RRE: A Game-Theoretic Intrusion Response and Recovery Engine

Saman A. Zonouz, Himanshu Khurana, William H. Sanders, *Fellow*, *IEEE*, and Timothy M. Yardley

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. The RRE applies attack-response trees (ART) to analyze undesired system-level security events within host computers and their countermeasures using Boolean logic to combine lower level attack consequences. In addition, the RRE accounts for uncertainties in intrusion detection alert notifications. The RRE then chooses optimal response actions by solving a partially observable competitive Markov decision process that is automatically derived from attack-response trees. To support network-level multiobjective response selection and consider possibly conflicting network security properties, we employ fuzzy logic theory to calculate the network-level security metric values, i.e., security levels of the system's current and potentially future states in each stage of the game. In particular, inputs to the network-level game-theoretic response selection engine, are first fed into the fuzzy system that is in charge of a nonlinear inference and quantitative ranking of the possible actions using its previously defined fuzzy rule set. Consequently, the optimal network-level response actions are chosen through a game-theoretic optimization process. Experimental results show that the RRE, using Snort's alerts, can protect large networks for which attack-response trees have more than 500 nodes.

Index Terms—Intrusion response systems, network state estimation, Markov decision processes, stochastic games, and fuzzy logic and control

1 Introduction

The severity and number of intrusions on computer networks are rapidly increasing. Generally, incident-handling [1] techniques are categorized into three broad classes. First, there are intrusion prevention methods that take actions to prevent occurrence of attacks, for example, network flow encryption to prevent man-in-the-middle attacks. Second, there are intrusion detection systems (IDSes), such as Snort [2], which try to detect inappropriate, incorrect, or anomalous network activities, for example, 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 [3]. Therefore, to reduce the severity of attack damage resulting from delayed response, an automated intrusion response is required that provides instantaneous response to intrusion.

This paper is built upon our previous work [4]. 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 multistep, sequential, hierarchical, nonzero-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 received IDS alerts to evaluate various security properties of the individual host systems within the network. ARTs provide a formal way to describe host 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. It is noteworthy that despite the mathematical cost *minimization*

- *S.A. Zonouz is with the Department of Electrical and Computer Engineering, University of Miami, Coral Gables, FL 33143. E-mail: s.zonouz@miami.edu.*
- *H. Khurana is with the Integrated Security Technologies Division, Honeywell Automation and Control Systems Lab, Minneapolis, MN 55418. E-mail: himanshu.khurana@honeywell.com.*
- *W.H. Sanders and T.M. Yardley are with the Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL 61801. E-mail: {whs, yardley}@illinois.edu.*

Manuscript received 15 Sept. 2012; revised 7 Aug. 2013; accepted 7 Aug. 2013; published online 19 Aug. 2013.
Recommended for acceptance by X. Li, P. McDaniel, R. Poovendran, and G. Wang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2012-09-0868.
Digital Object Identifier no. 10.1109/TPDS.2013.211.

in RRE that itself requires some time to complete in practice, RRE's ultimate objective is to *save/reduce* intrusion response costs and the system damages due to attacks compared to existing intrusion response solutions. 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 computers, 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. To support network-level intrusion response where the global security level is often a function of different specific properties and business objectives, RRE employs a fuzzy control-based technique that can take into account several objective functions simultaneously. In particular, reports from local engines are fed into the global response engine's fuzzy system as inputs. Then, the RRE calculates quantitative scores of the possible network-level response actions using its previously defined fuzzy rule set. The fuzzy rule set is defined using fuzzy numbers, and hence, various input parameters can take on qualitative values such as *high* or *low*; therefore, the real-world challenge that accurate crisp values of the involved parameters are not always known is addressed completely.

RRE extends the state of the art in intrusion response in three fundamental ways. 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 a partially observable Markov decision process that computes 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 to be practical. Third, for ease of design purposes, RRE allows network security administrators to define high-level network security properties through easy-to-understand linguistic terms for the particular target network. This is a crucial facility that RRE provides, because unlike system-level security properties, for example, the web server availability, which can be reused across networks, the network-level security properties usually should be defined specifically for each network by the security administrators manually. RRE achieves the above three 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, demonstrate that it is practical by studying commonly found power grid critical infrastructure networks. However, we believe that RRE has wide applicability to all kinds of networks.

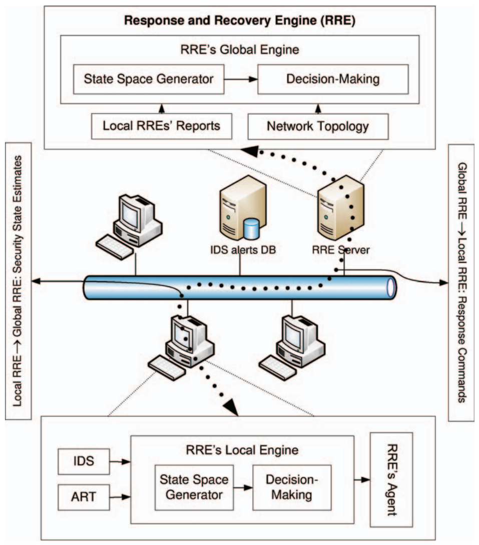


Fig. 1. High-level architecture of the RRE.

