A Supervisory Control Approach to Dynamic Cyber-Security

Mohammad Rasouli, Erik Miehling, and Demosthenis Teneketzis

Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI, USA

Abstract. An analytical approach for a dynamic cyber-security problem that captures progressive attacks to a computer network is presented. We formulate the dynamic security problem from the defender's point of view as a supervisory control problem with imperfect information, modeling the computer network's operation by a discrete event system. We consider a min-max performance criterion and use dynamic programming to determine, within a restricted set of policies, an optimal policy for the defender. We study and interpret the behavior of this optimal policy as we vary certain parameters of the supervisory control problem.

Keywords: Cyber-Security, Computer Networks, Discrete Event Systems, Finite State Automata, Dynamic Programming.

1 Introduction

Cyber-security has attracted much attention recently due to its increasing importance in the safety of many modern technological systems. These systems are ubiquitous in our modern day life, ranging from computer networks, the internet, mobile networks, the power grid, and even implantable medical devices. This ubiquity highlights the essential need for a large research effort in order to strengthen the resiliency of these systems against attacks, intentional and unintentional misuse, and inadvertent failures.

The study of cyber-security problems in the existing literature can be divided into two main categories: *static* and *dynamic*.

Static problems concern settings where the agents, commonly considered to be an attacker and a defender, receive no new information during the time horizon in which decisions are made. Problems of this type in the security literature can largely be classified under the category of *resource allocation*, where both the defender and attacker make a single decision as to where to allocate their respective resources. The main bodies of work involve infrastructure protection [3, 7, 9] and mitigation of malware and virus spread in a network [5, 6, 8, 16]. Some of the above works consider settings where the agents are strategic [3, 9]. The presence of strategic agents results in a game between the attacker and defender. The strategic approaches in the above works are commonly referred to as *allocation games*. The survey by Roy et al. [18], as well as [20], provide useful outlines of some static game models in security.

Dynamic security problems are those that evolve over time, with the defender taking actions while observing some new information from the environment.[1] The formulation of a security problem as a dynamic problem, instead of a static one, offers numerous advantages. The first advantage is clear; since real-world security problems have an inherently dynamic aspect, dynamic models can more easily capture realistic security settings, compared to static models. Also, most attacks in cyber-security settings are *progressive*, meaning more recent attacks build upon previous attacks (such as denial-of-service attacks, brute-force attacks, and the replication of viruses, malware, and worms, to name a few). This progressive nature is more easily modeled in a dynamic setting than in a static setting.

The literature within the dynamic setting can be further subdivided into two areas: models based on control theory [10, 13, 14, 17, 19] and models based on game theory [11, 18, 21, 22].

The control theory based security models in the literature differ in the ways in which the dynamics are modeled. The work by Khouzani et al. [10] studies the problem of a malware attack in a mobile wireless network; the dynamics of the malware spread are modeled using differential equations. A large part of the literature on control theory based models focuses on problems where the dynamics are modeled by finite state automata. The works of [13, 14, 19] implement specific control policies (protocols) for security purposes. The work of Schneider [19] uses a finite state automaton to describe a setting where signals are sent to a computer. Given a set of initial possible states, the signals cause the state of the computer to evolve over time. An entity termed the *observer* monitors the evolution of the system and enforces security in real-time. Extensions of Schneider's model are centered around including additional actions for the observer. Ligatti et al. [13] extend Schneider's model by introducing a variety of abstract machines which can edit the actions of a program, at run-time, when deviation from a specified control policy is observed. More recent work [14] develops a formal framework for analyzing the enforcement of more general policies. Another category of dynamic defense concerns scenarios where the defender selects an *adaptive attack surface*[2] in order to change the possible attack and defense policies. A notion termed *moving target defense* (a term for dynamic system reconfiguration) is one class of such dynamic defense policies. The work of Rowe et al. [17] develops control theoretic mechanisms to determine maneuvers that modify the attack surface in order to mitigate attacks. The work involves first developing algorithms for estimation of the security state of the system, then formalizing a method for determining the cost of a given maneuver. The model uses a logical automaton to describe the evolution of the state of the system; however, it does not propose an analytical approach for *determining* an optimal defense policy.

[1] This new information could consist of the attacker's actions, events in nature, or the state of a some underlying system.

[2] For example, changing the network topology.

The next set of security models in the literature are based on the theory of dynamic games. The work in [15] considers a stochastic dynamic game to model the environment of conflict between an attacker and a defender. In this model, the state of the system evolves according to a Markov chain. This paper has many elements in common with our model; however, it assumes the attacker and defender have perfect observations of the system state. In our paper, we consider the problem from the defender's point of view and assume that the defender has imperfect information about the system state. The work by Khouzani [11] studies a zero-sum two-agent (malware agent and a network agent) dynamic game with perfect information. The malware agent is choosing a strategy which trades off malware spread and network damage while the network agent is choosing a counter-measure strategy. The authors illustrate that *saddle-point strategies* exhibit a threshold form. The work of Yin et al. [22] (dynamic game version of [3]) studies a Stackelberg game where the defender moves first and commits to a strategy. The work addresses how the defender should choose a strategy when it is uncertain whether the attacker will observe the first move. Van Dijk et al. [21] propose a two player dynamic game, termed *Flipit*, which models a general setting where a defender and an attacker fight (in continuous time) over control of a resource. The results concern the determination of scenarios where there exist dominant strategies for both players. We refer the reader to Roy et al. [18], and references therein, for a survey on the application of dynamic games to problems in security.

While models based on game theory have generated positive results in the static setting, there has been little progress in the dynamic setting. We believe this is for two reasons; first, dynamic security has not been fully investigated in a non-strategic context and second, the results in the theory of dynamic games are limited.

In this paper, we develop a (supervisory) control theory approach to a dynamic cyber-security problem and determine the optimal defense policy against progressive attacks. We consider a network of K computers, each of which can be in one of four security states, as seen in Figure 1. The state of the system is the K-tuple of the computer states and evolves in time with both defender and attacker actions. We use a finite state logical automaton to model the dynamics of the system. The defender adjusts to attacks based on the information available.

Our model takes a different approach than the existing papers in the literature. One fundamental difference of our work from the existing literature that make use of automata is the development of an *analytical* framework for *determining optimal defense policies* within a restricted set of policies. Other works involving automata propose methods for *enforcing* a predetermined policy, rather than determining an optimal policy. Also, our control theoretic approach considers imperfect information regarding attacker actions, which we feel is an aspect that is engrained into security problems.

