IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received February 19, 2021, accepted March 13, 2021, date of publication March 26, 2021, date of current version April 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3069105

Hidden Markov Model and Cyber Deception for the Prevention of Adversarial Lateral Movement

MD ALI REZA AL AMIN[1], SACHIN SHETTY[1], (Senior Member, IEEE), LAURENT NJILLA[2], DEEPAK K. TOSH[3], AND CHARLES KAMHOUA[4], (Senior Member, IEEE)

[1]Computational Modeling and Simulation Engineering, Old Dominion University, Norfolk, VA 23508, USA
[2]Air Force Research Laboratory, Rome, NY 13441, USA
[3]Department of Computer Science, The University of Texas at El Paso, El Paso, TX 79968, USA
[4]Army Research Laboratory, Adelphi, MD 20783, USA

Corresponding author: Md Ali Reza Al Amin (malam002@odu.edu)

This work was supported by the Office of the Assistant Secretary of Defense for Research and Engineering [OASD (R & E)] under Grant FA8750-15-2-0120.

ABSTRACT Advanced persistent threats (APTs) have emerged as multi-stage attacks that have targeted nation-states and their associated entities, including private and corporate sectors. Cyber deception has emerged as a defense approach to secure our cyber infrastructure from APTs. Practical deployment of cyber deception relies on defenders' ability to place decoy nodes along the APT path optimally. This paper presents a cyber deception approach focused on predicting the most likely sequence of attack paths and deploying decoy nodes along the predicted path. Our proposed approach combines reactive (graph analysis) and proactive (cyber deception technology) defense to thwart the adversaries' lateral movement. The proposed approach is realized through two phases. The first phase predicts the most likely attack path based on Intrusion Detection System (IDS) alerts and network trace, and the second phase is determining optimal deployment of decoy nodes along the predicted path. We employ transition probabilities in a Hidden Markov Model to predict the path. In the second phase, we utilize the predicted attack path to deploy decoy nodes. However, it is likely that the attacker will not follow that predicted path to move laterally. To address this challenge, we employ a Partially Observable Monte-Carlo Planning (POMCP) framework. POMCP helps the defender assess several defense actions to block the attacker when it deviates from the predicted path. The evaluation results show that our approach can predict the most likely attack paths and thwarts the adversarial lateral movement.

INDEX TERMS Cyber deception, cyber defense, cyber decoy, lateral movement, intrusion detection system, hidden Markov Model, attack path prediction.

I. INTRODUCTION

Given the growing spate of cyber attacks, it is very imperative to design resilient cyber infrastructure. Organizations face substantial financial losses and challenges in maintaining core public services due to the increasing cyber attacks rate. According to McAfee's recent report in 2018, [1], cybercrime has reached nearly $600 billion. Adversaries have lately resorted to using Advanced Persistent Threats (APT) to conduct cybercrime. APT allows attackers to stay undetected in the network for long periods and steal organizations' data without being caught. In an APT attack, the attackers use social engineering, spear-phishing email, or vulnerability exploitation to gain the network's initial entry. After the network's initial entry, they maintain a low footprint and slowly gain their foothold by compromising one host to another within the organization's network. Lateral movement is the most critical step in the APT attack to maintain the presence in the network. Early detection of adversarial lateral movement can deter the ongoing APT attack. From the early detection, when a host is discovered as compromised, there are several forensic requirements we need to answer: What will be the end goal? What route the attacker can use to reach the end goal? To reach the end goal, the attacker may need to take several related attack steps (compromising hosts) and the identification of these steps can be used as an attack paths prediction process based on mathematical methods. Predicting the most likely attack path is an important technique that

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

enables the defender to react before the attacker reach the end goal by executing proactive responses.

Multi-Stage Attacks (MSAs) are cyber security threats where the attack campaign is performed through several attack stages. Each of the MSA stages comprises different attack steps where each step may not be malicious if implemented individually. APT has emerged as a complex version of MSAs in recent years [2]. The main objective of the APT attack is data exfiltration and intelligence appropriation. Adversarial lateral movement is of one the stage of the MSAs where the attacker stays in the network and slowly progress towards the target without raising an alert. Usually, this type of attack is conducted by highly skilled and motivated cyber criminals. The defender can use supervised/unsupervised machine learning approaches to detect each APT attack stage. The readers are encouraged to read different approaches, which are described in [3]. After detecting the lateral movement stage from an ongoing attack, the challenge of how the attacker will route to the end goal still remains. This challenge has brought interest in the research and development of new techniques to mitigate adversarial lateral movement. Therefore, in this paper, we use Hidden Markov Model (HMM) to identify the most likely attack path an attacker could take to reach the goal state. The HMM is a statistical model used for probability distributions over the sequence of observations [2]. HMM has also been utilized in [4], [5] to train models using observed network traffic under normal network conditions and to detect diverting sequences of traffic observations. Also, HMM can detect MSAs stages if an IDS misses the detection of any stages from the MSAs [2]. Therefore, HMM addresses the challenges of providing complete information on an attack campaign. After getting the most likely attack path, the defender needs to deter the lateral movement.

Defense methods to deter lateral movement are sometimes cost-effective in patching and resetting all suspicious entities. Moreover, patches are not available all the time, and sometimes it takes an extended period to develop the patch. Cyber deception techniques can help the cyber defender mitigate lateral movement without disrupting the organization's core services. Cyber deception has attracted attention from security researchers and practitioners as an approach for designing secure cyber infrastructure. Cyber deception can provide several advantages in mitigating cyber attacks, including providing the opportunity to learn the attacker's strategies, tactics, capabilities, intent and reduce the likelihood of adversarial success and cyber defense costs [6]. For a successful cyber deception, the defender needs to design the techniques appropriately. Deploying decoy nodes in the network is one of the cyber deception techniques. Researchers have proposed cyber deception approaches that introduce fake networks by varying system characteristics [7], manipulating attackers probes [8], [9], introduce virtual network interface controllers and route mutation [10]. All of the techniques focused on thwarting cyber reconnaissance mission. Preventing adversarial lateral movement using the cyber deception technique still needs a good amount of effort.

This paper proposes a method to predict the most likely attack path for adversarial lateral movement and deter the adversarial lateral movement using a cyber deception approach. Our proposed approach undergoes two main phases. The first phase predicts the most likely attack path based on Intrusion Detection System (IDS) alerts and *pcap* packet capture traces. The second phase is deployed decoy nodes along the predicted path. Our model assumes that the defender can detect the lateral movement stage from the APT life cycle. Rather than detecting the lateral movement, we focused on predicting the most likely attack path from the lateral movement stage. To predict the path, we use transition probabilities, present and past observations of the HMM. There are several works [11]–[13] on detecting lateral movement stage using machine learning approach. We used a network-based attack graph to correlate compromised hosts in the attack graph. We employ a state-based approach for alert correlation with the exploit activity, reducing the false positive alert. In the second phase, we utilize the predicted attack path to deploy decoy nodes. We employ a Partially Observable Monte-Carlo Planning (POMCP) framework to force the attacker towards the predicted path whenever the attacker deviates from the predicted path. POMCP helps the defender assess several defense actions to block the attacker in advance. The contribution of this work is summarized as follows:

- We present an approach to predict the most likely attack path for the adversarial lateral movement by leveraging HMM. This approach helps the defender understand the attacker's strategies and aims and plays a vital role for the security team to take the necessary actions (deploying decoy) before the attacker progresses into the predicted path and reaches the goal state.
- Incorporating POMCP in our model shows that the security defender can force the attacker towards deployed decoy paths whenever the attacker deviates from the predicted path. This module also provides insights on the optimal placement of decoy nodes.

The remainder of this paper is organized as follows. In Section II, the most relevant related works are reviewed. Section III provides an overview of the APT lifecycle. Section IV describes our proposed system's system architecture, and Section V presents the threat model of our proposed system. Section VI described our prediction model, and Section VII describes the assessment of various defense actions to block the attacker's path. Section VIII presents the performance evaluation of the proposed system and discusses the results. Finally, Section IX concludes the paper.

