A Model-Integrated Approach to Designing Self-Protecting Systems

Stefano Iannucci, *Member, IEEE,* Sherif Abdelwahed, *Senior Member, IEEE,* Andrea Montemaggio, Melissa Hannis, Leslie Leonard, Jason S. King, John A. Hamilton, Jr., *Senior Member, IEEE*

Abstract—One of the major trends in research on Self-Protecting Systems is to use a model of the system to be protected to predict its evolution. However, very often, devising the model requires special knowledge of mathematical frameworks, that prevents the adoption of this technique outside of the academic environment. Furthermore, some of the proposed approaches suffer from the curse of dimensionality, as their complexity is exponential in the size of the protected system. In this paper, we introduce a model-integrated approach for the design of Self-Protecting Systems, which automatically generates and solves Markov Decision Processes (MDPs) to obtain optimal defense strategies for systems under attack. MDPs are created in such a way that the size of the state space does not depend on the size of the system, but on the scope of the attack, which allows us to apply it to systems of arbitrary size.

Index Terms—Intrusion Response System, Autonomic Security Management

1 Introduction

Effectively protecting enterprise networks from cyber attacks is a challenging task due to their large scale and the heterogeneity of the underlying hardware and software components. Current Security Information and Event Management Systems (SIEM) [30] products (e.g., [4], [9], [28]) are focused on intrusion detection, leaving the responsibility of intrusion response to the system administrator, where he/she typically manually protects the system once an alert is raised, or configures a static mapping between every alert typology and its proper countermeasure. However, when trying to counter an attack, the time factor is critical [14] and any non-guided human resolution attempt could introduce a significant stress and delay to the execution of the proper response, thus providing to the attackers more time to accomplish their objectives [16].

Intrusion Response Systems (IRSs) (e.g., [29], [50], [54]) are usually classified according to their level of automation [52], i.e., notification systems, manual response systems and automated response systems. Notification systems do not usually provide automated protection, rather they provide a platform on top of which the system administrator can build his own countermeasure selection methodology. Manual response systems (e.g., [58], [60]) introduce some level of automation by providing a static mapping between the currently detected attack and the prospective responses. SIEMs usually provide this level of automation. However, an attack-response static mapping has been proven not to be an effective approach to protect a system [29], because of the scalability issues introduced by the massive amount of newly discovered attacks and by the ability of the attackers to bypass known protection mechanisms. Automated response systems (e.g., [11], [18], [34], [42], [51], [53]) are designed so that the system defense process does not require any human intervention. Usually they use a model of the system (e.g., [62]) and/or of the attacker (e.g., [19], [63]) in order to predict the evolution of the system itself and the attacker's strategy.

Most of the existing literature treats separately the intrusion detection and response problems and, to the best of our knowledge, none of the existing works aims at producing a comprehensive software architecture with the corresponding prototype that includes all the phases of defense life-cycle. To this end, we developed a model-based Autonomic Security Management (ASM) framework, based on the Monitor, Analyze, Plan, Execute and Knowledge (MAPE-K) loop for autonomic systems [23], [33]. In this paper, we focus on the Plan phase of the ASM, and we introduce an approach based on Model Integrated Computing (MIC) [46] to automatically instantiate, reduce, and solve the intrusion response problem.

MIC has been widely used to achieve autonomic performance optimization (e.g., [17], [44]), for cyber-security experimentation of cyber-physical systems [61] and for formal validation of cyber-security constraints [46]. The proposed approach relies on a meta-model that captures the architecture and dynamics of the system from security perspectives. System dynamics are used to predict the system evolution and, ultimately, to compute the protection plan. We focus in this paper on enterprise systems as a domain of application. However, the proposed approach can be easily extended to other domains such as, for example, health care systems and cyber-physical systems.

1.1 Contributions and Organization

In our previous works [25], [26], we introduced an approach based on a Markov Decision Process (MDP) [5] and stochastic games [8] for planning optimal system defense strategies

- *S. Iannucci is with the Department of Computer Science and Engineering, Mississippi State University, Starkville, MS.*
- *S. Abdelwahed is with the Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA*
- *A. Montemaggio, M. Hannis, J. A. Hamilton are with the Center for Cyber Innovation (CCI), Mississippi State University, Starkville, MS.*
- *L. Leonard and J. King are with the U.S. Army Engineer Research and Development Center (ERDC), Vicksburg, MS.*

assuming zero or partial knowledge of the attacker. However, one of the main limitations of the proposed approach was that a new MDP had to be formulated ad-hoc for every different system we intended to protect. For this reason, in this paper we propose an approach based on MIC to graphically design system models, more amenable to system administrators, that can be automatically transformed to an MDP formulation. Furthermore, MDPs are known to suffer the curse of dimensionality [5], because they are based on a state space that grows exponentially according to the size of the defended system. For this reason, we presented in [27] an MDP solver that was able to leverage the existence of Intel Xeon Phi many-core accelerators to reduce the planning time. However, we observe that exploiting accelerators is only a part of the solution, because only a constant speed-up can be obtained, whereas the problem has an exponential complexity. To this end, we introduce in this paper a technique for the creation of MDPs is such a way that the state space is not dependent on the size of the modeled system, but only on the scope of the attack, that is, on the amount of components of the system that are directly or indirectly affected by the attack. We also provide a formal demonstration that, under certain conditions, the optimality of the solution is maintained. In order to evaluate our approach in a quantitative fashion, we provide an experimental case study highlighting (i) the effectiveness of state space reduction techniques and (ii) that the number of the components in the system to be protected is not the key factor undermining its applicability to real scenarios.

The remainder of the paper is structured as follows. Section 2 introduces the enterprise system meta-model. Section 3 presents the theory underlying MDP-based planning, the transformation from system model to MDP, and a technique to reduce the state space while maintaining the optimality of the solution of the MDP. Section 4 experimentally shows the effectiveness of the proposed state space reduction technique. Threats to the validity of the proposed approach are discussed in Section 5, whereas related works are discussed in Section 6. Finally, Section 7 concludes the work and discusses future works.

2 Model-Integrated System Development

Having a model of the system allows to predict its evolution [2] and to produce a defense strategy. Model-Driven Engineering can be used to design and implement model-based systems, and several paradigms have been proposed, such as, Model-Driven Architecture (MDA) and Model-Integrated Computing [6] (MIC). On one hand, MDA employs an approach to system modeling that is based on three view-points: computation independent, platform independent, and platform-specific [20]. Each one of these view-points can be used to create its respective model, that is, Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). On the other hand, MIC extends MDA by letting the designer model generic view-points of the system. That is, MIC introduces some additional flexibility that is particularly useful to model complex systems with custom aspects. One of the most commonly used languages for MDA is Eclipse Modeling Framework (EMF, [55]), whereas Generic

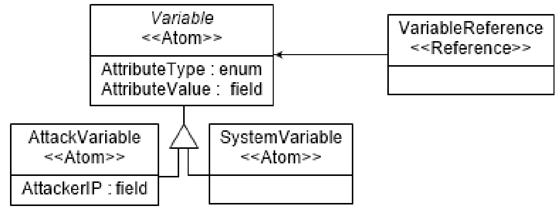


Fig. 1: Variable Meta-Model

Modeling Environment (GME, [35]) can be used for MIC. However, the authors of [6] have shown that, although non-trivial, a model transformation between GME to EMF is possible.

We adopt the MIC paradigm, and we use GME for its implementation. Although developed in an academic environment, GME is a production-ready tool used worldwide in industry and academia [32]. It envisages three user roles [38]: environment designers, domain experts, and component developers. Environment designers must have a deep knowledge of the domain, and a full understanding of how the GME toolbox works. They are in charge of building the meta-model for *a class* of systems, and the corresponding modeling language, which in turn is used by domain experts to build models of *instances* of systems. Finally, component designers leverage the structure provided by the meta-model to interpret models and build model-based software tools.

2.1 System Meta-Model

A meta-model is the model of a class of systems. In this paper we present the design of a meta-model that captures the structure and the dynamics of enterprise systems. The base concept on which the entire meta-model is founded is the `Variable` class. As shown in Figure 1, it can either represent the state of a specific system component, (`SystemVariable`, e.g., the status of a service, the CPU load of a particular server, the configuration of a network, and the version of an installed software), or the probability that a certain type of attack has currently been detected (`AttackVariable`). Variables are characterized by a type (e.g., `Boolean`, `Integer`, `Double`, and so on), and by a value.

The *Enterprise System Meta-Model* is represented in Figure 2. The latter defines the following hierarchical structure: the `SystemModel` is the highest level of abstraction of the system. It contains objects of type `Server`, `Network` and `NetworkConnection`. The `Server` class models real physical or virtual machines, possibly connected to a `Network` through a `NetworkConnection`. A `Firewall` is a specialization of a `Server` that models filtering rules and routing tables. Servers execute `Process` instances and contain `Data`, either in the form of a `File` or of a `Database` for a higher abstraction. Since different servers, processes and data can have different importance on a given system, we characterize the importance of a specific asset with a parameter named *Criticality*. We define the latter as an integer value ranging from 0 to 10, where 0 indicates that the considered asset is not critical for the system, while 10