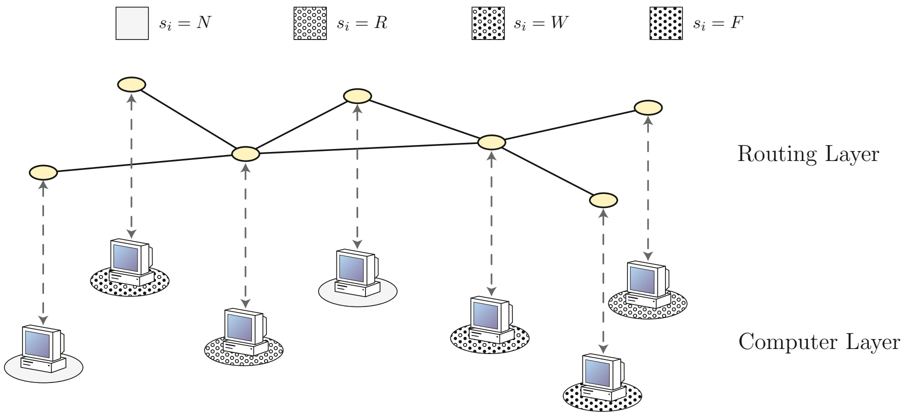


Fig. 1. *An instance of the problem that we consider. Computers are connected through a routing layer. Each computer can be in one of four security states:* normal *(N),* compromised *(R),* fully compromised *(W), or* remote compromised *(F).*

1.1 Contribution

The contribution of this paper is the development of a formal model for analyzing a dynamic cyber-security problem from the defender's point of view. Our approach has the following desirable features: (i) It captures the progressive nature of attacks; (ii) It captures the fact that the defender has imperfect knowledge regarding the state of the system; this uncertainty is a result of the fact that all attacks are uncontrollable and most are unobservable, by the defender; (iii) It allows us to quantify the cost incurred at every possible state of the system, as well as the cost due to every possible defender action; (iv) It allows us to quantify the performance of various defender policies and to determine the defender's optimal control policy, within a restricted set of policies, with respect to a min-max performance criterion.

1.2 Organization

The paper is organized as follows. In Section 2 we discuss our dynamic defense model. This is done by introducing the assumptions on the computer network and corresponding state, as well as the events which drive the evolution of the system state. In Section 3, we model the defender's problem of keeping the computer network as secure as possible while subjected to progressive attacks. We provide a simplified problem formulation that is tractable. In Section 4, we determine an optimal control policy for the defender based on dynamic programming. We discuss the nature of the optimal policy in Section 5. We offer conclusions and reflections in Section 6.

2 The Dynamic Defense Model

The key features of our model are characterized by assumptions **(A1)** – **(A6)**. We first describe the assumptions related to the *computer network*, discussed in assumption **(A1)**. In assumption **(A2)** we introduce the notion of the computer network *system state*. Next, in assumptions **(A3)** – **(A5)**, we discuss the *events* that can occur within the system. We describe how the events cause the system state to evolve, as well as specify which events are controllable and observable by the defender. In **(A6)** we discuss an assumption on the rules of interaction between the attacker and the defender. As mentioned in the introduction, we consider the cyber-security problem from the defender's viewpoint; the model we propose reflects this viewpoint.

Assumption 1 - Computer Network: *We assume a set of networked computers, $\mathcal{N} = \{1, 2, \dots, K\}$. Each computer, $i \in \mathcal{N}$, can be at security level $z^i \in \mathcal{M} = \{N, R, W, F\}$ where $\mathcal{M}$ is the set of security states.*

Each computer, $i \in \mathcal{N}$, is assumed to have three security boundaries, denoted by $\mathcal{B} = \{B_1, B_2, B_3\}$, representative of a layered structure to its security. These security boundaries partition the set of security states $\mathcal{M}$. Throughout this paper, we assume that the set of security states $\mathcal{M} = \{N, R, W, F\}$ is defined as follows.

- *Normal* ($z^i = N$): Computer i is in the *normal* state if none of the security boundaries have been passed by the attacker.
- *Compromised* ($z^i = R$): Computer i is *compromised* when security boundary B_1 has been passed by the attacker. In this state, the attacker has exploited some vulnerability on the computer and has managed to obtain user-level access privilege to the computer.
- *Fully Compromised* ($z^i = W$): Computer i is *fully compromised* when both boundaries B_1 and B_2 have been passed by the attacker. The attacker has exploited some additional vulnerability on the computer and has managed to obtain root level or execute privilege to the computer.
- *Remote Compromised* ($z^i = F$): Computer i is *remote compromised* when all security boundaries B_1, B_2, and B_3 have been passed by the attacker. The attacker has managed to obtain enough privileges to attack another computer and obtain user-level access privilege on that computer.

Assumption 2 - System State: *We assume that the computer network operates over an infinite time horizon, $\mathcal{T} = \{0, 1, 2, \dots\}$. The state of the computer network, Z_t, which evolves with time $t \in \mathcal{T}$, is the combination of the states of all the computers at time t. Each state Z_t has a corresponding cost.*

The state of the network, denoted $Z_t = (z_t^1, z_t^2, \dots, z_t^K) \in \mathcal{Z}$, is a K-tuple of all of the computer states.[3] The set $\mathcal{Z}$ denotes the set of all possible states,

[3] For example, a three computer network could have a network state of $Z_t' = (N, R, W)$. Notice that state Z_t' is distinct from state $Z_t'' = (R, N, W)$.

$\mathcal{Z} = \{Z^1, Z^2, \ldots, Z^{|\mathcal{M}|^K}\} = \{(N, N, \ldots, N), (N, N, \ldots, R), \ldots, (F, F, \ldots, F)\}$, where $|\mathcal{M}|^K$ is the number of system states.

The cost of the network state Z_t is defined by the costs of the states of the computers. We assign a cost, $c(z_t^i)$, to each computer i depending upon its state $z_t^i \in \mathcal{M}$. This cost is defined as follows

$$c(z_t^i) = \begin{cases} c_N & \text{if } z_t^i = N \\ c_R & \text{if } z_t^i = R \\ c_W & \text{if } z_t^i = W \\ c_F & \text{if } z_t^i = F \end{cases} \tag{1}$$

with $0 \leq c_N < c_R < c_W < c_F < \infty$. The cost of state Z_t is then defined as

$$C_{Z_t} = \sum_{i \in \mathcal{N}} c(z_t^i) \tag{2}$$

The state of the network, Z_t, evolves in time due to events, which we discuss in the next set of assumptions.

