A POMDP Approach to the Dynamic Defense of Large-Scale Cyber Networks

Erik Miehling, Mohammad Rasouli, and Demosthenis Teneketzis

***Abstract*— We investigate the problem of optimally mitigating the progression of an adversary through a network in real-time, decreasing the probability of it reaching its goal(s), while minimizing the negative impact to availability. Our model is based on a type of attack graph, termed a condition dependency graph, which models the dependencies between security conditions (attacker capabilities) and exploits. By embedding a state space on the graph, we are able to quantify the progression of the attacker over time. The defender is able to interfere with the attacker's progression by blocking some exploits from being carried out. The nature of the attacker's progression through the network is dictated by its private strategy, which depends on the defender's action. The defender's uncertainty of the attacker's true strategy is modeled by considering a finite collection of attacker types. Using noisy security alerts (exhibiting both missed detections and false alarms), the defender maintains a belief representing the joint distribution over the attacker's current capabilities and true strategy. The resulting problem of determining how to optimally interfere with the attacker's progression is cast as a partially observable Markov decision process. To deal with the large state space, we develop a scalable online defense algorithm for tracking beliefs and prescribing defense actions over time. Using the context provided by the state, we are able to efficiently process security alerts even in the presence of a high rate of false alarms. The behavior of the computed defense policy is demonstrated on an illustrative example.**

***Index Terms*— Network security, intrusion response systems, control theory, partially observable Markov decision processes, scalability.**

I. Introduction

THE high connectivity of modern cyber networks and devices has brought with it many improvements to the functionality and efficiency of our networked systems. Unfortunately, these benefits have come with the introduction of many new entry-points for attackers, making our systems much more vulnerable to intrusions. Recent events, such as information leakage & theft [2], car hacking [3], and denial-of-service attacks [4], have highlighted this vulnerability. Particularly concerning is that the operation of critical infrastructure is becoming increasingly reliant upon (potentially insecure) networked systems, generating significant vulnerabilities in many areas of society. As reported by the Department of Homeland Security's Industrial Control Systems Cyber Emergency Response Team (ICS-CERT), attacks on critical infrastructure sectors (such as manufacturing, energy, communication, water, and transportation systems) have remained persistent over the past few years, with 245 in 2014, 295 in 2015, and 290 in 2016 [5]. Unfortunately, due to the increased reliance of these systems on cyber networks, coupled with an escalation in the sophistication of cyber attacks, many of the recent intrusions have had the potential to inflict severe and widespread damage (an increasing number of attacks have reached the *control system layer* of the system [5]). It is imperative that methods are developed to detect and mitigate these attacks in order to ensure the secure operation of society's critical systems.

One approach to mitigating attacks is, upon discovery of a vulnerability, to develop and release a patch to remove the vulnerability. Unfortunately, the period between discovery of a vulnerability and the application of a patch (termed the *vulnerability exposure window*) is long, often lasting on the order of five months or more [6]. This significant delay results in many cyber networks being operational while multiple known vulnerabilities are present, resulting in significant risks to society. This concern necessitates the development of an active defense system that is capable of taking into account information in real-time, inferring the security status of the network, and translating this information into appropriate defense decisions that are able to immediately respond to and mitigate the progression of the attacker through the network.

The development of such a defense system is complicated by the fact that sophisticated and targeted cyber attacks, especially those carried out by nation-states, rarely consist solely of an exploitation of a single vulnerability. Rather, these attacks usually consist of a complex sequence of exploits, combining many vulnerabilities across multiple network elements, enabling the intruder to infiltrate deep within the cyber network. In an attempt to address these concerns, researchers in the security community have developed theoretical tools

Manuscript received July 14, 2017; revised December 30, 2017 and February 20, 2018; accepted March 13, 2018. Date of publication March 26, 2018; date of current version May 9, 2018. This work was supported in part by NSF under Grant CNS-1238962 and in part by ARO MURI under Grant W911NF-13-1-0421. A brief summary of a preliminary version of this work (as an extended abstract) can be found in [1]. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Walid Saad. *(Corresponding author: Erik Miehling.)*

E. Miehling is with the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA (e-mail: miehling@illinois.edu).

M. Rasouli is with the Department of Management Science and Engineering, Stanford University, Stanford, CA 94305 USA, and also with the Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: rasoulim@stanford.edu).

D. Teneketzis is with the Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: teneket@umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2018.2819967

1556-6013 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

(predominantly graphical approaches) to model the complex interactions between vulnerabilities. Attack trees/graphs are a popular formalism for modeling such interactions. First introduced by Schneier [7], attack trees model the dependencies between exploits and *system states*[1] in a cyber network, allowing one to construct the specific attack paths that intruders can take to enter a network. Unfortunately, attack trees and graphs can be enormously large even for modestly-sized systems [8], restricting their applicability to realistically-sized cyber networks. In order to improve scalability, researchers proposed an assumption on the attacker's behavior, termed *monotonicity* [9], which states that the success of a previous exploit will not interfere with the success of a future exploit. Monotonicity enables one to restrict attention to dependencies between exploits and *security conditions* (system attributes), in what is termed a *dependency graph*, avoiding the need to enumerate over all system states. This allows one to significantly reduce the amount of information required to describe attacks.

Knowledge of how an attacker can infiltrate a network offers a useful starting point for defining appropriate defenses; however, efficiently processing the available information and translating it into the prescription of an effective defense decision is still a difficult task. One difficulty arises from how to quantify the security status of the network at any given time. The security status is constantly changing as a function of both the attacker's progression through the network and the defender's actions. Furthermore, the defender does not know the true strategy of the attacker and is unable to perfectly observe the attacker's actions, resulting in a lack of certainty of the security status of the network at any given time. The defender only has access to a stream of noisy security information generated in real-time (for example, security alerts generated via intrusion detection systems). Oftentimes, this information suffers from a high-rate of false alarms, that is, alarms being triggered when nothing of concern has actually occurred. Furthermore, the defender's choice of a defense action is complicated by its uncertain effects on the security status of the network (due to the defender's uncertainty regarding the true security status of the system) as well as the need to strike a trade-off between enforcing security and maintaining the availability of network resources to trusted users.

In this paper, we propose a formal model based on the theory of stochastic control, for selecting defense actions in real-time in order to mitigate the progression of an attacker through the network while minimizing the negative impact to availability. We use a dependency graph to model how the attacker progresses through the cyber network over time. We represent the dependency graph as a *hypergraph*, where nodes represent possible security conditions and directed *hyperedges* (edges that connect a pair of sets of nodes) represent exploits, relating *preconditions*, the security conditions that must be true in order for the exploit to be attempted, to *postconditions*, the security conditions that become true if the exploit is successfully carried out. Each security condition can either be enabled or disabled, where an enabled condition is interpreted as the attacker possessing a particular capability.We define a *security state* to be the set of currently enabled security conditions. In this sense, the security state at any given time represents the current capabilities of the attacker. For a given security state, the attacker uses its current capabilities (the set of enabled security conditions) to attempt exploits, with the goal of reaching one or more *goal conditions*. The specific strategy that the attacker employs is its own private information and is assumed to dynamically adjust according to the deployed defense decision. In order to model the defender's uncertainty of the attacker's strategy, we consider the attacker to be one of a finite set of attacker *types*. Consideration of many types allows one to capture a wide range of potential attacker behavior. Each type characterizes the nature of both the security state dynamics (how the attacker progresses through the network, via *probabilities of attack and success* for each exploit) as well as the observation dynamics (the nature of how the intrusion detection system generates security alerts as a function of the attacker's progression, via *probabilities of detection* for exploit attempts and *probabilities of false alarm* for alerts). The defender is able to interfere with the progression of the attacker by performing system modifications that have the effect of blocking exploits from succeeding. The defender possesses uncertainty over both the current capabilities and the true strategy of the attacker and must make its defense decisions based on its *belief matrix*, that is, the joint distribution over security states and attacker types. The belief is constructed such that it is consistent with the defender's available information (the history of security alerts and previously deployed defense actions) and summarizes all of the necessary information for making an optimal decision. Through appropriate assignment of costs to both security states and defense actions, we are able to quantify the tradeoff between maintaining security and preserving availability of the system. The resulting defense problem is a *partially observable Markov decision process* (POMDP), the solution of which is a *defense policy* that maps the current belief (of the security state and attacker strategy) to a defense action.

Due to the high dimensionality of the defense problem, scalability of the solution approach is a primary concern. We employ an online algorithm, based on the *partially observable Monte-Carlo planning* (POMCP) algorithm [10], that simulates future possible state trajectories from the current belief in order to evaluate the effectiveness of various defense decisions, enabling the defender to make a selection in real-time. While forming the basis for our algorithm, the standard POMCP algorithm is not directly suitable for application to our problem. In particular, the belief update procedure does not scale as the observation space grows. As a result, we take advantage of the structure of the observation process, using the context provided by the (belief over the) security state, in order to design an efficient belief update procedure that effectively scales to larger settings. The proposed online defense algorithm enables us to compute effective defense policies for large

[1]System states represent an assignment of values to *system attributes* such as: active services (and the associated vulnerabilities), network connectivity, trust relationships between hosts, and attacker privileges on hosts [8].

instances of the defense problem, overcoming an important obstacle to deployment in realistic cyber network settings.

The consequence of adopting a control-theoretic approach for the network security problem warrants some discussion. Inherent to the control approach is that the attacker's behavior is hard-coded into the model. That said, inclusion of multiple attacker types allows one to embed a wide range of diverse attacker behavior. Allowing the defender to possess uncertainty over the true attacker type captures the defender's inherent uncertainty over the attacker's true strategy. As will be seen in the remainder of the paper, the defender learns the true attacker type from a stream of noisy security alert information and is able to prescribe effective defense actions that prevent the attacker from reaching its goal(s).

A. Literature Review

Systems that select defense actions in response to security alerts are referred to as *intrusion response systems* (IRSs) in the cybersecurity literature. Early IRSs took the form of *passive* systems, logging security information and notifying human operators in order for manual response actions to be selected. Unfortunately, this process is slow and has proven to be inadequate for defending networks against sophisticated modern-day attacks (as stated by Balepin *et al.* [11], "some of the most intense intrusions are automated"). Consequently, researchers have turned to the development of *active* systems that are capable of automatically responding to intrusions without the need for a human operator to intervene. Such systems are referred to as automated IRSs in the literature.

The past two decades have seen an increasing amount of research in automated IRSs. For literature reviews of the area, the reader is directed to the surveys by Foo *et al.* [12], Shameli-Sendi *et al.* [13], and Inayat *et al.* [14]. Automated IRSs can largely be categorized into two groups: *static* and *dynamic*. Static IRSs focus on designing an attack-response map that is capable of executing preprogrammed responses upon detection of attacks (see for example, the work by Ryutov *et al.* [15]). Static approaches, as the name suggests, use a fixed mapping (look-up table) from detected attacks to responses, and consequently, as stated by Lewandowski *et al.* [16], select responses that can be potentially predicted and exploited by an attacker. Furthermore, static IRSs do not take into account the potentially negative side-effects of deploying defense actions and can thus unintentionally inflict further damage to the system. Due to these concerns, researchers began to develop dynamic IRSs. Dynamic IRSs are capable of factoring in additional information, such as the effectiveness of previously deployed defense actions, *e.g.* [17], [18], or the cost of defenses, *e.g.* [19]–[21], in order prescribe a situation-dependent response to mitigate the attack. The ability of dynamic IRSs to modify their response based on new intrusion information raises the bar for the adversary, proving to be much more difficult to circumvent than static IRSs.

One class of dynamic IRSs, termed *state-based* approaches, has received an increasing amount of attention in recent years [16], [22]–[26]. State-based IRSs aim to quantify the security status of a network via the assignment of a security state and enable one to study how this state evolves as a function of both the attacker's and defender's actions. As argued by Iannucci *et al.* [25], and Iannucci and Abdelwahed [26], a state-based approach allows one to cast the problem of designing an automated IRS as a problem of choosing defense actions that ensure the security state remains in a desirable region of the state space. State-based approaches also allow one to avoid the issue of crafting individual response actions for each attack, since a single defense action may modify the dynamics of the security state's evolution in such a way as to prevent many attacks from being successfully carried out. One of the first to develop a state-based IRS was Lewandowski *et al.* [16]. The authors proposed a state-based approach in order to enable "global situational awareness," ensuring that the selection of a defense action benefits the entire system and not just a localized region. While significant in its contribution, the approach taken in [16] does not leverage any formal theory.

The nature of state-based IRSs make them a good fit for the application of formal tools. A well-designed IRS must be able to quickly select defense actions over time when provided with noisy security alert information (including false negatives and false positives) and evaluate the effectiveness of previous defense decisions, all while balancing inherent tradeoffs in the system, such as the conflicting objectives of security and availability. The tools found in control and game theory are well-suited for addressing these requirements, a fact that has been recognized by some in the security community, [22]–[26]. One of the first to apply formal theory, namely stochastic control theory, to the design of an automated IRS was Kreidl and Frazier [22] in the development of their system αLADS (ALPHATECH Lightweight Autonomic Defense System). The authors proposed a host-based IRS that receives alerts (as inputs) from an anomaly sensor in order to calculate the probability that the host is in an attack state. The approach uses a POMDP to select countermeasures in order to interfere with the progression of the attacker while attempting to minimize the negative impact to the normal operation of the system.

