A Probabilistic Approach to Autonomic Security Management

Stefano Iannucci
Distributed Analytics and Security Institute
Mississippi State University
Starkville, Mississippi
Email: stefano@dasi.msstate.edu

Sherif Abdelwahed
Department of Electrical and Computer Engineering
Mississippi State University
Starkville, Mississippi
Email: sherif@ece.msstate.edu

Abstract—**Intrusion Response Systems (IRSs) have been a major research topic in the last decade. At the core of an IRS is the response selection algorithm, which selects the best response action to counter the currently detected attack. Most of the IRSs proposed so far, statically or dynamically evaluate the mapping between response actions and specific attacks, ignoring the actual system state, thus providing only short-term decisions. In this paper we propose a controller based on Markov Decision Process (MDP) for an autonomic IRS. The proposed controller is able to compose atomic response actions to create optimal long-term response policies to protect a system. Experimental results show that long-term policies are always more effective than short-term policies and that they can reduce the threat resolution time up to 56% in the considered scenario.**

I. Introduction

The frequency of the attacks directed to computer systems and their complexity is increasing day by day. According to the Akamai's state of the Internet 2015 Q2 Report [1], the number of recorded attacks is more than doubled compared with a year ago. The complexity of the systems and the number of alerts raised by Intrusion Detection Systems (IDSs) makes it infeasible for a human being to timely address all the threats, therefore increasing the probability of attack success and the consequent damage to the system [7]. The need of automatic responses to the attacks motivated the research and development of Intrusion Response Systems (IRSs). The core of an IRS is its response selection algorithm which, according to the currently ongoing attack and a set of available response actions, selects the best one as a countermeasure.

Since early 2000s, researchers assisted to the development of mainly two kinds of IRSs: one based on a static mapping between the detected attack and its best countermeasure (e.g., [23]), the other based on a dynamic evaluation of all the response actions, according to the detected attack and to a list of evaluation criteria (e.g., [5], [22], [20], [6], [11], [18], [10]. The former immediately exhibited all the limit of a static approach, mainly due to scalability problems: a static mapping requires the system administrator to periodically update the set of known attacks and to associate them to the proper response. Given the limits of the static IRSs, the dynamic ones have been the main subject of study during the last years. Several approaches have been proposed to select the optimal response action, usually based on countermeasure ranking like in [20], [22], [5] or on a multi-objective optimization problem as in [10]. Even if more scalable than the static response selection approach, the dynamic one also exposes an evident scalability problem, as the administrator must identify a score (or the attributes scores) for all the considered response actions with respect to all the considered attacks. None of the considered works, with the exception of [18], take into consideration the system state, which connects the attack with the response actions. With a model composed by attack-related events, system states and response actions, the administrator's focus shifts from the static attack-response mapping to the appropriate attack response description based on the effects on the system state. As a result, it is possible to identify a set of *target* system states and the manual mapping done by the system administrator can be replaced by an automatic IRS, able to compose a set of atomic response actions into a complex response policy or plan. The latter drives the system towards the set of desired target states [3]. Response actions composition also means response actions re-use: instead of developing a single monolitic action for each attack typology, different combinations of the same atomic actions can be exploited to deal with different attacks, effectively, including unknown attacks.

In this paper we introduce a Markov Decision Process (MDP) based controller able to compose response policies using atomic response actions. Such a controller is the core of the proposed autonomic IRS.

A. System Overview

Generally, an autonomic system is composed of two different subsystems [24]: a controller that implements the self-management algorithms and a controlled subsystem concerned with the domain functionality. In the proposed IRS, the controller subsystem is designed according to the MAPE-K loop for autonomic systems [14], as shown in Figure 1. Specifically, the Monitor phase relies on a set of Network IDSs which analyze the network traffic of the controlled subsystem and on Host IDSs properly deployed on the controlled systems that analyze host attributes such as system loads and system loggers. The event generated by the IDSs are then collected and analyzed by the *IDS Event Manager* component, which implements the Analyze phase of the MAPE-K loop. Its task is

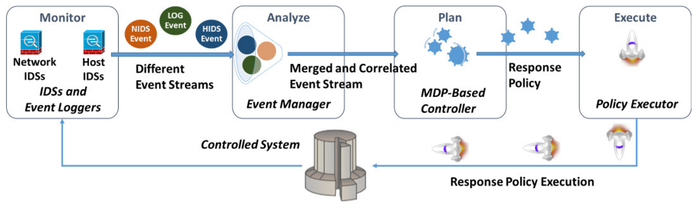

Fig. 1. High-Level System Architecture

to correlate and merge the data flows coming from the IDSs in order to produce a single, added value data flow to feed to the Plan phase. This is implemented by a controller based on a MDP. Given the inputs of the IDS Event Manager and the current system state, its task is to compute the *Response Policy* needed to drive the managed system towards the target state. Specific actuators on the controlled system named *Policy Executors* are in charge of applying the computed response policy, implementing the Execute phase of the MAPE-K loop. Finally, the Knowledge phase is contained into the MDP-based Controller. The latter, in fact, starts its execution with an initial parametrization provided by a domain expert, but then it continuously improves its behaviour by learning from the results of response actions applied on the managed system.

B. Contributions and Organization

In this paper we focus on the design, realization and evaluation of the MDP-based controller introduced in Section I-A. Its main characteristic is the ability to compose a sequence of atomic response actions to form an optimal response policy able to drive the system to the target state. Specifically, we provide the complete model and methodology needed to build MDP-based controllers for security planning. We show that, when a small degradation of the reward is allowed, such approach can be applied to large systems. Furthermore, we experimentally show that the adoption of long-term response policies can be more effective than the adoption of single response actions: in the best execution scenario we were able to reduce the threat resolution time by 56%, while in the worst case the threat resolution time was the same. All the experiments have been carried out using real-world attacks to exploit real vulnerabilities (more details on Section V-A). Note that the design and realization of the IDS Event Manager and the system learning behaviour of the controller are out of the scope of the present work.