Assumption 3 - Events: *There is a set of events, $\mathcal{E} = \mathcal{A} \cup \mathcal{D}$, where $\mathcal{A}$ are the attacker's actions and $\mathcal{D}$ are the defender's actions.*

We assume that the attacker has access to three types of actions. The set of attacker actions, $\mathcal{A} = \big\{N^a, \{P_n^i\}_{i \in \mathcal{N}, n \in \mathcal{B}}, \{H^{ij}\}_{i,j \in \mathcal{N}}\big\}$, is defined as follows.

- N^a*, null*: The attacker takes no action. The null action does not change the system state and is admissible at any state of a computer.
- P_n^i*, security boundary attack*: Attacking the n^{th} security boundary of computer i causes the security state of computer i to transition across the n^{th} security boundary. Specifically, $P_{B_1}^i$ causes computer i to transition from normal, $z^i = N$, to compromised, $z^i = R$; $P_{B_2}^i$ from $z^i = R$ to $z^i = W$; and $P_{B_3}^i$ from $z^i = W$ to $z^i = F$. Actions $P_{B_1}^i$, $P_{B_2}^i$, and $P_{B_3}^i$ are only admissible from states $z^i = R$, $z^i = W$, and $z^i = F$, respectively.
- H^{ij}*, network attack*: Using a computer i in state $z^i = F$ to attack any other normal or compromised computer j in the network that is in state $z^j = \{N, R\}$ to bring computer j to state $z^j = W$. The action H^{ij} is admissible at state $z^i = F$ for $z^j \in \{N, R, W\}$.

We assume that the defender knows the set $\mathcal{A}$ as well as the resulting state transitions due to each action in $\mathcal{A}$.

The defender has access to three types of costly actions. These actions are admissible at any computer state. The set of defender actions, denoted by $\mathcal{D} = \big\{N^d, \{E_i\}_{i \in \mathcal{N}}, \{R_i\}_{i \in \mathcal{N}}\big\}$, is defined as follows.

- N^d*, null*: The defender takes no action. The null action does not change the system state.
- E_i*, sense computer i*: The *sense* action, E_i, reveals the state of computer i to the defender. The sense action does not change the system state.

- *R_i, re-image computer i*: The *re-image* action, R_i, brings computer i back to the normal state from any state that it is currently in. For example, R_3 applied to state (N, R, F) results in (N, R, N).

The costs of the actions in $\mathcal{D}$ are defined by $\hat{C}(N^d)$, $\hat{C}(E_i)$, $\hat{C}(R_i)$, where $0 \leq \hat{C}(N^d) < \hat{C}(E_i) < \hat{C}(R_i) < \infty$ for all $i \in \mathcal{N}$.

Assumption 4 - Defender's Controllability of Events: *The actions in $\mathcal{A}$ are uncontrollable whereas the actions in $\mathcal{D}$ are controllable.*

Since the problem is viewed from the perspective of the defender, all actions in $\mathcal{D}$ are controllable. For the same reason, the defender is unable to control any of the attacker's actions $\mathcal{A}$.

Assumption 5 - Defender's Observability of Events: *All actions in $\mathcal{D}$ and some actions in $\mathcal{A}$ are assumed to be observable.*

Again, due to taking the defender's viewpoint, all actions in $\mathcal{D}$ are observable. Although we assume that the defender knows the set $\mathcal{A}$, we assume that it cannot observe N^a or any P_n^i actions; it can only observe actions of the type H^{ij}. One justification for this is that the the network attack H^{ij} involves passing sensitive information of computer j through the routing layer of the system to computer i.[4] We assume that the routing layer is able to detect the transfer of sensitive data through the network, and thus the defender is aware when an action of the form H^{ij} occurs.

Assumption 6 - Defender's Decision Epochs: *The defender acts at regular, discrete time intervals. At these time intervals, the defender takes only one action in $\mathcal{D}$. The attacker takes one action in $\mathcal{A}$ between each defender action.*

We require that the defender should consider taking a single action in $\mathcal{D}$ at regular time instances. We assume that between any two such instances, the attacker can only take one action in $\mathcal{A}$. This order of events is illustrated in Figure 2 for a given time $t = \tau$. We introduce *intermediate states*, denoted by $\tilde{\mathcal{Z}} = (\tilde{Z}^1, \tilde{Z}^2, \ldots, \tilde{Z}^{|\mathcal{M}|^K})$, which represent the system states at which events from $\mathcal{A}$ are admissible (that is, the states in which the attacker takes an action). The *system states*, denoted by $\mathcal{Z} = (Z^1, Z^2, \ldots, Z^{|\mathcal{M}|^K})$, are the states at which actions from $\mathcal{D}$ are admissible.

Assumption **(A6)** is, in our opinion, reasonable within the security context. Since time has value in security problems,[5] the defender should take actions at regular time intervals (note that at these instances the defender may choose N^d, that is, choose to do nothing). In general, a finite number of events in $\mathcal{A}$ may occur between any two successive defender actions; however, to reduce the dimensionality of the problem, we assume that only one event in $\mathcal{A}$ can occur.

[4] This sensitive information could be the login credentials of computer j.

[5] A computer that is compromised by the attacker for two time steps is more costly to the defender than a computer that is compromised for one time step.