II. RELATED WORK

State-of-art and State-of-practice intrusion detection and prevention systems have been proposed to detect and prevent several cyber threats. However, it is infeasible to design a cyber defense system that can defend against all threats. In this section, we present some recent research works on model-based approaches for intrusion detection and prevention.

The work described in [14]–[16], and [17] uses hidden states for characterizing risk. These approaches learn a single HMM model for any attack type. In [16], authors computed probability matrices, but they did shed details on the probability matrices' computation. Authors in [17] propose a model based on HMM, but there is no indication of model training, and the model uses random values for the transition matrix. In contrast with our work, we define specific algorithms to train the model and use alert sequences in the model training. We also use the alert sequence to compute the probability matrices for prediction.

Zonouz *et al.* [18] propose a security-oriented cyber- physical state estimation (SCPSE), based on the attack graph, to predict the attack paths that an attacker can traverse by exploiting vulnerabilities. In their methods, each state transition is achieved by exploiting vulnerabilities in the hosts. The AG is converted to an HMM, which is used to determine the attacker's attack path. The execution time increases as the network grows and is not practical in the real world. In contrast to our work, we also use the attack graph in our model and handle the execution time by incorporating an exploit dependency graph, which reduces the execution time.

Attack graphs were proposed as the first method for predicting cyber attacks [19]. To predict a cyber-attack using an attack graph required traversing the graph and searching for a successful attack path or using probability values of edges in the graph. Probability values can give the most probable attack path, but it does not consider the underlying different attack steps the attacker can take. Ramaki *et al*. [20] proposed a framework for multi-step attack scenarios detection and prediction. Despite proposing an attack graph in their work, the authors extensively use causal correlations to predict the attack path. The attack graph alone cannot predict the most probable attack; instead, it can project all possible attack paths. Our model uses HMM and Bayesian Attack Graph (BAG) to reduce searching space, thereby substantially improving computational efficiency.

In [21], the authors proposed a Hidden Colored Petri-Net (HCPN) model to predict the attacker's next goal. However, their model suffers from performance issues as preconditions and postconditions significantly grow as actions are added to the HCPN. The action set was refereed there based on different IDS alert set from a specific attack scenario. We handle the performance issue using the POMCP framework. The POMCP algorithm requires a sample region to construct the entire state space, allowing one to avoid the state space explosion problem.

The finite states machine (FSM) model is used in [22] to design a multi-attack response system. The model sends an alert only after there is a state change without predicting the whole attack path. The authors also define a weight for each state but not for any specific multi-step attack scenario. In contrast, we define a probabilistic model to predict the most likely attack path from the lateral movement stage. The work described in [23] is closely related to our work as their prediction model is based on the IDS database, National Vulnerabilities Database (NVD), and attack graph data sources. The authors' model assigns every state's weight manually, whereas we use the HMM model to automatically train the parameters and assign the weight in each state. The authors did not provide any results for their proposed model.

So far, we have presented reactive methods for predicting multi-step attack paths. In the following discussion, we present some recent works on proactive methods based on techniques to deceive the attacker or change the attack surface to make it difficult for the attacker to carry out the attack.

The authors in [24] addressed the insider threat problem with a deception-based approach. They deploy decoy data in the network to confuse and confound the attacker and make it difficult to differentiate between original and decoy data. These decoy data are automatically created and placed on a decoy system to entice the attacker with fake credentials, which triggers an alert when the attacker access those decoy data. Additionally, the authors also embedded a beacon in the decoy documents that signal a remote website when accessed. However, the authors did not mention how the decoy systems should be deployed in the system. It is very costly for an enterprise network to distribute the decoy system all over the network to entice the attacker.

Game-theoretical approaches are used in cyber deception to mix true and false information to thwart the attacker's cyber reconnaissance mission. In [7], the authors presented a Cyber Deception Game (CDG) model on how the defender can benefit the most from determining a mix of true, false, and obscure responses to deceive the attackers. The Cyber Deception Game (CDG) model captures the strategic interaction between the defender and an adversary in network security. The authors use a zero-sum Stackelberg game between the defender (e.g., network administrator) and an adversary (e.g., hacker). Game-theory cannot directly apply to predict the multi-step attack prediction as the game solution in game theory is not explicit. The most commonly used solution concept is the *Nash Equilibrium*. However, finding the Nash Equilibrium of a game is often computationally intractable [25].

Urias *et al.* [26] proposed an unpredictable and adaptable deception-based framework using virtualization and software-defined networking. The proposed framework can provide better insights into an adversary's actions by correlating the network's endpoint behavior data.

III. PRELIMINARIES AND ASSUMPTIONS

This section provides an overview of APT life-cycle.

A. ADVANCED PERSISTENT THREAT AND LIFE-CYCLE

A threat actor who remains undetected for a more extended period in the network with the aim of espionage and sensitive data exfiltration drive by a state-sponsored or a group of threat actors is called an APT. An APT actor requires a high degree of knowledge and stealthiness behavior to

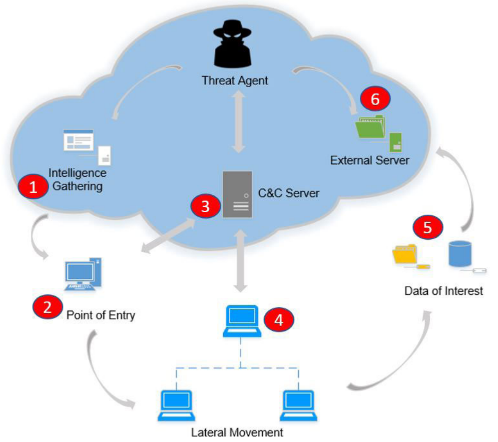


FIGURE 1. Typical stages of APT attack.

successfully carried out the attack. In Figure 1, we depict the different phases of an APT attack [27].

1) Intelligence gathering: This is the first step towards an APT attack where the attacker aims to collect intelligence information about the network as much as possible, including the organization's structure, IT structure, and sensitive information. The attacker uses public sources (Facebook, Linked In) and prepares a customized attack. Spear phishing email is the most commonly used technique to get to the point of entry [27].
2) Point of entry: After assessing security solution defenses and attack signatures that the victim might possess, the attacker narrows down the point of entry exploitation. Social engineering and spear-phishing email or vulnerability exploitation is the step used to penetrate the network. Another infection method is to plant malware into a website where organizations employees might visit.
3) Command and control (C&C) communication: In this stage, the communication between the infected host and the C&C server is performed through a secure socket layer (SSL), making it very difficult to identify whether traffic is malicious. Attackers may also use another technique, which is the domain flux technique [28]. In this technique, an infected host may try to connect to a large number of domain names to make it difficult to shut down all of these domain names.
4) Lateral movement and persistence: Once the attacker gains access to the target's network, the attacker will search for new hosts to infect and move laterally. There are several techniques that the attacker can use in this stage. One such attack is a brute force attack to obtain information such as a username and password or personal identification number (PIN). The attacker can also use internal spearfishing emails to gain access to other user's credentials. Another popular technique is the pass the hash (PTH) attack, where the attacker steals a hashed user credential and, without cracking it, reuses it to trick the authentication system.
5) Asset and data discovery: This stage aims to determine valuable assets within the target's network. Based on the asset and data discovery, the attacker determines the goal of future data exfiltration. Port scanning can be used for this step [29].
6) Data exfiltration: This is the final stage of APT, where the attacker tunneled data of interest into external servers with commonly used compressing and encryption techniques. Other techniques used in this stage include built-in file transfer via FTP or HTTP or the Tor anonymity network.