Zonouz *et al.* [23] formulate an automated, network-based IRS as a two-player, sequential Stackelberg stochastic game, termed the Response and Recovery Engine (RRE). The proposed scheme decomposes the problem into a hierarchical structure of local engines (hosts) and a global engine. Local engines contain graphs, termed *attack-response trees* (ARTs), that serve to quantify the security of the hosts based on noisy security alerts. The security information of each host is sent to the global engine which is responsible for computing defense actions. The defense actions are chosen using a (heuristic) fuzzy logic control-based technique under the behavioral assumption that the attacker will attempt to inflict maximum damage to the system. In previous work, [24], we developed a defense scheme that used Bayesian attack graphs (see [27] for the definition) to model the progression of the attacker and quantify the security state. Using noisy security alert information, the defender maintains a belief over the current progression of the attacker. The resulting problem of choosing defense actions over time as a function

of the belief is cast as a POMDP.[2] Lastly, Iannucci *et al.*, and Iannucci and Abdelwahed [26], proposed an autonomic IRS that uses a Markov Decision Process (MDP) to specify a sequence of defense actions to drive the system back to a normal operating state. They also offer a performance evaluation of their proposed solution method.

The IRS proposed in our paper differs from existing state-based approaches in multiple ways. First, in the host-based IRS developed by Kreidl and Frazier [22], only the state of the host is taken into consideration when determining the security status of the system. In our paper, embedding a state space on the dependency graph allows for the security of the entire network to be taken into account. Furthermore, due to the coarse-grained, small state space in [22], the scalability problem is not addressed. Second, while the network-based IRS introduced by Zonouz *et al.* [23] addresses the scalability problem via a hierarchical decomposition, our paper presents an alternate approach that addresses scalability by employing a Monte-Carlo sampling approach. Additionally, our paper uses an expected cost criterion, a less conservative objective than the worst-case cost found in [23]. Third, compared to our previous work, our current model is more expressive than the model we proposed in [24], allowing for one to consider more complex dependencies between exploits (the current model allows for exploits that have multiple postconditions), a more realistic observation model (alerts are triggered by exploit activity and are subject to false alarms), and private attacker strategies. Furthermore, we directly address the scalability concerns in this paper. Lastly, while Iannucci *et al.* [25], [26] address the state space explosion problem, their work assumes complete observability of the underlying state, whereas the model of our paper allows for imperfect observations.

B. Main Contributions

The formalism in this paper offers a quantitative model for the computation and analysis of defense policies under a wide range of attacker strategies. The specific contributions are as follows:

1) Quantification of Security: This paper is the first to embed a state space on a dependency graph for the purposes of designing a dynamic IRS. Such an approach allows one to accurately quantify the progression of the attacker along a combinatorial number of attack pathways, and provides valuable information for selecting defense actions that optimally mitigate the attacker's progression while minimizing the impact to availability. Furthermore, allowing the defender to possess uncertainty over the true underlying (dynamic) attack strategy leads to a more realistic model of attacker-defender interactions, permitting a more accurate quantification of the system's security status.

2) Management of False Alarms: The security state provides context for which exploits the attacker has already performed, and which exploits it needs to carry out in order to achieve its goals. Such information is valuable for efficiently processing security alerts, allowing the defender to weigh new security alert information by the likelihood of states in the current belief. That is, the belief is informative for determining whether the given alerts were generated by valid exploit attempts or were simply false alarms. This feature of our model, described in more detail in Section III-C, is particularly useful in settings where there is a high-rate of false alarms, a characteristic of many modern IDSs.

3) Scalability: Even though the number of security states can be very large for some instances of our model, the online defense algorithm (discussed in Section III-A) does not require one to construct the entire state space. Instead, the algorithm only samples regions of the state space relevant to the current defense decision, allowing one to avoid the state space explosion problem. This feature, combined with some problem-specific modifications (taking advantage of the structure of the observation process) allows for computation of defense policies in large domains.

II. The Dynamic Security Model

The proposed dynamic security model provides a formal basis for how a defender can detect and mitigate the infiltration of an attacker in a cyber network. Throughout the description of the model, the diagram of Fig. 1 will be useful. In particular, the remainder of Section II will describe the model for the attacker's progression through the cyber network (Section II-A), the defender's quantification of this progression via a security state (Section II-B), the evolution of the security state as a function of the interactions between the attacker and defender (Section II-C), the defender's information and its formation of consistent beliefs (Section II-D), and finally the formulation of the defender's problem (Section II-F).

A. The Condition Dependency Graph

Researchers and cybersecurity analysts have long been interested in how to represent the steps that intruders take when compromising a system. The concept of attack trees and graphs were developed with this goal in mind, allowing one to study all possible sequences of exploits that an intruder can take to infiltrate a network and reach its goal(s). An attack graph consists of *system states* (nodes) and *transition relations* (edges), which relate system states to each other via exploits. The construction of an attack graph requires one to enumerate over all system states, a process which generates graphs that quickly grow in dimension.

Making assumptions regarding the attacker's behavior allows us to greatly simplify attack graphs and reduce the amount of information required to describe an attack. One such assumption, termed monotonicity [9], states that the success of an exploit does not render the precondition of any other exploit invalid. In simpler terms, the success of one exploit does not interfere with the attacker's ability to carry out a future exploit.[3] Under monotonicity, one does not need to enumerate all system states in an attack graph, but can rather construct a *dependency graph* describing how exploits relate to *security*

[2]More recently, Nguyen *et al.* [28] investigated a multi-stage security game variant of the model presented in [24].

[3]The interested reader is directed to the discussion in [9, Sec. 2] for an explanation of how the majority of non-monotonic attacks can be modeled as monotonic under reasonable assumptions on the attacker's behavior.

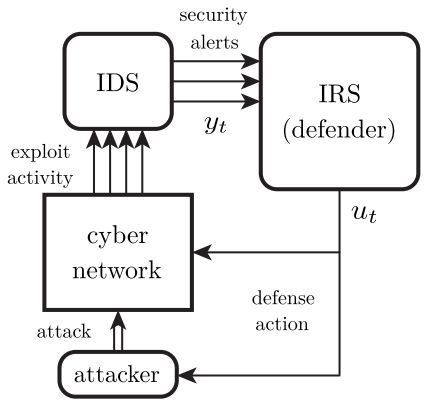

Fig. 1. *The dynamic security model*. The attacker progresses through the network by performing exploits, triggering security alerts via an intrusion detection system. The defender uses this intrusion information to construct a belief of the attacker's capabilities and strategy, which is then used to prescribe a defense action.

conditions [9], [29]. The appeal of the dependency graph representation is that the graph can more easily be constructed for large networks, proving to be especially useful in cases where the corresponding attack graph would be intractably large to generate. In the approach taken by Ammann *et al.* [9], the authors construct such a graph where nodes represent security conditions and edges represent exploits in what is termed a *condition dependency graph*.[4] Security conditions are atomic facts (they can either be true or false) that can reflect any of the aforementioned system attributes.[5] Exploits relate security conditions via *preconditions* and *postconditions*.

We adopt an approach similar to that of Ammann *et al.* [9] for modeling attack pathways, using a condition dependency graph to represent the dependencies between security conditions and exploits. As discussed by Ammann *et al.* [9], the edges in a condition dependency graph relate the security conditions "in a complex way," where a given exploit can have "both multiple preconditions and multiple postconditions." We formalize this notion by recognizing that such edges are in fact directed *hyperedges* (an "edge" that connects two *sets* of nodes rather than simply a pair of nodes). Furthermore, for simplicity, we adopt a slightly modified definition for the security conditions from the one found in [9]. The security conditions in [9] represent a mix of attributes that are true under the normal network configuration (termed *initial conditions*, such as default network connectivity and active services) and attributes that can be maliciously made true during an attack (which we term *attack conditions*, such as attacker privileges or unintended trust relationships between hosts). We do not include the conditions representing the normal network configuration (the initial conditions) explicitly in the dependency graph, but instead assume that the set of security conditions consists solely of attack conditions. This modification is purely for convenience; under the modified definition, the condition dependency graph for a network that has not yet been subject to an attack has all of its conditions set to false.

Formally, we represent a condition dependency graph as a directed acyclic hypergraph $\mathcal{H} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N} = \{c_1, \ldots, c_{n_c}\}$ is the set of security conditions (nodes) and $\mathcal{E} = \{e_1, \ldots, e_{n_e}\}$ is the set of exploits (hyperedges). The acyclic nature of the graph follows from the monotonicity assumption. As discussed earlier, each security condition $c_i \in \mathcal{N}$ in the hypergraph can either be true or false. The truth value of each condition is interpreted as follows: a true (*enabled*) condition means that the attacker possesses condition c_i, and a false (*disabled*) condition means that the attacker does not possess c_i, where an enabled condition is interpreted as the attacker having a particular capability. For example, an enabled condition could mean that the attacker has maliciously enabled a trust relationship between two hosts or has user access on a specific host (where a different privilege level on the same machine would be represented by a distinct condition). Some of the conditions in the hypergraph, when enabled, designate that an attacker has reached a goal. Such nodes are termed *goal conditions* and are denoted by the subset $\mathcal{N}^g \subseteq \mathcal{N}$. Goal conditions are defined by the defender and correspond to something that it wants to protect. For example, a goal condition could represent the attacker possessing root access on a critical host or access to a server that contains sensitive information. It is assumed that the attacker is attempting to enable one of these goal conditions; however, we (as the defender) do not know which one(s). Each hyperedge $e_i \in \mathcal{E}$ represents an exploit and takes the form of an ordered pair of sets, $e_i = (\mathcal{N}_i^-, \mathcal{N}_i^+)$, where $\mathcal{N}_i^- \subseteq \mathcal{N}$ represents e_i's *preconditions* and $\mathcal{N}_i^+ \subseteq \mathcal{N}$ represents e_i's *postconditions*. It is assumed that the attacker is only able to attempt exploit e_i if all preconditions $j \in \mathcal{N}_i^-$ are enabled. This is without loss of generality since for cases where multiple sets of conditions allow for an exploit to be attempted (a disjunction over preconditions), we simply duplicate the exploit for each of its sufficient sets of preconditions. There exist some exploits $e_i \in \mathcal{E}$ with $\mathcal{N}_i^- = \varnothing$, that is, an exploit with an empty set of preconditions. These exploits, termed *initial exploits* and denoted by $\mathcal{E}_0$, represent entry points for the attacker and reflect the fact that they can be performed without the attacker needing any prior (maliciously enabled) capabilities. If an attempted exploit is successful, all postconditions $j \in \mathcal{N}_i^+$ become enabled, increasing the attacker's set of capabilities, allowing it to perform additional exploits and penetrate further into the network. A pictorial representation of a condition dependency graph is provided in Fig. 2. We will use this example graph throughout the paper to aid in the explanation of the model and the results.

We assume that the dependency graph has already been constructed for the network (using vulnerability analysis tools such as the TVA tool of [30]) and instead focus on the formulation and solution of a real-time (dynamic) defense problem, using the dependency graph to describe the progression of the attacker.

[4]This graph has a dual representation termed an *exploit dependency graph* [29], [30].

[5]The important distinction between a system state in an attack graph and a security condition in a dependency graph is that, in attack graphs, each node represents a state, where each state is an assignment of values for *all* of the attributes, whereas in the condition dependency graph, each node represents a single attribute.