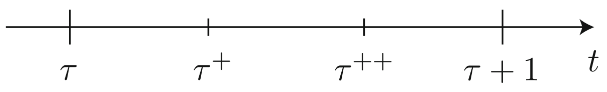


Fig. 2. *Order of events for a given time-step. At time $t=\tau$, the cost of the current state C_{Z_τ} is realized. At τ^+, the defender takes an action in $\mathcal{D}$ (the cost of which is realized immediately). The resulting system state due to the defender's action is denoted by $\tilde{Z}_{\tau^+} \in \tilde{\mathcal{Z}}$. At τ^{++} the attacker takes an action in $\mathcal{A}$. At $\tau+1$, the resulting system state is denoted by $Z_{\tau+1} \in \mathcal{Z}$.*

One important implication of assumption **(A6)** is related to the defender's observability of events in $\mathcal{A}$. By **(A6)**, the defender is aware *when* an event in $\mathcal{A}$ occurs. Since the event H^{ij} is observable, if the defender does not observe H^{ij} when an event in $\mathcal{A}$ is known to occur, then it knows that one of the unobservable events, N^a or one of $\{P_n^i\}_{i\in\mathcal{N},n\in\mathcal{B}}$, has occurred. To incorporate this fact into the defender's knowledge about the system's evolution, we group the above mentioned unobservable events into one event, denoted $X = \big\{N^a, \{P_n^i\}_{i\in\mathcal{N},n\in\mathcal{B}}\big\}$. This philosophy is used in constructing the system automaton from the defender's point of view, as well as in defining the defender's information state (discussed in Section 3). As a result of the above grouping, the set of events $\mathcal{A}' = \big\{X, \{H^{ij}\}_{i,j\in\mathcal{N}}\big\}$ is observable by the defender. Notice, however, that by performing this grouping, we have introduced non-determinism into the system; that is, the event X can take the system to many possible system states. All unobservable events in the problem have been eliminated due to Assumption **(A6)** and the grouping of unobservable events in $\mathcal{A}$.

As a result of assumptions **(A1)** – **(A6)**, the evolution of the system state, Z_t, from the defender's viewpoint, can be modeled by a discrete event system represented by a finite state automaton, which we term the *system automaton.* Due to assumption **(A6)**, we duplicate the system states by forming the set of intermediate states, denoted by $\tilde{\mathcal{Z}} = (\tilde{Z}^1, \tilde{Z}^2, \ldots, \tilde{Z}^{|\mathcal{M}|^K})$. The set of intermediate states represents the states at which an event from $\mathcal{A}$ can occur. The set of system states, denoted by $\mathcal{Z}$, are the states at which the defender takes an action $d \in \mathcal{D}$. The resulting automaton has $2|\mathcal{M}|^K$ states. The set of events that can occur is described by the set $\mathcal{E}' = \mathcal{A}' \cup \mathcal{D}$; the transitions due to these events follow the rules discussed in assumption **(A3)**. The system automaton takes the form of a bipartite graph, as seen in Figure 3. Notice that, like the null action, the sense actions, E_i, for all $i \in \mathcal{N}$, do not change the underlying system state. The purpose of sense is to update the defender's information state, which will be defined and explained in the following section.

3 The Defender's Problem

We now formulate the defender's problem – protecting the computer network. The defender must decide which costly action to take, at each time step, in order

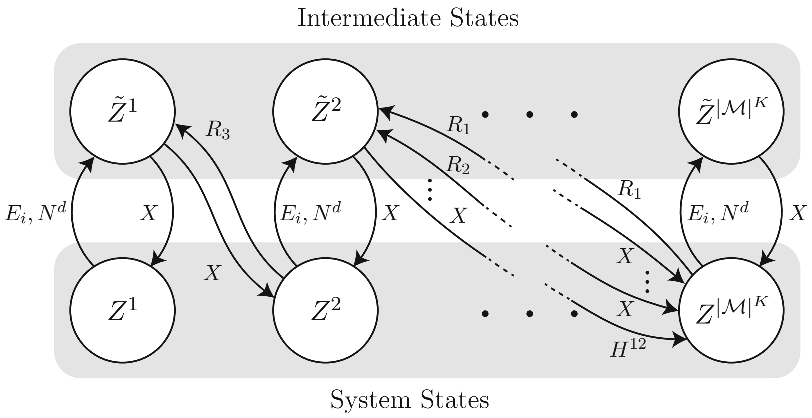


Fig. 3. *The system automaton represented as a bipartite graph of intermediate states,* $\tilde{\mathcal{Z}} = (\tilde{Z}^1, \tilde{Z}^2, \ldots, \tilde{Z}^{|\mathcal{M}|^K})$*, and system states,* $\mathcal{Z} = (Z^1, Z^2, \ldots, Z^{|\mathcal{M}|^K})$*, with events* $\mathcal{E}' = \mathcal{A}' \cup \mathcal{D}$*. Notice the non-determinism of the event* $X \in \mathcal{A}'$*.*

to keep the system as secure as possible given that it has imperfect knowledge of the network's state.

3.1 The Defender's Optimization Problem

Let $g := \{g_t, t \in \mathcal{T}\}$, denote a control policy of the defender, where

$$g_t : \mathcal{D}^{t-1} \times {\mathcal{A}'}^{t-1} \to \mathcal{D}, \tag{3}$$

and $\mathcal{D}^{t-1}$ and ${\mathcal{A}'}^{t-1}$ denote the space of the defender's actions and observations up to $t-1$, respectively. Let $\mathcal{G} := \{g \,|\, g_t : \mathcal{D}^{t-1} \times {\mathcal{A}'}^{t-1} \to \mathcal{D} \text{ for all } t \in \mathcal{T}\}$ denote the space of admissible control policies for the defender.

The defender's optimization problem is

$$\min_{g \in \mathcal{G}} \max_{\{Z_t^g \in \mathcal{Z}, t \in \mathcal{T}\}} \left\{ \sum_{t \in \mathcal{T}} \beta^t \left[C_{Z_t^g} + \hat{C}\big(d_t\big) \right] \right\} \qquad (P_D)$$
$$\text{subject to Assumptions } \textbf{(A1)} - \textbf{(A6)}$$

where $\{Z_t^g \in \mathcal{Z}, t \in \mathcal{T}\}$ denotes a sequence of states generated by control policy g and d_t is the defender's action at t generated according to Equation (3). Problem (P_D) is a supervisory control problem with imperfect observations.

3.2 Discussion of Problem (P_D)

The notion of an information state [12] is a key concept in supervisory (and general) control problems with imperfect information. Because of the nature of the performance criterion and the fact that the defender's information is imperfect, an appropriate information state for the defender at time t is $\sigma(\mathcal{D}^{t-1}, \mathcal{A}'^{t-1})$, the σ-field generated by the defender's actions and observations, respectively, up to $t-1$. Using such an information state, one can, in principle, write the dynamic program for Problem (P_D). Such a dynamic program is computationally intractable. For this reason, we formulate another problem, called (P'_D), where we restrict attention to a set of defense policies that have a specific structure; in this problem we can obtain a computationally tractable solution.

