Model-Based Response Planning Strategies for Autonomic Intrusion Protection

STEFANO IANNUCCI, Mississippi State University
SHERIF ABDELWAHED, Virginia Commonwealth University

The continuous increase in the quantity and sophistication of cyberattacks is making it more difficult and error prone for system administrators to handle the alerts generated by intrusion detection systems (IDSs). To deal with this problem, several intrusion response systems (IRSs) have been proposed lately. IRSs extend the IDSs by providing an automatic response to the detected attack. Such a response is usually selected either with a static attack-response mapping or by quantitatively evaluating all available responses, given a set of predefined criteria. In this article, we introduce a probabilistic model-based IRS built on the Markov decision process (MDP) framework. In contrast to most existing approaches to intrusion response, the proposed IRS effectively captures the dynamics of both the defended system and the attacker and is able to compose atomic response actions to plan optimal multiobjective long-term response policies to protect the system. We evaluate the effectiveness of the proposed IRS by showing that long-term response planning always outperforms short-term planning, and we conduct a thorough performance assessment to show that the proposed IRS can be adopted to protect large distributed systems at runtime.

CCS Concepts: • **Security and privacy → Artificial immune systems**;

Additional Key Words and Phrases: Intrusion response system, autonomic intrusion protection

ACM Reference format:
Stefano Iannucci and Sherif Abdelwahed. 2018. Model-Based Response Planning Strategies for Autonomic Intrusion Protection. *ACM Trans. Auton. Adapt. Syst.* 13, 1, Article 4 (April 2018), 23 pages.
https://doi.org/10.1145/3168446

1 INTRODUCTION

According to the Akamai's state of the Internet 2015 Q3 Report [2], there has been a 179.66% increase in total DDoS attacks with respect to the same period in 2014. Security mechanisms, such as firewalls, encryption, and properly configured access control policies have quickly shifted from being the defense mechanisms to being just the first line of defense [24]. The second line of defense is usually represented by signature-based or anomaly-based network intrusion detection systems (IDSs). The former are able to scan the content of the network packets looking for signatures of known attacks but are unable to identify unknown (0-days) attacks. To this end, anomaly-based

This work was partially supported by the Pacific Northwest National Laboratory under U.S. Department of Energy contract DE-AC05-76RL01830.
Authors' addresses: S. Iannucci, Mississippi State University, 665 George Perry Street, Mississippi State, MS, 39762; email: stefano@dasi.msstate.edu; S. Abdelwahed, Virginia Commonwealth University, 907 Floyd Avenue, Richmond, VA 23284; email: sabdelwahed@vcu.edu.

https://doi.org/10.1145/3168446

IDSs [5] have recently gained interest. They use machine learning or deep learning techniques [4] to find whether the protected system does not comply with the expected behavior and/or whether there is some anomaly in the network traffic flow.

The increasing number of cyberattacks makes it difficult and error prone for system administrators to manually handle all alerts generated by the IDSs. Intrusion response systems (IRSs) try to address this problem by automatically selecting the responses to the attacks detected by the IDSs [45]. Two main types of IRSs have been proposed so far [36]: static mapping and dynamic evaluation (e.g., [37, 41, 44]). With the static mapping type, the system administrators are expected to manually associate each category of detectable attacks with a prospective response action. However, periodically upgrading the response mapping can be overwhelming, given the massive amount of day-by-day newly discovered attacks and the ability of the attackers to bypass known protection mechanisms. The dynamic evaluation approach tries to overcome this limitation by letting the system administrator associate each single category of attack to a set of response actions. The response action is then chosen among the others according to an underlying system model and some evaluation criteria (e.g., the resolution time, cost, and impact) by solving a multi-objective optimization problem (e.g., [16]) or by ranking the alternatives (e.g., [9, 11, 16, 17, 35, 37, 41]).

Most of the works proposed so far either try to model the behavior of the attacker using attack graphs (e.g., [17]) or model the dependencies between the system components (e.g., [44]), but only a few of them introduce a comprehensive model able to describe the attacker behavior, the defender (IRS) behavior, and the actual system dynamics (e.g., [47]). Having a full model of the system associated with a control framework has several advantages, among them the possibility to simulate the behavior of the controlled system and to estimate its evolution over time [1].

In this article, we use the Markov decision process (MDP) framework to model a system controlled by an IRS. Unlike other approaches, we do not select a single short-term optimal response action; rather, we produce an optimal long-term policy—that is, an optimal sequence of response actions able to drive the system from its initial (under attack) state to a set of target (desired) states. We use the model to simulate the behavior of the system, and we show that long-term policies always outperform short-term policies by estimating average values and confidence intervals of the attack resolution time, cost, and impact for a system subject to real-world attacks. In addition, as an extension to our previous works [21, 22], we observe that being able to proactively react to a potential threat before it occurs is often better than waiting for it to reveal. To this end, we extend the previous single-agent MDP formulation, which only describes the system and the IRS behavior, by adding the attacker behavior to the model, using a competitive multiple-agent MDP implemented as a stochastic game [7]. We show that when this model is adopted, the IRS is either able to fully prevent an attack or at least anticipate some defense actions before the attack actually occurs.

Since the MDP state space grows exponentially with the number of features used to describe the system and the attacker states, we propose an algorithm able to instantiate the minimal MDP—that is, the MDP characterized by the minimum number of attributes and actions needed to drive the system to a secure state. Furthermore, we compare both the performance and the effectiveness of the single-agent formulation using state-of-the-art optimal and a suboptimal MDP planners implemented in the BURLAP library.[1] In addition, we extend the BURLAP library with a parallel Java implementation of the value iteration (VI) algorithm [3], which scales linearly with respect to the number of worker threads. The latter complements our previous work [23] on the performance

[1]Brown-UMBC Reinforcement Learning and Planning (BURLAP): http://burlap.cs.brown.edu/.

evaluation of VI on Intel MIC architecture [13]. Finally, we evaluate the performance of an optimal planner in the multiple-agent case.

The remainder of the article is organized as follows. The metamodeling framework and the system models are presented in Section 2. In Section 3, we compare the performance and the effectiveness of optimal and suboptimal MDP solvers and propose an algorithm for reducing the MDP state space. Detailed response policies evaluations are presented in Section 4. In Section 5, we present a discussion of related work. Section 6 concludes the work.

2 SYSTEM MODEL

The proposed system model is derived from the MDP framework. In the following, we provide an overview of the theoretical foundations of the single-agent and multiple-agent MDPs and explain how we use this framework to build a system model representing the behavior of an IRS when trying to protect a system from cyberattacks.

2.1 The MDP Modeling Framework

A single-agent discrete-time MDP is a stateful probabilistic approach to model the behavior and the runtime dynamics of a system. An MDP [3] is a tuple $\langle S, A, P, R, \gamma \rangle$, where S represents the state space that the agent can navigate and $s_k \in S$ represents the agent state at discrete-time k. Even if not explicitly considered in the MDP framework, a common practice is to characterize each state with a number of attributes. A is the finite set of actions available to the agent to navigate the state space. Specifically, by executing at time k an action $a_k \in A$ in the current state $s_k \in S$, the agent moves to a successor state $s_{k+1} \in S$. The transition dynamics from the current to the next state are given by the transition probability function P. This function specifies, for each source state $s_k \in S$, for each destination state $s_{k+1} \in S$, and for each action $a_k \in A$, the value $P(s_k, a_k, s_{k+1})$—that is, the probability value that by executing the action a in state s at time k, the resulting state will be s_{k+1}. When the transition probability function is time independent (*stationary*), we have $\forall k.P(s_k, a_k, s_{k+1}) = P(s_k, a, s_{k+1})$. Every time an action is executed, the MDP agent is rewarded with a bonus (or penalized with a cost), according to the reward function R. In other words, $R_k = R(s_k, a_k, s_{k+1})$ represents the reward that the agent will earn (or the cost the agent will pay) for executing at time k the action a in state s_k and being taken to some state s_{k+1}. Some MDP models use a different reward function, based only on s_k and a_k and not considering s_{k+1}. When the reward function is time independent (*stationary*), we have $\forall k.R(s_k, a_k, s_{k+1}) = R(s_k, a, s_{k+1})$. γ is the discount factor, usually defined in the interval $[0, 1]$, which specifies how much short-term rewards are preferred over long-term rewards.

The overall behavior of the agent is described by a deterministic or stochastic policy π. When π is deterministic, it specifies, for each s_k, the action a_k that the agent must execute. When π is probabilistic, it specifies a probability distribution such that $\pi : S \times A \rightarrow [0, 1]$. The objective of the agent is to find a policy π^* such that the discounted reward $R_k = \sum_{j=0}^{\infty} \gamma^j R_{k+j+1}$ is maximized.

Several optimal and suboptimal algorithms for solving MDPs have been proposed (e.g., [3, 27, 29, 31, 38]), but one of the most commonly used remains the VI algorithm [3] because of its simplicity. It is based on the concept of *state-value* function $V_\pi(s_k) = \mathbb{E}_\pi[R_k|s_k]$—that is, the expected reward achievable by the agent starting from state s_k and then following policy π. The base step of the algorithm is to assign an initial random state-value V^0 to all states and then execute the iterative refinement process described in Boutilier [6]:

$$V^{i+1}(s_k) = \max_{a_k \in A} R(s_k) + \gamma \sum_{s_{k+1} \in S} P(s_k, a_k, s_{k+1}) V^i(s_{k+1}). \quad (1)$$

The sequence of functions V^i converges linearly to the optimal value V^* in the limit and thus provides the expected maximum reward obtainable by following the optimal policy π^* from state s_k.