The attacker does not always need to use these stages in every APT attack. The author in [30] has discussed the APT life cycle model consisting of 7 stages such as (1) Initial Compromise, (2) Establish Foothold, (3) Escalate Privileges, (4) Internal Reconnaissance, (5) Move Laterally, (6) Maintain Presence and (7) Complete Mission. Ussath *et al.* [31] have discussed a 3 stage APT attack life cycle model focusing only on initial compromise, lateral movement, and command & control activity. Other modified versions of the APT attack life cycle model have been proposed in literature [3]. However, studies showed that this is the common life cycle followed by most of the APT attacks.

B. HIDDEN MARKOV MODEL

Hidden Markov Model is proposed to increase the usability of the Markov chain. A Markov chain states the probability of sequences of random variables. There is a strong assumption in the Markov chain that we need to only rely on the current state if we want to predict future states in the sequences. The previous state of the current state has no impact on the future state.

A Markov chain is applicable when we need to compute the probability for a sequence of observable events. That means the events we are interested in need to be directly observable. However, in many cases, we cannot observe them directly, which are called hidden states. HMM allows us to compute the probability of both observed events and hidden states. Figure 2 shows a Bayesian network representing the first-order HMM, where the hidden states are shaded in gray. In this model, an observation s_t at time t is produced by a stochastic process, but the state h_t of this process cannot be directly observed, i.e., it is hidden [32].

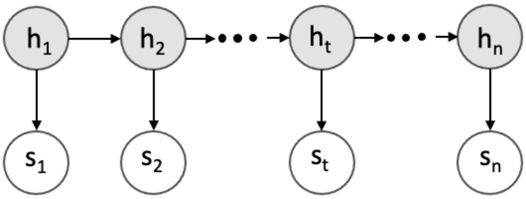


FIGURE 2. A Bayesian network representing a first-order HMM. The hidden states are shaded in gray.

Three fundamental problems such as training(learning), decoding, and evaluation need to be solved when a set of observations and the HMM are given:(1) Compute the probability of given observation sequence, (2) Compute the optimal (hidden) state sequence, and (3) Determine the optimal state transition probabilities and observation probabilities. The forward-Backward algorithm can use to solve the problem (1) [33], Viterbi algorithm solves the problem [33] (2), and Baum-Welch algorithm solves the problem (3) [34].

The Forward-Backward Algorithm: In the HMM, the actual state sequence is hidden, leading to considering all path probabilities to determine the observation probability. For N hidden states and T observations, there are N^T possible hidden sequences, leading to exponentially increasing possible paths. However, this complexity can be reduced by using Markov property and dynamic programming to efficiently compute values required to obtain the posterior marginal distributions in two passes. The first pass goes forward in time while the second goes backward in time. The Forward Algorithm (FW) computes the observation probability by summing over the probabilities of all possible state paths that could generate the observation sequence [34].

Viterbi Algorithm: The Viterbi algorithm is a dynamic programming algorithm used to find the most probable state sequence, also known as the decoding algorithm. The most probable state sequence can be computed by calculating the probability of the observation sequence for each possible path based on the Forward algorithm. In this approach, the most likely sequence is determined by tracing back the path with the highest likelihood value starting from the most likely state at the end of observation.

The Baum-Welch Algorithm: The BW algorithm is also known as the Forward-Backward algorithm. It is a dynamic programming approach and a special case of the expectation-maximization (EW) algorithm. The EW algorithm is an iterative method to find the maximum likelihood estimates of parameters in statistical models. The BW algorithm's main purpose is to tune the parameters of HMM, the state transition matrix A, the emission matrix B, and the initial probability distribution π_i. There are three phases in the BW algorithm: the initial phase, the forward phase, and the backward phase.

The list of assumptions that are made throughout the paper is listed here. First of all, it is assumed that the attacker is already in the network by performing social engineering and exploiting some vulnerabilities. Secondly, we assume that the defender can detect LM-based attacks in the network. Rather than focusing on detection, we focused on forestalling the attacker from reaching its goal(s) state. To capture the defender behavior in blocking vulnerabilities, we assume that the defender has some particular set of actions that restrict normal network configurations. For the illustrative example in the evaluation section, we assume that the attacker will move first and attempt an exploit.

TABLE 1. Symbols and their description.

Symbols	Description
δ	Maximum likelihood
$\xi_t(i,j)$	Probability of being a state
γ_t	Marginal probability
$\bar{\pi}_i$	Expected frequency
A	Transition probability
B	Emission probability
π	Initial probability distribution
ψ	Array of the argument
N	Node
E	Edge
s	Security state
$E(s_t)$	Available set of exploits
Z	Set of security alerts
β_t	Belief matrix
Φ	Attacker type
π^*	Optimal policy

IV. SYSTEM ARCHITECTURE

In this work, we considered lateral movement attacks from external threat agents and preventing the attack by deploying decoy nodes in the enterprise system. Here, we assume that the attacker is already in the network by performing social engineering and exploiting some vulnerabilities. How the attacker gains access to the system is beyond the scope of this paper.

Optimal deployment of decoy networks is always beneficial for the network administrator. It comforts the defender effort to drive the attacker towards deployed decoy nodes. During C2 communication in the APT life cycle, a set of infected hosts periodically sends a beacon to attacker-controlled servers and performed instructed operations. The operations include infecting other hosts in the network or gathering sensitive information about the network. Usually, the attacker uses HTTP(s), FTP, and SSH as a communication tool to evade easy detection. Attackers use several techniques to move laterally, including internal scanning, credential stealing, vulnerability exploitation, and privilege escalation. The exploitation of remote services is one of the techniques described by the MITRE post-compromise framework [35]. However, the attacker can use any of the techniques described in the framework. In this paper, we only consider the exploitation of remote services (T1210).

In Figure 3, we illustrate our basic system model architectural diagram. The first module in our architecture is *Lateral Movement Attack*, where the defender's job is to detect that attack using IDS alerts analysis and *pcap* (packet capture) traces. The alert dataset is needed to train the HMM parameters.

In the *Offline training module*, HMM parameters are trained based on observations of LM attacks. This module is explained in detail in the following section.

HMM Configuration File contains the algorithm to train HMM parameters and predicting the most likely attack path. This file also contains the procedure to obtain Local Conditional Probability Distribution (LCPD) for each node in the attack graph.

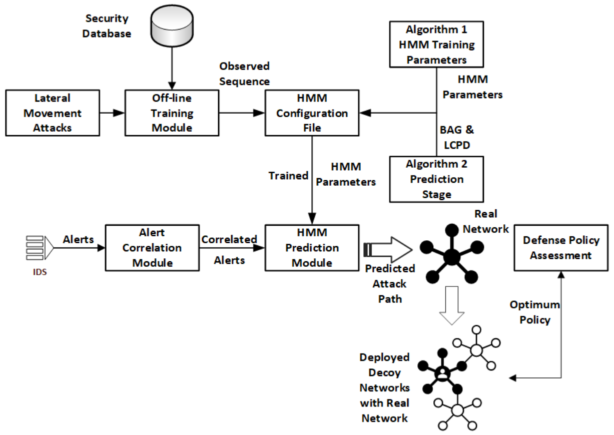


FIGURE 3. **System model architecture.**

The *Alert Correlation Module* receives alerts from IDS and uses the re-factorization and de-duplication technique to correlate alerts. This module also reduces the false positive alerts based on the state-based model, described in detail in the following sections.

The *HMM Prediction Module* uses two HMM algorithms, described in the following sections, to train the HMM parameters and predicting the most probable attack path. It also calculates the attack probability from the *Bayesian Attack Graph*.

The defender deploys decoy nodes along the predicted attack path to mislead the attacker. The defender can use the Defense Policy Assessment module to assess various defense actions in advance to force the attacker towards the decoy attack path whenever the attacker deviates from the predicted path.

The *Security Database* stores all the CVE information, including CVE name and CVSS scores, from the National Vulnerability Database (NVD).