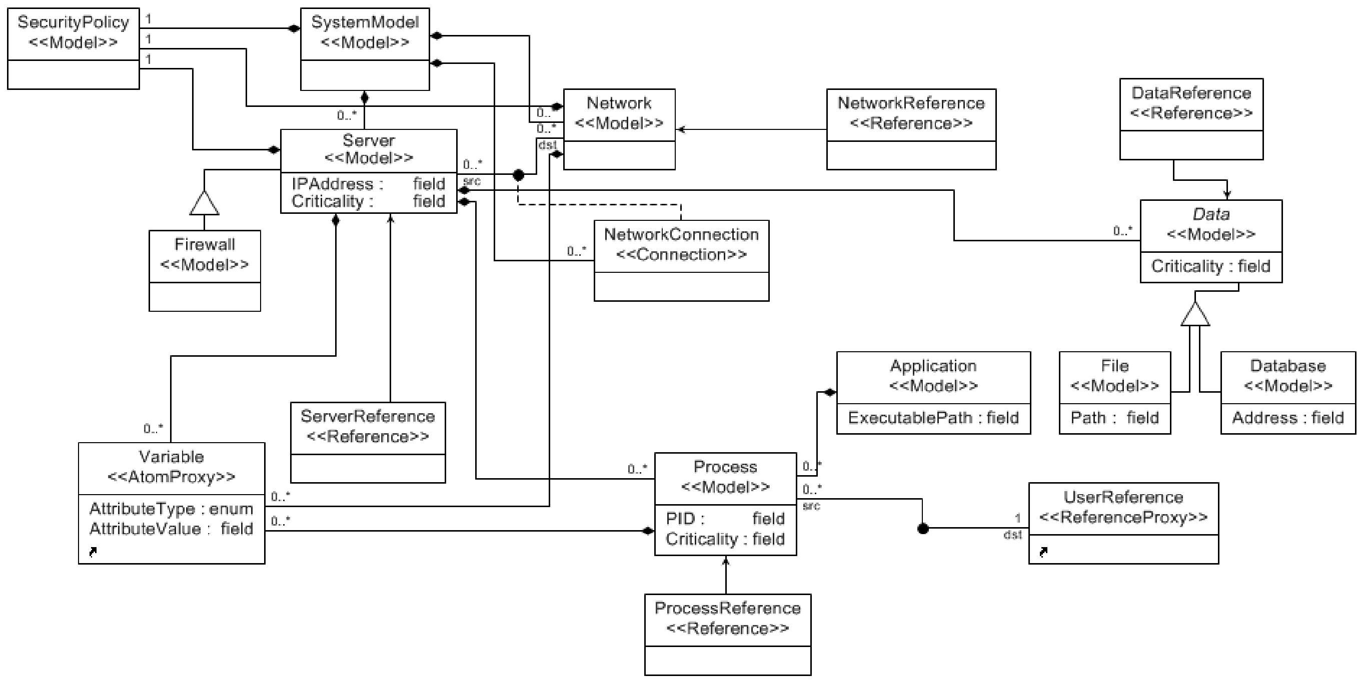


Fig. 2: Enterprise System Meta-Model

implies the highest possible criticality. We assume that the system administrator sets the appropriate criticality value to each one of the assets. Every process instance is characterized by a `UserReference`, which points to the owner of the process, and by the `Application` that spawned the process itself. Furthermore, each `Server`, `Process` and `Network` object includes an arbitrary number of `Variable` instances, that are used to model their attributes (e.g., whether or not a server is powered on, the system uptime, the path to the executable of a given process, and so on). The `SecurityPolicy` class models the security policy that the system has to comply with.

The hierarchical structure described so far is able to capture the static composition of a system. However, since our aim is to predict its evolution, we are also interested in its dynamic behavior. In other words, we need some tool that allows us to describe how the system state can change over time. To this end, we create an `Action` meta-model (not showed here for space reasons), with the `Action` class as its core component to model the execution of the defense commands on the system. Each `Action` instance uses a `VariableReference` instance to compose expressions regarding its pre-conditions and post-conditions. A `Precondition` instance is a boolean expression identifying the subset of states in which the action is executable; a `Postcondition` instance is a formula that sets new values to the referred variables. In other words, postconditions define the probability distribution of the next states of the system after the execution of an action. Actions can be executed on `Server`, `Process` and `Network` objects and are characterized by several parameters used in the defense strategy planning problem, such as:

- `cost`, representing the economic cost of the execution of a given action;
- `execTime`, representing the average amount of time needed to complete the execution of a given action;
- `confidentiality`, representing the impact on the Confidentiality attribute of the Confidentiality, Integrity, Availability (CIA) triad. This value ranges from 0 to 1, where 0 indicates no impact on the Confidentiality, while 1 implies total information disclosure;
- `integrity`, representing the impact on the Integrity attribute of the CIA triad. This value ranges from 0 (no impact) to 1 (total impairment);
- `availability`, representing the impact on the Availability attribute of the CIA triad. This value ranges from 0 (no impact) to 1 (asset unavailable).

An example of highest level of the hierarchy of a system model compliant with the proposed meta-model, an example of security policy, and an example of characterization of an action are depicted respectively in Figures 5, 6 and 7.

3 Intrusion Response Methodology

In the following, we provide a short introduction to the theory of MDP (Section 3.1); then, we discuss in detail how we transform a system model to a MDP-based representation (Section 3.2); finally, we present a technique for the reduction of the state space of the MDP (Section 3.3).

3.1 MDP-based Response Planner

Our approach to response planning is based on solving an instance of the MDP derived from the system model having the current state of the system as the initial state.

We define an MDP as a tuple $\langle S, A, P, R, T, \gamma \rangle$, where S is the state space that the agent can navigate, A is the finite set of actions available to the agent to navigate such a space and $T \subseteq S$ is a set of terminal states, i.e. the states from which the agent cannot move. The dynamics of the system are given by the transition probability function $P : S \times A \times S \mapsto [0,1]$ s.t. $P(s_0, a, s_1)$ is the probability that by executing the action a in state s_0, the next state is s_1. While moving through the state space, the agent is given bonuses and penalties according to the reward function $R : S \times A \times S \mapsto \mathbb{R}$, with $R(s_0, a, s_1)$ being the reward given to an agent that from the state s_0 moves to a state s_1 selecting the action a. Finally, the parameter $\gamma \in [0,1]$ is the discount factor, which models the agent's preference for short-term or long-term rewards.

Whilst the classical definition of an MDP [5] does not include the set T of terminal states, this notion is crucial to our approach. Indeed, we consider the set of terminal states as the set of all the states in which the security policy defined in the system model is satisfied, and we use it as a termination condition while solving the MDP instance.

In the domain of automated intrusion response, with the objective of simplifying the formulation of the system model, it is common to consider the actions independently from the state where they are executed [12], [26], [40], [53]. In this paper, we use a simplified reward function $\bar{R} : A \mapsto \mathbb{R}$, so that it only depends on the executed actions, that is, for all states $s_0, s_1 \in S$ and actions $a \in A$ we have that $R(s_0, a, s_1) = \bar{R}(a)$ holds.

According to this assumption, we define the reward function as a linear combination of five criteria: cost, execution time, confidentiality, integrity, availability.

$$\begin{aligned}\bar{R}(a) = Criticality \times (-w_t \frac{T(a)}{T_{max}} - w_c \frac{C(a)}{C_{max}} \\ - w_{conf} Conf(a) - w_i I(a) - w_a A(a))\end{aligned} \tag{1}$$

where $w_t, w_c, w_{conf}, w_i, w_a \in [0,1]$ reflect the importance of, respectively, execution time $T(a)$, cost $C(a)$, confidentiality $Conf(a)$, integrity $I(a)$, availability $A(a)$ optimization criteria for action a. $Criticality \in \{0, 1, \dots 10\}$ is taken as the maximum of the criticalities of the assets involved in the execution of action a and works as a negative reward amplifier that discourages, when possible, the execution of actions on critical components of the system. T_{max} and C_{max} represent respectively the maximum response time and the maximum cost for the considered response actions and are used to normalize their values.

The given definition of an MDP is quite impractical to work with, both for problem description purposes and to apply the state space reduction techniques discussed in Section 3.3, because the state space has no structure that can be exploited. For this reason, and in order to bridge the gap between the MDP theoretical framework and the behavioral meta-model described in Section 2.1, we adopt a factored representation of an MDP [21], derived automatically from a system model as described in Section 3.2. As we will formally define in the following, in this factored representation the state space is generated by the set of variables defined in the system model and the dynamics of the system state are described by a set of difference equations over the state variables associated with each action post-condition.

In the following, to keep the presentation clear, we assume the state variable values to range over a single arbitrary domain: the set of all the possible strings Σ^* from an arbitrary alphabet Σ. This does not hurt the generality since the discussion could be easily extended by introducing the set $\mathfrak{T}$ of the variable types as defined in the meta-model (see Section 2.1), the function $\mathfrak{D} : \mathfrak{T} \mapsto 2^{\Sigma^*}$ that maps a variable type to its domain (i.e. the set of all possible values a variable of that type can have, encoded as strings in Σ^*) and by adding some constraints between variables and their domain where required.

In order to formally define our factored representation of an MDP, to which we refer in the following as *MDP factored model*, we need to introduce some definitions and notation. Let V be a set of variables characterizing the state of a given system. We define the state space $\mathbb{S}_V$ generated by V as the set of all the functions $V \rightarrow \Sigma^*$, so that a system state is represented by a function that associates a value to each of the variables in V. In order to represent the post-condition dynamics, we define the family $\mathbb{E}_V$ of evaluation functions over the variables V as the set of all the functions $\mathbb{S}_V \rightarrow \Sigma^*$, namely the functions that evaluate a system state to a value in Σ^*. Similarly, to represent the action pre-condition and the system security policy (i.e., the MDP termination function) we define the family $\mathbb{B}_V$ of boolean evaluation functions as the set of all functions $\mathbb{S}_V \rightarrow \{true, false\}$, that assign a truth value to a system state. In the prototype implementation all these functions are represented as Spring Expression Language (SpEL) [43] strings and are evaluated at runtime with the system state as a context.

For a given variable set V, we define $\mathbb{P}_V \subseteq [0,1] \times \mathbb{D}_V$ as the set of all the post-conditions that can be represented over V, where $\mathbb{D}_V$ is the set of all functions $V \rightarrow \mathbb{E}_V$ that map the future value of each system variable to an evaluation function, viz. the set of all the representable system dynamics equations. Therefore, a post-condition is a tuple $\langle p, \lambda \rangle$ where p is the probability that the action outcome described by the system dynamics equation set λ occurs.