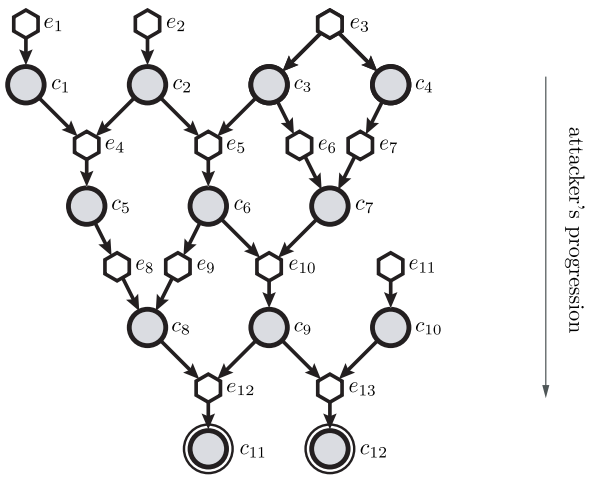

Fig. 2. *A sample condition dependency graph*. The above dependency graph $\mathcal{H} = (\mathcal{N}, \mathcal{E})$ consists of $n_c = 12$ security conditions and $n_e = 13$ exploits (in the form of hyperedges). Initial exploits, $\mathcal{E}_0 = \{e_1, e_2, e_3, e_{11}\}$, where $e_1 = (\varnothing, \{c_1\})$, $e_2 = (\varnothing, \{c_2\})$, $e_3 = (\varnothing, \{c_3, c_4\})$, $e_{11} = (\varnothing, \{c_{10}\})$ and exploits $e_4 = (\{c_1, c_2\}, \{c_5\})$, $e_5 = (\{c_2, c_3\}, \{c_6\})$, $e_6 = (\{c_3\}, \{c_7\})$, $e_7 = (\{c_4\}, \{c_7\})$, $e_8 = (\{c_5\}, \{c_8\})$, $e_9 = (\{c_6\}, \{c_8\})$, $e_{10} = (\{c_6, c_7\}, \{c_9\})$, $e_{12} = (\{c_8, c_9\}, \{c_{11}\})$, $e_{13} = (\{c_9, c_{10}\}, \{c_{12}\})$. We represent the graph in a layered structure in which preconditions are drawn above postconditions, *e.g.* exploit e_4 has preconditions $\{c_1, c_2\}$ and a single postcondition $\{c_5\}$. Goal conditions, $\mathcal{N}^g = \{c_{11}, c_{12}\}$, are represented by double-encircled nodes.

B. The Notion of a Security State

A primary objective of our dynamic security model is to quantify the level of security of the system over time. To this end, we define a *security state* to represent the current level of progression of the attacker in the network. The current security state, denoted by $s_t \subseteq \mathcal{N}$, is defined to be the set of currently enabled security conditions. Since an enabled condition is interpreted as the attacker having a particular capability, the current security state, s_t, describes the set of capabilities of the attacker.

The monotonicity assumption on the attacker's behavior implies a notion of *feasibility* for the security states, defined formally in Definition 1.

Definition 1 (Feasible Security State): A security state, $s \subseteq \mathcal{N}$, is called a *feasible security state* if for every condition $c_j \in s$, there exists at least one exploit $e_i = (\mathcal{N}_i^-, \mathcal{N}_i^+) \in \mathcal{E}$ such that $c_j \in \mathcal{N}_i^+$ and $\mathcal{N}_i^-, \mathcal{N}_i^+ \subseteq s$.

That is, in order for a security state to be feasible, every enabled condition must have been enabled through an exploit and all preconditions and postconditions of the associated exploit must also be enabled. An implicit assumption behind the feasibility condition is that our model for exploits is complete, in the sense that our model is not missing any exploits that would allow the attacker to enable security conditions.[6] Fig. 3 illustrates a few feasible security states for the graph $\mathcal{H} = (\mathcal{N}, \mathcal{E})$ of Fig. 2.

The state space of the dynamic security model consists of all feasible security states, denoted by $\mathcal{S} = \{s_1, \ldots, s_{n_s}\}$. We do not have a closed form expression for the number of feasible states n_s for a given graph; however, as will be discussed in Section III, the proposed online defense algorithm does not require one to construct the entire state space.

[6]Relaxing this assumption amounts to including nodes in s that are not associated with any hyperedge in $\mathcal{E}$, meaning that they can become enabled via an unknown influence. The inclusion of these *leaky nodes* greatly increases the state space and is not considered in this work.

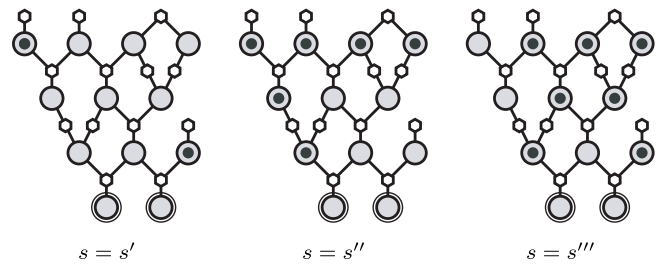

Fig. 3. *A collection of feasible security states*. Enabled security conditions are represented by shaded nodes. Notice that for each feasible security state, there is a path of exploits in $\mathcal{E}$ from enabled postcondition(s) of initial exploits to each enabled condition such that all preconditions and postconditions of the respective exploits are enabled.

C. Evolution of the Security State

The security state evolves probabilistically as a function of both the defender's and attacker's actions. In a given iteration of our problem, the defender is assumed to act first (as described in Section II-C.1), taking actions that interfere with the attacker's progression through the network by dynamically modifying the *attack surface* (the collection of various pathways that the attacker can use to infiltrate the system). As described by the threat model in Section II-C.2, the attacker then uses its set of current capabilities to attempt exploits, the dynamics of which are dictated by its (private) attack strategy. Finally, as shown in Section II-C.3, the attempted exploits that end up succeeding determine the transition to the next security state.

1) Defender's Actions: The defender is assumed to select actions that have the effect of restricting the normal network configuration (such as the network connectivity or active services). Performing such system modifications has the effect of *blocking* the exploits that depend on the network elements that were modified. As a simple example, some exploits depend on the existence of a connection between hosts via a specific port. By blocking this port between the hosts, we are able to block the corresponding exploits that depend on the port being open, preventing the attacker from using these exploits to progress through the network.[7]

In reality, the defender is not able to block individual exploits at will. The system modifications involved in blocking one exploit will, in general, block multiple exploits in the network (*e.g.* blocking a port or disabling a service). On the other hand, some exploits may not be able to be blocked by any of the defender's available system changes (*e.g.* an exploit of a local software vulnerability that results in the escalation of attacker's privilege on a specific host). This coarseness in the ability to block exploits translates into the defender having

[7]One can envision more general countermeasures which modify the graph structure in other ways (*e.g.* by disabling or removing conditions). We do not consider these alternate countermeasures in this work.

limited control over the attacker's progression through the network, a characteristic which is captured in our definition of the defender's set of actions (described below).

Formally, the defender is assumed to have access to $n_u + 1$ defense actions, represented by the set $\mathcal{U} = \{u^0, u^1, \ldots, u^{n_u}\}$. The defense action u^0 represents the null action and corresponds to the defender not blocking any exploits, allowing the network (and attacker) to operate uninterrupted. Each of the n_u remaining defense actions, u^i, $i = 1, \ldots, n_u$, corresponds to a set of system modifications that restrict the normal network configuration, such as restricting the network connectivity (*e.g.* by blocking a port between a pair of hosts) or the set of active services, and can be associated with blocking a specific set of exploits, denoted by $\mathcal{B}(u^i) \subseteq \mathcal{E}$. Notice that, as discussed earlier, the defender does not, in general, have the ability to block individual exploits, instead it must select a defense action $u \in \mathcal{U}$ which in turn induces a set of blocked exploits $\mathcal{B}(u) \subseteq \mathcal{E}$.

Each defense action $u \in \mathcal{U}$ corresponds to system modifications that interfere with the progression of the attacker but also, unavoidably, limit the availability of the system to trusted users. It is the goal of the defense scheme to optimally balance this tradeoff. As will be described in Section II-E, a cost is assigned to each defense action u in order to capture its impact to availability. Combined with the assignment of costs for undesirable security states, the defender is able to specify actions that limit the attacker while minimizing the negative impact to availability.

2) Threat Model: It is assumed that there is a single attacker attempting to infiltrate the system. At any given time-step, the attacker attempts to enable security conditions by performing exploits, in hopes of increasing its set of capabilities and allowing it to progress through the network. The specific nature of the attacker's progression is given by its (private) strategy, dictated by one of a finite set of attacker types. As will be described in the remainder of Section II, the attacker type dictates the statistics of both the security state and observation processes. Lastly, the attacker is assumed to be monotone. As discussed earlier, the monotonicity assumption states that the success of a previous exploit will not interfere with the success of a future exploit. In the context of our model, this implies that once the attacker enables a security condition, it remains enabled.

Formally, for a given security state s_t, the set of exploits that the attacker can attempt, termed the *available exploits*, is described by the set $\mathcal{E}(s_t)$. This set represents the complete set of exploits that are available from state s_t. The attacker does not necessarily know all of the elements in this set, $\mathcal{E}(s_t)$ simply represents what can be attempted using the capabilities described by s_t. The set of available exploits is given by Eq. (1), below.

$$\mathcal{E}(s_t = s) = \left\{ e_i = (\mathcal{N}_i^-, \mathcal{N}_i^+) \in \mathcal{E} \mid \mathcal{N}_i^- \subseteq s, \mathcal{N}_i^+ \not\subseteq s \right\} \quad (1)$$

In order for an exploit $e_i = (\mathcal{N}_i^-, \mathcal{N}_i^+)$ to be available to the attacker, it must satisfy two requirements. The first requirement, $\mathcal{N}_i^- \subseteq s$, states that all of the exploit's preconditions must be satisfied in the current security state. The second requirement, $\mathcal{N}_i^+ \not\subseteq s$, states that the exploit's postconditions must not all be satisfied. This latter requirement arises from the assumption that the attacker will not perform redundant exploits. This assumption is reasonable since the attacker will not gain any new capabilities by performing such exploits and will only increase its chances of being detected (discussed further in Section II-D). The caption of Fig. 4 describes the set of available exploits, for a given security state s_t, in the example condition dependency graph $\mathcal{H}$ of Fig. 2.

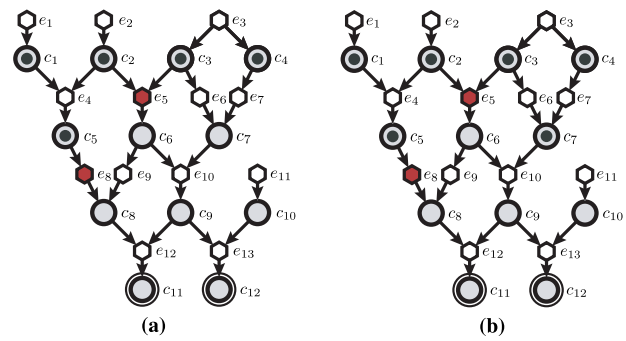

Fig. 4. *Sample evolution of the security state for a given state-type-action triple* (s_t, φ_t, u_t): **(a)** Consider the security state $s_t = \{c_1, c_2, c_3, c_4, c_5\}$ and defense action $u_t = u$ such that $\mathcal{B}(u) = \{e_5, e_8\}$ (blocked exploits are shown by red/shaded hyperedges). By Eq. (1), the set of available exploits is $\mathcal{E}(s_t) = \{e_5, e_6, e_7, e_8, e_{11}\}$; **(b)** The attacker attempts each exploit $e_j \in \mathcal{E}$ with a probability of attack $\alpha_{e_j}(\varphi_t, s_t, u_t)$ as described in Eq. (2). Each attempted exploit e_j that does not lie within the set of blocked exploits, $\mathcal{B}(u_t) \subseteq \mathcal{E}$, succeeds with a probability of success $\beta_{e_j}(\varphi_i)$ as described by Eq. (3). In this example, only exploit e_6 succeeded and thus $s_{t+1} = s_t \cup \mathcal{N}_6^+ = s_t \cup \{c_7\}$.

The specific strategy that the attacker employs is dictated by its type. The attacker is assumed to be one of n_a types, represented by the set $\Phi = (\varphi_1, \ldots, \varphi_{n_a})$. Each type $\varphi_i \in \Phi$ corresponds to a set of *conditional attack probabilities* over the exploits, $\alpha(\varphi_i, s_t, u_t) = \big(\alpha_{e_1}(\varphi_i, s_t, u_t), \ldots, \alpha_{e_{n_e}}(\varphi_i, s_t, u_t)\big)$, specifying the likelihood that the attacker will attempt each exploit from the current security state s_t under defense action u_t. The conditional attack probability for a given exploit $e_j \in \mathcal{E}$ is given by

$$\alpha_{e_j}(\varphi_i, s_t, u_t) = \begin{cases} \overline{\alpha}_{e_j}(\varphi_i) & \text{if } e_j \in \mathcal{E}(s_t) \setminus \mathcal{B}(u_t) \\ \underline{\alpha}_{e_j}(\varphi_i) & \text{if } e_j \in \mathcal{E}(s_t) \cap \mathcal{B}(u_t) \\ 0 & \text{if } e_j \notin \mathcal{E}(s_t). \end{cases} \quad (2)$$

By partitioning the set of available exploits into two components, the threat model describes how an attacker may modify its strategy based on the defender's action. Specifically, available exploits that are not blocked by the current defense action, $\mathcal{E}(s_t) \setminus \mathcal{B}(u_t)$, are attempted with probability $\overline{\alpha}_{e_j}(\varphi_i)$, whereas exploits that are blocked by the current defense action, $\mathcal{E}(s_t) \cap \mathcal{B}(u_t)$, are attempted with probability $\underline{\alpha}_{e_j}(\varphi_i)$. Exploits that are not available in the current security state are not attempted.