V. THREAT MODEL

If we consider the threat modeling from the attacker's perspective, we must evaluate the attacker's goal (intent), capability, methods (ways), and resources (means). Threat landscape helps us to define defense requirements. The threat landscape has been evolving, with approximately 80% of threats categorized as a commodity carried out by attackers using widely known tools. The next 10% are directed attacks carried out using standard tools by organized crime to make money. Finally, the last 10% are the most destructive attacks, including advanced persistent threats (APTs) whose attacks are crafted for a single target [36]. Operationalizing deception begins with the organization's objective to learn the adversary's tactics & capabilities. Once the organization defines the objective, the deception must be implemented within the organization. It is also required to determine the type of adversary's deception; small threats require small sticks, but the APT-based threat requires sophisticated measures that lead to knowledge of adversary tactics and intent. Deception is not a passive exercise and requires adversary engagement. When or where to engage with the adversary is also a decisive factor to consider. In our approach, we consider these aspects to ensure adversary engagement with the help of deception technology while the attacker moves laterally.

Deploying decoy networks in the real networks to slow down and thwart the ongoing attack is a deception technique. Once the defender identifies compromised hosts from the lateral movement stage and correlates hosts in the attack graph, the defender understands when to deploy the network's decoy targets. However, the question remains where to deploy the decoy targets. It is evident that for a small-scale network, there will be more than one attack path. Let us assume that there are 100 attack paths to reach the target node, and it is infeasible and not a cost-effective way to deploy decoy targets across all the attack paths.

Usually, the attacker moves forwards within the APT life cycle, which means the attacker does not go back to a previous stage. However, if the current attack fails, the attacker can go back to the previous stage and finds another way to complete the attack campaign. In that case, the defender needs to evaluate the effectiveness of various defense decisions from the current belief state. We use the Partially Observable Monte-Carlo Planning Framework (POMCP) to help the defender making the effective defense decision to block the attacker from moving forward. In the following subsection, we describe our HMM-based model to predict the most probable attack path.

We have selected HMM to predict the most likely attack path due to its inherent benefits over other AI-based algorithms. We cannot directly observe the underlying attack steps the attacker will take to reach the target node. We can only probabilistically identify the likely attack path. HMM is a generative, probabilistic model to model the distribution over observations' sequences.

VI. PREDICTION VALUES

Our HMM-based state estimation model is inspired by the work presented in [2], where authors presented an approach to predict the next APT stage based on HMM. In our approach, we use HMM to predict the most probable attack path where more than one attack path resides. These findings will ease the defender's effort to deploy the decoy networks along with the real network.

HMM consists of two stochastic processes: a hidden process that is not observable but can be observed through another set of stochastic processes by producing the sequence of observations. In our model, the different attack steps towards a target are the hidden stochastic process where the observations are the alerts generated by the attacker.

Definition 1: An HMM is specified by the following components: for a given set of N states, $S = (s_1, s_2, s_3, \ldots, s_N)$ *and discrete observation symbols,* $\bar{O}_M = \bar{o}_1, \bar{o}_2, \ldots, \bar{o}_m$*, the state transition matrix,* $A = \{a_{i,j}\}$*, the observation emission matrix,* $B = \{b_i(\bar{o}_k)\}$*, and initial matrix,* π_i*, where* $i, j \epsilon [1, \ldots, N]$

and $k \epsilon [1, \ldots, M]$ *[2]. The probability of moving from state i to j is represented by* $\{a_{i,j}\}$, $b_i(\bar{o}_k)$ *is the probability of an observation,* $\bar{o}_k$, *emitted at state i, and* π_i *is the initial probability of HMM to start in state i. So, an HMM can be fully described by* $\lambda = (A, B, \pi)$.

For a given sequence of observations, O_t, and a sequence of states, Q_t, a first-order HMM makes two assumptions, First, the probability of a particular state depends only on the previous state:

$$P(q_t|q_1 \ldots q_{t-1} = P(q_t|q_{t-1})$$

Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations:

$$P(o_t|q_1 \ldots q_i, q_T, o_1, \ldots, o_t, \ldots, o_T) = P(o_t|q_t)$$

Our goal here is to calculate the probability of attack given an HMM. To do so, first, we need to know the system state after the last observation. There is one way we can achieve this by calculating the probability of being a state near the end of the Markov chain. We can use the Baum-Welch algorithm to compute the conditional probability of each observation's most likely state. However, if the state transitions have zero probability, the state sequence could not be correct. We use the Viterbi algorithm to get the single best state sequence for the given observation sequence to solve this issue.

A. HMM TRAINING ALGORITHM

The most challenging problem in HMM is determining a method to adjust the model parameter (A, B, π) to maximize the observation sequence probability [34]. There is no known way to optimize the parameters even in the finite observation sequence as training data. However, we can choose the parameter to be locally maximized using an iterative procedure such as the Baum-Welch (BW) method. Baum-Welch algorithm is a learning algorithm used to optimize the HMM transition and emission probabilities.

The BW algorithm first uses the Forward-Backward (FW) algorithm parameters α and β. Then using Bayes theorem and expectation-maximization [34] to introduce the two parameters. We start with the parameter $\xi_t(i, j)$, which defines the probability of being in state i at time t, transitioning to state j at time $t + 1$, given the model and observation sequence.

To train the HMM parameters, we use the historical record of alert observations and the Baum-Welch algorithm shown in Algorithm 1.

In Algorithm 1, HMM parameters (A, B, π) are initialized randomly. At lines 2 & 3, the parameter from FW and BW algorithm is computed. At line 5, we calculate the probability of being state i at time t and transitioning to state j at time $t + 1$ given the model and observation sequence. Then the marginal probability over state j is calculated in line 6. Using line 5 & 6, we can reestimate the parameters of HMM. A set of reasonable reestimation formulas for the HMM parameters (A, B, π) are with $\bar{\pi}_i$ being the expected frequency spent in state s_i at time 1 is presented at line 8. Next, $\bar{a}_{i,j}$ is the expected number of transitions from state i to state j over the overall number of transitions from state i at line 9. The parameter $\bar{b}_i(\bar{o}_k)$ is defined by the number of expected transition from state i, when observation is $o_t = \bar{o}_k$, over the number of expected transitions is presented at line 10. Finally, from lines 8 to 10, optimized HMM parameters are computed.

Algorithm 1 HMM Parameters Training

```
Input: Correlated alert $O_T$ sequence
Output: Optimized $A, B, \pi$
Initialization: Random$(A, B, \pi)$
 1: To compute $\alpha_{t+1}(i)$(FW) and $\beta_t(i)$(BW):
 2: $\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{i,j}] b_j(o_{t+1})$
 3: $\beta_T(i) = \sum_{j=1}^{N} a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)$
 4: for state $i \rightarrow j$ do
 5:     $\xi_t(i, j) = \frac{\alpha_t(i) a_{i,j} b_j(o_{t+1} \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}$
 6:     $\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i) \beta_t(i)}$
 7: while iterate until convergence do
 8:     $\bar{\pi}_i = \gamma_1(i)$
 9:     $\bar{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$
10:     $\bar{b}_i(\bar{o}_k) = \frac{\sum_{t=1}^{T-1} \gamma_t(i), when\ o_t = \bar{o}_k, else\ 0}{\sum_{t=1}^{T-1} \gamma_t(i)}$
```

B. STATE SEQUENCE AND PROBABILITY

Let us assume that we have a sequence of observations, O_t, and we want to compute the most probable sequence of states, Q_t. One approach is to find the sequence of states is to calculate the probability of the observation sequence for each possible path based on the Forward algorithm. In this approach, the most likely sequence is determined by tracing back to the path with the highest likelihood value starting from the most likely state at the end of observation. The Viterbi algorithm uses the δ parameter, where it considers only the maximum likelihood value. It also uses another parameter ψ to keep track of the argument, which maximized δ for each t and j. The complete procedure for finding the best state sequence is stated as follows [34]:

- The initialization step $(t = 1)$:

$$\delta_1(i) = \pi_i b_i(o_1)$$
$$\psi_1(i) = 0$$

- The recursion step:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{i,j}] b_j(o_t), 1 \leq j \leq N \quad (1)$$
$$\psi_t(j) = arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{i,j}], 1 \leq j \leq N \quad (2)$$

- The termination step $(t = T)$:

$$P(T) = \max_{1 \leq i \leq N} \delta_T(i) \quad (3)$$
$$q_T = arg \max_{1 \leq i \leq N} \delta_T(i) \quad (4)$$
$$q_t = \psi_{t+1}(q_{t+1}) \quad (5)$$

The term $P(T)$, q_T, and q_t in the (3), (4), and (5) defines as maximum probability, best last state, and previous best state, respectively.