A multiple-agent discrete-time Markov decision process (MA-MDP), also known as a stochastic game, is an extension of the single-agent MDP. An MA-MDP with n agents is defined by the tuple $\langle S, A_1, \ldots, A_n, P, R_1, \ldots, R_n, \gamma \rangle$. With this formulation, different agents have possibly different sets of available actions and can have potentially different reward function. The transition from s_k to s_{k+1} is therefore given by the joint action of all agents: $a_k = [a_{k,1}, \ldots, a_{k,n}]$. According to the reward functions definitions, the agents could cooperate to reach a common objective or could behave selfishly (i.e., trying to achieve personal goals at the expense of the other agents). The former case is described in detail in Boutilier [6], whereas the competitive case is described in Busoniu et al. [7].

2.2 System and IRS Modeling With MDP

In this work, we use the single agent and stationary MDP modeling framework to model the behavior of a system responding to an attack. The agent represents the IRS, whose objective is to find an optimal policy to drive the system from a dangerous (under attack) state to a final (desired) state. We introduce two extensions to the general MDP framework: a *termination function* T and the *precondition function* PC. The termination function $T : S \rightarrow \{true, false\}$ is used to define the subset $S_{tgt} = \{s \in S | T(s) = true\}$ of the target states, which represents the set of the states where the agent stops its execution; the precondition function $PC : S \times A \rightarrow \{true, false\}$ is instead used to define whether an action $a \in A$ is executable in state $s \in S$. T and PC are introduced to simplify notations. They can be expressed directly in the base model with a target state $s_k \in S_{tgt}$ formulated as a state where $\forall a.P(s_k, a, s_{k+1}) = 0$, whereas a nonexecutable action a has the characteristic $\forall s_{k+1}.P(s_k, a, s_{k+1}) = 0$. The objective of the MDP agent is to drive the system from a starting state s to a target state $s' \in S_{tgt}$ such that the path between s and s' maximizes its reward.

In the following, we describe how we modeled a simple system, characterized by 14 system attributes and that could be subject to seven different attacks.

2.2.1 States Characterization. We use the object-oriented MDP representation introduced in Diuk et al. [12] in which each state is characterized by several attributes. Specifically, we compose the states by joining two macroattributes: the attack vector $\mathbf{p}$ and the system variables $\mathbf{v}$. The former contains as many variables as the number of attacks detectable by the IDSs, and each variable $p_i \in \mathbf{p}$ represents the probability value that the system is currently under attack i. The latter represent the current system status.

We consider seven different attacks and 14 system attributes. The attacks are modeled by the attributes p_{scan}, p_{vsftpd}, p_{smbd}, p_{phpcgi}, p_{ircd}, $p_{distccd}$, p_{rmi}, which represent the probability that the controlled system is being attacked respectively by a port scan, an exploit on the `vsftpd` daemon (OSVBD-73753), an exploit on the `smbd` daemon (CVE-2007-2447), an exploit on the execution of PHP as a CGI application (CVE-2012-1823), an exploit on the `ircd` daemon (CVE-2010-2075), an exploit on the `distccd` daemon (CVE-2004-2687), and finally an exploit on the `rmi` Java daemon (CVE-2011-3556). We specifically chose these attacks because their respective vulnerabilities are exposed by `metasploitable`,[2] an intentionally vulnerable Linux Virtual Machine that can be used to conduct security training, test security tools, and practice common penetration testing techniques. We consider the following system attributes:

[2]Virtual machine to test Metasploit: https://information.rapid7.com/metasploitable-download.html.

- $firewall \in \{true, false\}$ represents whether the system firewall is active.
- $\{blocked\_ips\}$ represents the set of currently blocked source IP addresses from the firewall of the considered system.
- $\{flowlimit\_ips\}$ represents the set of currently throughput-limited source IP addresses.
- $alert \in \{true, false\}$ represents whether the system administrator has been alerted about the ongoing attack.
- $\{honeypot\_ips\}$ represents the set of IP addresses whose traffic is currently being redirected to an honeypot.
- $logVerb \in \{0, 1, 2, 3, 4, 5\}$ represents the currently configured logging verbosity of the applications installed on the considered system.
- $active \in \{true, false\}$ represents whether the considered system is currently active and serving requests or if it has been shut down.
- $quarantined \in \{true, false\}$ represents whether the considered system is currently active and serving requests or if it has been isolated from the network.
- $rebooted \in \{true, false\}$ represents whether the considered system has been rebooted during the execution of the current policy.
- $backup \in \{true, false\}$ represents whether the considered system has ever been backed up during the execution of the current policy.
- $updated \in \{true, false\}$ represents whether the software installed on the controlled system is updated.
- $manuallySolved \in \{true, false\}$ represents whether there has been a manual intervention during the execution of the current policy.
- $everQuarantined \in \{true, false\}$ represents whether the system has been quarantined during the execution of the current policy.
- $everShutDown \in \{true, false\}$ represents whether the system has been shut down during the execution of the current policy.

2.2.2 Reward Function. Although several works aim at addressing the problem of evaluating the response cost to counter or mitigate an intrusion, a standard methodology has not yet been determined [43]. A common approach for cost evaluation is to take into consideration the effectiveness of the response action as in Ossenbuhl et al. [37] or to deal with the negative impact that it can have on the system [15, 18]. A third parameter usually contemplated for this evaluation is the operational cost that must be sustained to pay for hardware, software, and human resources needed to counter the attack [30, 43]. However, there are other factors that should be considered when planning a defense policy, among others: the confidentiality, integrity, availability (CIA) triad [10, 40] and the service-level agreement (SLA) [34] that the system is supposed to meet. The former is related to the data, which could be released, modified, or made inaccessible without authorization, causing confidentiality, integrity, and availability issues, respectively. The latter is instead related to the quality of service (QoS) that must be provided to the end users in terms of nonfunctional requirements such as applications response time and system reliability [8].

In this work, we consider all aforementioned attributes with the exception of the CIA triad for the defender's reward function. The latter is instead considered for the attacker's reward function and evaluated as discussed in Section 2.3.3. Furthermore, we also consider the response time—that is, the expected time needed to execute the defense policy on the target system. Specifically, we characterize the MDP reward function as a penalty score on the actions considered for inclusion in the defense policy. The reward function evaluates the response actions according to the following criteria:

- Response time $T(x) \in \mathbb{R}$ represents the time needed to apply the response action x.
- Cost $C(x) \in \mathbb{R}$ represents the operational cost of applying the response action x.
- Impact index $I(x) \in [0, 1]$ represents the impact index of the response action x on the system and is computed as follows:

$$\begin{cases} I(x) = w_R \frac{R(x)-R_0}{R_{max}} + w_D \frac{D_0-D(x)}{D_{min}}, & R(x) \leq R_{max}, D(x) \geq D_{min} \\ I(x) = 1, & otherwise, \end{cases}$$

  where $R(x)$ and $D(x)$ are the actual applications' execution time and reliability after the execution of the action x, R_0 and D_0 respectively are the applications' execution time and reliability during normal operations, and $\forall x. R(x) \geq R_0, D(x) \leq D_0$; R_{max} and D_{min} respectively are the SLA upper limit for the response time and the lower limit for the reliability, and w_R and w_D are custom weights with $w_R + w_D = 1$.

The reward function is then defined as follows:

$$R_{irs} = -w_t \frac{T(x)}{T_{max}} - w_c \frac{C(x)}{C_{max}} - w_i I(x), \tag{2}$$

where $w_t, w_c, w_i \in [0, 1]$ are custom weights used to balance the importance of the criteria in the multicriteria optimization problem. T_{max} and C_{max} respectively represent the maximum response time and the maximum cost over all considered response actions and are used to normalize their values. It is worth noting that in the reward function, we did not consider the effectiveness of the response action, because it is tightly integrated with the model of the system in the form of actions' postconditions, as described in Section 2.2.3.

2.2.3 Response Actions. To avoid activating potentially disruptive response actions when the system is not under severe attack and to better deal with the stochastic nature of the IDS inputs, we introduce two thresholds on the attack probability attributes, namely T_1 and T_2, $T_1 < T_2$. Thus, given an attack probability p, it can belong to one of the following four stages: ($p < T_1$) the IDSs have detected an insignificant anomaly that should be considered as noise, and no response actions should be triggered. With ($T_1 \leq p < T_2$), the IDSs have detected a significant anomaly, which cannot be classified as an attack. However, the system can start planning some response action to prevent possible attacks. With ($T_2 \leq p < 1$), the anomaly detected by the IDSs is considered to be an unidentified attack, and therefore the response plan generated by the IRS can only contain generic responses. When ($p = 1$), the attack has been identified and a specific response plan can be computed.

In the following, we describe some of the response actions that our IRS prototype is able to apply on the controlled system. For each of them, we provide a description of its behavior and the response time R, cost C, and impact I attributes, needed to compute the expected reward when planning the optimal policy. Each response is characterized by preconditions and postconditions. The former identify a subset of the states in which the actions can be executed; the latter are used instead to compute the state in which the system will be after the execution of the considered action. It is worth noting that although in this work we only consider statically defined transition probabilities as postconditions, it is possible to establish a feedback loop between the system and the IRS so that they can be updated at runtime to better mimic the actual system behavior. Eventual dependencies between response actions are not directly modeled: indeed, using preconditions, we are able to model the eventual dependency of a response action on a given subset of states, which in turn could imply that some dependent actions have been executed prior to the execution of the current action. Table 1 summarizes response time, cost, and impact attributes for the considered actions.