Constructing the threat model in such a way allows one to encode various levels of attacker knowledge. For example, if an attacker of type φ_i is not able to recognize that exploit e_j is blocked under a given defense action $u_t = u$, then $\overline{\alpha}_{e_j}(\varphi_i) = \underline{\alpha}_{e_j}(\varphi_i)$, reflecting the fact that the attacker is unable to modify its attack probability based on the defender's action. On the other hand, if the attacker knows with certainty

that exploit e_j has been blocked by the defender, then setting $\underline{\alpha}_{e_j}(\varphi_i) = 0$ reflects that the attacker would not attempt it. The threat model can also capture intermediate cases where the attacker has partial information and may attempt exploits that it believes are not blocked with a higher probability, *i.e.* $\overline{\alpha}_{e_j}(\varphi_i) \geq \underline{\alpha}_{e_j}(\varphi_i)$.

3) Security State Dynamics: For any given iteration, the defender first chooses a defense action, $u_t = u \in \mathcal{U}$, in turn blocking a set of exploits, $\mathcal{B}(u) \subseteq \mathcal{E}$. Next, given the current security state $s_t \in \mathcal{S}$ and defense action $u_t = u$, the attacker attempts a collection of available exploits according to its own private strategy, $\alpha(\varphi_i, s_t, u_t)$. Each of the attempted exploits succeeds with a *conditional probability of success*. The probability of success models the fact that attacks do not succeed with certainty (potentially due to the inherent difficulty in carrying out the attack or the existence of network defenses already in place). The probabilities are assumed to depend upon the attacker's type; this dependency arises from the fact that some attackers may possess greater knowledge of the exploit or be able to expend more resources. The set of conditional success probabilities is given by $\beta(\varphi_i, u_t) = \big(\beta_{e_1}(\varphi_i, u_t), \ldots, \beta_{e_{n_e}}(\varphi_i, u_t)\big)$, where the probability of success for a given exploit e_j is given by

$$\beta_{e_j}(\varphi_i, u_t) = \begin{cases} \beta_{e_j}(\varphi_i) & \text{if } e_j \notin \mathcal{B}(u_t) \\ 0 & \text{if } e_j \in \mathcal{B}(u_t) \end{cases} \tag{3}$$

Exploits that are blocked by the defender do not succeed. The exploit attempts that are successful enable the corresponding set of postconditions, forming the updated security state $s_{t+1} \in \mathcal{S}$. Fig. 4(b) illustrates one possible successor state s_{t+1} for a given state s_t, type φ_t, and defense action u_t.

In summary, the security state dynamics can be described by a controlled Markov chain, where the control is the defense action. The transition matrix of the Markov chain, for a given defense action u and attacker type φ_l, is $P^u(\varphi_l)$ with elements $p^u_{ijl} = P(S_{t+1} = s_j \mid S_t = s_i, \Phi_t = \varphi_l, U_t = u)$ (the analytical expression for this probability is given by Eq. (A.1) in the appendix). Note that defense actions only influence the attacker's progression. Blocking an exploit that already has all of its postconditions enabled does not disable any of the exploit's postconditions. An analogy to the physical security domain is useful: Consider an intruder attempting to break into a building to access a safe. If the intruder has already successfully broken through the front entrance, then barricading the entrance will have no effect on the attacker's ability to access the safe. However, securing the entrance before the attacker reaches it will prevent the attacker from using that entry point, forcing it to use another path, in turn increasing the attacker's effort and decreasing the likelihood of the safe being compromised.

Remark 1: The proposed security model also allows for the underlying type to vary in time according to a Markov chain (with transition matrix Q); however, for simplicity we consider a fixed, albeit unknown, underlying attacker type.

D. The Defender's Information

The defender lacks certainty of both the current security state and the underlying strategy of the attacker and must infer/learn both from a stream of noisy security information. The security information comes in the form of a sequence of *security alerts* generated by an intrusion detection system (IDS) as the attacker attempts exploits and progresses through the network (see Fig. 1). These security alerts are noisy, suffering from both missed detections (the IDS not seeing an exploit attempt) and false alarms (the IDS generating alerts when no attempt has occurred, *e.g.* alerts generated by legitimate network activity).

Let $\mathcal{Z} = \{z_1, z_2, \ldots, z_{n_z}\}$ represent the finite set of security alerts that may be generated by the IDS. Each exploit $e_i \in \mathcal{E}$, if attempted, has an associated set of alerts that can be generated, given by the set $\mathcal{Z}(e_i) = \{z_{\mathcal{A}_i(1)}, z_{\mathcal{A}_i(2)}, \ldots, z_{\mathcal{A}_i(a_i)}\} \in \mathscr{P}(\mathcal{Z})$, where $\mathcal{A}_i$ is the set of a_i alert indices from the set $\mathcal{A} = \{1, 2, \ldots, n_z\}$ and $\mathscr{P}(\mathcal{Z})$ is the power set of $\mathcal{Z}$. In general, more than one exploit can generate the same alert, that is $\mathcal{Z}(e_i) \cap \mathcal{Z}(e_j) \neq \varnothing$ for $e_i \neq e_j$. Also, some exploits may not generate any alerts, that is, $\mathcal{Z}(e_i) = \varnothing$ for some $e_i \in \mathcal{E}$ (such exploits are termed *stealthy*).

The IDS generates the security alerts probabilistically, based on detected exploit activity and false alarms, the statistics of which depend upon the underlying strategy of the attacker. Advanced attackers may be able to craft their attacks such that they are less likely to trigger security alerts or, alternatively, influence the false alarm rate of specific alerts to mask their true progression through the network. To capture this dependency, the security model allows for the *probabilities of detection* (the likelihood of seeing an alert given an exploit attempt) and the *probabilities of false alarm* (the likelihood of seeing an alert in the absence of an exploit attempt) to depend on the attacker's type. For an attacker of type φ_l, an attempt of exploit e_i will generate the alerts $\mathcal{Z}(e_i) = \{z_{\mathcal{A}_i(1)}, z_{\mathcal{A}_i(2)}, \ldots, z_{\mathcal{A}_i(a_i)}\}$ with corresponding probabilities of detection $\delta_{ij}(\varphi_l)$, $j \in \mathcal{A}_i$. Similarly, the probability of false alarm for each alert $z_i \in \mathcal{Z}$, under type φ_l, is dictated by $\zeta_i(\varphi_l)$. The vector of security alerts received by the defender at time $t+1$, denoted by $y_{t+1} \in \mathcal{Y} = \{0, 1\}^{n_z}$, consists of all security alerts triggered during the given iteration.

Using the received security alerts the defender constructs a belief, denoted by π_t, that summarizes its uncertainty over both the security state and the attacker type. This belief (also termed an *information state* [31], [32]) is constructed using all of the defender's available information at time t: the (distribution over the) initial security state and attacker type, the history of all defense actions from time 0 to time $t-1$, and all observations (security alerts) from time 0 to t, denoted by $h_t = (\pi_0, u_0, y_0, \ldots, u_{t-1}, y_t)$. The belief represents the joint probability distribution over security states and attacker types, and takes the form of a matrix, defined as

$$\pi_t = \begin{bmatrix} \pi_t^{1,1} & \pi_t^{1,2} & \cdots & \pi_t^{1,n_a} \\ \pi_t^{2,1} & \pi_t^{2,2} & \cdots & \pi_t^{2,n_a} \\ \vdots & \vdots & \ddots & \vdots \\ \pi_t^{n_s,1} & \pi_t^{n_s,2} & \cdots & \pi_t^{n_s,n_a} \end{bmatrix} \in \Delta(\mathcal{S} \times \Phi)$$

where $\pi_t^{il} = P(S_t = s_i, \Phi_t = \varphi_l \mid H_t = h_t)$ is the likelihood that s_i is the true security state and φ_l is the true type given the realized information h_t. The space $\Delta(\mathcal{S} \times \Phi)$ is the probability simplex over the state-type space $\mathcal{S} \times \Phi$. Notice that π_t is a doubly-stochastic matrix for each t; each row represents a probability mass function over the type space for a given state and each column represents a probability mass function over the space of security states for a given type.

The defender maintains the belief matrix over time, updating it as new information, consisting of the current defense action u_t and new observation y_{t+1}, is revealed. For a given defense action $u_t = u$ and observation $y_{t+1} = y_k$, the belief matrix update is defined as $\pi_{t+1} = [\mathcal{T}_{jm}(\pi_t, y_k, u)]_{\varphi_m \in \Phi, s_j \in \mathcal{S}}$ where the (j, m)'th update function, $\mathcal{T}_{jm}(\pi_t, y_k, u) = P(S_{t+1} = s_j, \Phi_{t+1} = \varphi_m \mid U_t = u, Y_{t+1} = y_k, \Pi_t = \pi_t)$, is given by

$$\pi_{t+1}^{jm} = \mathcal{T}_{jm}(\pi_t, y_k, u) = \frac{p_{jm}^u(\pi_t) r_{jk}^u(\pi_t)}{\sigma(\pi_t, y_k, u)}. \tag{4}$$

The above terms are defined as follows

$$\begin{aligned} p_{jm}^u(\pi_t) &= P(S_{t+1} = s_j, \Phi_{t+1} = \varphi_m \mid U_t = u, \Pi_t = \pi_t) \\ &= \sum_{s_i \in \mathcal{S}, \varphi_l \in \Phi} \pi_t^{il} p_{ijl}^u q_{lm} \end{aligned} \tag{5}$$

$$\begin{aligned} r_{jk}^u(\pi_t) &= P(Y_{t+1} = y_k \mid S_{t+1} = s_j, U_t = u, \Pi_t = \pi_t) \\ &= \sum_{s_i \in \mathcal{S}, \varphi_l \in \Phi} \pi_t^{il} r_{ijkl}^u \end{aligned} \tag{6}$$

$$\begin{aligned} \sigma(\pi_t, y_k, u) &= P(Y_{t+1} = y_k \mid U_t = u, \Pi_t = \pi_t) \\ &= \sum_{s_j \in \mathcal{S}, \varphi_m \in \Phi} r_{jk}^u(\pi_t) p_{jm}^u(\pi_t) \end{aligned} \tag{7}$$

where p_{ijl}^u is the probability of transitioning from state s_i to s_j under defense action u and attacker type φ_l, q_{lm} is the probability of transitioning between types (note that we assume that $q_{lm} = 1$ if $l = m$, zero otherwise, as mentioned in Remark 1), and $r_{ijkl}^u = P(Y_{t+1} = y_k \mid S_{t+1} = s_j, S_t = s_i, \Phi_t = \varphi_l, U_t = u)$ is the probability that the IDS generates observation vector y_k given a transition from state s_i to s_j under attacker type φ_l and defense action u. Analytical expressions for p_{ijl}^u and r_{ijkl}^u can be found in Eqs. (A.1) and (A.2), respectively, of the appendix.

The belief at any given time represents the *defender's view of the attacker's current capabilities and true strategy*. The trajectory of beliefs, $(\pi_0, \pi_1, \pi_2, \ldots)$, describes how this view changes over time. As evidenced by Eq. (4), the trajectory of beliefs (given an initial belief) is defined by the sequence of defense actions and observations (security alerts). Since security alerts are triggered probabilistically by exploit attempts and background events, the presence of an alert does not necessarily mean that the attacker is progressing through the network. That is, an exploit attempt may have triggered an alert but may not have succeeded, or an alert may have been triggered via a false alarm. Similarly, the absence of an alert may mean that an exploit was in fact attempted (and successful), but didn't trigger an alert (due to a missed detection or a stealthy exploit). Since the current belief π_t assigns mass to security states (and attacker types) that are consistent with the defender's available information, the belief trajectory may assign mass to worsening security states even in the cases where the underlying security state is unchanging or no alerts are generated. This characteristic highlights the importance of information in our model, reflecting that the defender's imperfect observations of the security state and attacker type contribute to a more pessimistic view of the system's security over time.

E. Assignment of Costs

In many systems, the cyber network needs to remain (at least partially) operational while subject to an attack. The defender thus has two objectives: 1) maintaining the availability of the system, and 2) keeping the attacker away from goal conditions. These two factors are largely in opposition of each other. If the defender were only concerned with maintaining the availability of the network, it would not perform any system modifications, leaving the system to run uninterrupted and in turn not interfering with the progression of the attacker, allowing it to reach its goal conditions undisturbed. On the other hand, if the defender were just concerned with preventing the attacker from reaching goal conditions, it would immediately execute aggressive system changes in order to block as many exploits as possible and maximally disrupt the attacker's progression through the network. Unfortunately, this latter option is clearly very costly to the availability of the network. It is evident that one must strike a trade-off between these two extremes.

In order to quantify this trade-off, we construct a cost function that takes into account both the quality of the current security state and the negative impact to availability of each defense action. Specifically, we assign a *security cost*, $c_s : \mathcal{S} \times \Phi \to \mathbb{R}$, to capture the cost of the system being in various security states $s \in \mathcal{S}$ under different attacker types, as well as an *availability cost*, $c_u : \mathcal{U} \to \mathbb{R}$, for each defense action that is deployed. Using the definition of a goal condition at the end of Section II-A, we define the notion of a goal state.