State probability can be expressed in terms of forward-backward variables [37]:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{6}$$

The steps in FW algorithm are given below:
Consider the forward variable $\alpha_t(i)$ which defined as,

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_T, q_t = s_i|\lambda)$$

To solve for $\alpha_t(i)$ inductively as follows:

- Step-1: The initialization step (t=1),

$$\alpha_1(i) = \pi_i b_i(o_1)$$

- Step-2: The induction step for $(1 < t \leq T)$,

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i)a_{i,j}]\, b_j(o_{t+1})$$

- Step-3: The termination step,

$$P(O_T|\lambda) = \sum_{i-1}^{N} \alpha_T(i)$$

The Backward Algorithm (BW) computes the β parameter, as follows [34]:

$$\beta_T(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T|q_t = s_i, \lambda) \tag{7}$$

Steps to solve for $\beta_T(i)$ inductively, as follows:

- Step-1: The initialization step for $(1 \leq j \leq N)$,

$$\beta_T(i) = 1$$

- Step-2: The induction step,

$$\beta_T(i) = \sum_{j=1}^{N} a_{i,j}b_j(o_{t+1})\beta_{t+1}(j),\ t = T-1, T-2, \ldots, 1$$

C. ATTACK PATHS PREDICTION

1) ALERT CORRELATION FRAMEWORK

The alert processing unit first aggregates all the alerts and then performs de-duplication processing to construct the prediction module's alerts log. In the alert log file, there are necessary 10 fields to do the analysis represented as 10-tuple (*StartTime*, *EndTime*, *Type*, *SrcIP*, *DstIP*, *SrcPort*, *DstPort*, *Times*, *Protocol*, *Content*).

In the alert log file, *StartTime* represents the time when the alert is started, *EndTime* represents the alert event finished time, *Type* represents the type of the alert, *SrcIP* represents the origin IP for that alert, *DstIP* represents the destination address, *SrcPort* represents the origin port number, *DstPort* represents the destination port number, *Times* represents the alert repetitions number, *Protocol* represents the protocol used in the alert, *Content* represents the contents in the alert.

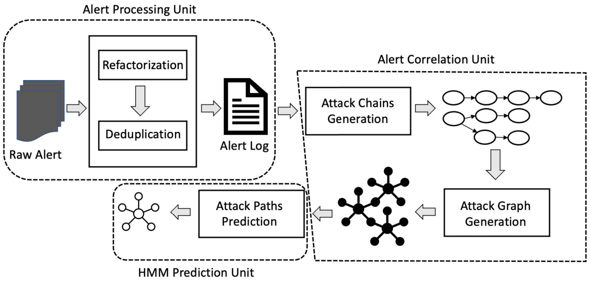


FIGURE 4. Attack path prediction framework.

De-duplication is applied to the aggregated alerts to reduce the number of alerts while keeping the source data. The de-duplication rule states that if the previous alerts IP, port, and type match the next alert, the latter alert will be discarded, and *EndTime* will be recorded with the previous alert. This alert correlation technique is used to predict the attacker's next state, whereas the state-based alert correlation technique is used to correlate the exploit activity for capturing the attacker's progression in the network. The state-based alert correlation model will discuss more in the following section.

After de-duplication, we get the set of alert logs where redundant alerts are removed based on the time information retention. For example, a single-step attack in a multi-step attack IDS may generate redundant alerts that may not belong to the same attack. In this way, de-duplication reduces the number of alerts and retains most of the source information. In the alert correlation unit, the attack chain is constructed using an alert graph based on the alert logs. We use the aggregation technique to remove redundant information. Here redundant information represents a redundant attack chain that does not belong to an ongoing attack. Lastly, the attack chains are obtained based on the depth-first-search (DFS) traversal algorithm. The attack graph generation module is responsible for presenting the association between attack chains intuitively. Attack graph generation module, first, converts the attack chains into a directed graph; second, it generates a dynamic Bayesian attack graph (BAG) from the attack graph.

2) HMM PREDICTION UNIT

To predict the next state of the attacker, we use Bayesian Attack Graph (BAG) [38], and Common Vulnerability Scoring System [39]. A Bayesian Attack Graph is a four tuple $BAG = (S, \tau, \epsilon, P)$ where $S = N_{internal} \cup N_{external} \cup N_{terminal}$ represents the set of attributes related to *internal*, *external*, and *terminal* node. The internal, $N_{external}$, represents the set of attributes, S_i, for postcondition of an attack. Similarly, $N_{internal}$, represents the set of attributes, S_j, between precondition and postcondition of an attack and $N_{terminal}$ is the set of attributes, S_k, for precondition of an attack. A set of ordered pairs, τ, represents the directed edges in the graph. Further, for $S_i \in S$, the set $P_a[S_i] = \{S_j \in S|(S_j, S_i) \in \tau\}$ is called the parent set of S_i. The relations of incoming connections

{AND, OR} of a node represents by ϵ. All the preconditions must be satisfied for AND, whereas if one or more preconditions are enough to exploit work, the relationship is defined as OR. To capture the success probability of an exploit, we use Local Conditional Probability Distribution (LCPD). Let, S_j, a local conditional probability distribution function when the preconditions are defined as AND [38]:

$$P_r(S_j|P_a[S_j]) = \left\{ \begin{matrix} 0, \exists S_i \in P_a[S_j]|S_i = 0, \\ P_r\left(\cap_{S_i=1} e_i\right), otherwise \end{matrix} \right\} \tag{8}$$

For OR,

$$P_r(S_j|P_a[S_j]) = \left\{ \begin{matrix} 0, \forall S_i \in P_a[S_j]|S_i = 0, \\ P_r\left(\cup_{S_i=1} e_i\right), otherwise \end{matrix} \right\} \tag{9}$$

When multiple exploits are present, for AND, each exploit has individual success probability. So, we use the product rule as follows:

$$P_r(\cap_{S_i=1} e_i) = \prod_{S_i=1} P_r(e_i) \tag{10}$$

For OR decomposition,

$$P_r(\cup_{S_i=1} e_i) = 1 - \prod_{S_i=1} [1 - P_r(e_i)] \tag{11}$$

To compute the LCPD, network administrator needs to estimate the success probability of a known exploit given in (21). The procedure of incorporating LCPD in our prediction algorithm is described in Algorithm 2.

Algorithm 2 Next State of the Attacker

```
Input: Optimized HMM parameters ($\bar{a}_{i,j}$, $\bar{b}_i(\bar{o}_k)$, $\bar{\pi}_i$), correlated alerts $O_T$, and LCPD from BAG
Output: The next state
1: for each of the next state $j = 1, 2, \ldots, N$ do
2:     for AND decomposition do
3:         $P_r(\cap_{S_i=1} e_i) = \prod_{S_i=1} P_r(e_i)$
4:     for OR decomposition do
5:         $P_r(\cup_{S_i=1} e_i) = 1 - \prod_{S_i=1} [1 - P_r(e_i)]$
6:     for each intermediate state $i = 1, 2, \ldots, N$ do
7:         $\alpha_t(i) = [\sum_{r=1}^{N} \alpha_{t-1}(i) a(r, i)] b_i(o_t)$ ▷ r=index of all possible prior states
8:     $P_{q_{t+1}=s_j} = \sum_{i=1}^{N} \alpha_t(i) P_r a_{i,j}$
```

This algorithm's inputs are as follows: optimized HMM parameters from Algorithm 1, correlated alerts O_T, and LCPD from BAG. For decomposition, the rule algorithm assigns edge probability to each node, which is presented at lines 1 to 5. Then, the FW α parameters of every intermediate stage, i, are calculated at line 7, and then the α parameters are multiplied by the transition probability and LCPD for the next state j. The state, which has the highest probability, is predicted as the attacker's next state.