Table 1. Response Actions Parameter Summary

Action Name	Resp. Time	Cost	Impact
Generate Alert	1	1	0
Firewall Activation	2	1	0
Block Source IP	1	3	0.3
Unblock Source IP	1	3	0
Flow Rate Limit	3	1	0.2
Unlimit Flow Rate	3	1	0
Redirect to Honeypot	3	3	0.1
Un-honeypot	3	3	0
Increase Log Verbosity	2	1	0.05
Decrease Log Verbosity	1	1	0
Quarantine Host	5	5	1
Unquarantine Host	5	5	0
Manual Resolution	3,600	200	0
System Reboot	60	6	0.7
System Shutdown	30	6	1
System Start	30	6	0
Backup Host	3,600	10	0.1
Software Update	600	300	0.1

Firewall activation. This starts the system's firewall in case it was not started previously. Its characteristics are the following:

- *Reward attributes*: $T = 2, C = 1, I = 0$
- *Preconditions*: $(p_{scan} \geq T_1 \vee p_{vsftpd} \geq T_1 \vee p_{smbd} \geq T_1 \vee p_{phpcgid} \geq T_1 \vee p_{distccd} \geq T_1 \vee p_{rmi} \geq T_1 \vee p_{ircd} \geq T_1) \wedge \neg firewall \wedge \neg quarantined \wedge active \wedge logVerb > 0$
- *Postconditions*: $Prob = 1, firewall = 1$.

This action can be executed when at least one entry of the attack probability vector $\mathbf{p}$ is greater than or equal to T_1, the firewall itself has not been activated yet, the system is active and it has not been quarantined, and the log verbosity is at least equal to 1. The resulting state after the execution of the action will be reached with probability 1 and is identical to the current state but with the *firewall* attribute set to *true*.

Block source IP `badIP`. This configures the system's firewall to drop IP packets originated by the IP `badIP`. Its characteristics are the following:

- *Reward attributes*: $T = 1, C = 3, I = 0.3$
- *Preconditions*: $p_{scan} \geq T_2 \wedge firewall \wedge \neg quarantined \wedge active \wedge badIP \notin blocked\_ips \wedge alert \wedge logVerb > 1$
- *Postconditions*: $Prob = 1, blocked\_ips = blocked\_ips \cup \{badIP\}, p_{scan} = 0$.

This action can be executed when the port scan attack probability is greater than or equal to T_2 and the firewall has been previously activated. Furthermore, it is required that the system is active and has not been quarantined and that its log verbosity is at least equal to 2. Finally, the system administrator must have been previously alerted, and the IP address of the attacker must not yet belong to the set of the blocked IPs. The resulting state after the execution of the action is identical to the current state but with the `badIP` included into the set of the blocked IPs and with

p_{scan} attribute set to 0. Setting the probability of an attack to zero for the next state means that the expected result in executing the given action is to certainly stop the attack.

Flow rate limit `badIP`. This configures the system's firewall to limit the traffic rate of IP packets originated by the IP `badIP`. Its characteristics are the following:

- *Reward attributes*: $T = 3, C = 1, I = 0.2$
- *Preconditions*: $p_{scan} \geq T_1 \wedge firewall \wedge badIp \notin flowlimit\_ips \wedge \neg quarantined \wedge active \wedge logVerb > 0$
- *Postconditions*:

$$\begin{cases} Prob = 0.5, & limited\_ips = limited\_ips \cup \{badIP\}, \\ & p_{scan} = 0 \\ Prob = 0.5, & limited\_ips = limited\_ips \cup \{badIP\}. \end{cases}$$

This action can be executed when a port scan attack probability is greater than or equal to T_1 and the firewall has been previously activated. Furthermore, it is required that the system is active and has not been quarantined and that its log verbosity is at least equal to 1. Finally, the IP address of the attacker must not belong to the set of the flow rate limited IPs. This action can drive the system to two different resulting states, with probability 0.5 each. In one case, the action is able to stop the attacker, and therefore we have $p_{scan} = 0$ together with the attacker IP address included in the set of flow rate limited IPs. In the other case, the action is unable to stop the attacker, and therefore we only obtain to limit the flow rate of the attacker's IP by adding it to the set of the flow rate limited IPs.

2.2.4 Termination Function. The policy planning terminates when the system reaches a target state $S_{tgt} = S_a \cup S_c$, where S_a is the subset of states in which the anomaly is harmless and S_c is the subset of states representing a fully clean system. Both subsets are identified with a Boolean expression on the state attributes, but for space reasons we only report the Boolean condition representing the fully clean system state:

$S_c = \{s \in S | p_{scan} < T_1 \wedge p_{vsftpd} < T_1 \wedge p_{smbd} < T_1 \wedge p_{phpcgi} < T_1 \wedge p_{irc} < T_1 \wedge p_{distcc} < T_1 \wedge p_{rmi} < T_1 \wedge blocked\_ips = \emptyset \wedge flowlimited\_ips = \emptyset \wedge honeypot\_ips = \emptyset \wedge logVerb = 0 \wedge active \wedge \neg quarantined\}$.

A clean system state is represented by an attack probability vector whose values are all under the T_1 threshold and there are no firewall limitation configured.

2.3 Attacker Modeling With MA-MDP

The model described so far is able to capture the dynamics of the underlying system and can be used to plan optimal long-term policies to defend the system against an attack. However, even if the long-term policies always outperform short-term policies (more details are provided in Section 4), an IRS built on such a model is not able to anticipate, and thus prevent, a possible multistep attack because the model does not describe the attacker behavior.

The competitive multiple-agent extension of the model aims at introducing a proactive defense mechanism by describing the system and its dynamics when subject to control actions executed by both the IRS and the attacker. Knowing what actions are available to the attacker and their interdependencies allows for the planning of proactive long-term response policies, able to block ongoing attacks and prevent an attack escalation.

In this extended model, each attack is characterized by three factors: an *attack belief*, representing the probability that the attacker will launch a specific attack in the future; an attack action, based on preconditions that can specify dependencies on other attacks or on targets on the system; and the effects that the attack has on the system attributes.

The multiple-agent extension of the model, based on a two-agent stochastic game, inherits all characteristics of the single-agent model and extends the set of attributes, the set of the available actions, and the reward function, which is now based on a joint action model.

2.3.1 Extended Attributes. Among the new attributes, the most important are the *timer* and the *nextAttackThreshold* attributes. These are used to take into account the execution time of both attacks and responses in the system to simulate the coordinated attack-response behavior. Specifically, the *timer* attribute has been added to the IRS model, whereas *nextAttackThreshold* has been added to the attacker model. The main system timer is kept by *timer*. Its value is incremented each time a response action is executed from the IRS side. From the attacker side, a newly launched attack increases *nextAttackThreshold* with the expected time needed to complete the attack. The attacker will not be able to launch new attacks until its threshold is greater than the system timer.

Attack beliefs are characterized by probability values. We add to the attacker agent model as many attributes as the number of executable attacks.

2.3.2 Extended Actions. Unlike the single-agent model, the multiple-agent one is based on the concept of *joint action*. $(x, y)_k$ is a joint action for a two-agent stochastic game, where $x \in A_{irs}$ and $y \in A_{attacker}$ represent the actions chosen at time k by the IRS and the attacker, respectively. The new set of actions $A_{attacker}$, available only to the attacker agent, contains *attackVsftpd, attackSmbd, attackPhpcgi, attackIrcd, attackDistccd, attackRmi, noOp*. The first six actions model actual attacks toward the system, whereas the last action represents a *void* attack, used to describe an attacker waiting for a running attack to complete or for the preconditions of some attack to become true.

Due to space limitations, we describe only the *attackVsftpd* and the *noOp* actions, the others similar to the *attackVsftpd* action. Similarly to the response actions, the attack actions are characterized by preconditions and postconditions. Specifically, Boolean preconditions can be used to model multistage attacks, where the next stage can be subject to the achievement of some previous step. Each attack action is characterized by a response time—that is, the time needed for the attack to complete.

attackVsftpd. This exploits the vulnerability OSVBD-73753 to attack the *vsftp* daemon:

- *Preconditions*: $p_{vsftpd} < T_1 \wedge$
  $p_{scan} \geq T2 \wedge \neg softwareUpToDate \wedge irsTimer \geq nextActionTimer$
- *Postconditions*: $Prob = 1, p_{vsftpd} = 1$.

The preconditions illustrate that the attack is executable by the attacker if it is not currently being executed ($p_{vsftpd} < T_1$) and when the port scan has been completed successfully (having the attribute $p_{scan} \geq T2$ at the end of the port scan means that the IRS was unable to run any action to counter the port scan). Furthermore, to successfully exploit the vulnerability, the software must not be updated ($\neg softwareUpToDate$), and finally the attack can be executed only when any eventual previous attack has been completed ($irsTimer \geq nextActionTimer$). When all preconditions are verified and the attack is launched, the system gets compromised with probability 1 and therefore its p_{vsftpd} attribute is set to 1.

noOp. This models an attacker currently unable to run an attack because either no preconditions are currently verified for any of the attack actions or an attack is already running:

- *Preconditions*: $true$
- *Postconditions*: $\emptyset$.