3.3 Specification of Problem (P'_D)

We define the *defender's observer* as follows. The defender's observer is built using the defender's observable events, $\mathcal{A}'$, and its actions, $\mathcal{D}$. The observer's state at time t, denoted by $S_t \subseteq \mathcal{Z}$, consists of the possible states that the network can be in at time t from the defender's perspective. We denote by $\mathcal{S}$ the space to which S_t belongs, for any $t \in \mathcal{T}$.

The evolution of the observer's state is described by the function $f : \mathcal{S} \times \mathcal{D} \times \mathcal{A}' \to \mathcal{S}$. The observer's state S_t follows the update

$$S_{t+1} = f(S_t, d_t, a'_t)$$

where $d_t \in \mathcal{D}$ is the realization of the defender's action and its effect at time t^+, and $a'_t \in \mathcal{A}'$ is the realization of the defender's observation at t^{++}. The precise form of the function f is determined by the dynamic defense model of Section 2. Thus, the dynamics of the defender's observer are described by a finite state automaton with state space $\mathcal{S}$ and transitions that obey the dynamics defined by the function $f(S_t, d_t, a'_t)$.

Using the defender's observer we formulate Problem (P'_D) as follows.

$$\min_{g \in \mathcal{G}'} \max_{Z_t^g \in \mathcal{Z}, t \in \mathcal{T}} \left\{ \sum_{t \in \mathcal{T}} \beta^t \left[C_{Z_t^g} + \hat{C}\big(d_t\big) \right] \right\} \qquad (P'_D)$$

$$\begin{aligned} \text{subject to} \quad & \text{Assumptions } \textbf{(A1)} - \textbf{(A6)}, \\ & d_t = g_t(S_t),\ t \in \mathcal{T}, \\ & Z_t^g \in S_t,\ t \in \mathcal{T}, \\ & S_{t+1} = f(S_t, d_t, a'_t),\ t \in \mathcal{T}. \end{aligned}$$

where $\mathcal{G}' := \{g \,|\, g := \{g_t, t \in \mathcal{T}\}, g_t : \mathcal{S} \to \mathcal{D} \text{ for all } t \in \mathcal{T}\}$.

4 Dynamic Programming Solution for the Defender's Problem

4.1 The Dynamic Program

We solve Problem (P'_D) using dynamic programming. The dynamic program corresponding to Problem (P'_D) is

$$V(S) = \min_{d \in \mathcal{D}} \max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V(S') \right]. \qquad (4)$$

for every $S \in \mathcal{S}$ (see [2,12]), where $\mathcal{Q}(S, d, Z)$ is the set of observer states that can be reached by S when the defender's action is d and the true system state in $\mathcal{S}$ is Z. The set $\mathcal{Q}(S, d, Z)$ is determined as follows. If at time t the observer's state is S and the defender takes action d then, before the effect of d at time t^+ and the observation at time t^{++} are realized, there will be several potential candidate observer states at $t+1$. Only a subset of these possible observer states can occur when the true state of the system at time t is $Z \in S$. This subset is $\mathcal{Q}(S, d, Z)$. We illustrate the form of the set $\mathcal{Q}(S, d, Z)$ by the following example.

Example 1. Assume a network of three computers and a current observer state of

$$S_t = \{(F, N, N),\ (F, N, R),\ (F, R, N)\}.$$

If the defender takes action E_2 then, before the effect of E_2 and the observation $H^{1,2}$ at t^{++} are realized, the possible observer states S_{t+1} are

$$\big\{\{(F, W, N),\ (F, W, R)\},\ \{(F, W, N)\}\big\}.$$

If the true system state is $Z_t = (F, N, R)$ then

$$\mathcal{Q}(S_t, E_2, Z_t) = \{(F, W, N),\ (F, W, R)\}.$$

$\triangle$

4.2 Solution of the Dynamic Program

We obtain the solution of the dynamic program, Equation (4), via *value iteration* [2,12]. For that matter, we define the operator T by

$$TV(S) := \min_{d\in\mathcal{D}} \max_{Z\in S} \left[C_Z + \hat{C}(d) + \max_{S'\in\mathcal{Q}(S,d,Z)} \beta V(S') \right]. \tag{5}$$

We prove the following result.

Theorem 1. *The operator T, defined by Equation (5), is a contraction map.*

Proof. We use Blackwell's sufficiency theorem (Theorem 5, [4]) to show that T is a contraction mapping. We show:

i) *Bounded value functions*: First, note that $|\mathcal{S}|, |\mathcal{D}| < \infty$, and that we have bounded costs, $C_Z \leq M_1 < \infty, \forall\, S \in \mathcal{S}$; $\hat{C}(d) \leq M_2 < \infty, \forall\, d \in \mathcal{D}$. Starting from any bounded value function, $V(S) \leq M_3 < \infty$ with $M_3 > \frac{M_1+M_2}{1-\beta}$ we have

$$\begin{aligned} TV(S) &= \min_{d\in\mathcal{D}} \max_{Z\in S} \left[C_Z + \hat{C}(d) + \max_{S'\in\mathcal{Q}(S,d,Z)} \beta V(S') \right] \\ &\leq M_1 + M_2 + \beta M_3 < M_3 < \infty \end{aligned}$$

for all $S \in \mathcal{S}$.

ii) *Monotonicity*: Assume $V_2(S) \geq V_1(S)\ \forall\, S \in \mathcal{S}$. Then, for all $S \in \mathcal{S}, Z \in S$ and $d \in D$,

$$C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V_2(S') \geq C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V_1(S')$$

Therefore, for all $S \in \mathcal{S}$ and $d \in D$

$$\max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V_2(S') \right] \geq$$
$$\max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V_1(S') \right]$$

Hence,

$$\begin{aligned} TV_2(S) &= \min_{d \in \mathcal{D}} \max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V_2(S') \right] \\ &\geq \min_{d \in \mathcal{D}} \max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V_1(S') \right] = TV_1(S). \end{aligned}$$

iii) *Discounting*: Assume $V_2(S) = V_1(S) + a$. Then, for all $S \in \mathcal{S}$

$$\begin{aligned} TV_2(S) &= \min_{d \in \mathcal{D}} \max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta (V_1(S') + a) \right] \\ &= \min_{d \in \mathcal{D}} \max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V_1(S') \right] + \beta a \\ &= TV_1(S) + \beta a. \end{aligned}$$