The paper is structured as follows: in Section II we describe several related works. The complete model design and methodology of the MDP-based controller is described in Section III; in Section IV we compare the performances of an optimal and a sub-optimal MDP solvers; in Section V we describe the realized testbed and we provide detailed experimental results. Finally, in Section VI we draw the conclusions.

II. Related Works

In this section we describe relevant works in the field of dynamic IRS. Specifically, we are interested in response selection methodologies. All the considered works try to make a balance between the positive and negative effects that a response could have on a running system and most of them try to model the system not by considering its internal status, but only considering the services it delivers and the inter-dependencies among them.

In [23] the authors represent a computer system by modeling: (i) the executed services; (ii) the system users; (iii) the network topology; (iv) the firewall rules. The model is used to find, given a dependency tree between the entities, the best firewall rule to apply in case of a detected attack. The best response is selected considering that very often the application of an additional firewall rule may interfere with the normal system operativeness. Therefore the response which provides the lowest penalty to the normal operativeness is applied.

In [20] the authors extend [23] by evaluating benefits and risks of a reaction, but also potential damages caused by the attack in case of no reaction. Penalty costs are modeled as Service Level Agreement costs related to the importance of a provided service. The response selection strategy proposed by REASSESS [20] considers both the positive and negative effects that the execution of an action could have on a running system. Such effects are included in a global index that is used to evaluate all the response actions, possibly using historical effectiveness data. Once all the actions are evaluated, an ordered list of response actions is produced and the most effective response action is selected to be applied to the system. However, the proposed approach neither consider the current system state, nor the explicit evaluation of different criteria with subsequent specific optimization.

In [19] the authors propose to compute a response policy rather than just a single response action to face an ongoing attack. The proposed model accounts for statically defined sequences of response actions which are statically mapped to the managed attacks. A dynamic behavior of the IRS controller is introduced in the run-time evaluation of the policies: while executing a given policy, the controller uses a threshold in order to decide whether to execute the next action in the policy or to stop its execution.

In [22] the authors map the attack detected by the IDS to an Attack Graph (AG), which is used to model multi-stage attacks. Based on the AG, defense points are assigned to each node and their negative impact is calculated independently of other responses. Finally, a Pareto-set is generated based on the response positive effect and negative impact values.

The work presented in [18] is the most similar to ours. The authors use a Bayesian Direct Acyclic Graph (DAG) to model a probabilistic attacker behavior. The DAG nodes describe system assets and system asset dependencies, while edges represent possible exploitation paths. System assets are described by the means of binary attributes, that characterize them as active or disabled. From the defense point of view, the authors propose binary response actions, that is, actions that are able to deactivate or activate a certain service or asset. The actions are evaluated according to the Confidentiality Integrity Availability (CIA) triad, but since all the considered actions deal with services or machines deactivation, their evaluation always increase confidentiality and integrity and always penalize the overall system availability. The response selection problem is formulated by building a Partially Observable MDP (POMDP) from the DAG and by selecting the optimal single response action according to an evaluation with infinite look-ahead, aimed at minimizing the overall execution cost. The entire work is built on the assumption that a partial shutdown is preferred to a scenario in which the attacker has partial control of the system. Even if this is a big step ahead towards dynamically building response policies to protect computer systems, we believe that an IRS should also be able to produce non-disruptive actions on the controlled system. Non disruptive behavior can be achieved either by developing specific and monolithic responses for every considered attack or by cleverly composing and reusing a set of fine-grained response actions.

III. System Model

The system model is based on a MDP. A MDP is a controlled stochastic process satisfying the Markov property with rewards assigned to state transitions. A solution to a MDP is a policy, mapping states to actions, that (perhaps stochastically) determines state transitions to maximize the reward according to a set of evaluation criteria [16].

Let S be the finite set of states; let A be the finite set of actions available to the MDP agent.

Let $P_a(s,s') = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$ be the probability that the response action $a \in A$ in state $s \in S$ at time t will lead to state $s' \in S$ at time $t+1$.

Let $S_{tgt} \subseteq S$ be the set of *target states*. Formally, $f \in S_{tgt}$ is a state where the controller does not take any action, i.e., $\mathbb{P}[A_t = a | S_t = f] = 0 \; \forall a \in A, \forall f \in S_{tgt}$. We assume here that there always exists a sequence of actions that can take the system from any non-target state to a target state.

Let R be the reward function $R : S \times A \to \mathbb{R}$. Rewards are computed using the Simple Additive Weighting (SAW) technique [12], that is: $R(s,a) = \sum_c w_{s,c} e_{s,a,c}$, where $e_{s,a,c}$ is the normalized evaluation of response action a under criterion c in state s and $w_{s,c}$ is the weight of criterion c in state s.

Let γ be the discount factor, $0 \leq \gamma < 1$ and let $G_t = \sum_{k=0}^{K} \gamma^k R_{t+k}$ be the discounted reward from time $t+1$ to time $t+K$. If $K \to \infty$, G_t is the *unbounded reward*, that is, the reward with infinite look-ahead.

$M = \langle S, A, P_a, R, \gamma \rangle$ is the MDP underlying the controller of the IRS. The aim of the MDP is to find an optimal policy π, that is, a sequence of actions that optimally drives the system from the current state to the set of the target states S_{tgt}.

We use a Object Oriented MDP representation (OO-MDP) [8], and each state is characterized by a number attributes. Specifically, states are composed by joining 2 macro-attributes: (i) the attack vector P and (ii) the system variables V. The former contains as many variables as the number of attacks detectable by the IDSs and each variable P_i represents the probability value computed by the Event Manager that the system is currently under attack i. The latter represent the current system status.

A. States Characterization

The system model presented in Section III is general and can be specialized in different ways according to the specific system that has to be protected. We implemented it considering 7 different attacks and 11 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 as computed by the Event Manager that the controlled system is being attacked, respectively by: a portscan, an exploit on the `vsftpd` daemon, an exploit on the `smbd` daemon, an exploit on the execution of PHP as a CGI application, an exploit on the `ircd` daemon, an exploit on the `distccd` daemon and finally and exploit on the `rmi` Java daemon. More details about the considered attacks are described in Section V-A.