Definition 2 (Goal state): A *goal state* is defined as a security state $s \in \mathcal{S}$ that contains one or more goal conditions, that is, there exists some $j \in s$ such that $j \in \mathcal{N}^g$.

We denote the space of all goal states by $\mathcal{S}^g \subseteq \mathcal{S}$. Goal states are undesirable from the perspective of the defender and are thus assigned a higher cost than non-goal states, that is, $0 \leq c_s(s', \varphi) \leq c_s(s'', \varphi) < \infty$ for $s' \notin \mathcal{S}^g$, $s'' \in \mathcal{S}^g$, $\varphi \in \Phi$. Although not a requirement, we can impose the additional property that for any two security states $s', s'' \in \mathcal{S}$ where $s' \subseteq s''$, we have $c(s', \varphi) \leq c(s'', \varphi)$, reflecting the fact that if the attacker has enabled more conditions, it should be more costly for the defender.

To model the availability factor, we assign an availability cost for each defense action, denoted by $c_u(u')$. Recall that each defense action $u' \in \mathcal{U}$ is a collection of system modifications. Some combinations of system modifications may have little to no impact on availability while other combinations may render important elements of the underlying network unavailable. The assignment of the costs $c_u(u')$, for each $u' \in \mathcal{U}$, allows one to incorporate such information (combinations of system modifications that severely impact

availability should be assigned a very high cost). We assume that $0 \leq c_u(u') < \infty$ for every $u' \in \mathcal{U}$. The cost for taking defense action u_t in security state s_t under attacker type φ_t is defined as

$$c(s_t, \varphi_t, u_t) = wc_s(s_t, \varphi_t) + (1 - w)c_u(u_t) \tag{8}$$

where $0 \leq w \leq 1$ is a weighting term that allows the defender to specify which factor is more important, where $w = 0$ ($w = 1$) corresponds to only being concerned with availability (resp. security).

F. Defender's Problem

The defender wishes to determine an optimal defense action to deploy for any belief that it may encounter. The decision rule determining this action is termed a *defense policy* and is represented by the function $\gamma : \Delta(\mathcal{S} \times \Phi) \to \mathcal{U}$, mapping a belief matrix $\pi \in \Delta(\mathcal{S} \times \Phi)$ to a defense action $u \in \mathcal{U}$. The problem of determining γ can be cast as a POMDP, represented by problem (9) below.

$$\min_{\gamma \in \Gamma} \ \mathbb{E}^{\gamma} \left\{ \sum_{t=0}^{\infty} \rho^t c(\Pi_t, U_t) \mid \Pi_0 = \pi_0 \right\} \tag{P}$$

$$\text{subject to} \quad U_t = \gamma(\Pi_t) \tag{P-1}$$

$$\Pi_{t+1} = \mathcal{T}(\Pi_t, Y_{t+1}, U_t) \tag{P-2}$$

where Γ is the space of admissible defense policies and $0 < \rho < 1$ is the discount factor. The function $c(\pi_t, u_t)$ represents the expected cost for being in belief state $\Pi_t = \pi_t$ when defense action $U_t = u_t$ is selected and is defined as $c(\pi_t, u_t) = \sum_{s_i \in \mathcal{S}, \varphi_l \in \Phi} \pi_t^{il} c(s_i, \varphi_l, u_t)$ where $c(s, \varphi, u)$ is the state-type-action cost function defined in Eq. (8). The current action U_t must be generated according to the defense policy γ, as demonstrated by constraint (P-1), and the next belief Π_{t+1} must obey the update $\mathcal{T}(\Pi_t, Y_{t+1}, U_t)$, constraint (P-2).

The solution to problem (9) is an optimal defense policy, denoted by $\gamma^* \in \Gamma$, which specifies an optimal defense action for every possible belief $\pi \in \Delta(\mathcal{S} \times \Phi)$ that the defender can possess. Following the optimal policy results in the minimum expected discounted cost over the infinite time-horizon. In other words, taking into account all uncertainty in the problem, the defense policy γ^* generates actions that achieve the desired tradeoff as dictated by the cost function in Eq. (8).

III. Computation of Defense Policies

While the embedding of a state space on the dependency graph allows for one to accurately quantify the level of progression of the attacker, the high dimensionality of the resulting defense problem leads to significant scalability concerns. One approach to solving the defense problem is to adopt an *offline* POMDP solver. Such solvers aim to explicitly solve the problem by computing the optimal action for every belief that can be encountered, prior to runtime. In spite of the fact that significant improvements have been made in the efficiency of offline solvers in recent years, *e.g.* [33], the requirement to specify an action for every possible encountered scenario often leads to an intractable problem. *Online* solvers represent an alternate paradigm in which one only considers the possible future scenarios from the current belief, constructing a local policy during runtime. Online methods interleave the computation and execution (runtime) phases of a policy [34], yielding a much more scalable approach than offline methods, making them a more natural fit for obtaining a solution to the defense problem.

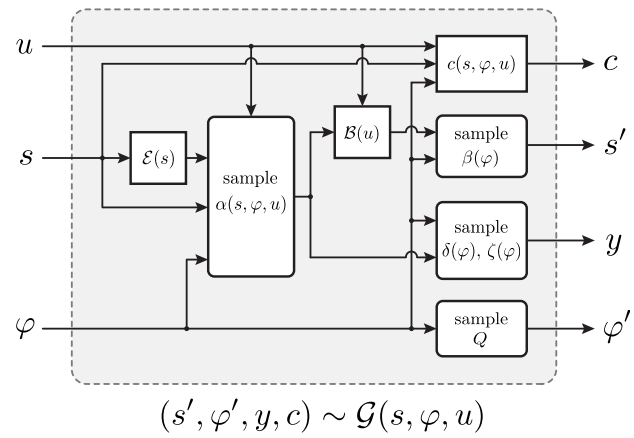

Fig. 5. *The generative model for the dynamic security model.* For a given state-type-action triple (s, φ, u), the generative model first determines the set of available exploits, $\mathcal{E}(s)$, then, taking into account the effect of the defender's action and attacker type, samples the probability distributions of the problem (probabilities of attack and success for exploits, and probabilities of detection and false alarm for observations) in order to generate a next state $s_{t+1} = s'$, updated type $\varphi_{t+1} = \varphi'$, an observation $y_{t+1} = y$, and a cost c.

The proposed algorithm for computing defense policies, which we term the *online defense algorithm*, is based on an existing online solver developed by Silver and Veness [10], termed the Partially Observable Monte-Carlo Planning (POMCP) algorithm. While no existing algorithm is immediately applicable to our problem, the POMCP algorithm is the best fit, requiring the fewest modifications to achieve efficient computation of defense policies.

A. The Online Defense Algorithm

The online defense algorithm is a heuristic search algorithm for determining defense actions in real-time as the attacker progresses through the network and security alerts are generated. The algorithm consists of two main stages: an *action selection* step and a *belief update* step. The action selection step of the online defense algorithm is similar to that of the standard POMCP algorithm (details pertaining to the specific operation of POMCP, as well as pseudocode for the algorithm, can be found in [10]). The belief update step has been modified by taking advantage of the structure of the observation process, enabling efficient updating in large-scale domains.

The action selection stage of the online defense algorithm operates by performing Monte-Carlo simulations from the current belief in order to estimate the quality of various defense actions. Each simulation consists of a call to a *generative model*, shown in Fig. 5. Specifically, a simulation begins by sampling a state-type pair, (s, φ), from the current belief matrix (approximated by a finite collection of state-type

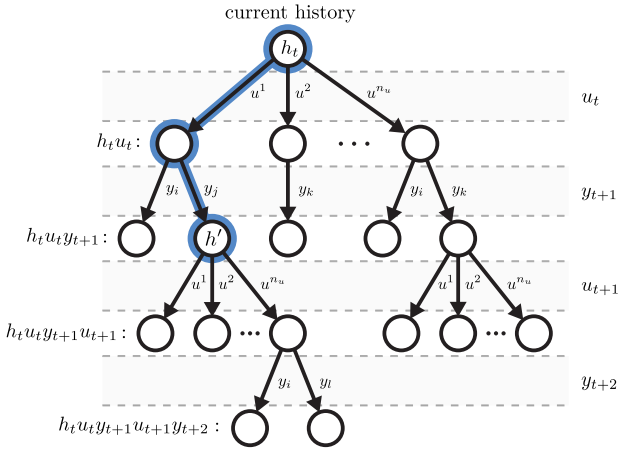

Fig. 6. *A search tree of histories.* The root node represents the current history from which simulations begin. Each descendant of the root node represents a possible future history. After a real-world action is taken, *e.g.* $u_r = u^1$, and a real-world observation is received, *e.g.* $y_r = y_j$, the history is updated to h' (represented by the blue/shaded path).

pairs, described in more detail in the following paragraph), and coupled with a given defense action u, generates a successor state s' and type φ', as well as an observation y and cost c, $(s', \varphi', y, c) \sim \mathcal{G}(s, \varphi, u)$.

Through successive sampling from the current belief and calls to the generative model, a search tree of histories is constructed, as shown in Fig. 6. Due to the partial observability of the underlying process, the search tree consists of nodes representing histories, where branches of the tree originating from the current history represent future possible histories. Each branch begins with the selection of a defense action, which is selectively sampled using a multi-armed bandit rule, termed UCB1 [35], in order to optimally balance exploitation (selecting presumably promising actions in order to decrease their estimation error) and exploration (checking other actions in order to rule out better alternatives). The estimation error associated with each defense action's quality decreases as the number of simulations (and the size of the search tree) grows. The online defense algorithm continues to perform simulations for the given history node, progressively expanding the tree, until a stopping condition is met (*e.g.* when a maximum number of simulations, n_{sim}, has been reached). The defense action that has the lowest estimated cost is then taken, termed the real-world action, denoted by u_r, and a real-world observation, denoted by y_r, is recorded. The relevant branches of the search tree are identified, the remaining tree is pruned, and a new root node is specified as the current history.

Once an updated history h' is realized, the defender's belief must be updated. Due to the computational complexity associated with updating the belief matrix analytically (see Eq. (4) and the appendix), the defender maintains a belief approximation, denoted by B_t, consisting of n_k state-type pairs, termed particles. The update of the belief approximation under the standard POMCP algorithm involves making multiple calls to the generative model in order to obtain samples (s', y) until y exactly matches the real-world observation y_r, at which point s' is accepted into the updated belief set B_{t+1} (repeating until n_k particles have been added). Instances of the security model with large observation spaces lead to scenarios where the sampled observation rarely matches the real-world observation, preventing the belief from being updated in reasonable time (or at all). To address this issue, we propose a modified belief update that takes advantage of the structure of the observations, that is, how security alerts are generated as a function of the security state and type. Instead of checking if the sample observation matches the real-world observation for every alert $z_i \in \mathcal{Z}$, the proposed belief update only checks if the alerts agree over a security state dependent subset of elements $z_i \in \mathcal{Z}(s) = \cup_{e \in \mathcal{E}(s)} \mathcal{Z}(e)$ and probabilistically accepts the particle if this modified condition is satisfied. The set $\mathcal{Z}(s)$ represents the set of alerts that can be generated by exploit attempts; alerts not in $\mathcal{Z}(s)$, *i.e.* any alert in $\bar{\mathcal{Z}}(s) = \mathcal{Z} \setminus \mathcal{Z}(s)$, cannot be generated by the attempt of any exploit available in state s, as dictated by Eq. (1). The rationale for restricting the comparison to the elements $\mathcal{Z}(s)$ is due to the fact that these are the only alerts that are informative for a change in the underlying state. The remaining alerts $\bar{\mathcal{Z}}(s)$ must have been triggered by false alarms under the current state s. Observations that pass the modified test are accepted into the updated belief with a probability of acceptance that depends on the security state and attacker type, $p_a(s, \varphi)$. The probability of acceptance is dictated by the likelihood that the state-type pair (s, φ) could have generated the real-world observation. It is defined as $p_a(s, \varphi) = \bar{p}_a(s, \varphi)/d$ where

$$\bar{p}_a(s, \varphi) = \Bigg(\prod_{i \in \mathcal{I}(y_r^{\bar{\mathcal{Z}}(s)} = 1)} \zeta_i(\varphi) \Bigg) \Bigg(\prod_{i \in \mathcal{I}(y_r^{\bar{\mathcal{Z}}(s)} = 0)} (1 - \zeta_i(\varphi)) \Bigg)$$

and $d = \max_{(s,\varphi) \in B_t} \bar{p}_a(s, \varphi)$ is a normalization term. The set $\mathcal{I}(y_r^{\bar{\mathcal{Z}}(s)} = 1)$ represents the indices of the alerts $z_i \in \bar{\mathcal{Z}}(s)$ where $y_r^i = 1$ is true (analogously for $\mathcal{I}(y_r^{\bar{\mathcal{Z}}(s)} = 0)$). The normalized probability of acceptance, $p_a(s, \varphi)$, ensures that particles are accepted into the updated belief more frequently than the standard POMCP belief update while ensuring that the relative mass under the modified belief procedure agrees with what would be achieved under the standard belief update. The pseudocode for the modified belief update is given in Algorithm 1 below.