With these basic definitions set up, we can say that our representation of an MDP factored model is a tuple $\langle V, A, \xi, \Lambda, \phi, \bar{R}, \gamma \rangle$ where V is the set of system variables, A is the set of actions, $\xi : A \mapsto \mathbb{B}_V$ is a function that maps each action to its pre-condition evaluation function, $\Lambda : A \mapsto 2^{\mathbb{P}_V}$ is a function that associates a set of post-conditions to an action, $\phi \in \mathbb{B}_V$ is a boolean evaluation function to serve as the MDP termination function, $\bar{R} : A \mapsto \mathbb{R}$ is a stateless reward function and $\gamma \in [0,1]$ is the discount factor. Moreover, for Λ to be valid, the following must hold:

$$\forall a \in A \sum_{\langle p, \lambda \rangle \in \Lambda(a)} p = 1$$

Given an MDP factored model $\mathcal{M} = \langle V, A, \xi, \Lambda, \phi, \bar{R}, \gamma \rangle$, in the following we denote as $\hat{\mathcal{M}} = \langle \mathbb{S}_V, A, P, T, \bar{R}, \gamma \rangle$ the MDP resulting from the interpretation of the factored model $\mathcal{M}$, s.t. the set of terminal states $T = \{\sigma \in \mathbb{S}_V \mid \phi(\sigma) = true\}$ is the set of states satisfying the termination function ϕ and the transition probability function P is defined in terms of the system dynamics as follows

$$P(\sigma_0, x, \sigma_1) = \begin{cases} p & \exists \langle p, \lambda \rangle \in \Lambda(x) : \sigma_0 \xrightarrow{\lambda} \sigma_1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

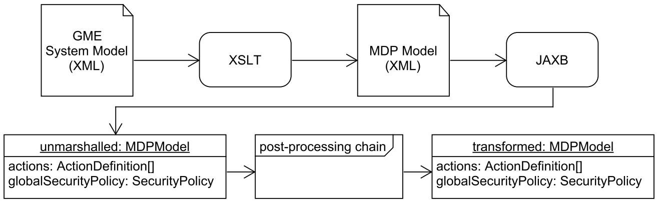


Fig. 3: The Model Transformation process

where $\xrightarrow{\lambda} \subseteq \mathbb{S}_V \times \mathbb{S}_V$ is a binary relation s.t. $\sigma_0 \xrightarrow{\lambda} \sigma_1$ holds iff $\forall v \in V\ \sigma_1(v) = \lambda(v)(\sigma_0)$.

From the implementation perspective, in the MDP solving phase the factored model is interpreted as an MDP suitable to be solved with the BURLAP library [1].

3.2 Transformation of the System Model to MDP

The diagram in Figure 3 shows the process of automatically transforming the GME system model into a MDP instance solvable by BURLAP. An XML of the exported system model maintains all of the configurations developed in GME. Since the GME model contains details that are not needed for the instantiation of the MDP problem (e.g., configurations of the monitoring agents, auto-deployment settings and so on), we apply a XSLT transformation to strip the model from all the unneeded details. Specifically, the transformed XML holds the extraction of: the variables for the creation of the MDP state structure, the attributes from each action that determine the MDP reward, the pre- and post-conditions of each action to create the MDP transition dynamics, and the security policy to build the MDP termination function. Afterwards, JAXB is used to unmarshall the MDP XML model into a Java instance of the MDP model. The post-processing chain enables additional arbitrary model-to-model transformations that we use to address the complexities that arise from some of the features provided by the modeling environment. For example, although not discussed in detail in this paper for space reasons, we defined the concept of *action templates* for actions that behave in a similar manner in the GME system model. In order to keep the MDP solver simple, these *action templates* are grounded into simple actions within the post-processing chain. Once the transformation process is complete, the instance of the MDP model is used as input to the BURLAP MDP solver.

3.3 System model reduction

In order to solve the minimal MDP in terms of size of the state space, we apply the *variables elimination* technique we developed to reduce the MDP state space acting on the MDP model, possibly keeping the ability to find optimal solutions. In the following, we introduce some more definitions to formally describe this technique. We define a function $\Psi_V : (\mathbb{E}_V \cup \mathbb{B}_V) \mapsto 2^V$ s.t. given an evaluation function in $\mathbb{E}_V$ or $\mathbb{B}_V$, returns all the variables in V the evaluated value depends upon. The Ψ_V function is defined as follows:

$$\Psi_V(f) = \big\{x \in V \,\big|\, \forall\sigma \in \mathbb{S}_V, v \in \Sigma^* f(\sigma) \neq f(\sigma[x\backslash v])\big\}$$

where with the $\sigma[x\backslash v]$ notation we mean a higher-order function $\mathbb{S}_V \times V \times \Sigma^* \mapsto \mathbb{S}_V$ s.t.

$$\sigma[x\backslash v](y) = \begin{cases} v & x = y \\ \sigma(y) & \text{otherwise} \end{cases}$$

Although in general this function could be challenging to compute, the factored representation makes its realization straightforward. From the implementation standpoint, since the evaluation functions are represented by expressions, the Ψ_V function is realized by returning all the variables referred in the given expression. For example, given an evaluation function $f(v_0, v_1)$ described by the expression $v_0 + v_1 + 1$, we have that $\Psi_V(f) = \{v_0, v_1\}$.

3.3.1 Variables elimination

Given an MDP factored model $\mathcal{M} = \langle V, A, \xi, \Lambda, \phi, \bar{R}, \gamma\rangle$, the variables elimination technique is aimed at building a reduced factored model $\mathcal{M}' = \langle V', A, \xi', \Lambda', \phi', \bar{R}, \gamma\rangle$, with $V' \subseteq V$ being the smallest subset of the original variable set, s.t. an optimal policy can still be found to bring the system into a state satisfying the system security policy.

The rationale behind this approach is to eliminate from the problem all the post-condition equations that do not directly or indirectly change the values of the variables referenced by the termination function. Thus, the main step in order to construct the reduced factored model, is to find the smallest subset of variables that preserves the possibility to evaluate the termination function.

Given an MDP factored model $\mathcal{M}$, let $L_V = \langle\, 2^V, \subseteq\, \rangle$ be the complete lattice of the power-set of V. For $\mathcal{M}$ we define the *dependency closure step* as a function $\delta_{\mathcal{M}} : 2^V \mapsto 2^V$ s.t.

$$\delta_{\mathcal{M}}(W) = W \cup \bigcup_{a \in A} \big\{x \in \Psi_V\big(\xi(a)\big) \cup \Psi_V(\lambda(w)) \,\big|\, \exists w \in W, \\ \langle p, \lambda\rangle \in \Lambda(a) : \Psi_V\big(\lambda(w)\big) \neq \emptyset\big\}$$

Indeed, each application of $\delta_{\mathcal{M}}$ returns the given set of variables (possibly) augmented with all the variables taken from the actions' pre-conditions and post-conditions equations, that directly influence the value of any other variable already present in the set.

Since $\delta_{\mathcal{M}}$ is defined to be an increasing function over L_V and V is finite, as a consequence of Tarski's fixed point theorem [59] we have that $\delta_{\mathcal{M}}$ has a least fixed point $\text{LFP}_{L_V}(\delta_{\mathcal{M}})$ and $\exists n \in \mathbb{N}, \text{LFP}_{L_V}(\delta_{\mathcal{M}}) = \delta^n_{\mathcal{M}}(\bot)$.

We are interested in finding the smallest subset of V still having all the variables used in the termination function, thus we work on the interval lattice $\hat{L}_V = \langle[V, \Psi_V(\phi)], \subseteq\rangle$, which is also a complete lattice, and we find the least fixed point $\Delta_{\mathcal{M}} = \text{LFP}_{\hat{L}_V}(\delta_{\mathcal{M}})$ iterating the application of the dependency closure step function over $\hat{L}_V$, starting from the infimum element $\Psi_V(\phi)$, i.e. the set of variables used by the termination function.

3.3.2 Construction of the reduced MDP model

Given an MDP factored model $\mathcal{M} = \langle V, A, \xi, \Lambda, \phi, \bar{R}, \gamma\rangle$, we build the reduced model $\mathcal{M}' = \langle V', A, \xi', \Lambda', \phi', \bar{R}, \gamma\rangle$ as follows:

$$
\begin{aligned}
V' &= \Delta_{\mathcal{M}} \\
\xi'(a)(\sigma) &= \bigvee_{\substack{\tau \in \mathbb{S}_V \\ \tau \restriction_{V'} = \sigma}} \xi(a)(\tau) \\
\Lambda'(a) &= \Big\{ \langle p, \lambda' \rangle \,\Big|\, \exists \langle p, \lambda \rangle \in \Lambda(a) : \\
&\qquad \lambda'(v) = \big\{ (\sigma, w) \mid \bigcap_{\substack{\tau \in \mathbb{S}_V \\ \tau \restriction_{V'} = \sigma}} \{\lambda(v)(\tau)\} = \{w\} \big\} \Big\} \\
\phi'(\sigma) &= \bigwedge_{\substack{\tau \in \mathbb{S}_V \\ \tau \restriction_{V'} = \sigma}} \phi(\tau)
\end{aligned}
$$

Since the state space $\mathbb{S}_{\mathbb{V}}$ is restricted to $\mathbb{S}_{V'}$, it may be the case that more than one state is mapped to the same reduced state of the restricted state space (see Figure 4). For this specific reason, the termination function ϕ' for a given reduced state is defined as the conjunction of the applications of the original termination function ϕ to all the states that map to the same reduced state.

As a consequence of how the post-conditions mapping Λ' is defined, we have the following property.

Lemma 1. *Given a post-condition dynamic equations set $\lambda : V \mapsto \mathbb{E}_V$ of some MDP factored model $\mathcal{M}$ and its corresponding equations set $\lambda' : V' \mapsto \mathbb{E}_{V'}$, built as shown in the definition of Λ' in the reduced factored model $\mathcal{M}'$ the following holds*

$$\forall \sigma_0, \sigma_1 \in \mathbb{S}_V \;\; \sigma_0 \xrightarrow{\lambda} \sigma_1 \iff \sigma_0 \restriction_{V'} \xrightarrow{\lambda'} \sigma_1 \restriction_{V'}$$

In the following, when we want to refer to a specific constituent of either an MDP factored model $\mathcal{M}$ or its interpreted MDP $\hat{\mathcal{M}}$, we will add a subscript to the symbol used to refer to that constituent, e.g. $V_{\mathcal{M}}$ for the set of variables of $\mathcal{M}$ and $P_{\hat{\mathcal{M}}}$ for the transition probability function of $\hat{\mathcal{M}}$.