Furthermore, we consider the following system attributes:

- $firewall \in \{true, false\}$ 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 quarantined (i.e., it has been isolated from the network).
- $rebooted \in \{true, false\}$ represents whether the considered system has ever 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.

B. Reward Function

We define the reward function as a penalty score on the considered actions. The reward function evaluates the response actions according to the following criteria:

- **Response Time** $R(x) \in \mathbb{R}$**.** This criteria represents the time needed to apply the response x.
- **Cost** $C(x) \in \mathbb{R}$**.** This criteria represents the economic cost of applying the response x.

- **Impact index $I(x) \in [0, 1]$.** This criteria represents the impact index of the response x on the normal system operativeness. The lower its value is, the lower is the impact on the system.

We define the following reward function:

$$R = -w_r \frac{R(x)}{R_{max}} - w_c \frac{C(x)}{C_{max}} - w_i I(x) \quad (1)$$

where $w_r, w_c, w_i \in [0, 1]$ are custom weights used to balance the importance of the criteria in the multicriteria optimization problem. R_{max} and C_{max} represent respectively the maximum response and the maximum cost over all the considered response actions and are used to normalize their values.

C. Response Actions

In order to avoid activating potentially disruptive response actions when the system is not under severe attack, we introduce two thresholds on the attack probability attributes, namely T_1 and T_2. Thus, given an attack probability p, it can belong to one of the following 4 stages:

- $p < T_1$. At this stage the IDSs have detected an insignificant anomaly that should be considered as noise and no response actions should be triggered;
- $T_1 \leq p < T_2$. At this stage the IDSs have detected a significant anomaly, which cannot be classified as an attack. However, the system can start planning some response action in order to prevent possible attacks;
- $T_2 \leq p < 1$. At this stage the anomaly detected by the IDSs is considered to be an unidentified attack ($p < 1$), therefore the response plan generated by the IRS can only contain generic responses;
- $p = 1$. The attack has been identified and a specific response plan can be computed.

In the following we describe, for space reasons, only some of the response actions that our IRS prototype is able to apply on the controlled system. For each of them, we will provide a description of its behavior and of the response time R, cost C and impact I attributes, needed to compute the expected reward when planning the optimal policy. These attributes are configurable by the system administrator to reflect the actual defended system. Each response will also be characterized by pre-conditions and post-conditions. The former are conditions that 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. Eventual dependencies between response actions are not directly modelled: indeed, using pre-conditions, 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 I summarizes response time, cost and impact attributes for all of the considered actions.

a) Firewall Activation: The response aim is to start the system's firewall in case it was not started previously. Its characteristics are:

- **Reward Attributes:** $R = 2, C = 1, I = 0$
- **Pre-Conditions:** $(P_{scan} \geq T_1 \vee P_{vsftpd} \geq T_1 \vee P_{smbd} \geq T_1 \vee P_{phpcgid} \geq T_1 \vee P_{distccd} \geq T_1 \vee P_{rmi} \geq T_1 \vee P_{ircd} \geq T_1) \wedge \neg firewall \wedge \neg quarantined \wedge active \wedge logVerb > 0$
- **Post-Conditions:** $Prob = 1, firewall = 1$

This action can be executed when at least one entry of the attack probability vector P is greater than 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 is identical to the current state, but with the $firewall$ attribute set to 1. The reason why a firewall could be inactive is that, as described in Section V, we are modeling an HPC system. In such systems usually the personal firewall is disabled to avoid any possible overhead.

b) Block Source IP `badIP`: This response configures the system's personal firewall in order to drop IP packets originated by the IP `badIP`. Its characteristics are:

- **Reward Attributes:** $R = 1, C = 3, I = 0.3$
- **Pre-Conditions:** $P_{scan} \geq T_2 \wedge firewall \wedge \neg quarantined \wedge active \wedge badIP \notin blocked\_ips \wedge alert \wedge logVerb > 1$
- **Post-Conditions:** $Prob = 1, blocked\_ips = blocked\_ips \cup \{badIP\}, P_{scan} = 0$

This action can be executed when the probability of having detected a port-scan attack 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 that it 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 blocked IPs set. 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. We set the impact to 0.3 because we consider that the observed source IP could be a router executing a source NAT. Therefore, blocking the IP would result in blocking the attacker as well as potentially non-malicious clients masqueraded by the source NAT.

c) Flow Rate Limit `badIP`: This response configures the system's personal firewall in order to limit the traffic rate of IP packets originated by the IP `badIP`. Its characteristics are:

- **Reward Attributes:** $R = 3, C = 1, I = 0.2$
- **Pre-Conditions:** $P_{scan} \geq T_1 \wedge firewall \wedge badIp \notin flowlimit\_ips \wedge \neg quarantined \wedge active \wedge logVerb > 0$
- **Post-Conditions:**

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

This action can be executed when the probability of having identified a port-scan attack 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 that it has not been quarantined and that its log verbosity is at least equal to 1. Finally, the IP address of the prospective 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 prospective 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.

d) Close Network Connection: This response closes an unauthorized TCP connection or UDP flow between any pair `(lhost:lport, rhost:rport)`. Its characteristics are:

- **Reward Attributes:** $R = 1, C = 1, I = 0.2$
- **Pre-Conditions:** $((P_{vsftpd} = 1 \vee P_{smbd} = 1 \vee P_{phpcgid} = 1 \vee P_{distccd} = 1 \vee P_{rmi} = 1 \vee P_{ircd} = 1) \vee \mathit{rebooted} \wedge (P_{vsftpd} \geq T_2 \vee P_{smbd} \geq T_2 \vee P_{phpcgid} \geq T_2 \vee P_{ircd} \geq T_2 \vee P_{distccd} \geq T_2 \vee P_{rmi} \geq T_2)) \wedge \neg \mathit{quarantined} \wedge \mathit{active} \wedge \mathit{alert} \wedge \mathit{rhost} \notin \mathit{blocked\_ips} \wedge \mathit{rhost} \notin \mathit{honeypotted\_ips} \wedge \mathit{logVerb} > 2$
- **Post-Conditions:**

$$\begin{cases} Prob = 0.1, & P_{vsftpd} = 0, P_{smbd} = 0, P_{phpcgid} = 0, \\ & P_{ircd} = 0, P_{distccd} = 0, P_{rmi} = 0 \\ Prob = 0.9, & \mathit{no\ changes} \end{cases}$$

This action can be executed immediately when at least one of the vulnerability has been exploited or it can be executed after a system reboot when at least one of the vulnerability has a probability greater than T_2 of being exploited. The system is required to be active and the remote host must not belong neither to the set of the blocked IPs nor to the set of the honeypotted IPs. Finally, the administrator must be alerted and the log verbosity must be greater than 2.