Being just a *void* action, *noOp* can always be executed by the attacker and it does not have any postcondition because it does not have any effect on the system.

The extended model also requires some modification to all IRS response actions to make them able (i) to manage the time concept adding to the *timer* attribute the response time needed for their execution and (ii) to be executed even when an attack has not been detected yet. To this end, among the others, we change the preconditions of *backup* and *softwareUpdate*. We only present here the former, but the same considerations apply to the latter.

backup. The purpose of this action is to model a system executing a backup, needed as a precondition for executing the *softwareUpdate* action:

- *Preconditions*: $((p_{scan} \geq T1 \vee p_{vsftpd} \geq T1 \vee p_{smbd} \geq T1 \vee p_{phpcgid} \geq T1 \vee p_{ircd} \geq T1 \vee p_{distccd} \geq T1 \vee p_{rmi} \geq T1)$ $\boldsymbol{\vee attP_{vsftpd} \geq T2 \vee attP_{smbd} \geq T2 \vee attP_{phpcgid} \geq T2 \vee attP_{ircd} \geq T2 \vee attP_{distccd} \geq T2 \vee attP_{rmi} \geq T2)} \wedge \neg quarantined \wedge active \wedge alerted \wedge logVerb > 1 \wedge backup \wedge \neg softwareUpToDate$
- *Postconditions*: $Prob = 1, timer+ = responseTime(backup), backup = true.$

The bold symbols represent the newly added preconditions. Thus, in the extended model, the action is executable either when any of the currently detected attack attributes are at least equal to $T1$ or when there is any attack belief greater than or equal to $T2$. The postcondition increments the timer with the time needed to perform the backup, as defined in Table 1, and sets the *backup* attribute to true.

2.3.3 Joint Reward Function. A *joint reward function* $R_k = (R_{k,irs}, R_{k,attacker})$ is used to model the reward of the agents in the stochastic game, where $R_{k,irs}$ represents the reward achieved by the IRS and $R_{k,attacker}$ represents the reward achieved by the attacker, both at discrete timestep k. The IRS reward is computed with the same reward function described in Section 2.2.2, whereas the reward of the attacker is evaluated according to the Common Vulnerability Scoring System (CVSS) [33] using the CIA triad as follows:

$$R_{k,attacker} = w_{sc} Score_C + w_{si} Score_I + w_{sa} Score_A, \tag{3}$$

where $w_{sc}, w_{si}, w_{sa} \in [0, 1]$, $w_{sc} + w_{si} + w_{sa} = 1$ are custom weights and $Score_C, Score_I, Score_A \in \{0, 0.5, 1\}$ respectively are the confidentiality, integrity and availability scores related to the attack action. A score equal to 0 means that the attack does not have any impact on the system; 0.5 means that the attack action has a partial impact on the system (e.g., considerable informational disclosure, modification of some system files, and reduced information availability); 1 means that the attack can completely compromise the target system, by achieving either a total information disclosure or the ability to modify any file or the total unavailability of the system. A complete discussion on the evaluation of the CIA triad is reported in Mell et al. [33].

For the purpose of this work, we attribute the rewards $0, 0.5, 1$ respectively to the *noOp, portScanAttack, attackVsftpd* actions. The stochastic game solver, based on a multiple-agent version of the VI algorithm, is set to maximize the disjunct reward. As a result, we simulate two selfish systems where each one does not know the internals of the other. An alternative would be to simulate two selfish systems that exactly know the counterpart using a zero-sum stochastic game, where $R_{k,irs} = -R_{k,attacker}$. In the latter case, maximizing a reward of a system means minimizing the reward of the other.

3 PERFORMANCE EVALUATION

VI is one of the most-used algorithms to plan an optimal policy for single-agent and multiple-agent MDPs. It produces successive approximations of the optimal value function until the expected objective value is stable for all MDP states. Unfortunately, even if each iteration can be performed in $O(|A||S|^2)$ steps [26], the number of states composing the MDP grows exponentially with the number of the defined attributes. The BURLAP library provides an implementation for both single-agent and multiple-agent VI, as well as an implementation of a suboptimal rollout-based Monte Carlo planning algorithm named UCT [29]. However, since all provided implementations are single threaded, we extended the library by adding a multithreaded implementation of the single-agent VI algorithm.

In this section, we compare the performance, intended as the planning time and reward gap in comparison to the optimal case, of the following algorithms: single-threaded, single-agent VI; multithreaded, single-agent VI (in the following, Parallel-VI); single-threaded UCT; and single-threaded multiple-agent VI. We show that, while obviously suffering the exponential state growth like the single-threaded implementation, Parallel-VI is able to scale almost linearly with the number of available cores. For systems where a small reward loss is acceptable, instead the UCT algorithm provides the best performance, improving the planning time by more than three orders of magnitude. Finally, the single-threaded multiple-agent VI algorithm is the one requiring the longest planning time.

The policy planners have been applied on a system characterized by up to 1,000 Boolean state attributes and up to 1,000 response actions. Each action is bound to one attribute, and it changes its Boolean value when executed to generate the full state space. The termination condition is based on an additional *termination* attribute that can be set to *true* by any action with probability 1/10. The reward function assigns the reward −1 to the actions with an even index and −2 to the actions with an odd index. All tests have been executed on a single compute node of the Shadow supercomputer at Mississippi State University, characterized by 20 cores and 512GB of RAM. Only a single core has been used for the single-threaded VI and for UCT, whereas up to 10 threads have been run for Parallel-VI.

The multiple-agent game inherits the same structure of the single-agent case but with the following changes: (i) the *termination* attribute is replaced by an integer attack counter and all response actions are capable of decreasing the counter of a single unit with probability 0.1, and (ii) a second agent modeling the attacker has been added to the system. This is capable of executing two actions: *noOp* and *attack*. The former does not provide the agent with any reward, whereas the latter provides the maximum reward. When the *attack* action is successful (with probability 0.05), it increases the attack counter by a single unit. The game ends when the IRS agent succeeds in zeroing the attack counter.

Figure 1 compares the planning time of all aforementioned planning algorithms. Specifically, all VI-based algorithms have been configured with $\gamma = 0.9$, whereas the UCT algorithm has been configured to perform 10, 20, or 30 rollouts and with a lookahead of 10 steps. Results highlight that UCT is able to scale linearly with the number of states, whereas the planning time of both VI and Parallel-VI grows exponentially, as well as the multiple-agent VI. Figure 2 shows that the speedup obtained by Parallel-VI is almost linear according to the number of threads. We used only half of the cores provided by the compute node to focus on the algorithm speedup avoiding architectural bottlenecks.

Figure 3 compares the VI and UCT rewards in the single-agent case. The rewards provided by the optimal planner VI are used as a baseline to compare the rewards provided by UCT. As expected, the average reward obtained by VI is close to −10, specifically −10.07 because it always chooses

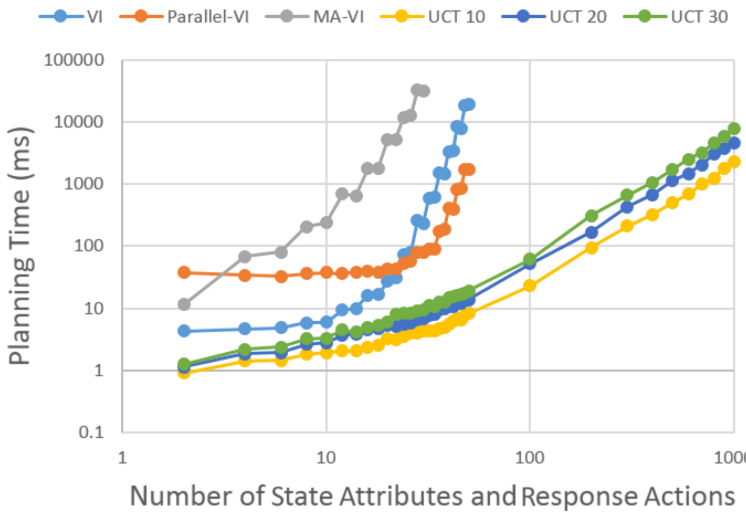

Fig. 1. Planning time comparison.

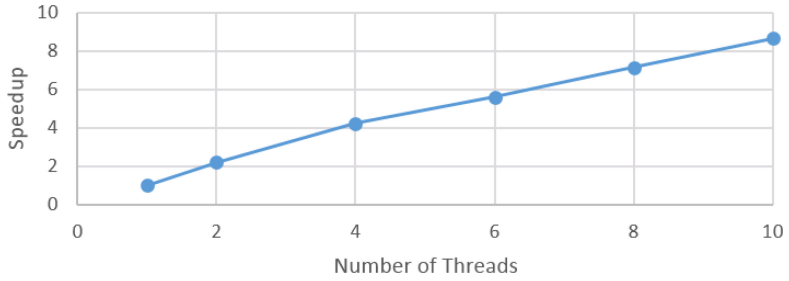

Fig. 2. Parallel-VI speedup.

the response actions characterized with the highest reward. By contrast, the UCT algorithm with 30 rollouts produces an average reward of −10.86.