2 Problem Formulation

We formulate the optimal response selection as a decision-making problem in which the goal is to choose the cost-optimal response action at each time instant. The optimal action m is picked out of the set of all possible response actions $m \in \mathcal{M}$, including the No-OPeration (NOP) action. For example, an intrusion response system can respond to SQL's buffer overflow exploitation by closing its TCP connection. The optimization problem is solved in the response system, given the following inputs:

$\mathcal{W}$: a set of the computing assets $w \in \mathcal{W}$, for example, an 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, for example, alerts from Snort [2] warning about a packet transferring the Slammer worm [5] that exploits a buffer overflow vulnerability in an SQL server.

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

The following sections are devoted to a solution to the response selection problem; 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 at two different layers, i.e., local and global. This hierarchical structure of RRE's architecture, as 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.

At the first layer, RRE's local engines are distributed in host computers. Their main inputs consist of IDS alerts and attack-response trees. All IDS alerts are sent to and stored in the alert database (see Fig. 1) to which each local engine subscribes to be notified when any of the alerts related to its host computer is received. It is noteworthy that the current RRE design assumes that the triggered alerts are trusted. 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, which the host computer could be in, are generated. 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, they may become malicious themselves if they get compromised. Furthermore, it could become very complicated, even impossible, for local engines to choose and take a global network-level response action, due to their limited local knowledge. To deal with these problems, RRE's global engine, as its second layer, obtains 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. Additionally, if a local engine is detected to be compromised or does not respond, the RRE's global engine takes network level actions to prevent further damage, for example, to quarantine the compromised node, and/or possibly recover from the attack, for example, to switch to the secondary repilca of the compromised node in the network. In addition to local security estimates from host computers, network topology as well as the global network access control policies are also fed into the global engine. RRE converts the network topology and access control policies into the competitive Markov decision process (CMDP) model automatically. Moreover, security administrators define the network security properties as a function of the security of the network's critical assets using easy-to-understand linguistic terms. RRE employs the defined network-level security properties as security metrics to select the optimal network-level response action by solving the generated network CMDP model.

The ART model in the global server within RRE formulates the high-level organizational objectives that are subjective and require human involvement by the security administrators to capture the attack consequences that affect those objectives. For instance, confidentiality of a logging server in a financial institute may be considered as a critical security property while it could be ignored in a process control network. Consequently, the single global ART model in RRE's global server needs to be designed manually; however, the local ART models within individual hosts, such as the Apache web server, capture the system level consequences, for example, the web server availability. Hence, the local ART models can be reused across systems in different networks as they are not dependent on the high-level objectives. The reusability of the ART models reduces the manual endeavor requirement for the overall system deployment.

4 Local Response and Recovery

We present the 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.

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 violated as result of an attack, given a received set of alerts. Attack trees [6] 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 for each computing asset, for example, an 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*, for example, an 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 $w \in \mathcal{W}$ is to define and analyze possible combinations of attack consequences that lead to violation of some security property of the asset. This security property, for example, integrity, is assigned to the root node of the tree that is also called the *top-event* node. In the current implementation of RRE's local engines, there are at most three ART graphs $\mathcal{G}_w = \{g_w^c, g_w^i, g_w^a\}$ 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 subconsequences. A node decomposition scheme could be based on either 1) an `AND` gate, where all of the subconsequences

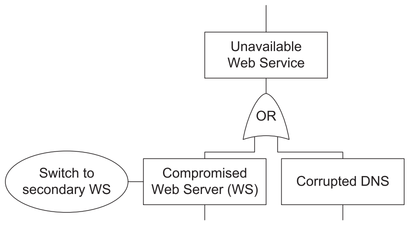


Fig. 2. Node decomposition in ART.

must happen for the abstract consequence to take place, or 2) an OR gate, where occurrence of any one of the subconsequences will result in the abstract consequence. For a gate, the underlying subconsequence(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 graph are tagged by response boxes that represent countermeasure (response) actions $m \in \mathcal{M}$ against the consequences to which they are connected. Fig. 2 illustrates how a sample abstract consequence node (output), i.e., an unavailable web service, is decomposed into two subconsequences (inputs) using an OR gate; this means that 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. Fig. 3 shows how a typical ART would finally look.

For every ART graph, a major goal is to probabilistically verify whether the security property specified by 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 in point, if the response box, that is connected to ART's root node is triggered, all nodes in the ART graph are reset to zero.

Dealing with uncertainties. In reality, determining 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 [7], 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 $\mathcal{O}_l^r \subseteq \mathcal{O}_l$:

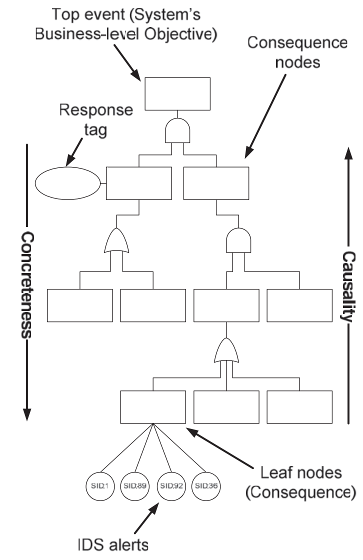


Fig. 3. Attack response tree.

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

where $\delta(l \mid O_l^r)$ represents the probability that the leaf node l's corresponding event has actually occurred given that the alerts O_l^r have been triggered by the IDSes. $P(l)$, 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* [8], 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, for example, 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 which verified that consequence l had occurred, and alert o_l had been received.