The *Close Network Connection* response action is very greedy for the MDP model because it is characterized by very low response time, cost and impact attributes, yet without providing good chances of successfully dealing with the ongoing attack. Therefore it could happen that in finding an optimal response policy the *Close Network Connection* response action could be selected tens of times, without having a positive effect on the system. For this reason, we generally introduce a dynamic reward computation based on an exponential increase of the penalty score of the considered response action every time that the action is actually executed. This dynamic reward computation is optional and can be activated on selected response actions. In our case it has been activated only on *Close Network Connection*.

D. Termination Function

We identify the set S_{tgt} of target states by defining a termination function $T : S \rightarrow \{true, false\}$. We allow the policy plan to terminate when the system has reached either a state of *controlled anomaly* or a state of *fully clean system*. We define a controlled anomaly state S_{ano} as: $S_{ano} = \{s \in S | (P_{scan} < T_2 \wedge P_{vsftpd} < T_2 \wedge P_{smbd} < T_2 \wedge P_{phpcgi} < T_2 \wedge P_{irc} < T_2 \wedge P_{distcc} < T_2 \wedge P_{rmi} < T_2) \wedge (P_{scan} \geq T_1 \vee P_{vsftpd} \geq T_1 \vee P_{smbd} \geq T_1 \vee P_{phpcgi} \geq T_1 \vee P_{irc} \geq T_1 \vee P_{distcc} \geq T_1 \vee P_{rmi} \geq T_1) \wedge (\mathit{firewall} \wedge \mathit{blocked\_ips} = \emptyset \wedge \mathit{flowlimited\_ips} \neq \emptyset \wedge \mathit{honeypot\_ips} = \emptyset \wedge \mathit{logVerb} > 0 \wedge \mathit{active} \wedge \neg \mathit{quarantined})\}$. We consider a system to be in a controlled anomaly state when the threshold T_2 is never hit by any of the considered attack probabilities and there is at least one attack probability greater than or equal to the threshold T_1. The anomaly is controlled because we impose as a requirement that there must be at least one entry in the set of the flow limited IPs.

Action Name	Response Time (sec)	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
Manual Resolution	3600	200	0
System Reboot	60	6	0.7
System Shutdown	30	6	1
Backup Host	3600	10	0.1
Close Network Connection	1	1	0.2
Software Update	600	300	0.1

TABLE I
ACTION PARAMETER SUMMARY

We define a fully clean system state S_{clean} as: $S_{clean} = \{s \in S | P_{scan} < T_1 \wedge P_{vsftpd} < T_1 \wedge P_{smbd} < T_1 \wedge P_{phpcgi} < T_1 \wedge P_{irc} < T_1 \wedge P_{distcc} < T_1 \wedge P_{rmi} < T_1 \wedge \mathit{blocked\_ips} = \emptyset \wedge \mathit{flowlimited\_ips} = \emptyset \wedge \mathit{honeypot\_ips} = \emptyset \wedge \mathit{logVerb} = 0 \wedge \mathit{active} \wedge \neg \mathit{quarantined}\}$. 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. We have therefore: $S_{tgt} = S_{ano} \cup S_{clean}$. We propose this specific termination function to provide an example of the entire lifecycle of the IRS, including both defense actions (e.g., *Block Source IP*, *Software Update*) and release actions (e.g., *Unblock Source IP*, *Decrease Log Verbosity*): in a real environment constraints such as no firewall rules configured are unlikely to appear.

IV. PERFORMANCE EVALUATION

The typical methods used to find the optimal policy π that maximizes the total reward of an MDP require the manipulation of a value function. A value function represents the expected objective value obtained following policy π from an initial state $s \in S$. One of the most used algorithm to compute the value function is the *Value Iteration* (VI) [4], which produces successive approximations of the optimal value function until the expected objective value is stable for all the MDP states. Unfortunately, even if each iteration can be performed in $O(|A||S|^2)$ steps [13], the number of states composing the MDP grows exponentially with the number of the defined attributes, thus limiting the applicability of the approach only to small systems. For this reason, in order to show the applicability of our model to large systems, we compare the performances (planning time and obtained reward) of the VI algorithm with the performances of the sub-optimal rollout-based Monte-Carlo algorithm named UCT introduced in [15]. The comparison will show that, for systems where a small reward degradation is acceptable, the planning time can be improved by more than 3 orders of magnitude.

The algorithms were applied in order to find a policy for a system composed by up to 1000 boolean state attributes and up

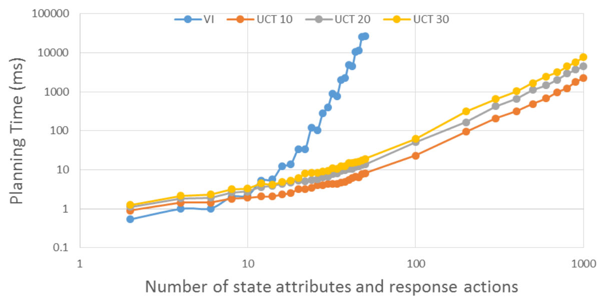

Fig. 2. Planning Time Comparison

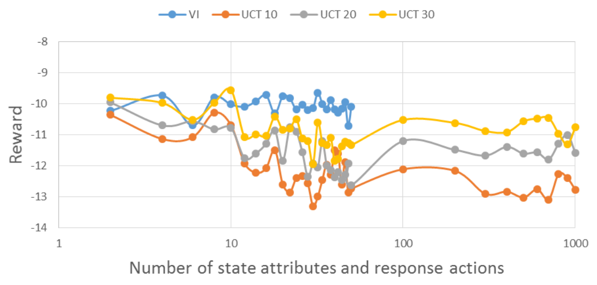

Fig. 3. Rewards comparison

to 1000 response actions. Each action is bound to one attribute and it changes its boolean value when executed, in order 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 the tests have been executed on a single compute node of the Shadow supercomputer at Mississippi State University, characterized by 20 cores and 512 GB of RAM. Anyway, the number of cores does not affect the overall planning time because both the algorithms have a single-thread implementation.