By Blackwell's sufficiency theorem, the operator T is a contraction mapping. □

Since T is a contraction mapping, we can use value iteration to obtain the solution to Equation (4), which we term the stationary value function, $V^*(S)$. From the stationary value function, we can obtain an optimal policy, g^*, as follows

$$g^*(S) = \arg\min_{d \in \mathcal{D}} \max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V(S') \right]$$

The optimal policy, $g^*(S)$, is not always unique. That is, for a given observer state $S \in \mathcal{S}$, there could be multiple $d \in \mathcal{D}$ which achieve the same minimum value of $\min_{d \in \mathcal{D}} \max_{Z \in S} \left[C_Z + \hat{C}(d) + \max_{S' \in \mathcal{Q}(S,d,Z)} \beta V(S') \right]$. We denote by $\mathcal{D}^*(S)$ the set of optimal actions for a given observer state S. In the event that $\mathcal{D}^*(S)$ is not a singleton for a given state S, we choose a single action $d^*(S) \in \mathcal{D}^*(S)$ based on a quantity we define as the *confidentiality threat*. The confidentiality threat is a measure of the degree to which computer i is presumed (by the defender) to be compromised and is defined as follows

$$\tilde{T}_i = \sum_{Z \in S} c(z^i), \ \ i \in \mathcal{N}$$

where $c(z^i)$, $z^i \in \mathcal{M}$, is the cost of the state, as defined in Equation (1), of the i^{th} computer in the candidate system state $Z \in S$. Summing over all candidate system states in the observer state S for a given computer i, we obtain the confidentiality threat $\tilde{T}_i$. Next, we compare the confidentiality threat of each computer and choose the action $d^*(S) \in \mathcal{D}^*(S)$ that corresponds to the highest confidentiality threat. In the case of equal confidentiality threats (which arise when the observer state is symmetric), we choose the action in $\mathcal{D}^*(S)$ corresponding to the computer with the lower index $i \in \mathcal{N}$.[6]

5 Optimal Defender's Policy

We now discuss the characteristics of the optimal policy for Problem (P_D'), henceforth referred to as the *optimal policy.* We illustrate sensitivity analysis via numerical results for both a two computer and a three computer network. We also discuss some qualitative observations of the optimal policy.

First we note that determining the set of observer states and its associated dynamics is not a trivial computational task, even for moderately sized networks. Our calculations show for the case of a two computer network, the defender's observer automaton consists of 87 states and 1207 transitions. Extending the system to a three computer network results in 1423 states with 65602 transitions. To automate the procedure, we have developed a collection of programs which makes use of the UMDES-LIB software library [1]. The specific procedure is discussed in Appendix A.

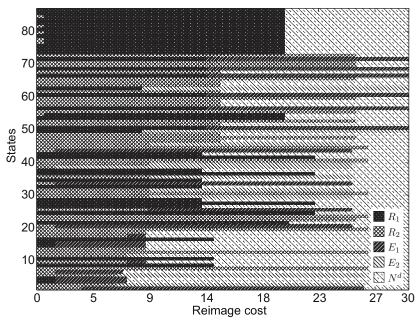
The sensitivity analysis studies how the cost of re-imaging affects the optimal policy. For both the two computer and three computer networks, we increase the re-image cost, $\hat{C}(R_i) = r$, $\forall i \in \mathcal{N}$, and observe how the optimal policy behaves. Since the number of observer states in the two computer network, denoted $|\mathcal{S}_2|$, is modest, $|\mathcal{S}_2| = 87$, we are able to plot the behavior for each observer state $S \in \mathcal{S}_2$, as seen in Figure 4(a).[7] In the three computer network, the size of observer state space, $|\mathcal{S}_3| = 1423$, is much larger than that of the two computer network. As a result, we plot the percentage of observer states that have the optimal action d, for all $d \in \mathcal{D}$, and analyze how the percentage changes as we increase r, as seen in Figure 4(b).

The behavior of the optimal policy due to increasing re-image costs, r, is intuitive. As r increases, the optimal policy exhibits a threshold form,[8] switching from specifying more expensive actions to less expensive actions. For very low re-image costs, the optimal policy specifies R_i in the majority of the observer states. As r increases, observer states for which R_i was optimal, switch to either sense, E_i, or null, N^d. Once the optimal action is null, it remains null for all higher values of r. For the observer states where the action switched to sense, a further increase in r may result in a switch to null; however, there exist some

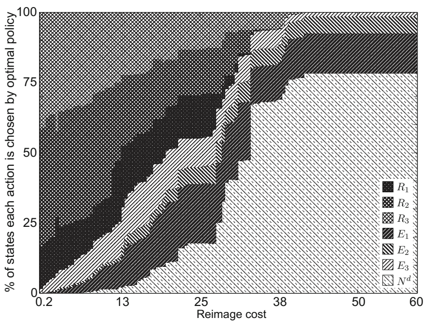
[6] This choice is arbitrary; we could randomize the choice as well.

[7] The "ordering" of these states is arbitrary.

[8] In the simulations that we have performed.



(a) *Two computer network.* Optimal actions for each observer state as a function of increasing re-image cost, $r = 3, \ldots, 30$.



(b) *Three computer network.* Percentage of observer states that have optimal actions $d \in \mathcal{D}$ as a function of increasing re-image cost, $r = 0.2, \ldots, 60$.

Fig. 4. *Sensitivity analysis for varying re-image cost r, where $r = \hat{C}(R_i)$ for all $i \in \mathcal{N}$. Other parameters are $\hat{C}(N_d) = 0$, $\hat{C}(E_i) = 0.1\ \forall i \in \mathcal{N}$, $c_N = 0$, $c_R = 1$, $c_W = 2$, $c_F = 8$, and $\beta = 0.9$.*

observer states where the optimal action is sense for all higher values of r. This threshold behavior is clearly depicted in Figure 4(a).

As a result of the aforementioned threshold behavior, for high enough values of r, the optimal policy eventually specifies N^d or E_i for all states $S \in \mathcal{S}$. The argument to see why there is no re-image action for high values of r is straightforward; at these values of r the cost of re-imaging is prohibitively expensive and the defender would rather incur the cost of being in a poor system state (see Equation (2)).