The memory usage is exponential in the number of attributes in the case of VI, reaching a peak of 71GB with 50 attributes and 50 actions. The trend is instead linear for all UCT configurations, resulting in a maximum usage of 5GB for UCT-30 with 1,000 attributes and 1,000 actions.

We observe that the planning time of all planners strictly depends on the cardinality of the state space, which in turn depends on the number of attributes. Therefore, regardless of the chosen planner, it is important to limit their number as much as possible to reduce the planning time. We also observe that the entire set of attributes and the entire set of actions do not necessarily need to be included in the MDP problem: while countering any threat, we only need to consider the attributes and the actions that, directly or indirectly, help in facing the threat. The rationale is that specific threats are supposed to impact only specific system attributes and not all of them.

To this end, we designed and implemented in the proposed IRS a dynamic attributes and actions selection engine, which is in charge of instantiating the MDP problem with the minimum number of attributes and actions.

Figure 4 describes the algorithm used to generate the minimum set of attributes and actions. It takes in input the set of abnormal attributes *abnormalAttributes*—that is, the set of attributes whose values differ from the values of the attributes belonging to the final states—and returns the

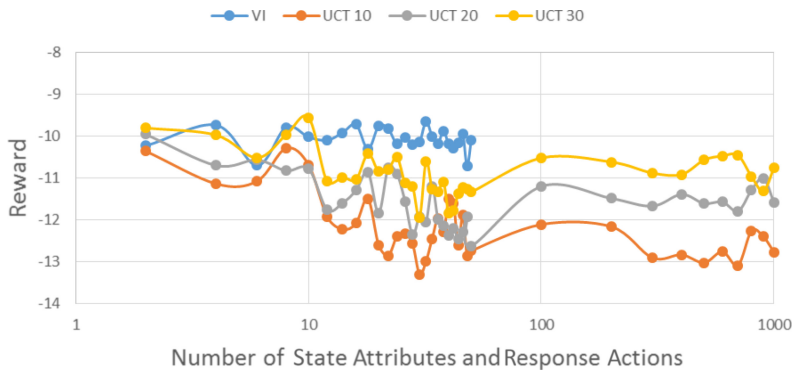

Fig. 3. Rewards comparison.

Input: abnormalAttributes
Output: attributes, actions
1: $Set\ attributes = \emptyset, Set\ actions = \emptyset$
2: $attributes \leftarrow abnormalAttributes$
3: **for** $Attribute\ att \in attributes$ **do**
4: $Set\ newActions = actLookup(att)$
5: $actions \leftarrow actions \cup newActions$
6: **for** $Action\ act \in newActions$ **do**
7: $Set\ newAttributes \leftarrow attLookup(act)$
8: $attributes \leftarrow attributes \cup newAttributes$
9: **end for**
10: **end for**
11: **return** $attributes, actions$

Fig. 4. Attributes and actions dynamic selection algorithm.

minimal sets of attributes and actions. For each abnormal attribute, the algorithm retrieves the set of actions that refer to it in its pre- and postconditions (line 4) and adds it to the output set of actions. For each newly discovered action, it then retrieves the list of attributes used as preconditions or postconditions for the considered action (line 7) and adds them to the output set of attributes. The proposed algorithm can be executed in $O(|Att| \times |A|)$, with $|Att|$ being the number of defined attributes, and it introduces a negligible overhead in the overall planning time.

The proposed attributes and actions selection engine breaks the connection between the number of attributes required to describe the system and the planning time, which is now only dependent on the maximum cardinality of attributes impacted by a threat. As a consequence, the proposed IRS is able to compute optimal response policies in less than 2 seconds using the Parallel-VI algorithm for threats impacting up to 50 system attributes and that require up to 50 different response actions to be countered, and thus we believe that it is possible to use it at runtime to protect large systems. Whether or not the attack should impact more attributes, the UCT algorithm provides a planning time that makes it feasible to run it at runtime, with an eventual reward degradation. In the multiple-agent case, the proposed IRS is able to deal at runtime with threats impacting up to 20 attributes.

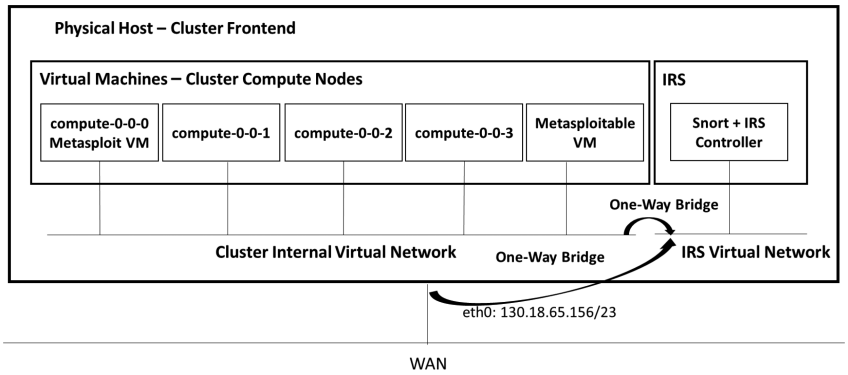

Fig. 5. Testbed architecture.

4 EXPERIMENTAL RESULTS

In this section, we describe the experiments we carried out to validate the proposed approach and to demonstrate that long-term planning outperforms short-term planning.

We set up a system composed by an HPC cluster based on Rocks,[3] a Snort IDS [39] and the IRS controller described in Section 2, as shown in Figure 5. The testbed is composed of a single physical machine that hosts two separate virtual networks, namely the Cluster Internal Virtual Network and the IRS Virtual Network. The former is attached to all compute nodes of the Rocks cluster, whereas the latter is attached to Snort and to the IRS. The two networks are constituted by two different layer-2 segments, and while the first is also bridged to physical WAN interface `eth0`, the IRS Virtual Network is instead isolated from external traffic. We run on the physical host two instances of the tool `daemonlogger`, which is used to mirror the traffic from the Cluster Internal Virtual Network and from `eth0` to the IRS Virtual Network. Traffic mirroring is accomplished at layer 2 and it is one-way—that is, frames captured on `eth0` or on the Cluster Internal Virtual Network are forwarded to the IRS Virtual Network but not vice versa.

We simulate a scenario in which an attacker already compromised one compute node in the cluster and is trying to exploit OSVDB[4] and CVE[5] vulnerabilities exposed by another compute node, namely OSVBD-73753, CVE-2007-2447, CVE-2012-1823, CVE-2010-2075, CVE-2004-2687, CVE-2011-3556. To this end, we set up five compute nodes: `compute-0-0-1` to `compute-0-0-3` are healthy VMs; `compute-0-0-0` is the VM compromised by the attacker; and finally `metasploitable` is a vulnerable, but not yet compromised compute node, target of the attacks. The compromised compute node is a VM in which we installed the Metasploit software [32]. We use this VM to scan the internal network and to launch attacks toward the vulnerable VM `metasploitable`.

4.1 Single-Agent Policies Evaluation

In the following, we compare the policies generated by the single-agent model–based IRS using both the VI and the UCT algorithms. Specifically, we configure the VI algorithm to run with two different settings: $\gamma = 0.9$ and $\gamma = 0$ (in the following, respectively VI-0.9 and VI-0); the UCT algorithm is instead configured with a lookahead of 30 steps (in the following, UCT-30). VI-0.9 fully

[3] http://www.rocksclusters.org.
[4] https://blog.osvdb.org/.
[5] https://cve.mitre.org/.

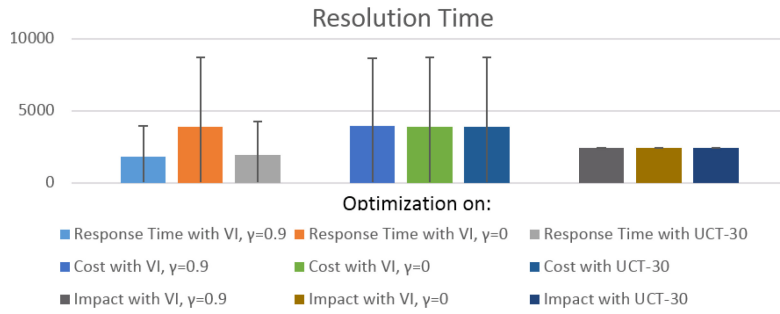

Fig. 6. Exploit resolution time comparison.

exploits the MDP features by planning response policies considering both immediate and future rewards. VI-0, by contrast, is only able to select the best short-term action as in Chen et al. [9] and Ossenbuhl et al. [37]. Therefore, we compare it to VI-0.9 to show the performance improvement achievable with long-term planning against short-term planning. Finally, UCT-30 is a suboptimal planner configured to consider long-term rewards.

We ran two different sets of experiments: a vulnerability exploit and a combination of port scan attack and vulnerability exploit. All experiments have been repeated 10,000 times, and the output of each single experiment is a response policy applicable on a real system, characterized by its own resolution time, cost, and impact, given by the sum of the respective attributes of the component response actions. For each metric, we compare the average values and the 95% confidence intervals. The reward function has been configured to optimize the response policy exclusively either on response time ($w_r = 1, w_c = 0, w_i = 0$), or cost ($w_r = 0, w_c = 1, w_i = 0$), or impact ($w_r = 0, w_c = 0, w_i = 1$).