Figure 2 compares the planning time of the VI algorithm configured with $\gamma = 0.9$ with the UCT algorithm configured to perform 10, 20 or 30 rollouts and with a look-ahead of 10. Results highlight that VI's planning time quickly becomes high even for systems with only 50 state attributes and 50 response actions. By contrast, the UCT with 10 rollouts algorithm is able to plan a policy in less than 2 seconds for a system with 1000 state attributes and 1000 response actions.

Figure 3 compares the obtained rewards. As expected, the average reward obtained by VI is close to -10, specifically -10.07 because it always choose the best response actions. By contrast, the UCT algorithm with 30 rollouts produces an average reward of -10.86.

V. Experimental Results

In this section we describe the experiments we carried out to validate the effectiveness of the proposed approach. We set up a system composed by: an HPC cluster based on Rocks [2], a Snort Intrusion Detection System (IDS) [21] and the

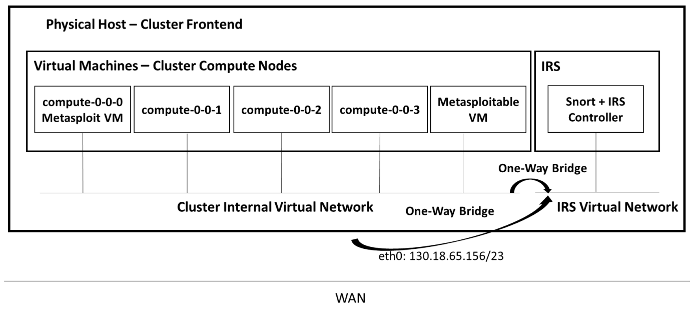

Fig. 4. Testbed Architecture

IRS controller described in Section III. The output of Snort is therefore considered as if it was the output of the Event Manager component described in Section I-A.

Figure 4 represents the architecture of the testbed, which is composed by a single physical machine which hosts two separate virtual networks, namely: (i) Cluster Internal Virtual Network and (ii) IRS Virtual Network. The former is attached to all the compute nodes of the Rocks cluster, while 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 completely isolated.

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 he is trying to exploit vulnerabilities exposed by another compute node. To this end, we set up 5 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. The compromised compute node is a VM in which we installed the Metasploit software [17]. We use this VM to scan the internal network and to launch attacks towards the vulnerable VM `metasploitable`. The latter is a publicly downloadable VM that exposes 6 vulnerabilities, as described in Section V-A.

A. Vulnerabilities

In this section we describe the six vulnerabilities that we consider in the experiments. We chose these vulnerabilities among the others because the software exposing them is freely available and ready to be exploited by downloading the `metasploitable` VM from the *Metasploit* web site [1].

1) **OSVBD-73753.** vsftpd on vsftpd.beasts.org Trojaned Distribution. The backdoor payload is constituted by a

[1]https://information.rapid7.com/metasploitable-download.html

":)" smiley face in the FTP username and the result is a TCP callback shell.

2) **CVE-2007-2447.** There is a command execution vulerability in Samba versions 3.0.20 through 3.0.25rc3 when using the non-default "username map script" configuration option. By specifying a username containing shell meta characters, attackers can execute arbitrary commands.
3) **CVE-2012-1823.** When run as a CGI, PHP up to version 5.3.12 and 5.4.2 is vulnerable to an argument injection vulnerability: in case an unescaped '=' is passed in the HTTP query string, the string is split on '+' (encoded space) characters, urldecoded, passed to a function that escapes shell metacharacters (the "encoded in a system-defined manner" from the RFC) and then passes them to the CGI binary.
4) **CVE-2010-2075.** UnrealIRCd 3.2.8.1, as distributed on certain mirror sites from November 2009 through June 2010, contains an externally introduced modification (Trojan Horse) in the DEBUG3_DOLOG_SYSTEM macro, which allows remote attackers to execute arbitrary commands.
5) **CVE-2004-2687.** distcc 2.x, when not configured to restrict access to the server port, allows remote attackers to execute arbitrary commands via compilation jobs, which are executed by the server without authorization checks.
6) **CVE-2011-3556.** The vulnerability is due to the default configuration of the RMI Registry and RMI Activation services allowing the loading of classes from a remote URL. A remote unauthenticated attacker can leverage this vulnerability by sending a crafted RMI message to a target server. In an attack scenario where code execution is successful the injected code will be executed within the security context of the target service.

All these vulnerabilities can be exploited by using already developed exploits available in the Metasploit DB.

B. Snort Configuration

Given the architecture of the testbed, Snort will work as an asynchronous IDS. That is, it will be able to detect malicious traffic, but not to stop it. Snort provides three rules sets, named respectively: *Community Rules*, *Registered Rules*, *Subscribed Rules*. Community rules are publicly available; registered rules are freely available upon registration; subscribed rules are instead available buying a specific Cisco subscription plan. We configured it to use both the Community and Registered rules and it was able to detect out-of-the-box only one of the six exploits that we launched from `compute-0-0-0` to `metasploitable`, specifically CVE-2012-2335. In order to create new Snort rules to handle unidentified attacks, we analyzed with the tool *Wireshark* the network traffic between `compute-0-0-0` and `metasploitable` during the attacks to find characteristic signatures. For space reasons, in the following we describe only one representative rule among the five rules added to Snort.