Algorithm 1 -Modified Belief Update

```
Initialize: n_k, B_{t+1} = ∅, numAdded = 0;
procedure ModifiedBeliefUpdate(B_t, u_r, y_r)
    while numAdded < n_k do
        (s, φ) ~ B_t
        (s', φ', y, −) ~ G(s, φ, u_r)
        if y^{Z(s)} = y_r^{Z(s)} then          ▷ If alerts Z(s) match
            B_{t+1} ← B_{t+1} ∪ {s', φ'} with probability p_a(s, φ)
            numAdded ← numAdded + 1
        end if
    end while
end procedure
```

In addition to the modified belief update procedure, a heuristic cost assignment can further improve the scalability of the online defense algorithm. A key bottleneck for tree-based

heuristic search algorithms in large-scale domains is the rate at which the search tree grows as a function of the depth from the root node, termed the *branching factor*. Problem instances with many actions and observations result in search trees with large branching factors, preventing the search algorithm from being able to search beyond a small depth, resulting in a poor quality, myopic policy. To help avoid this, we can assign non-zero costs to security states that are *close* to goal states. A simple procedure for such a cost assignment is to assign higher costs to states that require fewer successful exploits to reach a goal state. Such a heuristic cost assignment makes simulations more informative, decreasing the required search depth (and simulations) and resulting in more effective defense policies.

Using the modified belief update procedure and the heuristic cost assignment, we were able to effectively scale the online defense algorithm to large instances of our dynamic security model. Defense policies were successfully computed for a problem on a graph consisting of 134 conditions (nodes), 143 exploits (hyperedges), 64 defense actions, and 30 security alerts (resulting in over 10^9 possible observation vectors). The resulting number of security states exceeded 100 million.

B. An Illustrative Example

We now investigate an illustrative example of the defense problem using the sample dependency graph of Fig. 2. We assume $n_a = 3$ attacker types of varying aggression (described by their conditional probabilities of attack and success, dictating the rate of movement through the network), knowledge (described by the separation between $\overline{\alpha}_{e_j}(\varphi_i)$ and $\underline{\alpha}_{e_j}(\varphi_i)$ terms in Eq. (2)), and stealthiness (described by the probabilities of detection and false alarm). Specifically, the three attack types $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$ capture the following behavior.

	aggression	knowledge	stealthiness
φ_1	low	low	low
φ_2	moderate	high	high
φ_3	high	moderate	moderate

The problem parameters that capture the above behavior are now defined. Probabilities of attack for each exploit under each attacker type $\varphi_i \in \Phi$ are

$$\left(\overline{\alpha}_{e_j}(\varphi_1), \underline{\alpha}_{e_j}(\varphi_1)\right) = (0.5, 0.5) \text{ for all } e_j \in \mathcal{E}_0$$
$$\left(\overline{\alpha}_{e_j}(\varphi_1), \underline{\alpha}_{e_j}(\varphi_1)\right) = (0.3, 0.3) \text{ for all } e_j \in \mathcal{E} \setminus \mathcal{E}_0$$

$$\left(\overline{\alpha}_{e_j}(\varphi_2), \underline{\alpha}_{e_j}(\varphi_2)\right) = (0.8, 0.1) \text{ for } e_j \in \mathcal{E}_0$$
$$\left(\overline{\alpha}_{e_j}(\varphi_2), \underline{\alpha}_{e_j}(\varphi_2)\right) = (0.7, 0.3) \text{ for } e_j \in \{e_4, e_5, e_{10}, e_{12}, e_{13}\}$$
$$\left(\overline{\alpha}_{e_j}(\varphi_2), \underline{\alpha}_{e_j}(\varphi_2)\right) = (0.6, 0.4) \text{ for } e_j \in \{e_6, e_7, e_8, e_9\}$$

$$\left(\overline{\alpha}_{e_j}(\varphi_3), \underline{\alpha}_{e_j}(\varphi_3)\right) = (0.7, 0.4) \text{ for } e_j \in \mathcal{E}_0$$
$$\left(\overline{\alpha}_{e_j}(\varphi_3), \underline{\alpha}_{e_j}(\varphi_3)\right) = (0.6, 0.4) \text{ for } e_j \in \{e_4, e_5, e_{10}, e_{12}, e_{13}\}$$
$$\left(\overline{\alpha}_{e_j}(\varphi_3), \underline{\alpha}_{e_j}(\varphi_3)\right) = (0.6, 0.5) \text{ for } e_j \in \{e_6, e_7, e_8, e_9\}.$$

Notice the separation between $\overline{\alpha}_{e_j}(\varphi_i)$ and $\underline{\alpha}_{e_j}(\varphi_i)$ for attacker types φ_2 and φ_3, reflecting a higher level of assumed knowledge than type φ_1. Similarly, probabilities of success are

$$\beta_{e_j}(\varphi_1) = \begin{cases} 0.5 & \text{for } e_j \in \mathcal{E}_0 \\ 0.4 & \text{for } e_j \in \mathcal{E} \setminus \mathcal{E}_0 \end{cases}$$
$$\beta_{e_j}(\varphi_2) = \begin{cases} 0.6 & \text{for } e_j \in \mathcal{E}_0 \\ 0.5 & \text{for } e_j \in \mathcal{E} \setminus \mathcal{E}_0 \end{cases}$$
$$\beta_{e_j}(\varphi_3) = \begin{cases} 0.7 & \text{for } e_j \in \mathcal{E}_0 \\ 0.6 & \text{for } e_j \in \mathcal{E} \setminus \mathcal{E}_0. \end{cases}$$

TABLE I

TABLE OF PROBABILITIES OF DETECTION FOR EACH TYPE

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}
z_1	0.8	0.1	0	0	0	0	0	0	0	0	0.4	0	0
	0.3	0.4	0	0	0	0	0	0	0	0	0.3	0	0
	0.5	0.6	0	0	0	0	0	0	0	0	0.3	0	0
z_2	0	0.6	0.8	0	0	0	0	0	0	0	0.6	0	0
	0	0	0.4	0	0	0	0	0	0	0	0.2	0	0
	0	0.4	0.5	0	0	0	0	0	0	0	0.5	0	0
z_3	0	0	0	0.5	0	0.6	0.1	0	0	0	0	0	0
	0	0	0	0.4	0	0.4	0	0	0	0	0	0	0
	0	0	0	0.5	0	0.5	0.4	0	0	0	0	0	0
z_4	0	0	0	0	0.7	0	0.7	0	0	0	0	0	0
	0	0	0	0	0.3	0	0.5	0	0	0	0	0	0
	0	0	0	0	0.4	0	0.6	0	0	0	0	0	0
z_5	0	0	0	0	0	0	0	0.7	0.6	0	0	0	0
	0	0	0	0	0	0	0	0.4	0.3	0	0	0	0
	0	0	0	0	0	0	0	0.5	0.4	0	0	0	0
z_6	0	0	0	0	0	0	0	0	0.4	0.7	0	0	0
	0	0	0	0	0	0	0	0	0.2	0.5	0	0	0
	0	0	0	0	0	0	0	0	0.3	0.6	0	0	0
z_7	0	0	0	0	0	0	0	0	0	0	0	0.7	0
	0	0	0	0	0	0	0	0	0	0	0	0.4	0
	0	0	0	0	0	0	0	0	0	0	0	0.6	0
z_8	0	0	0	0	0	0	0	0	0	0	0	0	0.8
	0	0	0	0	0	0	0	0	0	0	0	0	0.4
	0	0	0	0	0	0	0	0	0	0	0	0	0.6

Probabilities of detection are provided in Table I. Columns represent attempted exploits wheres rows represent the triggered alert. Each entry represents the probability of detection, for a given exploit e_i (column) and alert z_j (row), for each type (from top to bottom), $\delta_{ij}(\varphi_1)$, $\delta_{ij}(\varphi_2)$, and $\delta_{ij}(\varphi_3)$. Notice that increased stealthiness is modeled by a lower probability of detection. Lastly, the probability of false alarm for each alert z_i under each type is $\zeta_i(\varphi_1) = 0.4$, $\zeta_i(\varphi_2) = 0.5$, and $\zeta_i(\varphi_3) = 0.6$. The space of defense actions is constructed as the powerset of a set of binary defense actions, that is $\mathcal{U} = \mathscr{P}(\{u^1, u^2, u^3, u^4\})$, resulting in a total of $|\mathcal{U}| = 2^4 = 16$ defense actions. Each binary defense action induces a set of blocked exploits, defined as $\mathcal{B}(u^1) = \{e_1, e_2, e_3, e_4\}$, $\mathcal{B}(u^2) = \{e_5, e_7, e_{11}\}$, $\mathcal{B}(u^3) = \{e_8, e_9, e_{10}\}$, and $\mathcal{B}(u^4) = \{e_{12}, e_{13}\}$. The set of exploits that a defense action $u_t \in \mathcal{U}$ blocks is equal to the union of the blocked exploits of the binary defense actions that it contains, that is, $\mathcal{B}(u_t) = \bigcup_{u^i \in u_t} \mathcal{B}(u^i)$. Security states are assigned a cost of 1 for each goal condition, $\mathcal{N}^g = \{c_{11}, c_{12}\}$, that is contained in the state. The cost of each binary defense action is $c_u(u^i) = 0.25$, for all $i \in \{1, 2, 3, 4\}$. The cost weight in Eq. (8) is set to $\omega = 0.5$ and the discount factor is $\rho = 0.95$. There are $n_s = 186$ security states (computed offline) and $n_z = 8$ security alerts leading to $|\mathcal{Y}| = 2^8 = 256$ distinct observation vectors. All simulations for the example use $n_k = 1200$ particles to approximate the belief. The problem is assumed to start from the empty (safe) security state $s_0 = \varnothing$. The defender is initially completely

uncertain of the true attacker type, reflected by a uniform belief over all attacker types. A sample evolution of the defense problem is illustrated in Fig. 7.

The computed defense policy is intuitive. Initially, in order to save on availability costs, the defense policy does not block any exploits. During this period of inaction, the defender's belief gradually assigns mass to worsening security states based on the received security alerts. The belief over the true attacker type (represented by the panel to the left of each graph in Fig. 7) is also updated as a function of the received alerts. Eventually the defense policy begins to deploy defense actions, blocking exploits that it believes are available to the attacker, as dictated by Eq. (1). The defense actions serve two purposes. First, the actions slow down the progression of the attacker through the network in the event that any of the blocked exploits are attempted. Second, the actions (along with the received observations) help the defender to gather information, serving to reduce its uncertainty of the true security state and attacker type. In order to lessen the negative impact to availability, the defense policy may prescribe the null action in some time-steps, as seen in $t = 17, 18$. In these cases, the defender will briefly wait for the attacker to progress before blocking exploits further downstream (as discussed at the end of Section II-C3, only blocking not yet successful exploits will impede the attacker's progression). This idle behavior only occurs in the early stages of the attack when the attacker is believed to be far from reaching a goal condition. When the defender's belief reflects that the attacker is close to reaching a goal condition,[8] the defense policy has no option but to block the exploit(s) that would allow the attacker to reach its goal(s), *e.g.* exploits e_{12} and e_{13} in time-steps $t = 40 - 42$ of Fig. 7. This defense action is persistent, resulting in the corresponding exploits being blocked for all subsequent time-steps. In summary, the defense policy initially behaves passively, placing priority on preserving availability, only deploying defense actions to slow the attacker and gain information. As the defender becomes more certain of the security state over time, it identifies and persistently blocks the exploits that would allow the attacker to reach a goal condition.

The defender's belief over the true attacker type exhibits more uncertainty than its belief over the security state. This is expected since the observed security alerts are more informative for the current progression of the attacker, *i.e.* the security state, than they are for inferring the true attacker type. In other words, the observed security alerts are largely consistent with the range of attacker behavior specified by the type space Φ. Nevertheless, the defender eventually becomes confident of the true attacker type (see time-steps $t = 40 - 42$ in Fig. 7), and even under the lack of complete certainty, is able to prescribe defense decisions that prevent the attacker from reaching the goal conditions.