An interesting (related) observation can be seen by analyzing the characteristics of the observer states and how these characteristics influence *when* the policy undergoes a switch as r increases. Consider Figure 4(a), and observe the behavior of the optimal policy around the re-image cost of $r = 20$. There is a collection of observer states (with indices 74 – 87) that contain the (F, F) element (both computers are in the remote compromised state) where the optimal policy specifies a switch from re-image to null. In these observer states, the defender believes that the true system state is so poor that, even if the a computer were to be re-imaged, the events in $\mathcal{A}$ would cause the system to transition back to a poor state in so few iterations that the defender would just be wasting its resources by re-imaging. That is, the number of time steps that it takes for the system to return to a poor state is not high enough to justify the cost that the defender must incur to keep the system in a secure operating mode. For this reason, in these observer states, the defender exhibits the passive behavior of *giving up* by choosing the cheapest action, N^d. An interesting related observation is that for other observer states in the system (the observer states that do not contain the element (F, F)) the optimal policy specifies a switch away from re-image at a higher re-image cost (around $r \in [25\ \ 26]$). In these observer states the defender views the process of securing the system as economically efficient because it can be returned to a secure operating mode in a small enough number of iterations (compared to the observer states that contain the system state (F, F)). This observed behavior reflects the fact that attacks are progressive and that time has value in our model.

Another observation is that there are sets of parameters for which the sense action is useful (as seen starting in Figure 4(a) around $r = 2$ and peaking in Figure 4(b) around $r = 25$). In these cases the act of sensing a computer results in a split observer state that has a lower future cost than if the defender were to choose either null or re-image. Thus, paying the cost to sense can result in the defender having a better idea of the underlying system state and thus make a wiser decision on which future action to take. However, for low values of r, we can see that the defender prefers to re-image over obtaining a better estimate of the system (and similarly for high values of r, the defender prefers to take the null action). This behavior highlights the duality between estimation and control.

Interestingly, sensing remains an optimal action even for high values of r when there is no re-image action prescribed in the optimal defense policy. In these cases, even though sensing does not change the state of the network, it refines the defender's information which then results in a lower future cost for the defender. Even though the sense action is more expensive than the null action, this lower future cost causes the defender to choose sense over null.

The intent of determining an optimal policy is to offer a set of procedures for the defender such that the network is able to be kept as secure as possible. After the defender specifies its costs for actions and costs for states, the optimal policy specifies a procedure that the defender should follow. For each action the defender takes, $d \in \mathcal{D}$, and for each event it observes, $a' \in \mathcal{A}'$, the resulting observer state is known through the dynamics of the observer state. For each of these observer states resulting from the sequence of defender actions and observed events, the optimal policy specifies whether to sense or re-image a particular computer, or to wait and do nothing. The resulting defender behavior will keep the network as secure as possible under the min-max cost criterion.

6 Conclusion and Reflections

In this paper we have proposed a supervisory control approach to dynamic cyber-security. We have taken the viewpoint of the defender whose task is to defend a network of computers against progressive attacks. Some of the attacker actions are unobservable by the defender, thus the defender does not have perfect knowledge of the true system state. We define an observer state for the defender to capture this lack of perfect knowledge.

We have assumed that the defender takes a conservative approach to preserving the security of the system. We have used the min-max performance criterion to capture the defender's conservative approach.

Dynamic programming was used to obtain an optimal defender policy to Problem (P'_D). The numerical results show that the optimal policy exhibits a threshold behavior when the cost of actions are varied. We have also observed the duality of estimation and control in our optimal policy.

We believe that our approach is suitable for modeling interactions between an attacker and a defender in general security settings. In general, we can use our approach to study dynamic defense against attacks in a network of N resources each with $\mathcal{M}$ (orderable) security levels and $\mathcal{M} - 1$ security boundaries. The attack actions can penetrate through some of these boundaries to compromise a resource, or use a compromised resource to attack other resources in the network. Some of these actions can be unobservable to the defender. On the other hand, the defender can take actions to change the state of resources to a more secure operating mode or sense the system state to obtain more refined information about the system's status.

The model we have defined is rich enough to be extended to capture more complicated environments. Some examples of such environments can be heterogeneity of the network's computers[9] or the introduction of a dummy computer[10] into the system so as to increase the network's resiliency to attacks.

One bottleneck of our approach is that the number of states and transitions grows exponentially with the number of computers. One solution to this is to use a hierarchical decomposition for the system. For example an *Internet Service Provider* (ISP) can model a collection of nodes in their network as one region (resource). Once a non-secure region is observed in the system, the ISP can more carefully analyze the nodes within that region and take appropriate actions. Approximate dynamic programming methods could also be useful in dealing with systems with a large number of computers.

Acknowledgement. This work was supported in part by NSF grant CNS-1238962 and ARO MURI grant W911NF-13-1-0421. The authors are grateful to Eric Dallal for helpful discussions.

References

[1] Umdes-lib (August 2000), `https://www.eecs.umich.edu/umdes/toolboxes.html`
[2] Bertsekas, D.P.: Dynamic programming and optimal control, vol. 1. Athena Scientific, Belmont (1995)
[3] Bier, V., Oliveros, S., Samuelson, L.: Choosing what to protect: Strategic defensive allocation against an unknown attacker. Journal of Public Economic Theory 9(4), 563–587 (2007)
[4] Blackwell, D.: Discounted dynamic programming. The Annals of Mathematical Statistics, 226–235 (1965)
[5] Bloem, M., Alpcan, T., Başar, T.: Optimal and robust epidemic response for multiple networks. Control Engineering Practice 17(5), 525–533 (2009)

[9] Placing an *importance* weight on each computer.
[10] The dummy computer contains no sensitive information and is meant to mislead the attacker.