Given the satisfaction probabilities of leaf nodes, the output probability of gate q with inputs $i \in \mathcal{I}$ 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 $\delta(i)$ denotes the probability that the consequence denoted by the input i had occurred. In the above equation, 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., that is 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 Section 5). Next, we will explain how ART graphs and their nodes' satisfaction probabilities are used in a game-theoretic algorithm to decide on the optimal response action.

Stackelberg game: RRE versus attacker. Reciprocal interaction between the adversary and response engine in a computer system is 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 [9] 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)* [10].

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 nonempty set endowed with the discrete topology. A is set of actions, 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 $K := \{(s, a, s') \mid a \in A(s); s, s' \in S\}$, and P is the transition probability function; 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 $P(s' \mid s, a)$, an immediate reward $r(s, a, s')$ is obtained by the player taking the action. γ is the discount factor, i.e., $0 < \gamma < 1$.

Automatic conversion: *ART-to-MDP.* Using the ART graphs, RRE's local engines automatically construct response Markov decision process (MDP) 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 graph under consideration. In other words, as a binary string, each MDP security state vector represents the ART leaf node consequences that have already been set to 1 according to the received alerts from IDS systems. For instance, an ART graph with n leaf nodes results in a generated MDP model with 2^n security states, i.e., n-bit vectors. For ART graphs 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.

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, which are 0 in s'. For 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 graph. 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, as discussed later, may result in the choice of another, less expensive response action, or in taking no action at all.

The second type of state-action-state transitions in CMDPs are those due to adversarial actions. During automatic ART-to-CMDP conversion in RRE, each leaf consequence node l in the ART graph 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, but the bit related to l is 1.

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; moreover, reward functions $r : K \to \Re$ 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 state s':

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

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.

Having automatically generated CMDP using the ART graph, 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 ART graphs are mapped to subsets of IDS alerts, the system's current state s consists of 1s for bits which 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:

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

where $\mathcal{L}$ is the set of leaf nodes in the ART graph; $\delta(l)$ 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. As discussed, each belief state represents a probability distribution over state space $b : s \to [0, 1]$ s.t. $\sum_{s \in S} b(s) = 1$, i.e., the probability that the system in each state $s \in S$ when then system is in belief state b. The equation demonstrates that how RRE calculates the value of the belief state b for the state s that is by definition a vector of bits each of which represent individual security incidents (modeled by an ART leaf node). Assuming that the incidents are independent of each other, the probability that the system is in the state s is the multiplication of the probabilities that the each state notion bit is correct, i.e., $\delta(l)$ if the bit is 1 and $1 - \delta(i)$ otherwise.

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.

It is noteworthy that the rationale behind having the ART models within RRE rather than having the security administrators to design the state-based Markov decision processes manually is that the ART trees are easier to understand and hence to design manually partially because of their tree structure and recursive design process for individual subtrees. As discussed earlier, RRE uses the manually designed ART model to construct the CMDP state space automatically. In addition to ARTs' easier understandability, to design the Markov decision process requires more effort than designing its corresponding ART tree because the number of security states in a Markov decision process is exponentially more compared to the number of nodes (more accurately number of leaf nodes) in its corresponding ART tree.[1]

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 formulize 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 [13], 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 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: $V^*(b) = \max_{a_r \in A_r(b)} \Psi(V^*, b, a_r)$

$$\begin{aligned}\Psi(V, b, a) = \sum_{o \in \mathcal{O}} P(o \mid b, a) \cdot \{\rho\big(b, a, b'_{b,a,o}\big) \\ + \sqrt{\gamma} \cdot [\min_{a_a \in A_a(b'_{b,a,o})} \sum_{o' \in \mathcal{O}} P(o' \mid b'_{b,a,o}, a_a) \cdot \\ \big(\rho(b'_{b,a,o}, a_a, b''_{b',a_a,o'}) + \sqrt{\gamma} \cdot V\big(b''_{b',a_a,o'}\big)\big)]\},\end{aligned} \tag{6}$$

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 (6), in which ρ is the POCMDP reward function. ρ is computed using reward function r in the inherent CMDP: $\rho(b, a, b') = \sum_{s,s' \in S} b(s)b'(s')r(s, a, s')$. 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:

$$\begin{aligned}b'_{b,a,o}(s') &= P(s' \mid b, a, o) \\ &= \frac{P(o \mid s') \sum_{s \in S} P(s' \mid s, a) b(s)}{\sum_{s'' \in S} P(o \mid s'') \sum_{s \in S} P(s'' \mid s, a) b(s)},\end{aligned} \tag{7}$$

1. Recall that each CMDP state is represented by a bit-vector in which each bit denotes an individual ART leaf node; therefore, a manually designed ART tree with n leaf nodes results in a CMDP model with 2^n security states.