VII. DEFENSE POLICY ASSESSMENT

Knowing how an attacker can progress in the network offers a useful starting point for defining appropriate defense actions. Attack graph can be leveraged to get the attacker's progression map in the network. However, it is still challenging to prescribe effective defense decisions as the defender has uncertainty over the network's security status at a given time, the attacker's true strategy, and attacker types. The defender only has information about the history of security alerts and previously deployed defense actions. The defender must make the defense decisions based on the belief matrix he possesses over the attacker's capabilities. The belief matrix is the joint probability distribution over the security states and attacker capabilities. Forcing the adversary towards deployed fake networks by taking actions (e.g., blocking vulnerabilities, applying security control) is a Partially Observable Markov Decision Process (POMDP) problem. There are two main primary objectives in our defense policy assessment model:1) quantify the security state, and 2) taking the optimum defense actions based on the attacker's capabilities. To quantify the security state, we define the security state as the set of currently enabled security conditions. In this sense, the security state at any given time represents the current capabilities of the attacker. One of our paper's objective is to quantify the level of security of the system as attacker progress. To capture the security level, we define the security state as a current level of the network's attacker progression.

A. CAPTURING ATTACKER's PROGRESSION

Researchers and cyber security professionals are always interested in projecting different attack steps an attacker can take to compromise a system. The attack graphs were developed and allowed to study all possible combinations of exploits an adversary can use to reach its goal(s). An attack graph consists of *system states* (nodes) and *transition relations* (edges). System states are related to each other via exploits. Attack graphs must enumerate all possible steps, which allow the graph to grow in dimension quickly. According to the monotonicity assumption [40], we can greatly simplify the attack graph and reduce the amount of information required to describe an attack. The monotonicity assumption states that one exploit's success does not interfere with the attacker's ability to carry out a future exploit. In a simpler term, we do not need to enumerate all system states in an attack graph, rather we can construct an exploit dependency graph which describes how an exploit is related to security conditions [40]. In [40], the authors construct such a graph where nodes represent security conditions, and edges represent exploits. Exploits are used to relate the security conditions via *preconditions* and *postconditions*. As discussed in [40], the edges in an exploit dependency graph relate the security conditions in a complex way, which means a given exploit can have both multiple preconditions and multiple postconditions. We formalize this behavior by acknowledging that such edges are directed *hyperedges*. Here, in this paper, the meaning of hyperedge is that an edge connects two *sets* of nodes rather than a pair of nodes.

To capture the attacker progression, we use an exploit dependency graph [41], a directed acyclic hypergraph, $H = (N, E)$, where, $N = \{c_1, c_2, \ldots, c_n\}$ is the set of security conditions and $E = \{e_1, e_2, \ldots, e_n\}$ is the set of exploits. The security conditions in the graph can be either true or false. When the security condition is in a true state, the attacker has a particular set of capabilities. In contrast, the false value represents the attacker does not possess any condition from hypergraph H. For example, a true security condition could mean an attacker may maliciously build the trust relationship between two hosts or the attacker reached the goal state. The distinct condition would represent the same host with different privilege levels. To specify the goal state, we define a parameter representing the goal node $N_r^g \subseteq N$, $N_f^g \subseteq N$ where N_r^g and N_f^g are real and fake network goal node, respectively. The defender's main objective is to protect the N_r^g and drive the attacker towards N_f^g.

Each exploits from hyperedges has two conditions, termed as $N_i^-(pre)$ and $N_i^+(post)$. To attempt an exploit e_i, an attacker needs to set true all of the preconditions of that exploit termed as $j \in N_i^-$ [41]. There are some exploits without having any preconditions, $N_i^- = \emptyset$, termed as initial exploits and denoted by E_0. To attempt initial exploits attacker does not need any prior capabilities (maliciously enabled). When an exploit is successful, all of its postconditions become enabled and let the attacker penetrate more into the network.

In Figure 5, we present an exploit dependency graph generated using Topological Vulnerability Analysis (TVA) [42] tool to explain the model and the results. Whenever a condition is enabled, it means an attacker has a particular set of capabilities where the current security state, s_t, describes the attacker's set of capabilities. A security state, $s \subseteq N$, is called a feasible security state if for every condition $c_j \in S$ there exists at least one exploit $e_i = (N_i^-, N_i^+) \in E$ such that $c_j \in N_i^+$ and $N_i^-, N_i^+ \subseteq s$ and set $S = \{s_1, \ldots, s_n\}$ represents the state space for this model. So, for a feasible security state, every enabled condition must have been enabled through an exploit, and all preconditions and postconditions associated with that exploit must also be enabled. Here, we made an implicit assumption for security state feasibility that our model is not missing any exploits that could allow the attacker to enable security conditions. This assumption makes sense because some nodes can be added in s, which are not associated with any hyperedge E. These nodes can become enabled via an unknown influence. We did not consider these nodes in our work because the state space greatly increases.

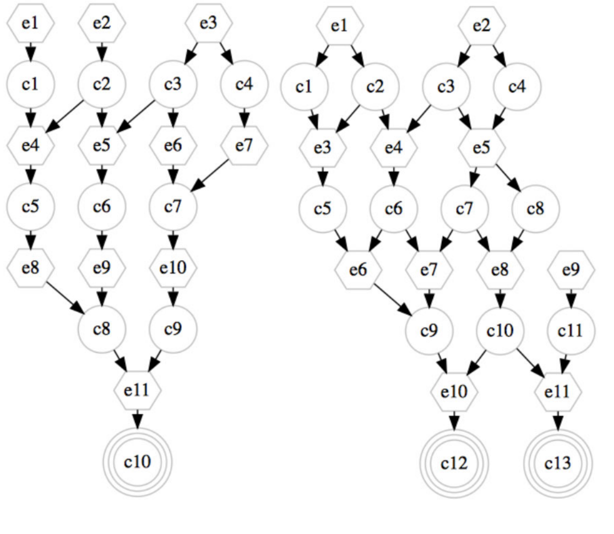


FIGURE 5. A sample Exploit Dependency Graph with a real network (left) and a fake network (right). The above dependency graph for real & fake network $H = (N, E)$ consists of $n_{cr} = 10$ security conditions, $n_{er} = 11$ exploits (in the form of hyperedges), $n_{cf} = 13$ security conditions and $n_{ef} = 11$ exploits respectively. Triple-encircled nodes are representing as goal conditions $N_r^g = \{c_{10}\}$ and $N_f^g = \{c_{12}, c_{13}\}$.

The security state evolves probabilistically as a function of the defender's and attacker's action [10]. The defender is assumed to select actions that have the impact of restricting normal network configuration. This action includes changing network configuration or shut down a port or any active services. However, in reality, the defender cannot block any individual vulnerability; instead, the defender's action induces a set of blocked vulnerabilities [41]. Blocking a set of vulnerabilities also helps us to capture some of the zero-day attacks. However, design a system to capture all unknown attacks is infeasible. To capture the defender's behavior in terms of blocking vulnerabilities, we assume that the defender has some particular set of actions that have the effect of restricting normal network configurations. The action will block the vulnerabilities and influence of an attacker to choose a different attack path.