- **OSVBD-73753 Exploit Analysis.** As described in OSVDB, the exploit tries to log into the FTP server by using a username ending with the string ":)". We therefore added the following rule in the Snort DB:
```
alert tcp any any -> any 21 (msg:"vsFTPd
backdoor detected"; sid:80000000;rev:1;
classtype:suspicious-login; content:"|3a 29|";)
```
This rule generates an alert when the payload of TCP segments coming from any source IP and any source port and directed to any destination IP and port 21 contain the hexadecimal string: "3a 29", corresponding to the ASCII characters: ":)".

Please note that the added rules are not intended to be used in a production environment: specifically, the presented rule would trigger a suspicious login alert every time a ":)" string is found in the payload of any analysed packet, thus generating a lot of false positives.

C. Simulation of the Controller Behavior

We ran three different sets of simulations: the first set was carried out in order to show how the controller is able to compute optimal policies to respond to a portscan attack; in the second set the system is subject to a vulnerability exploit; finally, in the third set, we show the controller behavior when used to face multiple concurrent attacks. All the simulations have been repeated 10000 times, with the controller configured to use the VI algorithm and the reward function configured to optimize the response policy exclusively either on response time, or cost or impact. For each experiment we report the overall average resolution time, cost and impact attributes, each one computed, respectively, as average sum of the response times, costs and impacts of the actions included in the response policy. Since the produced policies are probabilistic, we also provide the confidence intervals of each attribute. It is important to note that, given the probabilistic nature of the planned policies, they represent different possible evolutions of the system. For space reason, we limit our discussion only to the most likely system evolution scenarios and we will show results only for the resolution time metric, but the same considerations apply for the other cases. Furthermore, we show how the γ discount factor, which dictates the short- or long-term nature of the policy, impacts the obtained results.

1) Portscan Attack: a portscan attack is usually one of the first steps an attacker carries out in order to discover the vulnerabilities of a system. It is a really common attack and it does not necessarily imply that the system has been compromised. Rather it must be considered as a first alarm in order to prepare the system to a prospective future attack. For this reason, the model has been designed so that not all the response actions are available to counter the portscan attacks: disruptive actions such as *System Reboot*, *Quarantine Host* and so on are only applied when there is the evidence of an exploited vulnerability, as specified in the action preconditions described in Section III-C.

Figure 5 compares the average resolution times obtained with the different optimization methods. The lowest resolution

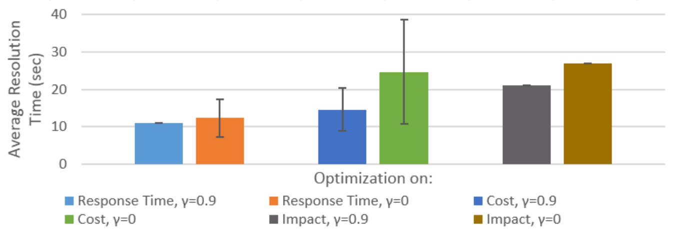

Fig. 5. Average Portscan Resolution Time

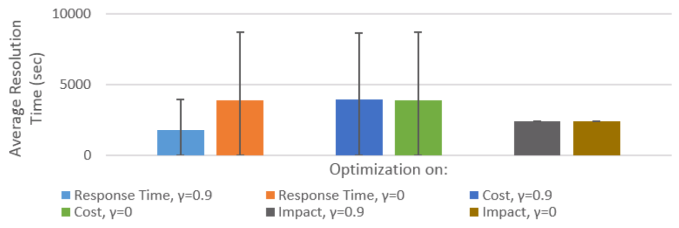

Fig. 6. Average Vulnerability Exploit Resolution Time

time has been achieved when the controller was set to optimize on response time and with $\gamma = 0.9$. Generally speaking, the results obtained with $\gamma = 0.9$ are always better than the results obtained with $\gamma = 0$. This means that when the controller is configured to produce long-term policies, it always provides better results. Specifically, when the controller was set to optimize on response time and $\gamma = 0.9$, it produced 6 equivalent policies, and the most likely was: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, blockSrcIP, unblockSrcip, decreaseLogVerb, decreaseLogVerb*. We can split such a policy in three blocks: (i) preparation, (ii) response and (iii) conclusion. The preparation phase regarded the alert generation, the increase of the verbosity of the logging system and the firewall activation. The core response action is *blockSrcIp*, which configures the firewall to drop incoming packets from the IP that is generating the malicious traffic. Finally, conclusion actions are carried out to restore the system to normal conditions by unblocking the source IP address and by decreasing the log verbosity. The action *increaseLogVerb*, as well as the action *decreaseLogVerb*, have been added twice to the response policy because the action *blockSrcIp* needs a log verbosity at least equal to 2. The other 5 generated policies in this case only differ in the order of the actions planned for the preparation and conclusion phases.

When the controller was configured to optimize on response time, but with $\gamma = 0$, it produced 20 different policies. Even if the most likely was the same that was computed in the $\gamma = 0.9$ case, in this case a lot of locally optimal, but globally sub-optimal policies have been planned. In fact, setting $\gamma = 0$ results in a greedy approach, which is not necessarily optimal in a long-term perspective. Indeed, besides the most likely policy, the controller planned a lot of policies similar to: *generateAlert, increaseLogVerb, increaseLogVerb, increaseLogVerb, increaseLogVerb, increaseLogVerb, activateFirewall, blockSrcIP, unblockSrcip, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb*. As described in Table I, the *Firewall Activation* and *Increase Log Verbosity* actions have both the same response time. Therefore, after having selected the fastest action (*Generate Alert*), since in the resulting state the action *Block Source IP* is unavailable because it first needs the firewall to be activated, the greedy approach is not able to distinguish between the *Firewall Activation* and *Increase Log Verbosity* because they both have a response time equal to 2. The choice between them is therefore probabilistic with $1/2$ probability each and it might happen that the controller chooses five times in sequence to increase the log verbosity, even if a log verbosity equal to 2 is sufficient for the execution of the main action *blockSrcIP*.