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

$$P(o \mid s) = \prod_{l \in \mathcal{L}} (1_{[s_l=1]} \cdot P(o \mid l) + 1_{[s_l=0]} \cdot P(\bar{o} \mid l)), \qquad (8)$$

where $P(\bar{o} \mid l) = 1 - P(o \mid l)$, and $P(o \mid l)$ is simply obtained using (2). Once the partially observable decision process is formulized, the optimal response action is chosen based on the optimal value function. There are different techniques to obtain the optimal value function. The decision-making unit in RRE uses a value iteration technique [14] $V_t(b) = \max_{a_r \in A_r(b)} \Psi(V_{t-1}, b, a_r)$, that applies dynamic programming updates to gradually improve on the value until it converges to the ε-optimal value function, i.e., $\mid V_t(b) - V_{t-1}(b) \mid < \varepsilon$. 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:

$$\pi^*(b) = \arg \max_{a_r \in A_r(b)} \Psi(V^*, b, a_r), \qquad (9)$$

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 graph.

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 are sent by local engines to RRE's server consist of root probabilities δ_g, as computed in (3), of local ART graphs. 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.

Agents. In the above-mentioned 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 commander, i.e., 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.

5 GLOBAL RESPONSE AND RECOVERY

Although host-based intrusion response is taken into account by RRE's local engines using local ART graphs and the IDS rule-set for computing assets, for example, 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. As discussed, in the distributed local response engines, most of the security properties (ARTs' root nodes) are (objective) system-level concepts, for example, *Is the apache process available?*, and can be measured simply using the Boolean logic expressions (ART trees) and the triggered IDS alerts. In RRE, global network intrusion response is resolved in the central server. Unlike in local engines, in the global intrusion response engine, global network-level (possibly subjective) security properties, for example, *Is the network currently secure?*, are to be determined. Such global security properties do not always take on only binary values. As a case in point, in a large scale enterprise network, a web server compromise affects the network's current security level, but it does not mean that the network is completely insecure. Additionally, various network assets often have different levels of criticality and impact on accomplishment of the enterprise's overall business objective, and hence, affect the global security level differently.

To address the above-mentioned challenges in assessing the global security properties for the whole network given the real-time reports from local response engines, we propose a multiobjective *network* security reward function $\delta_n : S \rightarrow [0, 1]$ (see Section 5.2) that uses a fuzzy logic-based controller to calculate, at each time instant, a security measure value for the whole network. Throughout this paper, we will concentrate on the generic security property of *"Is the network currently secure?"*. All other network-level security properties can be calculated similarly.

5.1 Automatic CMDP Generation

To generate the CMDP model, RRE analyzes the network topology input to find out about the set of known system vulnerabilities and individual host computers, i.e., privilege domains. Given the set of system vulnerabilities, the connectivity matrix is updated accordingly to encode adversarial paths only. In particular, RRE automatically generates a CMDP by traversing the connectivity matrix and concurrently updating the CMDP. First, RRE creates the CMDP's initial state ($\emptyset$) and starts the CMDP generation with the network's entry point (Internet) node in the connectivity matrix. Considering the connectivity matrix as a directed graph, RRE runs a depth-first search (DFS) on the graph. While DFS is recursively traversing the graph, it keeps track of the current state in the CMDP, i.e., the set of privileges already gained through the path traversed so far by DFS. When DFS meets a graph edge $[i, j]$ that crosses over privilege domains w_i to w_j, a state transition in CMDP is created if the current state in CMDP does not include the privilege domain of the host to which the edge leads, i.e., w_j. The transition in CMDP is between the current state and the state that includes exactly the same privilege set as the current state plus the host w_j directed by the graph edge $[i, j]$. The CMDP's current state in the algorithm is then updated to the latter state, and the algorithm proceeds until no further updates to CMDP are possible according to the connectivity matrix.

In addition to the adversarial transitions, the above algorithm also updates the CMDP regarding possible response and recovery actions $m \in \mathcal{M}$. In particular, host redundancies, specified by the network topology input, help RRE to create responsive state transitions. As a case point, consider that for a web server in an enterprise network there exists a redundant hot spare server designated for intrusion tolerance purposes. To model such a proactive design, RRE creates a responsive state transition, denoting the *recover the web server* action, from any state in which the server is compromised to states containing the same privileges except the web server. At that point, the offline CMDP generation is complete, and by design, the CMDP includes all possible attack paths launching from remote (Internet) host systems against the network as well as response and recovery scenarios. Matrix elements are denoted as red arrows among network component pairs, and as illustrated, a redundant web server can take on web request processing if the main server gets compromised.

5.2 Multiobjective System Security Reward Function

Local engines send their local security estimates, i.e., root node probabilities δ_g of their ART graphs, to the RRE server. RRE considers the network's global security as a multiobjective reward function for the response selection procedure. Each objective is represented by a specific system-level security property, and quantified by the δ_g values, which are calculated in the local engines. In our multiobjective game scheme, there is usually not a single solution that simultaneously minimizes each objective to its fullest. In each case, we are looking for a solution for which each objective has been optimized to the extent that if we try to optimize it any further, then the other objective(s) will suffer as a result. RRE makes use of a fuzzy-logic based controller that merges the involved objective function values using an information fusion algorithm according to the network security definition, and consequently, result in a single scalar reward value.

Fuzzy logic is a form of multivalued logic derived from fuzzy set theory to deal with reasoning that is approximate rather than precise. In contrast with binary sets which follow the binary logic, the fuzzy logic variables may have a membership value of not only 0 or 1. Just as in fuzzy set theory, with fuzzy logic, the set membership values can range (inclusively) between 0 and 1, and the degree of truth of a statement, for example, *The network is currently secure.*, can range between $0:\texttt{false}$ and $1:\texttt{true}$ and is not constrained to only two digital values as in classic propositional logic. In particular, RRE calculates the global network security level, i.e., the truth degree of the *"The network is currently secure"* predicate, using a fuzzy control system [15] that analyzes analog input values in terms of logical variables (system-level security properties) from local response engines that take on continuous values δ_g, and produces the network-level security measure values.