Hereinafter, we use the following definition of the q-value function $\mathbb{Q}^*_{\hat{\mathcal{X}}}$ for an MDP $\hat{\mathcal{X}}$:

$$\mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, a) = \bar{R}_{\hat{\mathcal{X}}}(a) + \gamma \sum_{\sigma'} P_{\hat{\mathcal{X}}}(\sigma, a, \sigma') \cdot \mathbb{V}^*_{\hat{\mathcal{X}}}(\sigma') \tag{3}$$

where $\mathbb{V}^*_{\hat{\mathcal{X}}}$ is the state value function of the optimal policy $\pi^*_{\hat{\mathcal{X}}}$ for $\hat{\mathcal{X}}$, both defined as follows

$$\mathbb{V}^*_{\hat{\mathcal{X}}}(\sigma) = \max_{a \,\in\, \Xi_{\mathcal{X}}(\sigma)} \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, a) \tag{4}$$

$$\pi^*_{\hat{\mathcal{X}}}(\sigma) = \operatorname*{arg\,max}_{a \,\in\, \Xi_{\mathcal{X}}(\sigma)} \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, a) \tag{5}$$

where the function $\Xi_{\mathcal{X}} : \mathbb{S}_{V_{\mathcal{X}}} \mapsto 2^{A_{\mathcal{X}}}$ s.t. $\Xi_{\mathcal{X}}(\sigma) = \{a \in A_{\mathcal{X}} \mid \xi_{\mathcal{X}}(a)(\sigma)\}$ gives the subset of the actions defined in the factored model $\mathcal{X}$ that are available in a certain state.

Theorem 1. *For a given MDP $\hat{\mathcal{X}}'$ interpreted from the reduced factored model $\mathcal{X}'$, which was derived from a factored model $\mathcal{X}$ by applying the construction given in Section 3.3.2, the following holds*

$$\forall x \in A, \sigma \in \mathbb{S}_V \;\; \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, x) = \mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma \restriction_{V'}, x) \tag{6}$$

with V and V' being the variable sets, respectively, of $\mathcal{X}$ and $\mathcal{X}'$ and A being the action set of $\hat{\mathcal{X}}'$.

Theorem 2. *For a given MDP $\hat{\mathcal{X}}'$ interpreted from the reduced factored model $\mathcal{X}'$, which was derived from a factored model $\mathcal{X}$ by applying the construction given in Section 3.3.2, the following holds*

$$\forall \sigma \in \mathbb{S}_V \;\; \mathbb{V}^*_{\hat{\mathcal{X}}}(\sigma) = \mathbb{V}^*_{\hat{\mathcal{X}}'}(\sigma \restriction_{V'}) \tag{7}$$

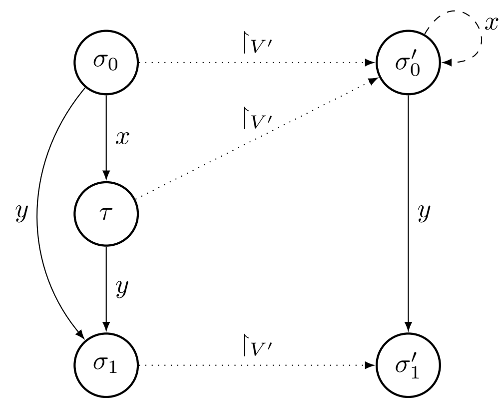


Fig. 4: State space restriction. The state transition graphs of both an MDP $\hat{\mathcal{M}}$ (left) and the reduced MDP $\hat{\mathcal{M}}'$ (right) are shown. A dotted line is drawn between a state of $\hat{\mathcal{M}}$ and the reduced state in $\hat{\mathcal{M}}'$, obtained by the restriction $\restriction_{V'}$ of the original state to the set $V' = \Delta_{\mathcal{M}}$. In the depicted case, two states τ and σ_0 are restricted to the same state $\sigma'_0 = \sigma_0 \restriction_{V'} = \tau \restriction_{V'}$. The action x is represented with a dashed line to mean it can be removed from the reduced MDP model $\mathcal{M}'$ due to its lack of utility (see lemma 2).

Moreover, if the reward function is always negative as the one defined in (1), we can optimize the reduced MDP further by removing post-conditions and potentially entire actions, according to the following result.

Lemma 2. *Given an MDP factored model $\mathcal{X}$ and an action $a \in A_{\mathcal{X}}$, under the hypothesis (h0) that the reward function is always negative, any post-condition $\langle p, \lambda \rangle \in \Lambda_{\mathcal{X}}(a)$ s.t.*

$$\forall \sigma \in \mathbb{S}_{V_{\mathcal{X}}} \; \sigma \xrightarrow{\lambda} \sigma$$

can be removed without changing the state value $\mathbb{V}^*_{\hat{\mathcal{X}}}(\sigma)$.

As shown in theorem 2, when the dependency closure step is iterated starting from the set of variables used in the security policy, the application of the variables elimination technique to an MDP model produces a new reduced MDP model, whose optimal policy leads to the same total reward the agent would have on the original MDP. In this case we say that we are applying conservative variables elimination, since after the optimization we are still able to find optimal policies. Furthermore, this kind of optimization can be applied in an offline fashion, i.e. with no dependency on the actual system state.

The same technique can be applied starting from a different set of variables, trading the possibility to find optimal solutions for a more aggressive MDP state space reduction. This is useful especially during the runtime of the ASM, when a number of heuristics can be applied to the information produced by the sensors to build subsets of the system variables to be used as a starting set. As an example, the attack information provided by the IDS can be used to determine the subset of the variables impacted by the attack and this subset can be used as the initial set for applying variables elimination. Another heuristic can be realized by analyzing the system state changes in which the initial state satisfies the security policy, while the final state does not. Hereafter, we will refer to this technique as *divergence compensation*. Given such a state change, the only variables that changed their value can be used as the starting subset for variables elimination. The rationale behind this heuristic is to solve the reduced MDP model having only the actions that can act on these changed values. Whilst possibly not optimal, the resulting policy will compensate the change bringing back the system in a state satisfying the security policy.

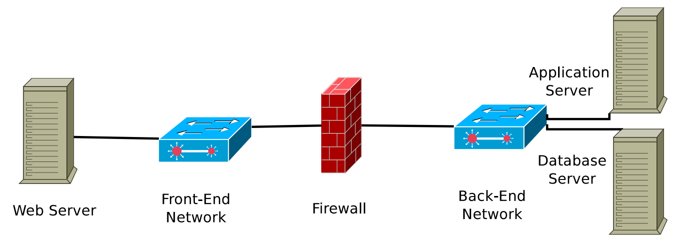


Fig. 5: Topology for the sample system model.

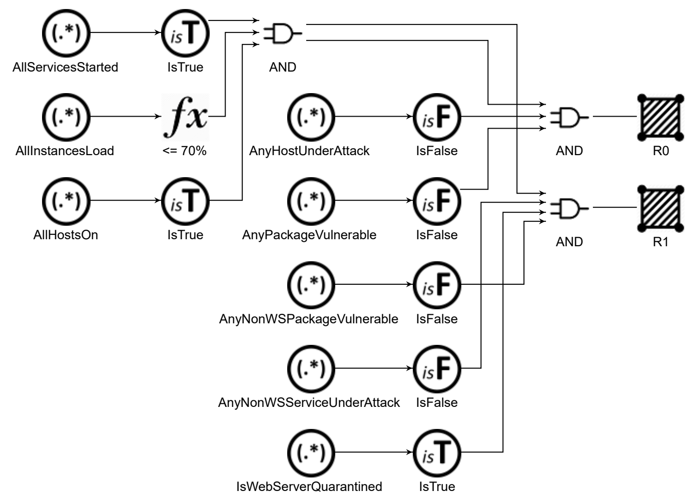


Fig. 6: Global security policy for the sample system model. In order to promote compactness and flexibility in the representation of security policies, we employ pattern meta-variables in formulas that are expanded to ground formulas in the model transformation stage as we do for action templates (see Section 3.2). For instance, the pattern meta-variable `AnyHostUnderAttack` in the formula $\neg$`AnyHostUnderAttack` is parametrized by the regular expression `Is(.+)UnderAttack`, thus it is expanded to $\neg$`IsAppServerUnderAttack` $\wedge$ $\neg$`IsWebServerUnderAttack` $\wedge$ $\neg$`IsDBServerUnderAttack`, i.e. the conjunction of the ground formulas bound to the state variables matching the pattern.

4 Case Study

In this section we present experimental results obtained by letting the Planner plan a defense strategy to protect the system represented by a sample system model.

After characterizing the sample system model adopted, a description of the methodology used to conduct the experiments follows. The section ends with a discussion on the results, where we evaluate the produced policies and the effectiveness of the state space reduction techniques applied.

4.1 Sample system model

The sample system model includes three hosts, a firewall and two networks, as depicted in Figure 5. The `AppServer` host runs the `Tomcat` service provided by the `Tomcat` package, while the `WebServer` host runs `Httpd` and `Vsftpd` services provided, respectively, by the packages `Apache` and `Vsftp`. Finally, the `DBServer` host runs the `Mysqld` service provided by the `Mysql` package.

In order simplify the system model design, our prototype supports parametric action templates for behavior specification. In Table 1 all the action templates defined in the sample system model are described, whilst Table 3 shows the actual (ground) actions resulting from the application of templates to components, as defined in the system model itself, along with the action-dependent reward function parameters. As an example, in Figure 7 the definition of the `Quarantine{H}` action template is shown, in which the pre-condition requires the `Is{H}Quarantined` variable to be $false$ for the action to be executed, while the post-conditions model the possible outcomes: the successful one sets the `Is{H}Quarantined` to $true$ with a probability of 0.9, whereas the other models a failure by leaving the state unchanged.

The global security policy for the sample system model is shown in Figure 6. Two safe regions have been defined, with different levels of confidence: `R0` requires all the components not to be under attack and all the installed packages not to be vulnerable, whilst `R1` relaxes these constraints on the `WebServer` host, provided that it gets quarantined. Both of the safe regions require all the hosts to be on with a maximum load per instance of 70% and all the services to be started.