2) Vulnerability Exploit: a vulnerability exploit happens when an attacker successfully manages to breach a system by exploiting an exposed vulnerability. Contrarily to the portscan attack, most of times it implies that the system has been compromised. In this case we try to respond to the attack with all the set of actions. However, the *Block Source IP*, *Flow Rate Limit* and *Redirect to Honeypot* are not able to solve the problem because the exploitation of the considered vulnerabilities opens a reverse shell in which the compromised host directly connects, as a client, to the machine of the attacker. Since most firewalls are stateful, we modeled such actions so that they are not able to solve the problem because in the real system they would not affect already established connections. However, it is possible to tailor the controller behavior by just modifying the pre-conditions and the post-conditions of the aforementioned actions in the model.

Figure 6 compares the average resolution times and confidence intervals obtained with the different optimization methods. The lowest resolution time, along with the smallest confidence interval, has been achieved with the controller set to optimize on response time and with $\gamma = 0.9$. Specifically, when the controller was set to optimize on response time with $\gamma = 0.9$, the most likely policy was: *increaseLogVerb, generateAlert, activateFirewall, increaseLogVerb, increaseLogVerb, increaseLogVerb, increaseLogVerb, systemReboot, backup, softwareUpdate, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb*. We can split such a policy in five blocks: (i) first preparation; (ii) first response attempt; (iii) second preparation; (iv) second response attempt; (v) conclusion. It is worth noting how the failure in solving the problem with the first attempt leaded to a countermeasure escalation for the second attempt. Specifically, the first preparation phase increased the log verbosity to the maximum level (required to issue a system reboot) and activated the firewall. The first response attempt is the system reboot which, according to the model, has a 0.3 probability of successfully countering the currently ongoing attack. In this case the system reboot was not able to successfully face the attack and therefore the controller planned a software update in order to remove the vulnerability at its roots. The second preparation phase is therefore the execution of a backup before the software update. Finally the software update solved the problem and conclusion

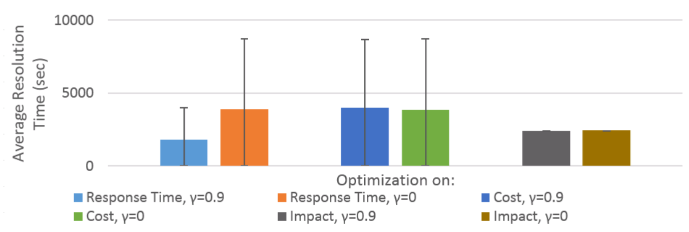

Fig. 7. Average Combined Attack Resolution Time

actions have been selected to complete the policy.

When the controller was set to optimize on response time, but with $\gamma = 0$, the most likely policy was: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, increaseLogVerb, increaseLogVerb, increaseLogVerb, systemReboot, quarantineSystem, backup, manualResolution, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb.* The first steps are the same as in the case with $\gamma = 0.9$. Even if from the first selected response actions it could seem that the controller is doing a good job, this merely is a coincidence: the first preparation actions are the same only because those are the actions with the lowest possible response time. After the system reboot, following the greedy approach, the action with the lowest response time has been selected (*quarantineSystem*). This action has not been taken before because it first requires a system reboot. After the system has been quarantined, the only available choices were *Backup* or *manualResolution*, both of them with the same response time. The former was wrongly selected (it does not actually solve the problem and it is only a prerequisite for the *softwareUpdate* action) and only as a last resort the *manualResolution* action has been chosen.

3) Combined Vulnerability and Portscan Attack: in this section we will describe the policies planned by the controller when the controlled system was subject to a combined flow of portscan and vulnerability attacks. In this test all the response actions have been used to counter the ongoing attack.

Figure 7 compares the average resolution times and confidence intervals obtained with the different optimization methods. The lowest resolution time, along with the smallest confidence interval, has been registered when the controller was set to optimize on response time and with $\gamma = 0.9$. Specifically, when the controller was set to optimize on response time, with $\gamma = 0.9$, the most likely policy was: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, blockSrcIP, increaseLogVerb, increaseLogVerb, increaseLogVerb, systemReboot, backup, softwareUpdate, unblockSrcip, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb.* We can split such a policy in seven blocks: (i) first preparation; (ii) first response attempt; (iii) second preparation; (iv) second response attempt; (v) third preparation; (vi) third response attempt; (vii) conclusion. It is worth noting how this time the countermeasure escalation occurred twice. In fact, this policy contains three response attempts for the ongoing attacks. In detail, the first preparation phase activated the firewall and increased the log verbosity to the minimum level required to block the source IP address. The first response attempt is the configuration of the firewall to block the IP address originating the malicious traffic. Even if this response, according to the model, is able to effectively protect the system from the portscan attack, it does not address the ongoing vulnerability exploit. The second defence attempt begins by increasing the log verbosity to its maximum level, required for the system reboot. Unfortunately, as in the previous case, the system reboot has a low probability of successfully countering the vulnerability exploit attack, therefore a third response attempt, consisting in a software update preceded by a system backup, has been planned by the controller. This last response attempt was actually able to face the ongoing attack and, finally, the controller selected the response actions needed to restore its standard functionality level.

When the controller was set to optimize on response time, but with $\gamma = 0$, the most likely policy was: *generateAlert, increaseLogVerb, activateFirewall, increaseLogVerb, blockSrcIP, increaseLogVerb, increaseLogVerb, increaseLogVerb, systemReboot, quarantineSystem, backup, manualResolution, unblockSrcip, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb, decreaseLogVerb.* Comparing this policy to the most computed one in the $\gamma = 0.9$ case, it is easy to note the greedy behavior: the first four blocks of the policy are exactly the same because the globally optimal path matches with the greedy path. However, after the system reboot, instead of executing the most time consuming backup and then the less time consuming software update, the policy preferred to take immediately the less time consuming action, that is, to quarantine the system. Unfortunately, at that point, the only further action that it could take was the manual resolution.