4.1.1 Vulnerability Attack. Figure 6 compares the average resolution times and confidence intervals of the planned response policies. The first set of columns compares the resolution times obtained by VI-0.9, VI-0, and UCT-30 while optimizing on response time; the second set of columns compares the resolution times while optimizing on cost; finally, the third set of columns compares the resolution time while optimizing on impact. The lowest resolution time has been obtained by VI-0.9 with optimization on response time, whereas the worst result has been obtained with VI-0, with a 115% overhead; the UCT-30 overhead is instead 6%. The following is the most frequently planned response policy with VI-0.9 and optimization on response time: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, increaseLogVerb, increaseLogVerb, increaseLogVerb, systemReboot, backup, softwareUpdate, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb*. It is interesting to note how the planner tries to minimize the average resolution time in the planned the policy; the latter can be indeed easily split in four phases: preparation, first defense attempt, second defense attempt, and conclusion. The first response attempt (*systemReboot*) has a response time equal to 60 seconds, and it is able to face the attack with probability 0.3. Therefore, even if most of the time rebooting the machine would not be a successful resolution of the attack, it offers a very good alternative that can be used to lower the average resolution time that, in case of a backup and software update, would always be equal to 4,200 seconds.

Figure 7 compares the costs incurred by the IRS to face the vulnerability exploit. In this case, the performance obtained by the three planning algorithms with optimization on cost are perfectly comparable. This happened because most of the time the locally optimal policy was also the global

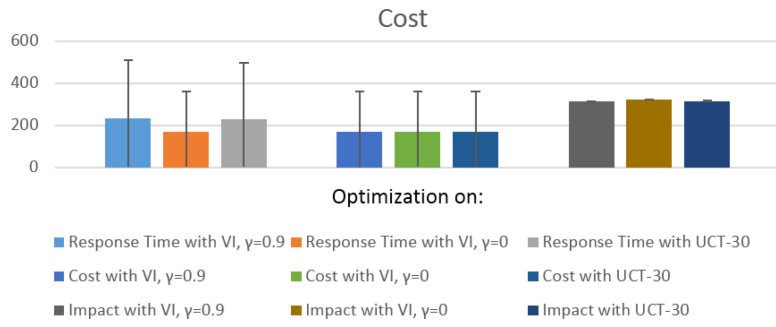

Fig. 7. Exploit cost comparison.

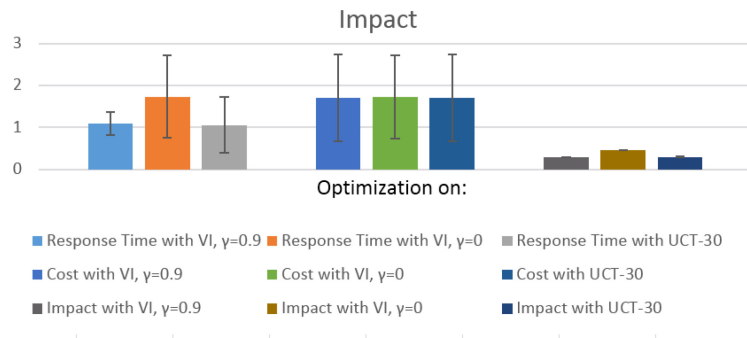

Fig. 8. Exploitimpact comparison.

optimal policy. The following is the most frequently planned response policy with VI-0.9 and optimization on cost: *generateAlert, increaseLogVerb, increaseLogVerb, increaseLogVerb, systemReboot, activateFirewall, increaseLogVerb, increaseLogVerb, quarantineSystem, backup, manualResolution, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb*. Like in the optimization on response time case, here the IRS tries to lower the average cost by preferring the *systemReboot* action over the *quarantineSystem, backup, manualResolution*.

In Figure 8, a comparison of the impacts produced by the computed policy is shown. The lowest impact on the real system has been obtained with VI-0.9 and optimization on impact; VI-0 introduced an impact overhead of 50%, whereas UCT-30 did not introduce any impact overhead. The following is the only planned response policy with VI-0.9 and optimization on impact: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, backup, softwareUpdate, decreaseLogVerb, decreaseLogVerb*. Here the *systemReboot* action is not taken into consideration because it has a high impact on the system. Instead, *backup* and *softwareUpdate* are always chosen.

In conclusion, VI-0.9 always outperformed both VI-0 and UCT-30, and the latter outperformed VI-0 in all experiments.

4.1.2 Simultaneous Port Scan and Vulnerability Attack. Figure 9 compares the resolution times obtained with the different planning algorithms and different optimization strategies. The lowest resolution time has been obtained with VI-0.9 configured to optimize on response time. UCT-30 introduced a 6% overhead, outperforming VI-0, which introduced a 116% overhead. The following is the most frequently planned response policy with VI-0.9 and optimization on response time: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, blockSrcIP, increaseLogVerb,*

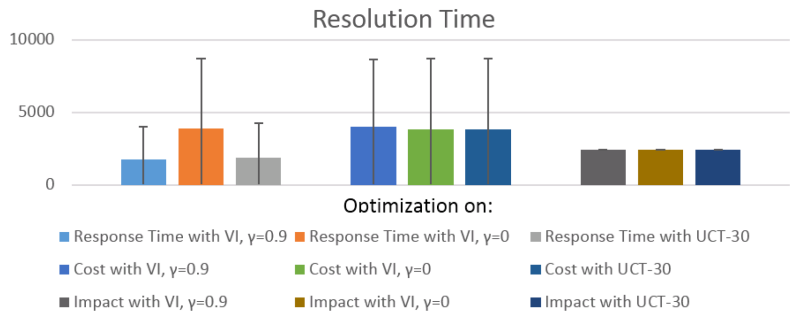

Fig. 9. Combined attack resolution time comparison.

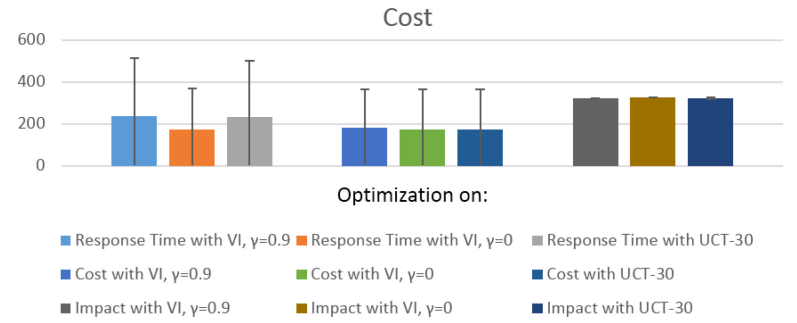

Fig. 10. Combined attack cost comparison.

increaseLogVerb, increaseLogVerb, systemReboot, backup, softwareUpdate, unblockSrcip, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb. To counter the port scan, the *blockSrcIP* main defense action has been taken because it is the one with the lowest response time; to counter the vulnerability exploit, instead, a *systemReboot* followed by *backup* and *softwareUpdate* have been planned, where the *systemReboot* has been used to lower the average resolution time.

The execution costs obtained by the three planning algorithms with optimization on cost and shown in Figure 10 are perfectly comparable, therefore evidencing that most of the time the locally optimal policy is also the best globally optimal one. The following is the most frequently planned response policy with VI-0.9 and optimization on cost: *generateAlert, increaseLogVerb, increaseLogVerb, increaseLogVerb, increaseLogVerb, systemReboot, increaseLogVerb, quarantineSystem, backup, manualResolution, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb.* Unlike the previous case, here `blockSrcIP` has not been added to the policy because the chosen *manualResolution* response action, which is the one with lowest cost to face the vulnerability exploit, is also able to deal with the port scan attack.

Finally, Figure 11 compares the impacts produced by the computed policy. The lowest impact has been obtained by VI-0.9 with optimization on impact, whereas VI-0 introduced an impact overhead of 22% and UCT-30 an impact overhead of 39%. The following is the most frequently planned response policy with VI-0.9 and optimization on impact: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, increaseLogVerb, redirectToHoneypot, backup, softwareUpdate, disableHoneypot, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb.* In conclusion, VI-0.9 always outperformed both VI-0 and UCT-30.

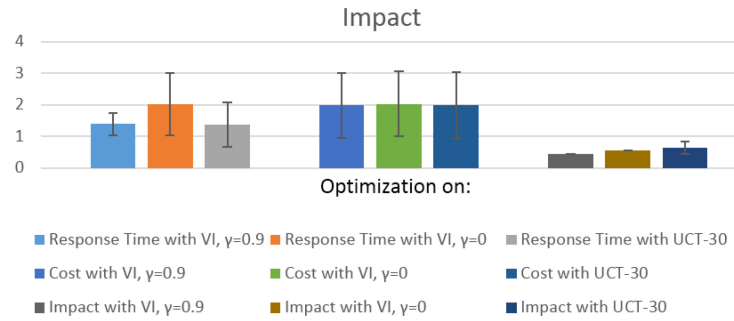

Fig. 11. Combined attack impact comparison.