4.2 Methodology

The Planner does not enumerate the entire state space, that could be possibly infinite, but explores only the states that are reachable from a given initial state. In order to generate an MDP with an increasing number of states given a single system model, we start from a safe state σ_0 to build a sequence of unsafe states $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where n is the number of system variables involved in the global security policy, s.t. $\sigma_i = \sigma_{i-1}[x_i \setminus v_i]$ for $1 \leq i \leq n$, with x_i being the i-th variable from an arbitrary total order and v_i being any value in the domain of x_i turning the global security policy to false. As shown in Table 2, we followed the lexicographic order over the names of the variables within the same host

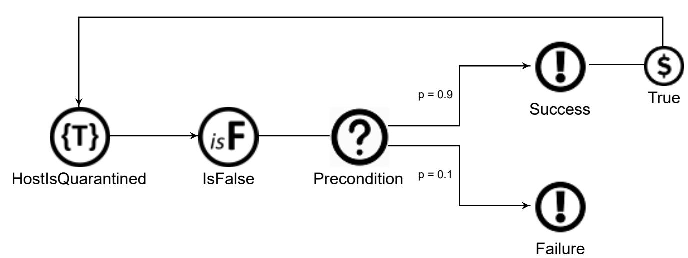


Fig. 7: Definition of the `Quarantine{H}` action template.

Name	Parameters	Variables	Semantics
Cut{H}Cord	H:Host	Is{H}On, Is{H}UnderAttack, Is{H}FSCorrupted	Powers off the host H, possibly causing filesystem corruption or teminating an ongoing attack.
Fsck{H}	H:Host	Is{H}FSCorrupted	Repairs filesystem(s) of the host H.
Quarantine{H}	H:Host	Is{H}Quarantined	Puts the host H in an insulated network environment.
ScaleUp{H}	H:Host	{H}Instances, {H}InstanceLoad	Increases the number of instances for the host H.
ScaleDown{H}	H:Host	{H}Instances, {H}InstanceLoad	Decreases the number of instances for the host H.
Startup{H}	H:Host	Is{H}On, Is{H}FSCorrupted	Starts up the operating system of H, if the filesystem is not corrupted.
Shutdown{H}	H:Host	Is{H}On, Is{H}UnderAttack	Shuts-down the operating system of H, thus possibly terminating the attack condition.
Start{S}	H:Host, S:Service	Is{S}Started, Is{H}On	Starts the service S, while the host H is on.
Stop{S}	H:Host, S:Service	Is{S}Started, Is{H}On, Is{H}{S}UnderAttack	Stops the service S, while the host H is on and S is running.
Update{P}	H:Host, P:Package, S:Service	Is{H}{P}Vulnerable, Is{H}{P}NewVersionAvailable, Is{S}Started, Is{H}On	While the host H is on, if the package P is vulnerable and a new version is available, then P is updated and the related service S restarted.

TABLE 1: System model action templates.

Attack scope	Host	Variable	Safety condition	Unsafe value
$i = 1$	AppServer	AppServerInstanceLoad	≤ 70	80
$i = 2$	AppServer	IsAppServerTomcatUnderAttack	false	true
$i = 3$	AppServer	IsAppServerTomcatVulnerable	false	true
$i = 4$	AppServer	IsAppServerUnderAttack	false	true
$i = 5$	WebServer	IsWebServerApacheVulnerable	false	true
$i = 6$	WebServer	IsWebServerHttpdUnderAttack	false	true
$i = 7$	WebServer	IsWebServerUnderAttack	false	true
$i = 8$	WebServer	IsWebServerVsftpVulnerable	false	true
$i = 9$	WebServer	IsWebServerVsftpdUnderAttack	false	true
$i = 10$	WebServer	WebServerInstanceLoad	≤ 70	80
$i = 11$	DBServer	DBServerInstanceLoad	≤ 70	80
$i = 12$	DBServer	IsDBServerMysqlVulnerable	false	true
$i = 13$	DBServer	IsDBServerMysqldUnderAttack	false	true
$i = 14$	DBServer	IsDBServerUnderAttack	false	true

TABLE 2: Incremental changes of the variables with the increasing of the attack scope.

while changing their values, starting the attack from the `AppServer` host and extending it to the `WebServer` host and finally to the `DBServerHost`.

Each unsafe state σ_i of the sequence, has i divergent variables, namely the variables whose values do not satisfy the safety conditions expressed in the global security policy. In the following, we refer to i as the *attack scope*, and use it as a simple measure of how extended is an attack: the higher i is, the more components of the system are involved.

Given such a sequence of unsafe states and the MDP factored model $\mathcal{M}$ derived from the sample system model, for each unsafe state σ_i in the sequence we generate a reduced factored model $\bar{\mathcal{M}}_i$ by applying variables elimination (see Section 3.3.1) with the set of variables that changed their value between the safe state σ_0 and σ_i as the set of variables to preserve. Before deriving the reduced models from $\mathcal{M}$, a variables elimination pass is performed to retain only the variables referred by the security policy. In the following, we refer to the reduced version of $\mathcal{M}$ as the full model.

In order to measure the effectiveness of the divergence compensation technique, for each unsafe state σ_i in the generated sequence, the MDP interpreted from both the original factored model $\mathcal{M}$ and the reduced model $\bar{\mathcal{M}}_i$ has been solved with the Value Iteration algorithm to find the optimal policies π_i^* and $\bar{\pi}_i^*$, respectively, using a discount factor $\gamma = 0.99$ and even reward function weights $w_t = w_c = w_{conf} = w_i = w_a = 0.2$.

Finally, the average cumulative reward accumulated by

AppServer Action a	$T(a)$	$C(a)$	$Conf(a)$	$I(a)$	$A(a)$
CutAppServerCord	1	0	0	0	1
FsckAppServer	1800	0	0	0	0
ScaleUpAppServer	10	10	0	0	0
ScaleDownAppServer	10	0	0	0	0
StartupAppServer	60	0	0	0	0
ShutdownAppServer	60	0	0	0	1
StartTomcat	5	0	0	0	0
StopTomcat	5	0	0	0	1
UpdateTomcat	1800	0	0	0	0
DBServer Action a	$T(a)$	$C(a)$	$Conf(a)$	$I(a)$	$A(a)$
CutDBServerCord	1	0	0	0	1
FsckDBServer	1800	0	0	0	0
ScaleUpDBServer	10	10	0	0	0
ScaleDownDBServer	10	0	0	0	0
StartupDBServer	60	0	0	0	0
ShutdownDBServer	60	0	0	0	1
StartMysqld	5	0	0	0	0
StopMysqld	5	0	0	0	1
UpdateMysql	1800	0	0	0	0
WebServer Action a	$T(a)$	$C(a)$	$Conf(a)$	$I(a)$	$A(a)$
CutWebServerCord	1	0	0	0	1
QuarantineWebServer	5	0	0	0	0.8
FsckWebServer	1800	0	0	0	0
ScaleUpWebServer	10	10	0	0	0
ScaleDownWebServer	10	0	0	0	0
StartupWebServer	60	0	0	0	0
ShutdownWebServer	60	0	0	0	1
StartHttpd	5	0	0	0	0
StopHttpd	5	0	0	0	1
UpdateApache	1800	0	0	0	0
StartVsftpd	5	0	0	0	0
StopVsftpd	5	0	0	0	1
UpdateVsftp	1800	0	0	0	0

TABLE 3: Reward function parameters for all the ground actions defined in the sample system model.

the agent over 10,000 policy roll-outs was used to compare the policies π_i^* and $\bar{\pi}_i^*$, for each i.

4.3 Evaluation

For each unsafe state σ_i in the sequence, two kinds of data have been collected while calculating the optimal policies π_i^* and $\bar{\pi}_i^*$: (i) the size of the state space that the MDP explored starting from σ_i and (ii) the total cumulative reward the agent accumulated.

In the following, we use (i) to evaluate the effectiveness of the state space reduction provided, while (ii) is used to show the potential sub-optimality of the policies computed on the reduced models.

Figure 8a shows that, although the size of the explored state space still grows exponentially with the attack scope, the reduced model MDPs have a state space that can be several orders of magnitude smaller in size than the one generated by the full model. Indeed, by leveraging the information about the state change, we can obtain a more tractable problem, whose size depends on the extension of the attack.

As shown in theorem 2 of Section 3.3.2, if the reduced model contains all the variables referenced by the security policy and their dependencies, the optimal policy for the reduced model is still optimal for the full model. The reduced models generated by the divergence compensation technique contain a smaller set of variables, so the resulting optimal policies are not guaranteed to be optimal for the full model.

Starting with $i = 5$, the attack scope extends beyond the `AppServer` component and the Figure 8b shows that $\bar{\pi}_i^*$ starts to be sub-optimal w.r.t. π_i^*, i.e. the policy computed for the full model. This situation arises since, in our sample model, the only component for which there exists an instance of the `Quarantine`$\{$`H`$\}$ action template is the `WebServer` host and the relaxed safe region `R1` exists admitting safety whenever that component is quarantined, regardless the value of the vulnerability related variables `IsWebServerApacheVulnerable` and `IsWebServerVsftpVulnerable`, or the attack related variables `IsWebServerHttpdUnderAttack` and `IsWebServerUnderAttack`.

From Table 2 we have that, for $i = 5$, the only changed variable that is added to the set to be preserved is `IsWebServerApacheVulnerable`, but since the `QuarantineWebServer` action does not reference that variable, variables elimination does not preserve that action in the reduced model. As a consequence, the MDP generated from the full model knows about the `QuarantineWebServer` action and chooses it to counter the attack to the `WebServer` component, whilst the reduced model cannot and is forced to counter the attack in a less rewarding manner. A similar argument explains the sub-optimality for all $i \geq 5$.