[6] Bloem, M., Alpcan, T., Schmidt, S., Basar, T.: Malware filtering for network security using weighted optimality measures. In: IEEE International Conference on Control Applications, CCA 2007, pp. 295–300. IEEE (2007)

[7] Böhme, R., Félegyházi, M.: Optimal information security investment with penetration testing. In: Alpcan, T., Buttyán, L., Baras, J.S. (eds.) GameSec 2010. LNCS, vol. 6442, pp. 21–37. Springer, Heidelberg (2010)

[8] Chen, T.M., Jamil, N.: Effectiveness of quarantine in worm epidemics. In: IEEE International Conference on Communications, ICC 2006, vol. 5, pp. 2142–2147. IEEE (2006)

[9] Hart, S.: Discrete colonel blotto and general lotto games. International Journal of Game Theory 36(3-4), 441–460 (2008)

[10] Khouzani, M., Sarkar, S., Altman, E.: Maximum damage malware attack in mobile wireless networks. IEEE/ACM Transactions on Networking 20(5), 1347–1360 (2012)

[11] Khouzani, M., Sarkar, S., Altman, E.: Saddle-point strategies in malware attack. IEEE Journal on Selected Areas in Communications 30(1), 31–43 (2012)

[12] Kumar, P.R., Varaiya, P.: Stochastic systems: Estimation, identification and adaptive control. Prentice-Hall, Inc. (1986)

[13] Ligatti, J., Bauer, L., Walker, D.: Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security 4(1-2), 2–16 (2005)

[14] Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Transactions on Information and System Security (TISSEC) 12(3), 19 (2009)

[15] Lye, K., Wing, J.M.: Game strategies in network security. International Journal of Information Security 4(1-2), 71–86 (2005)

[16] Mastroleon, L.: Scalable resource control in large-scale computing/networking infrastructures. ProQuest (2009)

[17] Rowe, J., Levitt, K.N., Demir, T., Erbacher, R.: Artificial diversity as maneuvers in a control theoretic moving target defense. In: National Symposium on Moving Target Research (2012)

[18] Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A survey of game theory as applied to network security. In: 2010 43rd Hawaii International Conference on System Sciences (HICSS), pp. 1–10. IEEE (2010)

[19] Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (2000)

[20] Schwartz, G.: Blotto games for security, review and directions. Presented at NetEcon Meeting. University of California, Berkeley (Private communication) (2013)

[21] Van Dijk, M., Juels, A., Oprea, A., Rivest, R.L.: Flipit: The game of stealthy takeover. Journal of Cryptology 26(4), 655–713 (2013)

[22] Yin, Z., Korzhyk, D., Kiekintveld, C., Conitzer, V., Tambe, M.: Stackelberg vs. nash in security games: Interchangeability, equivalence, and uniqueness. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 1139–1146, International Foundation for Autonomous Agents and Multiagent Systems (2010)

A Appendix – UMDES-LIB

The UMDES-LIB library [1] is a collection of C-routines that was built to study discrete event systems that are modeled by finite state automata. Through specification of the states and events of a system automaton (along with the controllability and observability of events), the library can construct an entity termed the *observer automaton.* In our problem the observer automaton is the defender's observer automaton, since we take the viewpoint of the defender. Thus, the observer automaton consists of the defender's observer states.

In this appendix we describe an automated process[11] for extracting the defender's observer state from the system automaton that makes use of UMDES-LIB. This requires first *constructing the system automaton* in an acceptable format for the library while preserving all the features of our model. After running the library on the provided system automaton, we *extract the defender's observer state* from the observer automaton output. This method allows one to construct the defender's observer state for any number of computers.[12]

Constructing the System Automaton. The input that we provide to UMDES-LIB is the system automaton from the defender's viewpoint, as illustrated earlier in Figure 3.

In order to preserve all features of our model in the resulting observer automaton, we need to introduce additional sensing actions. Recall that the sense action, $\{E_i\}_{i\in\mathcal{N}}$, causes the system automaton to transition to the same state as the null action, N^d (see Figure 3). However, as stated in Section 2, the sense action updates the information state of the defender. In order to ensure that UMDES-LIB captures this functionality, we expand the sense action E_i for each computer i into $|\mathcal{M}|$ distinct actions, denoted by $E_i^{z^i}$, which represent sensing computer i when it is in state $z^i \in \mathcal{M}$. This results in a reduced level of uncertainty for the defender as it splits the observer state into, at most, $|\mathcal{M}|$ possible sets of observer states. The admissible actions from $\{E_i^{z^i}\}_{z^i\in\mathcal{M}}$, at a given system state, are the sense actions that correspond to the true system state. For example, from the system state $Z_t = (N, R, W)$, the admissible sense actions are E_1^N, E_2^R, and E_3^W. The above example of the expanded sense action is perhaps worrisome at first glance – if the only admissible sense actions from the current state are the ones that correspond to the current state of the computer, then the defender will know what the current state of each computer is, eliminating the need for a sense action. However, the observer state that is obtained from each expanded sense action is the same as the observer state that is obtained if the defender were to observe the true, unknown state of a computer.

Running UMDES-LIB on the system automaton with the expanded sense actions results in the observer automaton.

[11] Source code is available upon request.

[12] The only bottleneck being the (potentially large) dimensionality of the problem.

Extracting the Defender's Observer State. The output of UMDES-LIB is the observer automaton, from which we must extract the defender's observer state. First, since the defender does not have the ability to choose the expanded sense actions, $E_i^{z^i}$, we re-group them into a single, non-deterministic action, $E_i \in \mathcal{D}$, for each $i \in \mathcal{N}$. Next, we need to extract the function, $f : \mathcal{S} \times \mathcal{D} \times \mathcal{A}' \to \mathcal{S}$ from the observer automaton. The observer automaton, generated by UMDES-LIB, takes the form of a bipartite graph; one collection of states of the bipartite graph is observer states over system states $\mathcal{Z}$, denoted $\mathcal{S}$, whereas the other collection is observer states over intermediate states $\tilde{\mathcal{Z}}$, denoted $\tilde{\mathcal{S}}$. Defender actions, $\mathcal{D}$, are the only admissible actions from observer states $\mathcal{S}$. The defense action $d \in \mathcal{D}$ causes a transition[13] to an observer state in $\tilde{\mathcal{S}}$, where only events in $\mathcal{A}'$ are admissible. Each event $a' \in \mathcal{A}'$ causes a transition back to an observer state in $\mathcal{S}$. Repeating this process for all observer states in $\mathcal{S}$, actions $d \in \mathcal{D}$, and events $a' \in \mathcal{A}'$, the function $f : \mathcal{S} \times \mathcal{D} \times \mathcal{A}' \to \mathcal{S}$ is defined. To construct the set $\mathcal{Q}(S, d, Z)$ we follow the approach described in Section 4.1 and illustrated by Example 1.

[13] This transition may be non-deterministic due to the sense action.