4.2 Multiple-Agent Policies Evaluation

In this section, we discuss the defense policies generated executing the IRS configured with the multiple-agent model. Specifically, we model an attack consisting of a port scan followed by an exploit of the vulnerability OSVBD-73753 of the *vsftp* daemon. We compare the evolution of the multiple-agent system to the evolution of the single-agent one. In the single-agent case, the behavior of the attacker is not modeled. Therefore, when the attacker executes the port scan, the IRS reacts by planning a policy similar to *increaseLogVerb, generateAlert, activateFirewall, increaseLogVerb, blockSrcIP, unblockSrcip, decreaseLogVerb, decreaseLogVerb*. Depending on the time needed by the attacker to complete the port scan and on the time needed by the IRS to counter the attack, it could happen that the attacker manages to complete the port scan before the IRS could complete the deployment of the response actions. Being able to complete the port scan allows the attacker to discover the vulnerability, and as a consequence, it immediately launches the exploit on the *vsftp* daemon. At this point, the IRS reacts again by executing one of the policies described in Section 4.1.1. We observe that (i) the disjoint execution of the policies brings an overhead in terms of repeated actions and that (ii) the execution of the second policy happens when the system has already been compromised. Since in the multiple-agent case the behavior of the attacker is instead modeled, when the attacker launches the port scan, the IRS can use the information of the attacker's attack belief to guess what the next attack is and to proactively deploy a response policy. In the following, we show two games between the attacker and the IRS, represented as a list of stages of the form *(attackerAction / irsAction)*. In the first game, the port scan attack is assumed to require 60 seconds, enough for the IRS to deploy all needed countermeasures. In the second game, instead, the port scan attack is assumed to require only 5 seconds. Due to limited space, we removed all stages of the game regarding exclusively log verbosity increase or decrease.

4.2.1 *Game 1: Full Prevention.*

(1) *portScanAttack / generateAlert*
(2) *noOp / activateFirewall*
(3) *noOp / blockSrcIP*
(4) *noOp / unblockSrcIP*
(5) *noOp / backup*
(6) *noOp / softwareUpdate*

We observe that the attacker, after having launched a 60-second port scan attack, waits for it to finish with a series of *noOp*. During the waiting, the IRS is able not only to generate a policy to

protect against the port scan (with the *blockSrcIp*) but also to proactively address the prospective vulnerability exploit. This prevents the attacker from being able to launch further attacks.

4.2.2 Game 2: Reaction and Prevention.

(1) *portScanAttack / generateAlert*
(2) *noOp / activateFirewall*
(3) *attackVsftpd / increaseLogVerbosity*
(4) *noOp / blockSrcIP*
(5) *noOp / quarantineSystem*
(6) *noOp / backup*
(7) *noOp / manualResolution*
(8) *noOp / unblockSrcIP*
(9) *noOp / softwareUpdate*

In this case, the attacker manages to complete the port scan before its IP gets blocked by the firewall. The *blockSrcIP* does not have effect on the attack because it is based on a reverse shell, and therefore the firewall rule just added does not work. Afterward, the IRS planned to counter the attack with a manual resolution on a quarantined system, after having executed a backup. This step solves the ongoing attack but leaves the system vulnerable to other threats because the software has not been updated yet. Therefore, as a last step, the IRS plans to update the software.

5 RELATED WORKS

The field of autonomic systems, specifically self-protecting systems, is an already established research field [19, 28]. The research produced so far on self-protection is mostly focused on intrusion detection rather than protection [45]. However, the increasing amount cyberattacks [2] makes it infeasible to manually handle all generated alerts. IRSs try to address this problem by selecting the appropriate responses to the detected attacks.

Existing works on dynamic IRS can be classified according to the following dimensions: (i) multiobjective planning, or the ability of an IRS to select the optimal response action (or response plan) according to a custom set of weighted criteria; (ii) IDS uncertainty, or the ability of the IRS to deal with stochastic IDS alerts; (iii) long-term policies, or the ability of the IRS of planning policies that span over a long period of time rather than just selecting the immediate optimal response; (iv) system model, or whether the IRS is based on a model of the system, which can be used to describe the system dynamics due to an ongoing attack or due to the application of a response action (plan); (v) attacker model, or whether the IRS is based on a model describing the attacker behaviur and the potential graph of achievable targets.

The authors of Toth and Kruegel [44] introduce a network model, consisting of resources, system users, network topology, and firewall rules. The model is then used to specify direct and indirect dependencies among the resources and between the users and the resources to be able to predict the impact of a service unavailability on dependent services and on dependent users. Such a model is used by the IRS to choose the response action able to avert a certain threat to minimize the overall impact on the system and, ultimately, on the users.

ADEPTS [17] focuses on attack containment—that is, on restricting the effect of the intrusion to a subset of the services. The presented approach maximizes the availability of the overall system at the expenses of the features compromised by the attack, which are isolated from the rest of the system. Unlike Toth and Kruegel [44], in which the proposed model is system centric, in this work the authors propose an attack-centric model, based on intrusion graphs. The latter support OR, AND, and QUORUM nodes and can be used to easily represent attack propagation and escalation.

The Compromised Confidence Index (CCI) is used to compute the confidence of the detected alert and, by traversing the graph, the confidence of a particular system breach. The response action is then selected from a response repository by evaluating the effectiveness and the potential disruptiveness of all available responses.

The authors of Stakhanove et al. [42] propose an IRS that takes into consideration the stochastic nature of the detections made by the IDS, and the response action is only triggered if the confidence level of the detected attack is greater than a specified threshold. In Fessi et al. [16], an optimal response selection is proposed based on financial cost, reputation loss, and processing resource. A modified version of the classical genetic algorithm is used to represent the association between each response action with the system resources affected by the execution of the action.

A long-term response planning is presented in Mu and Li [36]. The work uses a Hierarchical Task Network (HTN) to model the IRS goal, the high-level response actions, and the mechanisms to enable response actions. This work uses a fixed set of goals and statically maps each goal to a sequence of high-level response actions.

A partially observable MDP (POMDP) with a single-objective reward function is used in Zan et al. [46] to model an IRS able to plan optimal response policies. Since the POMDP is subject to an exponential growth of the states according to the number of the considered attributes, the authors propose a hierarchical decomposition to reduce the computational complexity.

The authors of Tiehling et al. [35] use a Bayesian directed acyclic graph (DAG) [25] to model attacker behavior. The DAG nodes describe system assets and their dependencies, whereas edges represent possible exploitation paths. Responses are evaluated according to the CIA triad preferring confidentiality and integrity over availability.

The authors of Tonouz et al. [47] propose a game-theoretic model named *RRE*, based on a nonzero sum stochastic game. The core of the work is represented by attack response trees (ARTs). A leaf node of an ART represents a binary system attribute, which is set to 1 if an IDS alert that includes such an attribute is triggered or to 0 otherwise. Binary attributes are then combined using `AND` and `OR` logical ports to define a path toward the impairment of the system's functionalities and ultimately toward the global system when the impairment reaches the root node. Each node of the tree, with the exception of the leaves, can be labeled with a *response tag*. The latter represents a response action that, when executed, is able to set to 0 all attributes specified by the leaves in the corresponding subtree. In the same way as our approach, the ART model is not built on the basis of the attack itself but on the consequences that the attack has on the system. RRE includes a component that is in charge of computing the security level of the system based on a set of `if-then` rules manually defined by the system administrator, according to his personal system knowledge. The computed security level is represented as a string such as `low, medium, high`.

All reviewed works, with the exception of Fessi et al. [16] and Mu and Li [36] make use of either a system or an attacker model to compute the optimal response action. However, Fessi et al. [16] and Zonouz et al. [47] are the only works considering a multiobjective optimization, which is fundamental for a fine tuning of the produced response plans. Miehling et al. [35], Mu and Li [36], and Zonouz et al. [47] are the only works considering a long-term planning. However, the first only introduces static long-term plan templates, whereas the second only produces the immediate optimal response action evaluated with infinite lookahead. Zonouz et al. [47], instead, proposes a long-term response plan based on the evaluation of a stochastic game between the attacker and the IRS. The authors deal with the exponential growth of the state space using approximation techniques, but they do not provide hints about the gap that the approximated policies have with respect to the optimal policies.

This work aims at providing the entire set of features. Specifically, we introduce a reward function based on the simple additive weighting (SAW) technique [20] to support multiobjective

planning; we handle IDS uncertainty by modeling attack probabilities as state attributes; we use the MDP framework to produce optimal stochastic long-term policies; and we provide both the system and the attacker model. The former is statically described with state attributes and dynamically described with the state transitions; the latter is described as a multiple-agent stochastic game.

6 CONCLUSIONS AND FUTURE WORKS

In this work, we presented an IRS, based on the MDP framework, that supports multiobjective long-term planning. The proposed approach models both the attacker and the defended system behavior and takes into accounts the uncertainty of IDS detections. We presented the MDP framework as a model for building reactive and proactive IRSs. We used a single-agent MDP to model the behavior of a reactive IRS applied to a system subject to several attacks, and we used a competitive multiple-agent MDP to model a game between a proactive IRS and an attacker. Since the state space of the models grows exponentially according to the number of the attributes used to describe the protected system, we introduced a dynamic attributes and actions selection algorithm, which is able to instantiate the minimal MDP problem given the currently ongoing threat. The performance assessment showed that the proposed IRS is able to optimally plan response policies at runtime with threats affecting up to 50 system attributes and requiring up to 50 different response actions to be countered with the proposed parallel version of the VI algorithm. Should the threat involve more attributes or actions, the IRS is anyway able to solve the MDP and drive the system toward a protected state in a suboptimal way. Finally, a thorough effectiveness validation showed that long-term policies always outperform short-term ones and that stochastic games can be effectively used to proactively protect a system.