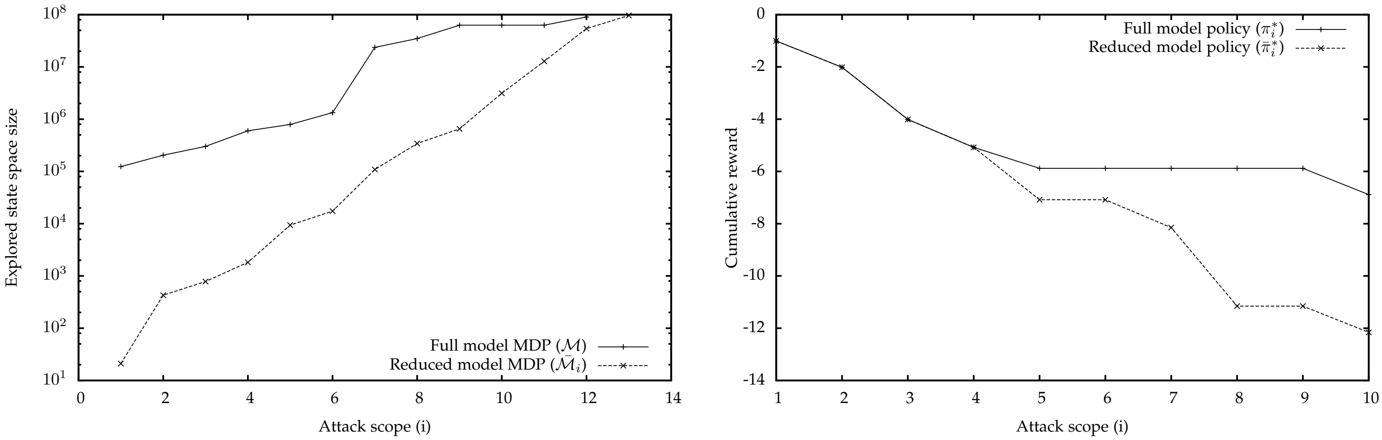
It is worth noting that, while the `QuarantineWebServer` action can solve the issues on the `WebServer` component for $5 \leq i \leq 9$, it is not able to counter the violation of the safety condition for the variable `WebServerInstanceLoad` that occurs with $i = 10$. As a consequence, the reward of the π_i^* policy does not change for $5 \leq i \leq 9$, and continues to decay for $i \geq 10$. The quicker decay of the reward of policies $\bar{\pi}_i^*$ is due to the fact that in the reduced model more actions are to be executed to bring the system in the safe region.

In conclusion, we found that the proposed approach provides multiple benefits: (i) given the same computation effort, it allows to deal with larger systems, possibly with optimal results, (ii) the computation effort to counter an attack does not depend on the size of the system, but depends on its complexity and on the attack scope.

5 Threats to Validity

Modeling enterprise systems is well documented. Hamilton, Nash and Pooch noted [22]: "Broadly speaking, there are two classes of strategies which may be used to validate a model": (i) axiomatic [48] and (ii) empirical. The methodology proposed in this paper is well suited to validation by both axiomatic and empirical methods: the former can be used when operations such as model minimization have to be performed, as shown in Section 3; the latter are instead particularly useful to measure the divergence of the model-based predictions with real observations on the system.

Indeed, the primary threat to validation is an inaccurate model of the target enterprise system. A model is valid if it can produce the outputs that are equivalent to the



(a) State space size reduction effectiveness.

(b) The agent return of the optimal policies for full and reduced models. The $\bar{\pi}_i^*$ policy shows its sub-optimality starting with an attack scope $i = 5$.

Fig. 8: Evaluation of the divergence compensation technique.

ones that would be observed given the same inputs in the environment being modeled [22]. In other words, the model is capable of predicting the behavior of the system being modeled within a specified tolerance. This threat is particularly evident in non-stationary systems like enterprise systems, where their behavior could change over time, for instance, due to the addition or removal of components, due to changes on the software base, and due to changes in the behavior of the users. Non-stationarity could be addressed in several ways. One of the possibilities is to record historical data regarding the behavior of the system and periodically update the parameters of the model to more accurately reflect the current or the predicted behavior, for instance by using filters like Exponential Weighted Moving Average (EWMA [24]) or Kalman's [7], but this would require the re-execution of a computationally expensive planning every time a change is detected. A more interesting alternative is to use Reinforcement Learning techniques that can leverage the MDP-based structure of the problem, such as, Q-Learning, SARSA [57] and Artificial Neural Networks [56], that allow the planner to automatically evolve as the system does, without the need for a computationally expensive planning when a change to the system occurs.

Another important threat to validity is given by the scope of the model. The one that has been presented in this paper only considers the defense side of the problem, and assumes that the attacker does not make any move during the execution of the defense plan. In the real world, of course, this assumption is not realistic. This problem can be either mitigated by considering the existence of external controls, and by re-planning the defense strategy every time an external control occurs, or embraced by extending the model so to include the behavior of the attacker. The same structure of the MDP-based formulation can then be used to realize a multi-agent stochastic game, which potentially allows the agents to execute proactive actions.

However, extending the model and defining the problem as a multi-agent stochastic game, also has the potential to increase the planning time, thus reducing the effectiveness of the proposed methodology. For this reason, optimal and/or sub-optimal model reduction techniques, like the one presented in this paper, must be considered in order to mitigate this issue.

Other threats to validity can come from the completeness of the model: undocumented, undiscovered and unknown systems connected to target system are not going to be in the system model. If the static model is incomplete, dynamic modeling based on that static model will have some errors and omissions.

6 Related Works

Autonomic computing comprises four main dimensions as described in [33], namely, self-configuration, self-optimization, self-healing and self-protection. Most of the proposed autonomic frameworks are focused on performance and reliability management to maximize Quality of Service (QoS) under uncertain operating conditions (e.g., [10], [36], [37], [39], [47]), and only few address the self-protection aspect. In the following, we present a summary of the current state of the art research on autonomic frameworks supporting self-protection and related approaches to software development using Model Integrated Computing [15].

In [12] the authors developed an autonomic security management for healthcare information systems, that includes vulnerability assessment, intrusion estimation, intrusion detection and intrusion response. The anomaly-based intrusion detection system uses system features such as CPU load, memory utilization, and network and disk throughputs to verify whether or not the system is in the safe region. Should the system not reside in the safe region, the intrusion response component selects a proper countermeasure according to a list of actions executable on the system, ordered according to the effectiveness in countering the attack.

The authors of [13] propose an autonomic framework named *SHAPE* for self-healing and self-protecting enterprise systems. It uses the same features proposed by [12] to identify potential software anomalies on the hosts, whereas Snort is used for network intrusion detection. The framework provides a wide range of monitoring sensors (e.g., hardware monitoring agent, software agents, memory agents, network agents) for handling different aspects of fault management and self-protection. The output of the monitoring modules is correlated and signatures are generated when a deviation from the secure region is detected. Currently, SHAPE only provides a limited set of countermeasures (i.e., hardware/software restart and job resubmission) and a static mapping approach is used to select the best possible intrusion response action.

In [49], the authors describe an approach to architecture-based self-protection. The main features of the proposed framework, named *Rainbow*, reside in the separation between application logic and control layer and the usage of system models for reasoning and deciding the countermeasure to deploy. Rainbow is designed according to the MAPE loop for autonomic systems and, in the same way as [13], it can be used for self-healing in addition to self-protection by implementing the MAPE phases with the proper tools. In this work, the authors show how Rainbow is able to defend the system from a Denial of Service (DoS) attack.

In [45] the authors introduce a framework based on MAPE for the self-protection of computer networks. The monitor phase works with network traffic as well as with the same set of system features used by [12], [13] for anomalous behavior analysis. Filtered data is then continuously streamed to the anomaly analysis module, that uses sliding windows of different sizes to detect attacks with different time granularity. The planning phase relies on boolean expressions that define acceptable operations and are associated with actions that describe how the boolean condition would change. Since the main focus of this work is network protection, routers are the main components subject to control and target of the execute phase.

The authors of [3] describe how the human, that is usually considered to be in-the-loop, should instead be brought on-the-loop, therefore having only the responsibility to oversee the automated process and eventually validate the results of the automated analysis. The work highlights how modeling the attacks with attack graphs (e.g., [31]) without modeling the system behavior introduces some limitations to the predictive behavior of the protection tool, because the attack graphs are not able to capture the likelihood of each attack pattern and/or the real impact on the enterprise system. To this end, they propose a framework based on a system model that also includes the effects of attacks on the present vulnerabilities, based on a service dependency analysis. Once an attack pattern has been detected, the framework is able to automatically generate a ranked list of prospective actions to execute to minimize its impact.

7 Conclusions and Future Works

Although often with a limited rate of false positives, the rate of the alerts generated by intrusion detection tools is too high to be manually handled by a human operator. Intrusion response systems replace the human operator with an algorithm in charge of automatically finding a (possibly) optimal countermeasure to the detected threat. Most of the works proposed so far treat separately the intrusion detection and the intrusion response steps and, to the best of our knowledge, none of the existing research proposes a comprehensive model-based framework that integrates system monitoring, intrusion detection, intrusion response selection, intrusion response execution and realize a working prototype.

In this paper, we presented the design and implementation of the Plan phase of an Autonomic Security Management system architected according to the Monitor, Analyze, Plan, Execute loop for autonomic systems. In order to make the solution suitable to be used outside an academic environment, we employed Model Integrated Computing for the automatic generation of MDP from a system model, and we proposed a novel technique that shifts the curse of dimensionality from the size of the system to the scope of the attack. Experimental results show that it is practically possible to obtain a reduction of several orders of magnitude of the state space of the MDP, while maintaining optimal or near-optimal solutions, according to the initial set of attributes chosen for the execution of the heuristic.

The MAPE approach proposed in this work is in theory sufficient to protect a system that behaves exactly in the way it is described into the model. That is, everything works perfectly with the assumption of a perfect model. However, the effects of the defense actions executed on the system might change over time, leading thus to a non-stationary process. This is due to different reasons, among which, shifts in the system configurations, updates to the software base, changes of the users behavior. One possible way to address this issue would be to monitor the execution of the actions and to update the parameters of the model, for instance by using filters like EWMA or Kalman's, but this would require the re-execution of a computationally expensive planning every time a change is detected. For these reasons, as a future work, we will investigate the realization of a self-adaptive controller, by changing the reinforcement learning paradigm from model-based to model-free. Specifically, we will perform research on learning agents based on widely used algorithms and technologies, such as, Q-Learning, SARSA, Expected SARSA, QVLearning, double Q-Learning [57] and Artificial Neural Networks [56], to let the controller automatically evolve as the environment does, without the need for a computationally expensive planning when a change to the system occurs.

Finally, we are planning to conduct a Cognitive Task Analysis [41] to compare the cognitive load required from the end-user in presence of the ASM and without it. This experiment will be particularly useful to identify the stages of the planning that require the most cognitive activity from the user, thus giving hints on what aspects of the ASM should be enhanced.

Acknowledgment

Funding for this work was partially provided by the U.S Army Engineer Research and Development Center (ERDC), under Contract W912HZ-17-C-009.

References

[1] Brown-umbc reinforcement learning and planning (burlap). http://burlap.cs.brown.edu/.