Formally, inputs to the fuzzy controller, that is in charge of calculating the global network-level security measure values for individual network security states $\delta_n : S \rightarrow [0, 1]$, represent root node values of the ART trees within the local response engines δ_g $g \in \mathcal{G}$. Before getting into technical details, as a clarifying example scenario, let us consider that the fuzzy controller defines the global network-level security as a function of two inputs: $A(WS)$: availability of the web server and $I(DB)$: integrity of the database server. So, given degrees of the system availability, for example, $A(WS) = \texttt{high}$, and system integrity, for example, $I(DB) = \texttt{low}$, in a sample network belief state b, the fuzzy controller computes the security status of the system, for example, $\delta_n(b) = \texttt{medium}$.

The global engine's fuzzy controller is composed of the following four elements:

1. A rule-base (a set of If-Then rules), which contains a fuzzy logic quantification of the experts linguistic description of how to achieve accurate global network-level security measure estimates.
2. An inference module, which emulates the experts' decision-making in interpreting and applying knowledge about how best to estimate the global network-level security measure values.
3. A fuzzification interface, which converts the controller inputs δ_g from local response engines into information that the inference mechanism can easily use to activate and apply rules.
4. A defuzzification interface, which converts the conclusions of the inference mechanism into real number values as inputs to the game-theoretic intrusion response system to pick the cost-optimal response action.

Here, we will discuss each of the above-mentioned components of a fuzzy control system according to our game-theoretic security battle formulation.

5.2.1 System Security Rule-Bases

Rule-base uses a linguistic qualification technique to specify a set of rules that capture the expert's evaluation of each belief state's security level in Stackelberg game scheme used by the intrusion response system (discussed in Section 4). It is noteworthy that, as discussed below, the expert's input on individual states (of the infinite continuous state space) is not required, and instead, an abstract discretized state space, with manageable number of states, is used for querying the expert. The general form of the linguistic rules is:

If premise **Then** consequent

The premises are associated with the fuzzy controller inputs (i.e., the system-level security properties representing local ARTs' root nodes) and are on the left-hand side of the rules. The consequents are associated with the fuzzy controller output (e.g., global network security) and are on the right-hand side of the rules. As a case in point, following the earlier example on two-input fuzzy controller, we may define security measure rules as follows:

$$\begin{aligned}&\textbf{If } \text{WS-Availability is } \texttt{high} \textbf{ and } \text{DB-Integrity is } \texttt{low}\\&\textbf{Then } \text{Global Network Security is } \texttt{medium}.\end{aligned} \tag{10}$$

The rule listed above is a linguistic security metric rule, since it is formed solely from linguistic variables and values to qualitatively measure the security level of each state. The main benefit of using linguistic variables and values over traditional numerical security metrics [16] is that they are at

TABLE 1
A Sample Global Security Assessment Rule-Base with Two Inputs (WS-Availability and DB-Integrity)

WS-Availability/DB-Integrity	low	medium	high
low	low	low	medium
medium	medium	high	high
high	high	high	high

a level of abstraction that the system operators are often comfortable with in terms of specifying how to measure the global network security level under different circumstances.

A tabular representation of one possible set of rules for the above-mentioned example is shown in Table 1. The body of the table lists the qualitative values of the rules, and the left column and top row of the table contain the premise terms, i.e., availability and integrity. As a case in point, the $(3, 1)$ entry in Table 1 represents our earlier rule sample in (10).

5.2.2 Fuzzification of the Local Engine Reports

Fuzzy sets are usually defined by their membership functions that describe the certainty that an element belongs to the set as opposed to the traditional binary set memberships. For instance, a fuzzy membership could quantify whether (or more precisely, how much) any system situation (belief state) is a member of the *network global security* set. The fuzzification block converts the fuzzy system inputs, i.e., δ_g's from local engines, to fuzzy sets that are used by the fuzzy system's inference engine later. Similar to [17], RRE uses Gaussian and triangular membership functions. In particular, we define $\mu_l(\delta_g)$ to represent the degree that δ_g is `low`, i.e., belongs to the `low` set. And $\mu_m(.)$ and $\mu_h(.)$ membership functions are similarly defined for the `medium` and `high` sets. A Gaussian fuzzy set membership degree in name is defined as follows:

$$\mu_l(\delta_g) = \exp - \frac{(\delta_g - c_l)^2}{2\sigma^2}, \tag{11}$$

where c_l and σ are the fuzzy number mean and standard deviation and are assigned initially. In RRE, $\sigma = 0.1$, and as the inputs δ_g take values in the range $[0, 1]$, we assigned $c_l = 0.15$, $c_l = 0.5$, $c_l = 0.85$. Using (11), RRE determines the membership degree of the corresponding system-level security property. As a clarifying example, if the database's integirty is believed to hold with 0.15 based on the reports from the local engine and detection systems running on the database server, the database integrity is completely a member of the `low` set, i.e., $\mu_l(0.15) = 1$. Similarly, the membership functions for all the fuzzy system inputs are specified. In RRE, we use the same fuzzy sets, i.e., `low`, `medium`, and `high`, for all the inputs. Consequently, using these membership functions, RRE's fuzzification component converts individual inputs, i.e., δ_g's, into fuzzy set membership degrees so that they can be used by the global security assessment system.