As future work, we plan to establish a feedback loop between the controller and the managed system to let the actions' postconditions probabilities evolve according to the real system evolution. Furthermore, we plan to consider nondeterministic MDPs [14] to produce a set of near-optimal decision policies from which the system administrators could pick the best one according to his or her personal knowledge. Such a semiautomatic behavior could be particularly useful in industrial control systems (SCADA), which are used extensively in critical infrastructures.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Qian Chen of Savannah State University for her comments that improved the overall quality of the article.

REFERENCES

[1] Sherif Abdelwahed, Jia Bai, Rong Su, and Nagarajan Kandasamy. 2009. On the application of predictive control techniques for adaptive performance management of computing systems. *IEEE Transactions on Network and Service Management* 6, 4, 212–225.

[2] Akamai. 2015. Akamai's State of the Internet: Q3 2015 Report. Retrieved March 2, 2018, from https://www.stateoftheinternet.com/resources-cloud-security-2015-q3-web-security-report.html.

[3] R. E. Bellman. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ.

[4] Yoshua Bengio. 2009. Learning deep architectures for AI. *Foundations and Trends® in Machine Learning* 2, 1, 1–127.

[5] Monowar H. Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal Kumar Kalita. 2014. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys and Tutorials* 16, 1, 303–336.

[6] Craig Boutilier. 1996. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*. 195–210.

[7] Lucian Busoniu, Robert Babuska, and Bart De Schutter. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38, 2, 156–172.

[8] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. 2012. Moses: A framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering* 38, 5, 1138–1159.

[9] Qian Chen, Sherif Abdelwahed, and Abdelkarim Erradi. 2014. A model-based validated autonomic approach to self-protect computing systems. *IEEE Internet of Things Journal* 1, 5, 446–460.
[10] Yulia Cherdantseva and Jeremy Hilton. 2013. A reference model of information assurance and security. In *Proceedings of the 2013 8th International Conference on Availability, Reliability, and Security (ARES'13)*. IEEE, Los Alamitos, CA, 546–555.
[11] Chun-Jen Chung, Pankaj Khatkar, Tianyi Xing, Jeongkeun Lee, and Dijiang Huang. 2013. NICE: Network intrusion detection and countermeasure selection in virtual network systems. *IEEE Transactions on Dependable and Secure Computing* 10, 4, 198–211.
[12] Carlos Diuk, Andre Cohen, and Michael L. Littman. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*. ACM, New York, NY, 240–247.
[13] Jianbin Fang, Henk Sips, Lilun Zhang, Chuanfu Xu, Yonggang Che, and Ana Lucia Varbanescu. 2014. Test-driving Intel Xeon Phi. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*. ACM, New York, NY, 137–148.
[14] Mahdi Milani Fard and Joelle Pineau. 2011. Non-deterministic policies in Markovian decision processes. *Journal of Artificial Intelligence Research* 40, 1–24.
[15] Ahmed Fawaz, Robin Berthier, and William H. Sanders. 2016. A response cost model for advanced metering infrastructures. *IEEE Transactions on Smart Grid* 7, 2, 543–553.
[16] B. A. Fessi, S. Benabdallah, N. Boudriga, and M. Hamdi. 2014. A multi-attribute decision model for intrusion response system. *Information Sciences* 270, 237–254.
[17] Bingrui Foo, Yu-Sung Wu, Yu-Chun Mao, Saurabh Bagchi, and Eugene Spafford. 2005. ADEPTS: Adaptive intrusion response using attack graphs in an e-commerce environment. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, Los Alamitos, CA, 508–517.
[18] Mansoureh Ghasemi, Hassan Asgharian, and Ahmad Akbari. 2016. A cost-sensitive automated response system for SIP-based applications. In *Proceedings of the 2016 24th Iranian Conference onElectrical Engineering (ICEE'16)*. IEEE, Los Alamitos, CA, 1142–1147.
[19] Salim Hariri, Bithika Khargharia, Houping Chen, Jingmei Yang, Yeliang Zhang, Manish Parashar, and Hua Liu. 2006. The autonomic computing paradigm. *Cluster Computing* 9, 1, 5–17.
[20] C. L. Hwang and K. Yoon. 1981. *Multiple Criteria Decision Making*. Lecture Notes in Economics and Mathematical Systems. Springer.
[21] Stefano Iannucci and Sherif Abdelwahed. 2016. A probabilistic approach to autonomic security management. In *Proceedings of the 13th IEEE International Conference on Autonomic Computing (ICAC'16)*.
[22] Stefano Iannucci and Sherif Abdelwahed. 2016. Towards autonomic intrusion response systems. In *Proceedings of the 2016 IEEE International Conference on Autonomic Computing (ICAC'16)*.
[23] Stefano Iannucci, Qian Chen, and Sherif Abdelwahed. 2016. High-performance intrusion response planning on many-core architectures. In *Proceedings of the 2016 25th International Conference on Computer Communication and Networks (ICCCN'16)*.
[24] Zakira Inayat, Abdullah Gani, Nor Badrul Anuar, Muhammad Khuram Khan, and Shahid Anwar. 2016. Intrusion response systems: Foundations, design, and challenges. *Journal of Network and Computer Applications* 62, 53–74.
[25] Finn V. Jensen. 1996. *An Introduction to Bayesian Networks*. Vol. 210. UCL Press, London, England.
[26] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4, 237–285.
[27] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning* 49, 2–3, 193–208.
[28] J. O. Kephart and D. M. Chess. 2003. The vision of autonomic computing. *IEEE Computer* 36, 1, 41–50.
[29] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*. Springer, 282–293.
[30] Wenke Lee, Wei Fan, Matthew Miller, Salvatore J. Stolfo, and Erez Zadok. 2002. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security* 10, 1–2, 5–22.
[31] L. Li, M. L. Littman, and L. Littman. 2008. *Prioritized Sweeping Converges to the Optimal Value Function*. Technical Report DCS-TR-631. Rutgers University.
[32] Carlos Joshua Marquez. 2010. An Analysis of the IDS Penetration Tool: Metasploit. Retrieved March 2, 2018, from https://www.infosecwriters.com/text_resources/pdf/jmarquez_Metasploit.pdf.
[33] Peter Mell, Karen Scarfone, and Sasha Romanosky. 2007. A Complete Guide to the Common Vulnerability Scoring System: Version 2.0. Retrieved March 2, 2018, from https://www.first.org/cvss/v2/guide.
[34] Daniel A. Menascé. 2002. QoS issues in Web services. *IEEE Internet Computing* 6, 6, 72–75.
[35] Erik Miehling, Mohammad Rasouli, and Demosthenis Teneketzis. 2015. Optimal defense policies for partially observable spreading processes on Bayesian attack graphs. In *Proceedings of the 2nd ACM Workshop on Moving Target Defense*. ACM, New York, NY, 67–76.

[36] Chengpo Mu and Yingjiu Li. 2010. An intrusion response decision-making model based on hierarchical task network planning. *Expert Systems with Applications* 37, 3, 2465–2472.
[37] Sven Ossenbuhl, Jessica Steinberger, and Harald Baier. 2015. Towards automated incident handling: How to select an appropriate response against a network-based attack? In *Proceedings of the 2015 9th International Conference on IT Security Incident Management and IT Forensics (IMF'15)*. IEEE, Los Alamitos, CA, 51–67.
[38] Martin L. Puterman and Moon Chirl Shin. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science* 24, 11, 1127–1137.
[39] Martin Roesch. 1999. Snort—lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration (LISA'99)*. 229–238.
[40] Jerome H. Saltzer and Michael D. Schroeder. 1975. The protection of information in computer systems. *Proceedings of the IEEE* 63, 9, 1278–1308.
[41] Alireza Shameli-Sendi and Michel Dagenais. 2015. ORCEF: Online response cost evaluation framework for intrusion response system. *Journal of Network and Computer Applications* 55, 89–107.
[42] Natalia Stakhanova, Samik Basu, and Johnny Wong. 2007. A cost-sensitive model for preemptive intrusion response systems. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications (AINA'07)*. 428–435.
[43] Christopher Roy Strasburg, Natalia Stakhanova, Samik Basu, and Johnny S. Wong. 2008. The methodology for evaluating response cost for intrusion response systems. In *Recent Advances in Intrusion Detection*. Lecture Notes in Computer Science, Vol. 5230. Springer, 390–391.
[44] Thomas Toth and Christopher Kruegel. 2002. Evaluating the impact of automated intrusion response mechanisms. In *Proceedings of the 2002 18th Annual Computer Security Applications Conference*. IEEE, Los Alamitos, CA, 301–310.
[45] Eric Yuan, Naeem Esfahani, and Sam Malek. 2014. A systematic survey of self-protecting software systems. *ACM Transactions on Autonomous and Adaptive Systems* 8, 4, 17.
[46] Xin Zan, Feng Gao, Jiuqiang Han, Xiaoyong Liu, and Jiaping Zhou. 2010. A hierarchical and factored POMDP based automated intrusion response framework. In *Proceedings of the 2010 2nd International Conference on Software Technology and Engineering (ICSTE'10)*. IEEE, Los Alamitos, CA, 410.
[47] Saman A. Zonouz, Himanshu Khurana, William H. Sanders, and Timothy M. Yardley. 2014. RRE: A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems* 25, 2, 395–406.

Received September 2016; revised April 2017; accepted November 2017