[2] 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.

[3] M. Albanese, H. Cam, and S. Jajodia. Automated cyber situation awareness tools and models for improving analyst performance. In *Cybersecurity Systems for Human Cognition Augmentation*, pages 47–60. Springer, 2014.

[4] H. ArcSight. Security intelligence for a faster world, 2012.

[5] R. Bellman. Dynamic programming. *BellmanDynamic Programming1957*, 1957.

[6] J. Bézivin, C. Brunette, R. Chevrel, F. Jouault, and I. Kurtev. Bridging the generic modeling environment (gme) and the eclipse modeling framework (emf). In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA*, volume 5, 2005.

[7] R. G. Brown, P. Y. Hwang, et al. *Introduction to random signals and applied Kalman filtering*, volume 3. Wiley New York, 1992.

[8] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008.

[9] D. Carasso. Exploring splunk. *published by CITO Research, New York, USA, ISBN*, pages 978–0, 2012.

[10] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola. Moses: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering*, 38(5):1138–1159, 2012.

[11] 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.

[12] Q. Chen, J. Lambright, and S. Abdelwahed. Towards autonomic security management of healthcare information systems. In *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on*, pages 113–118. IEEE, 2016.

[13] I. Chopra and M. Singh. Shape—an approach for self-healing and self-protection in complex distributed networks. *The Journal of Supercomputing*, 67(2):585–613, 2014.

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

[15] J. Davis. Gme: the generic modeling environment. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 82–83. ACM, 2003.

[16] M. J. Druzdzel and R. R. Flynn. Encyclopedia of library and information science, chapter decision support systems, 2003.

[17] J. M. Ewing. *Autonomic Performance Optimization with Application to Self-Architecting Software Systems*. PhD thesis, George Mason University, 2015.

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

[19] 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.

[20] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

[21] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

[22] J. A. Hamilton, D. A. Nash, and U. W. Pooch. *Distributed simulation*, volume 8. CRC Press, 1997.

[23] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, 2006.

[24] J. S. Hunter. The exponentially weighted moving average. *Journal of quality technology*, 18(4):203–210, 1986.

[25] S. Iannucci and S. Abdelwahed. A probabilistic approach to autonomic security management. In *Proceedings of the 13th IEEE International Conference on Autonomic Computing (ICAC)*, 2016.

[26] S. Iannucci and S. Abdelwahed. Model-based response planning strategies for autonomic intrusion protection. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 13(1):4, 2018.

[27] S. Iannucci, Q. Chen, and S. Abdelwahed. High-performance intrusion response planning on many-core architectures. In *Workshop on Network Security Analytics and Automation (NSAA)*, 2016.

[28] IBM. *Admin Guide: IBM Security QRadar SIEM Version 7.2*. 2013.

[29] Z. Inayat, A. Gani, N. B. Anuar, M. K. Khan, and S. Anwar. Intrusion response systems: Foundations, design, and challenges. *Journal of Network and Computer Applications*, 62:53–74, 2016.

[30] M. Irfan, H. Abbas, Y. Sun, A. Sajid, and M. Pasha. A framework for cloud forensics evidence collection and analysis using security information and event management. *Security and Communication Networks*, 9(16):3790–3807, 2016.

[31] S. Jajodia, S. Noel, P. Kalapa, M. Albanese, and J. Williams. Cauldron mission-centric cyber situational awareness with defense in depth. In *2011-MILCOM 2011 Military Communications Conference*, pages 1339–1344. IEEE, 2011.

[32] G. Karsai. Generic modeling environemnt: Building tools that build tools, 2013. https://engineering.vanderbilt.edu/innovations-2013/building-tools.php.

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

[34] N. Kheir, H. Debar, N. Cuppens-Boulahia, F. Cuppens, and J. Viinikka. Cost evaluation for intrusion response using dependency graphs. In *Network and Service Security, 2009. N2S'09. International Conference on*, pages 1–6. IEEE, 2009.

[35] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary*, volume 17, page 1, 2001.

[36] D. Menasce, H. Gomaa, J. Sousa, et al. Sassy: A framework for self-architecting service-oriented systems. *Ieee Software*, 28(6):78–85, 2011.

[37] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. End-to-end support for qos-aware service selection, binding, and mediation in vresco. *IEEE Transactions on Services Computing*, 3(3):193–205, 2010.

[38] Z. Molnár, D. Balasubramanian, and Á. Lédeczi. An introduction to the generic modeling environment. In *Proceedings of the TOOLS Europe 2007 Workshop on Model-Driven Development Tool Implementers Forum*, 2007.

[39] O. Moser, F. Rosenberg, and S. Dustdar. Domain-specific service selection for composite services. *IEEE Transactions on Software Engineering*, 38(4):828–843, 2012.

[40] 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.

[41] M. A. Neerincx. Cognitive task load analysis: allocating tasks and designing support. *Handbook of cognitive task design*, 2003:283–305, 2003.

[42] 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.

[43] Pivotal. Spring expression language (spel). https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#expressions, 2018.

[44] S. Pradhan, A. Dubey, T. Levendovszky, P. S. Kumar, W. A. Emfinger, D. Balasubramanian, W. Otte, and G. Karsai. Achieving resilience in distributed software systems via self-reconfiguration. *Journal of Systems and Software*, 122:344–363, 2016.

[45] G. Qu, O. A. Rawashdeh, and D. Che. Self-protection against attacks in an autonomic computing environment. *IJ Comput. Appl.*, 17(4):250–256, 2010.

[46] G. Rasche, E. Allwein, M. Moore, and B. Abbott. Model-based cyber security. In *Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the*, pages 405–412. IEEE, 2007.

[47] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software engineering for self-adaptive systems*, pages 164–182. Springer, 2009.

[48] R. Sargent. An overview of verification and validation of simulation. In *Proceedings of Winter Simulation Conference*, 1987.

[49] B. Schmerl, J. Cámara, J. Gennari, D. Garlan, P. Casanova, G. A. Moreno, T. J. Glazier, and J. M. Barnes. Architecture-based self-protection: composing and reasoning about denial-of-service mit-

igations. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, page 2. ACM, 2014.

[50] A. Shameli-Sendi, M. Cheriet, and A. Hamou-Lhadj. Taxonomy of intrusion risk assessment and response system. *Computers & Security*, 45:1–16, 2014.

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

[52] A. Shameli-Sendi, N. Ezzati-Jivan, M. Jabbarifar, and M. Dagenais. Intrusion response systems: survey and taxonomy. *Int. J. Comput. Sci. Netw. Secur*, 12(1):1–14, 2012.

[53] N. Stakhanova, S. Basu, and J. Wong. A cost-sensitive model for preemptive intrusion response systems. In *AINA*, volume 7, pages 428–435, 2007.

[54] N. Stakhanova, S. Basu, and J. Wong. A taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1(1-2):169–184, 2007.

[55] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[56] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[57] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 2018.

[58] S. Tanachaiwiwat, K. Hwang, and Y. Chen. Adaptive intrusion response to minimize risk over multiple network attacks. *ACM Trans on Information and System Security*, 19:1–30, 2002.

[59] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. 1955.

[60] 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.

[61] W. Yan, Y. Xue, X. Li, J. Weng, T. Busch, and J. Sztipanovits. Integrated simulation and emulation platform for cyber-physical system security experimentation. In *Proceedings of the 1st international conference on High Confidence Networked Systems*, pages 81–88. ACM, 2012.

[62] X. Zan, F. Gao, J. Han, X. Liu, and J. Zhou. A hierarchical and factored pomdp based automated intrusion response framework. In *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, volume 2, pages V2–410. IEEE, 2010.

[63] S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley. Rre: A game-theoretic intrusion response and recovery engine. *Parallel and Distributed Systems, IEEE Transactions on*, 25(2):395–406, 2014.

Stefano Iannucci is an Assistant Professor of Computer Science and Engineering at Mississippi State University and an affiliated member of the Center for Cyber Innovation (CCI) department of the High Performance Computing Collaboratory at Mississippi State University. He received his Ph.D. in 2015 from the University of Rome Tor Vergata. His research focuses on cyber-security automation, autonomic computing, Internet of Things and performance modeling and benchmarking. He published over 20 papers on top journals and conferences. Dr. Iannucci has chaired several international workshops and has been the workshops chair for IEEE ICCAC, one of the leading conferences in autonomic computing. He is member of IEEE.

Sherif Abdelwahed is a Professor of Electrical and Computer Engineering (ECE) at Virginia Commonwealth University (VCU). Before joining VCU in August 2017, he served as the associate director of the Distributed Analytics and Security Institute at Mississippi State University (MSU). He was also is also an Associate Professor in the ECE Department at MSU. He received his Ph.D in 2002 from the Department of Electrical and Computer Engineering at the University of Toronto. Prior to joining Mississippi State University, he was a research assistant professor at the Department of Electrical Engineering and Computer Science and senior research scientist at the Institute for Software Integrated Systems, Vanderbilt University, from 2001-2007. From 2000-2001 he worked as a research scientist with the system diagnosis group at the Rockwell Scientific Company. Dr. Abdelwahed has chaired several international conferences and conference tracks, and has served as technical committee member at various national and international conferences. He received the StatePride Faculty award for 2010 and 2011, the Bagley College of Engineering Hearin Faculty Excellence award in 2010, and recently the 2016 Faculty Research Award from the Bagley College of Engineering at MSU. He has more than 140 publications and is a senior member of the IEEE.

Andrea Montemaggio is a Research Engineer at Center for Cyber Innovation (CCI), Mississippi State University (MSU). In 2017 he earned his B.E. degree in Computer Engineering, with honors, from the University of Rome Tor Vergata. He has been working as a Software Engineer since 2003, both for private companies and as a free-lance, gaining experience as a team leader, project manager and software architect mainly in the retail and financial industries. In 2009, he has been an Associate Editor of the proceedings of the Open Source in the Public Administration (OSPA) workshop sponsored by the LUISS Guido Carli University in Rome.