The performance of the online defense algorithm improves as the number of simulation iterations n_{sim} increases, as shown by the plots in Fig. 8. For low simulation counts, *e.g.* $n_{\text{sim}} = 500$, the defense policy makes selections based on poor quality estimates of the actions. This causes the defense policy to be overly aggressive initially, prescribing to block

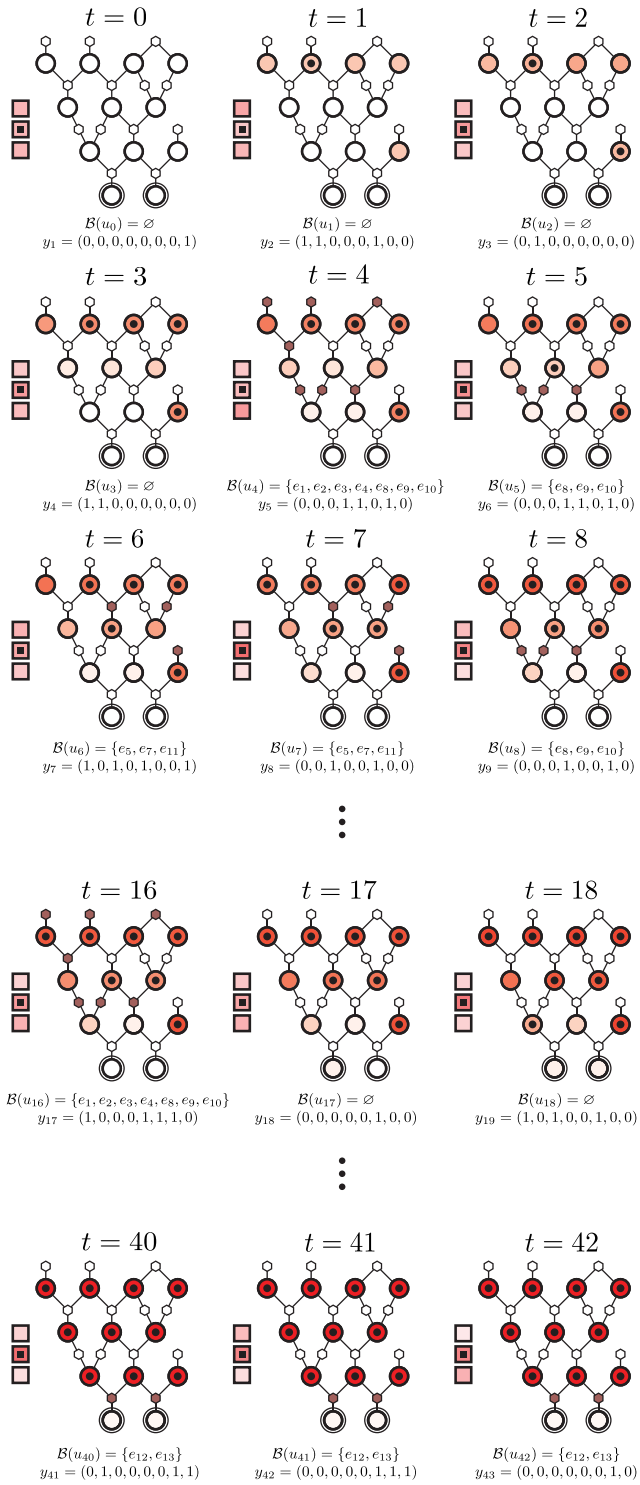

Fig. 7. *Sample evolution of the defense problem*. The current (true) security state s_t is represented by the tagged nodes. The true attacker type is represented by the tagged node in the panel to the left of each graph; the true type for the above simulation is φ_2. The defender's (marginal) probability for both the security state and true type is represented as a *heat-map* (representing values via shades), computed from the current belief state π_t, where a darker shade represents a higher probability. Blocked exploits $\mathcal{B}(u_t)$, represented by red/shaded hyperedges, and the observation vector y_{t+1} are displayed beneath each graph. The above sample evolution was performed using $n_{\text{sim}} = 5000$ simulations.

[8]It has been observed that for sample paths where the attacker has gained many conditions in a short amount of time, the defense policy is more aggressive, prescribing many (exploratory) actions until the defender's uncertainty is reduced.

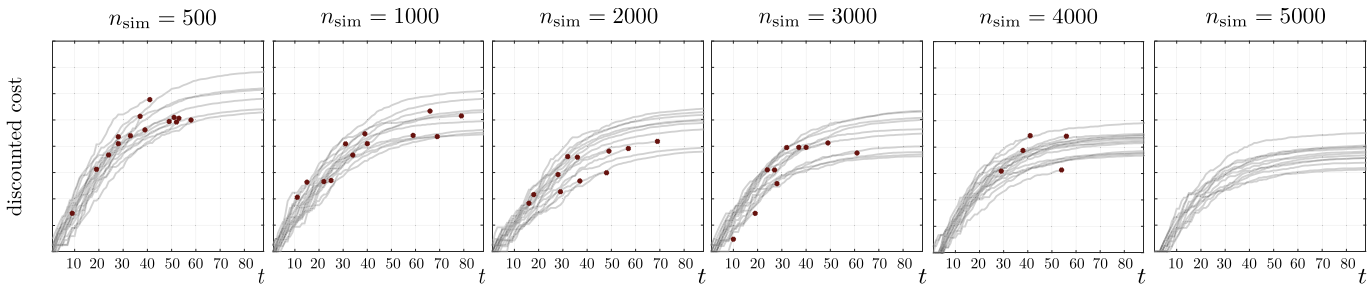

Fig. 8. *Discounted costs of sample paths for various simulation counts.* The behavior of the defense policy (for the example system of Fig. 2) is demonstrated for simulation counts $n_{\text{sim}} = 500, 1000, 2000, 3000, 4000, 5000$. The simulations for each value of n_{sim} are initiated by randomly assigning a true attacker type uniformly from $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$. For each value of n_{sim}, the discounted cost is plotted (versus the time-step) for 20 sample paths. Trajectories that terminate in a marker represent sample paths where the attacker reached a goal state.

exploits from time-step $t = 0$ and unnecessarily restricting availability. Furthermore, due to the poor quality estimates, the resulting defense policy also allows the attacker to reach a goal condition in many of the sample runs. As the number of simulations increases, more possible future histories are taken into account, resulting in higher quality estimates of actions and a better performing defense policy (as evidenced by the remaining plots in Fig. 8). The number of times that the attacker reaches a goal state decreases as the number of simulations increases. For $n_{\text{sim}} = 5000$, the attacker failed to reach any goal in all of the 20 sample runs.

C. A Remark on the Processing of Security Alerts

A particularly desirable feature of our state-based dynamic security model is in regard to how security alerts are processed, specifically false alarms. Existing approaches, such as the CSM (cooperating security managers) system of White *et al.* [36] or the EMERALD (event monitoring enabling response to anomalous live disturbances) system developed by Porras and Neumann [37], attempt to deal with false alarms by defining *metrics* that reflect both the severity of an attack and the confidence that it is a real intrusion. In the presence of a high rate of false alarms, these metric-based approaches can incorrectly classify benign security alerts as real intrusions. The state-based approach of our model avoids this drawback. Since the security state precisely describes which exploits are available to the attacker, via the set $\mathcal{E}(s)$ in Eq. (1), the defender is able to use the likelihood of the individual security states in its belief to weigh new security information. To see this, consider the following example: Consider belief matrices $\pi, \pi' \in \Delta(\mathcal{S} \times \Phi)$ such that for some security state $s_i \in \mathcal{S}$ and type $\varphi_l \in \Phi$, $\pi^{il} > 0$ and $\pi'^{il} = 0$. Let there be a single available exploit in state s_i, that is, $\mathcal{E}(s_i) = \{e\}$ and assume that if exploit e is attempted, it generates the unique alert z (that is, no other exploit attempt can trigger alert z). If the defender possesses belief π and sees alert z, then the belief update allows for the possibility of the alert being generated by an attempt of exploit e. On the other hand, if π' is the current belief and alert z is received, the defender can say with certainty that the alert was a false alarm. In general, the likelihood of the individual security states in the current belief influence how security alert information is processed. In our simulations, we have observed that even in situations where the false-alarm rate is high, the defender is able to accurately track the true security state over time.

IV. Conclusion & Future Work

The complex nature of sophisticated cyber attacks necessitates the development of a defense system that is capable of prescribing defense actions in real-time that both mitigate the attack and preserve availability, all while enabling a solution that scales to realistically-sized cyber networks. Furthermore, the defense scheme must be able to operate under uncertainty of the attacker's strategy, the inherently noisy security alert information generated by the intrusion detection system, as well as be capable of using the evaluated effectiveness of previously deployed defense actions to influence future defense decisions. The state-based model introduced in this paper addresses all of the above mentioned concerns. Specifically, using ideas from stochastic control theory, we can precisely model the evolution of the system's security status and formulate how the defender can use its imperfect information to specify optimal defense actions. Scalability is achieved via a sample-based, online defense algorithm that takes advantage of the structure of the security model to enable computation in large-scale domains.

The proposed dynamic security model provides a formal description of the evolution of multi-stage attacks and the information that is generated as they unfold, and thus provides a basis for the formulation of many interesting problems in the network security domain. For example, one research direction concerns modeling situations in which the defender may possess uncertainty over which vulnerabilities (exploits) the attacker knows. The defender's uncertainty can be quantified as uncertainty over the structure of the true dependency graph. Defense of the cyber network would then involve not only the prescription of defense actions, but also learning the structure of the graph. Another research direction concerns modeling situations in which the attacker is deceptive. That is, the attacker may perform actions so as to mislead the defender into thinking it is attempting to reach a goal condition that is distinct from its true goal. Addressing these problems involves studying the proposed model in the context of more complex behavioral models that are able to describe such interactions (*e.g.* dynamic games with asymmetric information [38]).

APPENDIX
DEFENDER'S BELIEF STATE UPDATE

Recall the belief update of Eq. (4)

$$\pi_{t+1}^{jm} = \mathcal{T}_{jm}(\pi_t, y_k, u) = \frac{p_{jm}^u(\pi_t) r_{jk}^u(\pi_t)}{\sigma(\pi_t, y_k, u)}$$

where $p_{jm}^u(\pi_t) = \sum_{s_i \in \mathcal{S}, \varphi_l \in \Phi} \pi_t^{il} p_{ijl}^u q_{lm}$, $r_{jk}^u(\pi_t) = \sum_{s_i \in \mathcal{S}, \varphi_l \in \Phi} \pi_t^{il} r_{ijkl}^u$, and $\sigma(\pi_t, y_k, u) = \sum_{s_j \in \mathcal{S}, \varphi_m \in \Phi} r_{jk}^u(\pi_t) p_{jm}^u(\pi_t)$ as dictated by Eqs. (5), (6), and (7), respectively.

To define the transition probability, p_{ijl}^u, consider the set of *transition events*, denoted by $\mathcal{F}(s_i, s_j, \varphi_l, u)$, denoting the set of exploit events that could have caused the transition between s_i and s_j under type φ_l and action u. Each event in $\upsilon \in \mathcal{F}(s_i, s_j, \varphi_l, u)$ is a binary assignment (successful, not successful) to each of the available exploits not blocked by the current defense action, $\mathcal{E}(s_i) \setminus \mathcal{B}(u)$. Since events in $\mathcal{F}(s_i, s_j, \varphi_l, u)$ are disjoint, the transition probability is

$$p_{ijl}^u = \sum_{\upsilon \in \mathcal{F}(s_i, s_j, \varphi_l, u)} \left(\prod_{e_m \in \upsilon_1} \alpha_{e_m}(\varphi_l) \beta_{e_m}(\varphi_l) \cdot \prod_{e_m \in \upsilon_0} \left(1 - \alpha_{e_m}(\varphi_l) \beta_{e_m}(\varphi_l)\right) \right) \quad \text{(A.1)}$$

where the set υ_1 (resp. υ_0) denotes the collection of exploits in υ that must succeed (resp. must not succeed).