The space of the defender's available action set is represented by $U = \{u^0, u^1, \ldots, u^n\}$. Here, u^0 represents the defender's null action, which means the defender will not block any exploit. The remaining actions from the set of U signify the network changes, which will induce a set of blocked exploits. Each action associated with the set of blocked exploits influences the attacker to seek the available paths. Defender's action will have an impact on the availability of the system to the trusted users. So, it is a defender's goal to make the trade-off between network availability and network security. To capture this behavior, we assign a cost to each of the defender's action sets. Based on the cost, the defender can choose an action that will limit the attacker's progression throughout the network and minimize the system availability's negative impact.

Based on the single attacker who is trying to infiltrate the system, it can only increase its capability by exploiting more vulnerabilities. On the other hand, it also increases the chance of being detected. The defender's goal is to prevent vulnerability exploitation in the real network and let the exploitation in the fake network. From the monotonicity assumption, we know that once an attacker enables a condition, it remains enabled all the time. For a given security state, s_t, the attacker will have some set of available exploits described by $E(s_t)$. From the available set of exploits, the attacker will attempt exploits based on capabilities. The available set of exploits is defined by [41],

$$E(s_t = s) = \{e_i = (N_i^-, N_i^+) \in | N_i^- \subset s, N_i^+ \not\subseteq s\} \quad (12)$$

Two important requirements that must be satisfied for an exploit $e_i = (N_i^-, N_i^+)$ to be available:(1) $N_i^- \subset s$, i.e. all of the exploit's preconditions must be satisfied:(2) $N_i^+ \not\subseteq s$, i.e. the exploit's postconditions must not all be satisfied [41]. The second requirement depends on the assumption that the attacker will not perform any redundant exploits. This is a reasonable assumption since the attacker is not gaining new capabilities by performing redundant exploits. It only increases the chance of being detected.

Figure 6 represents the sample evolution of the security state for a given state-action (s_t, u_t): (a) Consider the security state $s_t = \{c_1, c_2, c_3, c_4, c_5\}$ (green circle) and defense action $u_t = u$ where $B(u) = \{e_5, e_6\}$ (here blocked exploits are shown with red shaped hyperedge). So, the available set of exploits using (12) is $E(s_t) = \{e_5, e_6, e_7, e_8\}$ and (b) attacker attempt each exploit which does not lie within a set of blocked exploits, with a probability of attack and success. In this example, only exploits $\{e_7, e_8\}$ are succeeded and the updated security state is $s_t = \{c_7, c_8\}$ (green circle). Doubled circle shaded shape represents the security state.

As soon as the exploit attempts are successful, it enables all the postconditions, which eventually form the updated security state, as shown in Figure 6. Defender's lack of information regarding the current security state and the attacker's true strategy can be learned from noisy security alerts. The next section describes how the defender uses that information to construct the belief by getting security alerts from the Intrusion Detection System (IDS).

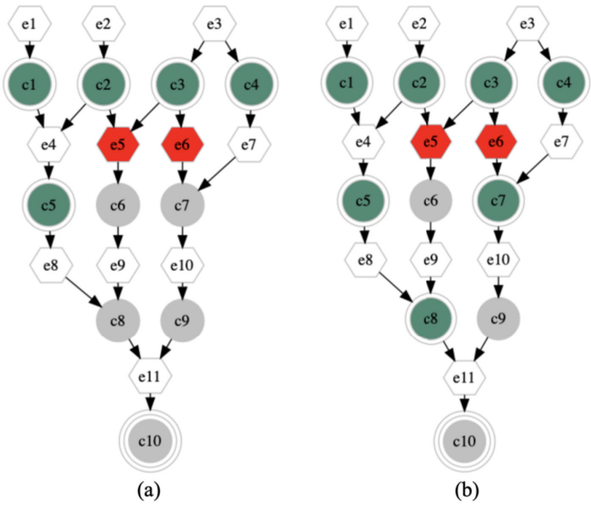


FIGURE 6. Sample evolution of the security state.

B. DEFENDER's AVAILABLE INFORMATION

Intrusion Detection System (IDS) is a major component in this model because the defender's certainty over the security state depends on security alerts. IDS generates security alerts in a sequential form when an attacker attempts to exploit and progress through the network. Those security alerts are not free from noisy alerts termed as false positives and false negatives. Sometimes, there will be no alert for the exploit activity, which solely depends on the attacker's capability (stealthiness), termed as a false negative. Similarly, it generates an alert for legitimate user activity termed as false positive. It is critically important for the defender to know which exploit activity is going on. Based on the alerts, the defender will choose his defensive action to drive the attacker towards deployed fake networks. Filtering out the noisy alerts from true alerts is essential in improving the defender's efficiency when it turns in real-time. In this work, we are considering only known vulnerabilities.

Let $Z = \{z_1, z_2, \ldots, z_n\}$ represents the set of security alerts generated by the IDS, which is the defender's observation set. Each exploit $e_i \in E$, when attempting can generate a set of alerts, given by the set $Z(e_i) = \{z_{A_i(1)}, z_{A_i(2)}, \ldots, z_{A_i(ai)}\} \in P(Z)$ where $P(Z)$ is the power set of Z [41]. There is a possibility that two or more exploits can generate the same alert, that is, $Z(e_i) \cap Z(e_j) = \emptyset$ for $e_i \neq e_j$. Some exploits $e_i \in E$ may not generate any alerts, that is, $Z(e_i) = \emptyset$.

To capture the uncertainty over the security state and attacker type, we construct a belief matrix denoted by κ_t. It combines all the defender's available information into the matrix, which includes initial security state, attacker type, history of all defense action from time 0 to $t-1$ and all observations (security alert) from time 0 to t denoted by $h_t = (\kappa_0, u_0, y_0, \ldots, u_{t-1}, y_t)$. The belief matrix represents the joint probability distribution over security states, and the attacker types [41], is given below as a matrix form,

$$\kappa_t = \begin{bmatrix} \kappa_t^{1,1} & \kappa_t^{1,2} & \ldots & \kappa_t^{1,n_a} \\ \kappa_t^{2,1} & \kappa_t^{2,2} & \ldots & \kappa_t^{2,n_a} \\ \vdots & \vdots & \vdots & \vdots \\ \kappa_t^{n_s,1} & \kappa_t^{n_s,2} & \ldots & \kappa_t^{n_s,n_a} \end{bmatrix} \in \Delta(S \times \Phi)$$

The space $\Delta(S \times \Phi)$ represents the probability distribution over state-type $(S \times \Phi)$. In the matrix, κ_t presented in the double-stochastic matrix for each t. Each row in the matrix represents the probability mass function over the type and space for a given state and each column represents a probability mass function over the space of security states for a given type. For any defense action $u_t = u$ and observation $y_{t+1} = y_k$, the belief update is defined as $\kappa_{t+1} = [T_j(\kappa_t, y_k, u)]_{s_j \in S}$ where j is the update function, $T_j(\kappa_t, y_k, u) = P(S_{t+1} = s_j \mid U_t = u, Y_{t+1} = y_k, K_t = \kappa_t)$ is given by [41],

$$\kappa_{t+1}^j = T_j(\kappa_t, y_k, u) = \frac{p_j^u(\kappa_t) r_{jk}^u(\kappa_t)}{\rho(\kappa_t, y_k, u)} \quad (13)$$

The above terms are defined below,

$$p^u_j(\kappa_t) = P(S_{t+1} = s_j \mid U_t, K_t) = \sum_{s_i \in S} \kappa^i_t \, p^u_{ij} \tag{14}$$

$$r^u_{jk}(\kappa_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t, K_t) = \sum_{s_i \in S} \kappa^i_t \, r^u_{ijk} \tag{15}$$

$$\rho(\kappa_t, y_k, u) = P(Y_{t+1} \mid U_t, B_t) = \sum_{s_j \in S} r^u_{jk}(\kappa_t) \, p^u_j(\kappa_t) \tag{16}$$

where p^u_{ij} is the transition probability from state s_i to s_j under defense action u, and $r^u_{jk}(\kappa_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t = u, K_t = \kappa_t)$ is the probability that IDS will generate observation vector y_k when transitioning from the state s_i to s_j under a defense action u. The trajectory of beliefs based on security alerts termed as observations and series of actions defined in (14). Under a defense action u, transition probability s_i to s_j is controlled by a set of exploit events. For the available set of exploits from (12), each event in the set of exploit is in binary form (successful and unsuccessful).