VI. Conclusions and Future Works

During the last decade a number of IRSs have been proposed in order to face the ever growing frequency and complexity of attacks directed to computer systems. All the proposed approaches, however, only considered either a static mapping of the best response action to the currently detected attack or the dynamic evaluation of the available response actions according to a set of pre-defined attributes. As a consequence, they were only able to produce optimal short-term response policies composed by a single response action. Since this single action had to be resolutive for the ongoing attack, all the response actions had to be designed and developed as monolithic applications or functions, therefore becoming hardly reusable to counter attacks different from the one they have been designed for.

In this paper we introduced a MDP-based controller for an autonomic IRS. Its novelty resides in the planning of long-term response policies by composing atomic response actions. This long-term planning exploits the concept of system state, which decouples the attacks from the responses. This decouplement makes it possible for the system administrator to focus on the description of the attacks and of the responses as effects on the system state, rather than on the attack-response bindings. As a consequence, there is no more need for monolithic response actions expressly thought to face specific attacks, but the already existing atomic actions can be composed in order to face known and, possibly, unknown attacks.

We provided a complete overview of the model and of the methodology of the proposed controller, together with a detailed description of the testbed used to test its features. We showed that, when a small reward degradation is acceptable, the proposed approach can be applied to large systems. We experimentally proved its effectiveness using real-world attack scenarios. Specifically, experimental results show that long-term planned policies always provide better results than the short-term ones and the threat resolution time can be reduced up to 56% in the considered scenario.

The main limitation about the applicability of the proposed approach in practice still remains the initial work that has to be carried out by the system administrator in (i) capturing the minimal set of system attributes required, (ii) describing the effects of the attacks on the system attributes and (iii) describe the effects of the responses on the system attributes. Even if the amount of work is reduced in comparison to the attack-response mapping (it is $O(|Att| + |A|)$ instead of $O(|Att| \times |A|)$, where $|Att|$ is the number of considered attacks), it probably requires a more skilled system administrator. For this reason, as a future work, we plan to realize a meta-model in which we will define standard components and connections that could be used by the system administrators to visually design the model of their system. Having such a meta-model will enable the development of standard attacks and response libraries that, integrated with the personalized system model, will allow the IRS to provide response policies tailored for the specific system.

At this time the optimal response policies are computed by maximizing a reward function based on a weighted sum of normalized criteria. However, we plan to introduce a constraint-based optimization problem, useful to model for instance thresholds on the maximum resolution time or on the maximum allowed cost. Furthermore, since the produced policies are based on a probabilistic evolution of the system, it is important to establish a feedback loop between the controller and the managed system, in order to check whether the current system state corresponds to the planned one and eventually to update the action's post-condition probabilities according to the real system evolution. The current work is aimed at producing reactive policies, that is, policies that defend the system after the attack has been detected. We plan to introduce proactive response policies using Multi-Agent competitive MDPs modeling an attacker-IRS game to predict the future attacker's behavior. Moreover, we plan to consider non-deterministic MDPs [9] in order to produce a set of near-optimal decision policies from which the system administrator could pick the best one according to his/her personal knowledge.

Acknowledgement

Funding for this work was (partially) provided by the Pacific Northwest National Laboratory, under U.S. Department of Energy Contract DE-AC05-76RL01830.

References

[1] Akamai's state of the internet: Q2 2015 report. https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report.

[2] Rocks cluster. http://www.rocksclusters.org.

[3] S. Abdelwahed, J. Bai, R. Su, and N. Kandasamy. On the application of predictive control techniques for adaptive performance management of computing systems. *Network and Service Management, IEEE Transactions on*, 6(4):212–225, 2009.

[4] R. Bellman. Dynamic programming. princeton, nj: Princeton universitypress. *BellmanDynamic Programming1957*, 1957.

[5] Q. Chen, S. Abdelwahed, and A. Erradi. A model-based validated autonomic approach to self-protect computing systems. *Internet of Things Journal, IEEE*, 1(5):446–460, 2014.

[6] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang. Nice: Network intrusion detection and countermeasure selection in virtual network systems. *Dependable and Secure Computing, IEEE Transactions on*, 10(4):198–211, 2013.

[7] F. Cohen. Simulating cyber attacks, defences, and consequences. *Computers & Security*, 18(6):479–518, 1999.

[8] C. Diuk, A. Cohen, and M. L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.

[9] M. M. Fard and J. Pineau. Non-deterministic policies in markovian decision processes. *J. Artif. Intell. Res.(JAIR)*, 40:1–24, 2011.

[10] B. A. Fessi, S. Benabdallah, N. Boudriga, and M. Hamdi. A multi-attribute decision model for intrusion response system. *Information Sciences*, 270:237–254, 2014.

[11] B. Foo, Y.-S. Wu, Y.-C. Mao, S. Bagchi, and E. Spafford. Adepts: adaptive intrusion response using attack graphs in an e-commerce environment. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 508–517. IEEE, 2005.

[12] C. Hwang and K. Yoon. *Multiple Criteria Decision Making, Lecture Notes in Economics and Mathematical Systems*. Springer, 1981.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285, 1996.

[14] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[15] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[16] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc., 1995.

[17] C. J. Marquez. An analysis of the ids penetration tool: Metasploit. *The InfoSec Writers Text Library, Dec*, 9, 2010.

[18] E. Miehling, M. Rasouli, and D. Teneketzis. Optimal defense policies for partially observable spreading processes on bayesian attack graphs. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 67–76. ACM, 2015.

[19] C. Mu and Y. Li. An intrusion response decision-making model based on hierarchical task network planning. *Expert systems with applications*, 37(3):2465–2472, 2010.

[20] S. Ossenbuhl, J. Steinberger, and H. Baier. Towards automated incident handling: How to select an appropriate response against a network-based attack? In *IT Security Incident Management & IT Forensics (IMF), 2015 Ninth International Conference on*, pages 51–67. IEEE, 2015.

[21] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[22] A. Shameli-Sendi and M. Dagenais. Orcef: Online response cost evaluation framework for intrusion response system. *Journal of Network and Computer Applications*, 2015.

[23] T. Toth and C. Kruegel. Evaluating the impact of automated intrusion response mechanisms. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 301–310. IEEE, 2002.

[24] D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(1):8, 2012.