The observation probability r_{ijkl}^u is now defined. Let E_t represent the set of exploits attempted from state S_t. Then,

$$\begin{aligned} r_{ijkl}^u &= P(Y_{t+1} = y_k \mid S_{t+1} = s_j, S_t = s_i, \Phi_t = \varphi_l, U_t = u) \\ &= \sum_{\mathcal{E}_a \in \mathscr{P}(\mathcal{E}(s_i))} P(Y_{t+1} = y_m \mid E_t = \mathcal{E}_a, S_{t+1} = s_j, S_t = s_i, \\ &\qquad \Phi_t = \varphi_l, U_t = u) \\ &\quad \cdot P(E_t = \mathcal{E}_a \mid S_{t+1} = s_j, S_t = s_i, \\ &\qquad \Phi_t = \varphi_l, U_t = u) \\ &= \sum_{\mathcal{E}_a \in \mathscr{P}(\mathcal{E}(s_i))} P(Y_{t+1} = y_m \mid E_t = \mathcal{E}_a, \Phi_t = \varphi_l) \\ &\quad \cdot P(E_t = \mathcal{E}_a \mid S_{t+1} = s_j, S_t = s_i, \\ &\qquad \Phi_t = \varphi_l, U_t = u) \end{aligned} \quad \text{(A.2)}$$

where we have used the fact that the event $\{Y_{t+1} = y_m\}$ is independent of the event $\{S_{t+1} = s_j, S_t = s_i, U_t = u\}$ given the exploit attempt event $\{E_t = \mathcal{E}_a\}$. The probability of seeing a given observation vector given a set of exploit attempts, $P(Y_{t+1} = y_m \mid E_t = \mathcal{E}_a, \Phi_t = \varphi_l)$, is defined as

$$\begin{aligned} P(Y_{t+1} &= y_m \mid E_t = \mathcal{E}_a, \Phi_t = \varphi_l) \\ &= \prod_{j \in \mathcal{A}} P(Y_{t+1}^j = y_m^j \mid E_t = \mathcal{E}_a, \Phi_t = \varphi_l) \end{aligned}$$

where separability of the above terms follows from the fact that the elements of the observation vector are conditionally independent given the exploit attempt. Defining $\mathcal{E}(z_j)$ as the set of exploits that can trigger alert z_j, that is, $\mathcal{E}(z_j) = \{e_i \in \mathcal{E} \mid z_j \in \mathcal{Z}(e_i)\}$, each term in the above product is

$$P(Y_{t+1}^j = y_m^j \mid E_t = \mathcal{E}_a) = \begin{cases} \left(1 - \zeta_j(\varphi_l)\right) \prod_{e_i \in \mathcal{E}_a \cap \mathcal{E}(z_j)} \left(1 - \delta_{ij}(\varphi_l)\right) & \text{if } y_m^j = 0 \\ 1 - \left(1 - \zeta_j(\varphi_l)\right) \prod_{e_i \in \mathcal{E}_a \cap \mathcal{E}(z_j)} \left(1 - \delta_{ij}(\varphi_l)\right) & \text{if } y_m^j = 1 \end{cases}$$

The probability of exploit attempts given a transition from s_i to s_j under type φ_l and action u, $P(E_t = \mathcal{E}_a \mid S_{t+1} = s_j, S_t = s_i, \Phi_t = \varphi_l, U_t = u)$, is

$$\begin{aligned} &P(E_t = \mathcal{E}_a \mid S_{t+1} = s_j, S_t = s_i, \Phi_t = \varphi_l, U_t = u) \\ &= \frac{\begin{aligned} &P(S_{t+1} = s_j \mid E_t = \mathcal{E}_a, S_t = s_i, \Phi_t = \varphi_l, U_t = u) \\ &\qquad \cdot P(E_t = \mathcal{E}_a \mid S_t = s_i, \Phi_t = \varphi_l) \end{aligned}}{P(S_{t+1} = s_j \mid S_t = s_i, \Phi_t = \varphi_l, U_t = u)}. \end{aligned}$$

To define the probability $P(S_{t+1} = s_j \mid E_t = \mathcal{E}_a, S_t = s_i, \Phi_t = \varphi_l, U_t = u)$, let $\mathcal{F}(s_i, s_j, \varphi_l, u, \mathcal{E}_a)$ denote the set of attempted exploit events that could have resulted in a transition to state s_j given that exploits $\mathcal{E}_a$ were attempted in state s_i under type φ_l and action u. Each event $\upsilon \in \mathcal{F}(s_i, s_j, \varphi_l, u, \mathcal{E}_a)$ is a binary assignment (successful, not successful) to each of the available exploits that were attempted and not currently blocked, $(\mathcal{E}_a \cap \mathcal{E}(s_i)) \setminus \mathcal{B}(u)$. The probability is then

$$\begin{aligned} &P(S_{t+1} = s_j \mid E_t = \mathcal{E}_a, S_t = s_i, \Phi_t = \varphi_l, U_t = u) \\ &\qquad = \sum_{\upsilon \in \mathcal{F}(s_i, s_j, \varphi_l, u, \mathcal{E}_a)} \prod_{e_l \in \upsilon_1} \beta_{e_l}(\varphi_l) \prod_{e_l \in \upsilon_0} \left(1 - \beta_{e_l}(\varphi_l)\right). \end{aligned}$$

The probability of exploit attempts $\mathcal{E}_a$ given the current security state s_i, $P(E_t = \mathcal{E}_a \mid S_t = s_i, \Phi_t = \varphi_l)$, is

$$\begin{aligned} &P(E_t = \mathcal{E}_a \mid S_t = s_i, \Phi_t = \varphi_l) \\ &\qquad = \prod_{e_l \in \mathcal{E}_a \cap \mathcal{E}(s_i)} \alpha_{e_l}(\varphi_l) \prod_{e_l \in \mathcal{E}(s_i) \setminus \mathcal{E}_a} \left(1 - \alpha_{e_l}(\varphi_l)\right) \end{aligned}$$

and $P(S_{t+1} = s_j \mid S_t = s_i, \Phi_t = \varphi_l, U_t = u)$ is the transition probability given by p_{ijl}^u.

ACKNOWLEDGEMENT

The authors are grateful to the reviewers for their insightful comments that greatly improved this manuscript. They also thank Michael Wellman, Sushil Jajodia, and Sridhar Venkatesan for their valuable feedback, as well as Mason Wright for tips regarding the implementation of the online defense algorithm.

REFERENCES

[1] E. Miehling, M. Rasouli, and D. Teneketzis, "A dependency graph formalism for the dynamic defense of cyber networks," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2017, pp. 511–512.

[2] J. Finkle and D. Skariachan. *Target Cyber Breach Hits 40 Million Payment Cards at Holiday Peak*. Accessed: Sep. 9, 2016. [Online]. Available: http://www.reuters.com/article/us-target-breach-IdUSBRE9BH1GX20131219

[3] A. Greenberg. *Hackers Remotely Kill a Jeep on the Highway—With me in it*. Accessed: Dec. 15, 2016. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

[4] D. Etherington and K. Conger. *Large DDoS Attacks Cause Outages at Twitter, Spotify, and Other Sites*. Accessed: Nov. 20, 2016. [Online]. Available: https://techcrunch.com/2016/10/21/many-sites-including-twitter-and-spotifysuffering-outage/

[5] Department of Homeland Security. *Industrial Control Systems Cyber Emergency Response Team (ICS-CERT)*. Accessed: Mar. 30, 2016. [Online]. Available: https://ics-cert.us-cert.gov/

[6] B. Gorenc and F. Sands. *Hacker Machine Interface: The State of SCADA HMI Vulnerabilities*. Accessed: May 1, 2017. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/the-state-ofscada-hmi-vulnerabilities

[7] B. Schneier, "Attack trees," *Dr. Dobb's J.*, vol. 24, no. 12, pp. 21–29, 1999.

[8] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proc. IEEE Symp. Secur. Privacy*, May 2002, pp. 273–284.

[9] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proc. 9th ACM Conf. Comput. Commun. Secur.*, 2002, pp. 217–224.

[10] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2164–2172.

[11] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using specification-based intrusion detection for automated response," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2003, pp. 136–154.

[12] B. Foo, M. W. Glause, G. M. Howard, Y.-S. Wu, S. Bagchi, and E. H. Spafford, "Intrusion response systems: A survey," in *Information Assurance: Dependability and Security in Networked Systems*. Burlington, MA, USA: Morgan Kaufmann, 2008, ch. 13, pp. 377–416.

[13] A. Shameli-Sendi, N. Ezzati-Jivan, M. Jabbarifar, and M. Dagenais, "Intrusion response systems: Survey and taxonomy," *Int. J. Comput. Sci. Netw. Secur.*, vol. 12, no. 1, pp. 1–14, 2012.

[14] Z. Inayat, A. Gani, N. B. Anuar, M. K. Khan, and S. Anwar, "Intrusion response systems: Foundations, design, and challenges," *J. Netw. Comput. Appl.*, vol. 62, pp. 53–74, Feb. 2016.

[15] T. Ryutov, C. Neuman, K. Dongho, and Z. Li, "Integrated access control and intrusion detection for Web servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 9, pp. 841–850, Sep. 2003.

[16] S. M. Lewandowski, D. J. Van Hook, G. C. O'Leary, J. W. Haines, and L. M. Rossey, "SARA: Survivable autonomic response architecture," in *Proc. IEEE DARPA Inf. Survivability Conf. Expo. II*, vol. 1. Jun. 2001, pp. 77–88.

[17] D. J. Ragsdale, C. A. Carver, J. W. Humphries, and U. W. Pooch, "Adaptation techniques for intrusion detection and intrusion response systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, vol. 4. Oct. 2000, pp. 2344–2349.

[18] 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 *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, Jun./Jul. 2005, pp. 508–517.

[19] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok, "Toward cost-sensitive modeling for intrusion detection and response," *J. Comput. Secur.*, vol. 10, nos. 1–2, pp. 5–22, 2002.

[20] T. Toth and C. Kruegel, "Evaluating the impact of automated intrusion response mechanisms," in *Proc. IEEE 18th Annu. Comput. Secur. Appl. Conf.*, Dec. 2002, pp. 301–310.

[21] N. Kheir, N. Cuppens-Boulahia, F. Cuppens, and H. Debar, "A service dependency model for cost-sensitive intrusion response," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2010, pp. 626–642.

[22] O. P. Kreidl and T. M. Frazier, "Feedback control applied to survivability: A host-based autonomic defense system," *IEEE Trans. Rel.*, vol. 53, no. 1, pp. 148–166, Mar. 2004.

[23] S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley, "RRE: A game-theoretic intrusion response and recovery engine," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 395–406, Feb. 2014.

[24] E. Miehling, M. Rasouli, and D. Teneketzis, "Optimal defense policies for partially observable spreading processes on Bayesian attack graphs," in *Proc. 2nd ACM Workshop Moving Target Defense*, 2015, pp. 67–76.

[25] S. Iannucci, Q. Chen, and S. Abdelwahed, "High-performance intrusion response planning on many-core architectures," in *Proc. IEEE 25th Int. Conf. Comput. Commun. Netw.*, Aug. 2016, pp. 1–6.

[26] S. Iannucci and S. Abdelwahed, "A probabilistic approach to autonomic security management," in *Proc. 13th IEEE Int. Conf. Autonomic Comput.*, Jul. 2016, pp. 157–166.

[27] Y. Liu and H. Man, "Network vulnerability assessment using Bayesian networks," *Proc. SPIE*, vol. 5812, pp. 61–71, Mar. 2005. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/5812/0000/Network-vulnerability-assessment-using-Bayesian-networks/10.1117/12.604240.full?SSO=1

[28] T. H. Nguyen, M. Wright, M. P. Wellman, and S. Baveja, "Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis," in *Proc. ACM Workshop Moving Target Defense*, 2017, pp. 87–97.

[29] S. Noel and S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," in *Proc. ACM Workshop Vis. Data Mining Comput. Secur.*, 2004, pp. 109–118.

[30] S. Jajodia, S. Noel, and B. O'Berry, "Topological analysis of network attack vulnerability," in *Managing Cyber Threats: Issues, Approaches, and Challenges*. Boston, MA, USA: Springer, 2005, ch. 9, pp. 247–266.

[31] K. J. Åström, "Optimal control of Markov processes with incomplete state information," *J. Math. Anal. Appl.*, vol. 10, no. 1, pp. 174–205, 1965.

[32] P. R. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1986.

[33] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proc. Robot. Sci. Syst. IV*, 2008, pp. 65–72.

[34] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, "Online planning algorithms for POMDPs," *J. Artif. Intell. Res.*, vol. 32, pp. 663–704, 2008.

[35] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.

[36] G. B. White, E. A. Fisch, and U. W. Pooch, "Cooperating security managers: A peer-based intrusion detection system," *IEEE Netw.*, vol. 10, no. 1, pp. 20–23, Jan. 1996.

[37] P. A. Porras and P. G. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proc. 20th Nat. Inf. Syst. Secur. Conf.*, 1997, pp. 353–365.

[38] Y. Ouyang, H. Tavafoghi, and D. Teneketzis, "Dynamic games with asymmetric information: Common information based perfect Bayesian equilibria and sequential decomposition," *IEEE Trans. Autom. Control*, vol. 62, no. 1, pp. 222–237, Jan. 2017.

Erik Miehling received the B.A.Sc. and the M.A.Sc. degrees in electrical engineering from The University of British Columbia, Vancouver, BC, Canada, in 2009 and 2011, respectively, and the Ph.D. degree in electrical engineering systems from the University of Michigan, Ann Arbor, MI, USA, in 2017.

He is currently a Post-Doctoral Research Associate with the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign. His research interests concern stochastic control, decision/game theory, graph theory, and decentralized optimization with applications to the security of large-scale computer networks and critical infrastructure systems.

Mohammad Rasouli received the B.Sc. and M.Sc. degrees in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2009 and 2011, respectively, and the M.Sc. degree in economics and the Ph.D. degree in electrical engineering systems from the University of Michigan, in 2014 and 2017, respectively. He is currently a Post-Doctoral Scholar with the Management Science and Engineering Department and the Civil and Environmental Engineering Department with Stanford University. His research interests include mechanism design and stochastic control with applications in energy and security systems.

Demosthenis Teneketzis received the Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently a Professor of electrical engineering and computer science with the University of Michigan, Ann Arbor, MI, USA. His research interests are in stochastic control, decentralized systems, queuing and communication networks, stochastic scheduling and resource allocation problems, mathematical economics, and discrete-event systems.