Melissa Hannis is a Research Engineer at Mississippi State University (MSU) in the Center for Cyber Innovation (CCI) department at The High Performance Computing Collaboratory (HPC2). She graduated from MSU with a BS in Computer Science in 2015, and is currently working on finishing her MS in Computer Science. Much of her interest in research was acquired during her time as a student worker at The Center for Advanced Vehicular Systems (CAVS), where she worked on a number of research projects. She started working at CAVS in 2014 as an undergraduate student researcher and later as a Graduate Research Assistant. This interest in research is what encouraged her to pursue a job as a full-time Research Engineer at CCI.

Leslie Leonard is a Computer Scientist at the U.S Army Engineer Research and Development Center (ERDC). She serves as the Cybersecurity Research lead for the High Performance Computer Modernization Program's (HPCMP) Security team. She received her Ph.D. in 2015 from the University of Maryland. Dr. Leonard leads Research and Development (R&D) for new technologies, tools, and techniques that enable the HPCMP to defend, mitigate, and secure five Defense Supercomputing Resource Centers (DSRCs) and the Defense Research and Engineering Network (DREN). In 2013, 2015, and 2016, she was recognized by the ERDC director, ERDC commander, and U.S. Army Corps of Engineers (USACE) Command Sergeant Major for her professional excellence. In 2017, she received the USACE Cybersecurity Professional of the Year award. Dr. Leonard and her cybersecurity research team also received the ERDC Research and Development Achievement Award for Technical Collaboration and Technical Achievement. She is a published author and served as a member of various technical committees.

Jason S. King joined the High Performance Modernization Program (HPCMP) at the Engineer Research and Development Center (ERDC) shortly after graduating with a Bachelor's degree in Computer Science from Mississippi State University in May, 2016. Mr. King began working with a team developing a state of the art intrusion detection system built around a Bro sensor network. The Cybersecurity Environment for Detection, Analysis and Response (CEDAR) was designed to provide analysts with a more accurate, integrated framework of processing tools to reduce the time of cyber-incident response. More recently, Mr. King has been integral in the establishment of the HPCMP as a Cybersecurity Service Provider (CSSP) utilizing the CEDAR IDS, acting as both Detect and Respond and Cyber Threat Intelligence Team Leads. Mr. King is part of a team of researchers exploring the use of HPC systems to develop new algorithms for use in cybersecurity incident detections. Current research includes exploring data science techniques to identify malicious content using ssl/tls handshakes.

John A. Hamilton, Jr. is a Professor of Computer Science & Engineering at Mississippi State University where he directs two research centers: the Distributed Analytics and Security Institute and the Center for Cyber Innovation. Previous faculty appointments were at Auburn University, the US Military Academy and the US Naval Postgraduate School. Dr. Hamilton earned his doctorate in computer science from Texas A&M University and is a distinguished graduate of the US Naval War College.

APPENDIX

PROOF OF LEMMA 1

Proof: By contradiction. We show that a contradiction can be derived from both the following cases.

$$\exists \sigma_0, \sigma_1 \in \mathbb{S}_V \ \sigma_0 \overset{\lambda}{\nrightarrow} \sigma_1 \wedge \sigma_0 \restriction_{V'} \xrightarrow{\lambda'} \sigma_1 \restriction_{V'} \tag{i}$$

$$\exists \sigma_0, \sigma_1 \in \mathbb{S}_V \ \sigma_0 \xrightarrow{\lambda} \sigma_1 \wedge \sigma_0 \restriction_{V'} \overset{\lambda'}{\nrightarrow} \sigma_1 \restriction_{V'} \tag{ii}$$

For the case (i), we have that there must exist a state $\sigma_2 \neq \sigma_1$ s.t. $\sigma_0 \xrightarrow{\lambda} \sigma_2$. By applying the definition of the transition relation and the definition of λ' we have

$$\begin{gathered}\forall v \in V \ \lambda(v)(\sigma_0) = \sigma_2(v) \\ \wedge \\ \forall v' \in V' \bigcap_{\substack{\tau \in \mathbb{S}_V \\ \tau \restriction_{V'} = \sigma}} \{\lambda(v')(\tau)\} = \{\sigma_1 \restriction_{V'} (v')\}\end{gathered}$$

By resolving the intersection and since $V' \subseteq V$ this can be rewritten as $\forall v \in V \ \lambda(v)(\sigma_0) = \sigma_2(v) \wedge \forall v' \in V', \tau \in \mathbb{S}_V \ \lambda(v')(\tau) = \sigma_1(v')$, from which if we can derive the following contradiction

$$\forall v \in V \ \lambda(v)(\sigma_0) = \sigma_2(v) \wedge \forall v' \in V' \ \lambda(v')(\sigma_0) = \sigma_1(v')$$

With a similar argument it can be shown that the case (ii) also leads to a contradiction. □

PROOF OF THEOREM 1

Proof: By the definition of the q-value function (3), we have that its value depends upon the reward function and the transition probability function only. The construction does not change the reward function, so we have to show that the following holds

$$\begin{gathered}\forall x \in A, \forall \sigma_0, \sigma_1 \in \mathbb{S}_V \\ P_{\hat{\mathcal{X}}}(\sigma_0, x, \sigma_1) = P_{\hat{\mathcal{X}}'}(\sigma_0 \restriction_{V'}, x, \sigma_1 \restriction_{V'})\end{gathered} \tag{8}$$

For every action $x \in A$ and post-condition $\langle p, \lambda \rangle \in \Lambda_{\mathcal{X}}(x)$ of x, the construction of $\Lambda_{\mathcal{X}'}$ does not alter the probability distribution, hence the equation (8) does not hold iff for some $x \in A$ and $\sigma_0, \sigma_1 \in \mathbb{S}_V$ (i) the LHS is 0 while the RHS is p, or (ii) the LHS is p while the RHS is 0.

For the case (i), by applying (2) we have that

$$\begin{gathered}\exists x \in A, \exists \sigma_0, \sigma_1 \in \mathbb{S}_V \\ (\forall \langle p, \lambda \rangle \in \Lambda_{\mathcal{X}}(x) \quad \sigma_0 \overset{\lambda}{\nrightarrow} \sigma_1) \\ \wedge \\ (\exists \langle p, \lambda' \rangle \in \Lambda_{\mathcal{X}'}(x) \quad \sigma_0 \restriction_{V'} \xrightarrow{\lambda'} \sigma_1 \restriction_{V'})\end{gathered}$$

which contradicts lemma 1.

With a similar argument it can be shown that even the case (ii) leads to a contradiction. □

PROOF OF THEOREM 2

Proof: By reductio ad absurdum, we show that a contradiction follows from

$$\exists \sigma \in \mathbb{S}_V \ \mathbb{V}^*_{\hat{\mathcal{X}}}(\sigma) \neq \mathbb{V}^*_{\hat{\mathcal{X}}'}(\sigma \restriction_{V'}) \tag{9}$$

From (9) and by definition of the value function we obtain

$$\exists \sigma \in \mathbb{S}_V \ \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, x) \neq \mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma \restriction_{V'}, y) \tag{10}$$

where $x = \pi^*_{\hat{\mathcal{X}}}(\sigma)$ and $y = \pi^*_{\hat{\mathcal{X}}'}(\sigma \restriction_{V'})$ are the actions selected by the optimal policies for the MDPs $\hat{\mathcal{X}}$ and $\hat{\mathcal{X}}'$, respectively.

For (10), either (i) $x = y$ or (ii) $x \neq y$ must hold. If (i) holds, this would be in contrast with theorem 1. Otherwise, for the case (ii) we have that x is chosen over y by the optimal policy $\pi^*_{\hat{\mathcal{X}}}$, whereas y is chosen over x by the optimal policy $\pi^*_{\hat{\mathcal{X}}'}$ of the reduced MDP.

Hence, from the definition of optimal policy (5), there must exist a state $\sigma \in \mathbb{S}_V$ s.t. $\mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, x) > \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, y) \wedge \mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma \restriction_{V'}, x) < \mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma \restriction_{V'}, y)$ holds, from which we derive a contradiction due to theorem 1. □

PROOF OF LEMMA 2

Proof: Let $\mathcal{X}$ be an MDP factored model and $\mathcal{X}'$ the factored model derived from $\mathcal{X}$ by removing a post-condition $\langle p, \lambda \rangle \in \Lambda_{\mathcal{X}}(a)$ s.t. $\forall \sigma \in \mathbb{S}_V = \mathbb{S}_{V_{\mathcal{X}}} = \mathbb{S}_{V_{\mathcal{X}'}}$, $\sigma \xrightarrow{\lambda} \sigma$, for some action $a \in A = A_{\mathcal{X}} = A_{\mathcal{X}'}$, and let $\bar{R}$ be the reward function of both $\mathcal{X}$ and $\mathcal{X}'$ s.t. $\forall x \in A \ \bar{R}(x) < 0$, strictly negative by hypothesis (h0).

In order to show that the removal of such a post-condition does not change the state-value it is enough to show that the following holds

$$\begin{gathered}\forall x \in A, \sigma \in \mathbb{S}_V \\ \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, x) > \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, a) \implies \mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma, x) > \mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma, a)\end{gathered} \tag{11}$$

Hence, by reductio ad absurdum we show that a contradiction can be derived if both the following relations hold.

$$\mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, x) - \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, a) > 0 \tag{i}$$

$$\mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma, x) - \mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma, a) \leq 0 \tag{ii}$$

By the definition of the q-value function (3), in (ii) we rewrite $\mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma, a)$ in terms of $\mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, a)$, by subtracting the term related to the removed post-condition $\langle p, \lambda \rangle$. Furthermore, the derivation of $\mathcal{X}'$ did not change the action x, thus we can write $\mathbb{Q}^*_{\hat{\mathcal{X}}'}(\sigma, x) = \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, x)$ to derive

$$\mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, x) - \mathbb{Q}^*_{\hat{\mathcal{X}}}(\sigma, a) \leq -\gamma p \mathbb{V}^*_{\hat{\mathcal{X}}'}(\sigma) \tag{12}$$

For both (i) and (12) to hold, $-\gamma p \mathbb{V}^*_{\hat{\mathcal{X}}'}(\sigma) \leq 0$ must hold. Since the sign of the state-value function $\mathbb{V}^*_{\hat{\mathcal{X}}'}$ depends only upon the sign of the reward function $\bar{R}$, it should be that $\forall x \in A \ \bar{R}(x) \geq 0$, which contradicts the hypothesis (h0) about the strict negativity of $\bar{R}$. □