5.2.3 Security Metric Inference

Having encoded the knowledge and preferences of the security administrators about the network in the rule-base, we explain how RRE's inference component uses the fuzzy rules in the rule-base to come to conclusions about system-wide security measure values.

Briefly, given a particular input vector by the local engines, for example, $(\delta_g(\textit{WS-Availability}), \delta_g(\textit{DB-Integrity})) = (0.15, 0.9)$, RRE uses an information fusion algorithm [15] to determine the subset of rules that are needed for the global security assessment. Briefly, the premises of individual rules are compared to the inputs to identify the rules that apply to the current situation, and the certainty that each rule applies is determined. Consequently, the recommendations of rules that apply to the current situation with higher certainty are taken into account more strongly.

RRE's information fusion component calculates the applicability of individual rules using the membership degree functions (see Section 5.2.2). In particular, RRE first quantifies the meaning of the premise of each rule that is composed of one or more input term(s). For instance, to quantify the rule shown in (10) for the above input $(0.85, 0.6)$, RRE calculates the $\mu_h(0.85) = 1.0$ and $\mu_l(0.6) = 0.005$ for the premise's input terms. The main remaining part is how to quantify the logical "**and**" operation that combines the meaning of two linguistic terms into a single premise membership function μ_{prem}. RRE picks the minimum [15] of the membership degrees of the input terms in each rule's premise as the certainty measure that the rule applies, and hence should contribute to the global network security assessment. Hence, in the above example, we will have $\mu_{prem} = 0.005$. In particular, we call a specific rule *ON* if its premise membership function is greater than zero. In the next step, the inference mechanism will combine the recommendations from the ON rules to come up with a single conclusion about the global network security measure estimate.

5.2.4 Overall System Security Measure Defuzzification

RRE's defuzzification component operates on the applicable rules produced by the inference module and combines their effects to provide the most certain output, i.e., system security level. The RRE employs the center of gravity (COG) defuzzification method [18] for combining the recommendations by all of the applicable rules. Let ξ_i denote the mean for the membership function of the rule i's consequent. The COG of a given belief state is calculated using the following equation:

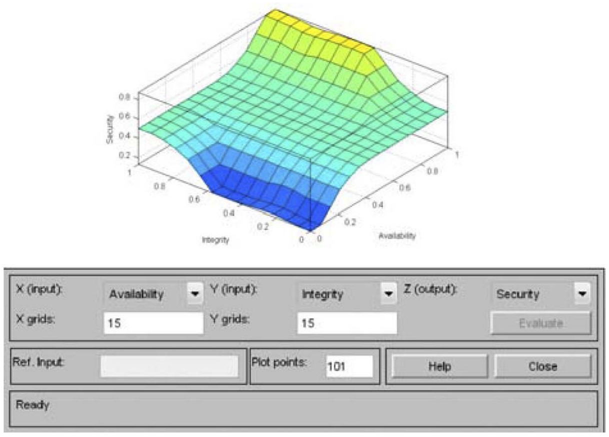
$$COG(b) = \frac{\sum_i^q \left[\xi_i \cdot \int_{-\infty}^{\infty} \mu^i(x, \mu^i_{prem}) \cdot dx\right]}{\sum_i^q \int_{-\infty}^{\infty} \mu^i(x, \mu^i_{prem}) \cdot dx}, \tag{12}$$

where q denotes the number of the ON rules, which are applicable to the current network security belief state. $\int_{-\infty}^{\infty} \mu^i(x, \mu^i_{prem}) \cdot dx$ denotes the area under the function μ^i that is defined as

$$\mu^i(x, \mu^i_{prem}) = \begin{cases} \mu^i_{cons}(x), & \text{if } \mu^i_{cons}(x) < \mu^i_{prem}, \\ x, & \text{otherwise,} \end{cases} \tag{13}$$

where μ^i_{cons} represents the rule i's consequent membership function (e.g., μ_m for the rule in (10)).

Fig. 4 shows an implementation of the RRE's defuzzification engine for our two-objective network security measure example. In particular, the X and Y axes represent



X (input): Availability Y (input): Integrity Z (output): Security
X grids: 15 Y grids: 15
Ref. Input: Plot points: 101
Ready

Fig. 4. The reward function fuzzy logic surface.

the web server availability and database integrity inputs, while the Z axis denote the global network security measure estimate that is calculated by the RRE's defuzzification component according to the provided rule set.

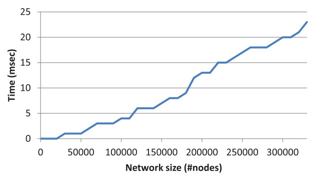
6 Experimental Evaluation

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 [2], 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.

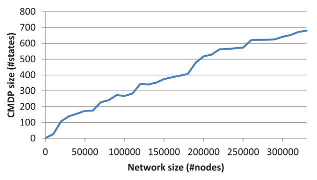
Model generation performance. Although network topology analyses and CMDP model generation in RRE is performed during an offline phase, for practical usages, it is still important to complete those steps within a reasonable time interval. To validate RRE's efficiency on various networks with different sizes and topologies, we measured how long RRE takes to generate the CMDP model for randomly generated networks. Figs. 5a and 5b show the CMDP generation time requirement and the model's size. The results were averaged over 1,000 runs. As illustrated, for large-scale power networks with 330K host computers, RRE analyzed the inputs and generated the CMDP model within 24 milliseconds.

Response optimization scalability. To evaluate how RRE handles complex networks consisting of a large number of host systems, we measured the time required by RRE to compute the optimal response action versus various metrics. Fig. 6d shows the average time-to-response over ten runs versus the attack-response tree order, i.e., the maximum number of children for each node.[2] 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. The ε-optimality termination criterion in Bellman's equation and discounting factor were set to $\varepsilon = 0.1$ and $\gamma = 0.99$, respectively. Then, a decision process is constructed and solved, and the total time spent is recorded (see Fig. 6d). As expected, the figure shows that increasing the ART order leads to rapid growth of the required time-to-response by the response engine.

2. The graphs, in this section, are not completely smooth as the results are averaged over only three runs.



(a) Generation overhead

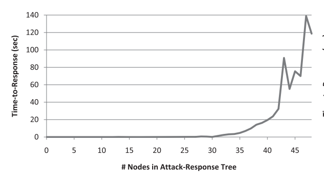


(b) Graph size

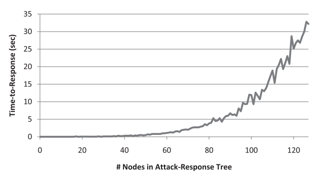
Fig. 5. Offline automated CMDP graph generation.

In another scalability evaluation experiment, we measured time-to-response versus the number of nodes in balanced ART trees of order 2. Fig. 6 shows average results on 10 runs for three scenarios. 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. 6a, 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 four steps was generated and solved. As illustrated in Fig. 6b, limited expansion improves a solution's convergence speed and increases the solvable ART size to trees with up to 120 nodes within 30 seconds. Third, to further improve RRE's scalability, we evaluated how fast a decision process is solved with an upper expansion limit of 2. Fig. 6c shows that ART trees with more than 900 nodes are still solvable in less than 40 seconds. By solving ART trees with about 900 nodes in a minute, RRE can protect large-scale computer networks.

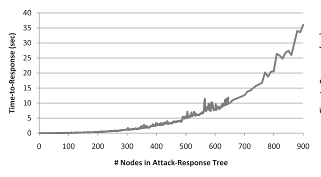
Comparison. This section evaluates how beneficial RRE is compared to static intrusion response systems, particularly those that statically choose and take response actions from a lookup table that stores alert-response mappings. In our experiments, both IRS systems, i.e., RRE and static engines, were given a sample ART tree with $|\mathcal{L}| = 6$ 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



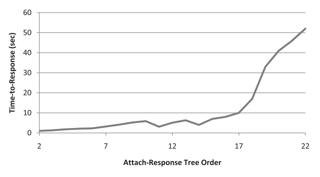
(a) Complete State Space Generated



(b) Finite Lookahead: 4 Steps



(c) Finite Lookahead: 2 Steps



(d) Tree Order vs. Time-to-Response

Fig. 6. Response system scalability analysis.

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 result graphs are omitted due to space constraint; 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 two actions. In five scenarios (out of 64), RRE caused more recovery cost than its static competitor, the reason being that the RRE chooses the optimal response action under the assumption that the action ratio is 1.

7 Related Work

EMERALD [19], a dynamic cooperative response system, introduces a layered approach to deploy monitors through different abstract layers of the network. Analyzing IDS alerts and coordinating response efforts, the response components are also able to communicate with their peers at other network layers. AAIRS [20] 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 [21], a host-based automated defense system, uses a partially observable Markov decision process to account for imperfect state information; however, αLADS is not applicable in general-purpose distributed systems due to their reliance on local responses and specific profile-based IDS. Balepin et al. [22] address an automated response-enabled system that is based on a resource type hierarchy tree and a directed graph model called a *system map*. Both αLADS and the IRS in [22] can be exploited by the adversary, since none 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 and Wing [23] use a game-theoretic method to analyze the security of computer networks. The interactions between an attacker and the administrator are modeled as a two-player simultaneous game in which each player makes decisions without the knowledge of the strategies being chosen by the other player; however, in reality, IDSes help administrators probabilistically figure out what the attacker has done before they decide upon response actions, as in sequential games. AOAR [24], 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 multistep security attacks in which the attacker significantly damages the system with an intelligently chosen sequence of individually negligible adversarial actions.

8 Conclusions

A game-theoretic intrusion response engine, called the response and recovery engine, 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. Experiments show that RRE efficiently takes appropriate countermeasure actions against ongoing attacks that save system damage and intrusion response cost compared to existing static and dynamic IRS solutions.

References

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

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

[3] F. Cohen, "Simulating Cyber Attacks, Defenses, and Consequences," http://all.net/journal/ntb/simulate/simulate.html, 1999.

[4] S.A. Zonouz, H. Khurana, W.H. Sanders, and T.M. Yardley, "RRE: A Game-Theoretic Intrusion Response and Recovery Engine," *Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks (DSN),* pp. 439-448, 2009.

[5] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm," *IEEE Security and Privacy,* vol. 1, no. 4, pp. 33-39, July/Aug. 2003.

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

[7] A. Valdes and K. Skinner, "Adaptive, Model-Based Monitoring for Cyber Attack Detection," *Proc. Recent Advances in Intrusion Detection,* pp. 80-92, 2000.

[8] C. Kruegel, W. Robertson, and G. Vigna, "Using Alert Verification to Identify Successful Intrusion Attempts," *Information Processing and Comm.,* vol. 27, pp. 220-228, 2004.

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

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

[11] S. Hsu and A. Arapostathis, "Competitive Markov Decision Processes with Partial Observation," *Proc. IEEE Int'l Conf. Systems, Man and Cybemetics,* vol. 1, pp. 236-241, 2004.

[12] L. Kaelbling, M. Littman, and A. Cassandra, "Partially Observable Markov Decision Processes for Artificial Intelligence," *Proc. German Conf. Artificial Intelligence: Advances in Artificial Intelligence,* vol. 981, pp. 1-17, 1995.

[13] E. Sondik, "The Optimal Control of Partially Observable Markov Processes," PhD thesis: Standford Univ., 1971.

[14] R. Bellman, *Dynamic Programming,* Princeton Univ. Press, 1957.

[15] D.F. Jenkins and K.M. Passino, "An Introduction to Nonlinear Analysis of Fuzzy Control Systems," *J. Intelligent and Fuzzy Systems,* vol. 7, no. 1, pp. 75-103, http://dl.acm.org/citation.cfm?id=1314668.1314674, Jan. 1999.

[16] E. LeMay, M.D. Ford, K. Keefe, W.H. Sanders, and C. Muehrcke, "Model-Based Security Metrics Using Adversary View Security Evaluation (Advise)," *Proc. Int'l Conf. Quantitative Evaluation of Systems (QEST),* pp. 191-200, 2011.

[17] R.C. Berkan and S. Trubatch, *Fuzzy System Design Principles,* first ed. Wiley-IEEE Press, 1997.

[18] S.-J. Chen and S.-M. Chen, "Fuzzy Risk Analysis Based on Similarity Measures of Generalized Fuzzy Numbers," *IEEE Trans. Fuzzy Systems,* vol. 11, no. 1, pp. 45-56, Feb. 2003.

[19] P. Porras and P. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *Proc. Information Systems Security Conf.,* pp. 353-65, 1997.

[20] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch, "Adaptation Techniques for Intrusion Detection and Intrusion Response System," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics*, pp. 2344-2349, 2000.

[21] O.P. Kreidl and T.M. Frazier, "Feedback Control Applied to Survivability: A Host-Based Autonomic Defense System," *IEEE Trans. Reliability*, vol. 53, no. 1, pp. 148-166, Mar. 2004.

[22] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using Specification-Based Intrusion Detection for Automated Response," *Proc. Int'l Symp. Recent Advances in Intrusion Detection*, pp. 136-154, 2003.

[23] K. Lye and J. Wing, "Game Strategies in Network Security," *Int'l J. Information Security*, vol. 4, pp. 71-86, 2005.

[24] M. Bloem, T. Alpcan, and T. Basar, "Intrusion Response as a Resource Allocation Problem," *Proc. Conf. Decision and Control*, pp. 6283-6288, 2006.

Saman A. Zonouz received the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 2011. He is an assistant professor in the Electrical and Computer Engineering Department at the University of Miami. He has worked on intrusion response and recovery, information flow-based security metrics for power-grid critical infrastructures, online digital forensics analysis and monitorless recoverable applications. His research interests include: computer security and survivable systems, control/game theory, intrusion response and recovery systems, automated intrusion forensics analysis, and information flow analysis-based security metrics.

Himanshu Khurana received the MS and PhD degrees from the University of Maryland, College Park. He is the senior technical manager for the Integrated Security Technologies section at Honeywell Automation and Control Systems Research Labs, where he is currently working on developing security solutions for messaging systems and the electric grid infrastructure, among other things. Before joining Honeywell, he was a principal research scientist at the Information Trust Institute at the University of Illinois, Urbana-Champaign, and served as the coprincipal investigator and principal scientist for the Trustworthy Cyber Infrastructure for Power center.

William H. Sanders is a Donald Biggar Willett professor of engineering and the director of the Coordinated Science Laboratory (www.csl.illinois.edu) at the University of Illinois at Urbana-Champaign. He is a professor in the Department of Electrical and Computer Engineering and affiliate professor in the Department of Computer Science. He was the founding director of the Information Trust Institute (www.iti.illinois.edu) at Illinois. His research interests include secure and dependable computing and security and dependability metrics and evaluation, with a focus on critical infrastructures. He is a fellow of the IEEE and the ACM, a past chair of the IEEE Technical Committee on Fault-Tolerant Computing, and past vice-chair of the IFIP Working Group 10.4 on Dependable Computing.

Timothy M. Yardley is the assistant director for Testbed Services and a senior researcher and lead for many projects in ITI. He works to define the vision and direction for testbed initiatives under ITI and engages in research to address ITI's mission. His work addresses trustworthiness and resiliency in critical infrastructure, with a particular focus on cyber security, and includes analysis and development of techniques for securing components, systems, and networks. His research in those areas includes work on control systems, telecommunications systems, critical incident response, and simulations of real-world systems.