C. BALANCING SECURITY AND AVAILABILITY COST

In cyber deception, it is possible to leverage the availability cost over the security cost. There are two benefits when the attacker is in the fake network: 1) defender can collect as much intelligence information on the adversary, which helps to derive the attacker's capability, intentions, and targets, 2) defender can maximize the network availability to the trusted user during a cyber attack. An availability cost, c_a, for each action defender takes to drive the adversary towards the fake network. There will be no impact on the system's availability for some defense action, and sometimes there will be a more significant impact. To formalize this notion, we represent the availability cost $c_a : U \rightarrow \mathbb{R}$ for each defense action taken by the defender. Similarly, the security cost $c_s : S \times U \rightarrow \mathbb{R}$ represents the cost while the system is in various security states under defense action u. Here, we consider a node's availability regarding end-to-end packet delay (considering the IT system).

D. END-TO-END PACKET DELAY

A packet starts the journey from a host (source), passes through a series of routers, and ends its journey in another host (destination). It is assumed that d_E and N represent the total delay and number of devices between a source and destination. The end-to-end delay defined in [6] as

$$d_E = N(d_{proc} + d_{trans} + d_{prop} + d_{queue}) + d_{proco} \tag{17}$$

The terms are in (17) defined as following d_{proc} = processing delay, d_{trans} = transmission delay, d_{prop} = propagation delay, d_{queue} = queuing delay and d_{proco} = processing overhead because of authentication, integrity and confidentiality. For an uncongested enterprise network, $d_{queue} \simeq 0$ and the distance between the source and the destination node is very small so that $d_{prop} \simeq 0$. The processing delay, d_{proc}, is often negligible; however, it strongly influences a router's maximum throughput, which is the maximum rate at which a router can forward packets [6]. So that, (17) can be reduced to

$$d_E = N \times d_{trans} \tag{18}$$

where $d_{trans} = L/R$, L = packet size and R = transmission rate. For every defense action, the defender will measure the total end-to-end packet delay. So, the availability cost in terms of delay is defined as following $c_u = d_E$. We assign more cost to the goal conditions (attacker's target node) as the defender's goal is to keep the attacker from achieving the goal. The total cost in terms of a security state and defense action is defined as

$$c(s_t, u_t, \varphi_t) = (1-f)c_s(s_t, \varphi_t) + f * d_E(u_t) \tag{19}$$

Here, f is a weighted factor, determines which cost focused more (f = 0 represents defender is concerned only with security cost, $f = 1$ means defender is only concerned with availability cost).

E. THE DEFENSE ALGORITHM

An optimization of defense algorithm is a heuristic search algorithm for determining defense actions in real-time as the attacker progresses through the network and the security alerts are generated. The scalability is achieved via a sample-based online defense algorithm that takes advantage of the security model structure to enable computation in large-scale domains. For a large-scale network, computing optimal action while deceptively interacting with the attacker is a challenge. Offline POMDP solver aims to compute the optimal action for each belief state before runtime. Although such solvers have improved their efficiency [43], capturing the optimal action can be intractable for large networks. To resolve this issue, Silver and Veness [44] developed an online algorithm termed Partially Observable Monte-Carlo Planning (POMCP) to handle large-scale networks while computing optimal action. Online methods interleave the computation and execution (runtime) phases of policy, yielding a much more scalable approach than offline methods. POMCP algorithm is based on POMDP [24]. There are two types of nodes in POMCP: belief nodes representing a belief state and action nodes, which are their children nodes that can be reached by performing an action. In this work, the action selection procedure is the same as the POMCP algorithm described in [44], and the belief update procedure is modified to solve the large observation space problem as the belief update procedure in POMCP does not scale as the observation space grows.

Our defense policy assessment algorithm's action selection stage starts by performing Monte-Carlo simulations from the current belief state to estimate the various defense actions' quality. Each simulation starts by calling a *generative model* shown in Figure 8. A generative model makes predictions of all future events [45]. The predictions include what the model is meant to make. For example, a generative model will predict whether flipping over card 1 in 10 time-steps will

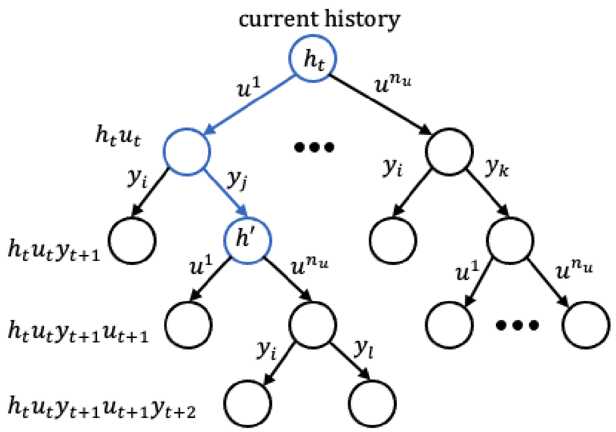


FIGURE 7. **An illustration of the search tree. The root node represents the current history. Each child node from the root node represents the possible future history. How the history is updated to h' after a real-world action (u^1)is taken and real-world observation (y_j) is received represented by the blue path [41].**

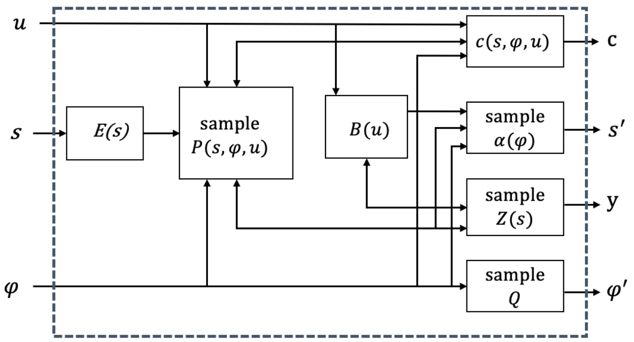


FIGURE 8. **The generative model.**

reveal the ace or whether cards 1 and 2 will be swapped in the next time-slot. An agent begins the simulation by calling the generative model that provides a sample successor state, observation, and cost given a state and action, $(s', \varphi', y, c) \sim G(s, \varphi, u)$. Calling the generative model and successive sampling from the current belief creates search tree histories, as shown in Figure 7.

As the process is partially observable, the search tree in Figure 7 consists of nodes representing histories, and branches from the original tree represent possible future histories. The multi-armed bandit rule, termed as UCB1 [46], is used to sample the selection of a defense action that begins from the branch of the search-tree. It also optimally balance the exploitation (to decrease the estimation error in terms of promising selection actions) and exploration (finding better alternatives by checking other actions). Here, the estimation error decreases as the number of simulations increases. The online algorithm performs the simulation until a stopping condition is met (the max number of simulation n_{sim}). After the simulation, defense action, which has the lowest value of the estimated cost, is taken. Then, a real-world action u_r and a real-world observation y_r is recorded. A new root node is specified as the current history node, and relevant branches of the search tree are identified, and lastly, the remaining tree is pruned.

As soon as the updated history h' is obtained, the defender's belief must be updated. However, the computation of the defender's belief analytically is complex, as shown in (13). This is why the defender maintains a belief approximation, $\mathfrak{B}_t$, a state-type pair called particles. This belief approximation updating procedure involves calling the generative model several times to obtain samples (s', y) until it matches the real-world observation vector y_r and s' is accepted into the updated belief set $\mathfrak{B}_{t+1}$. This procedure continues until n_k particles have been added. However, with the large observation spaces, the sampled observation rarely matches the real-world observation, causes the belief update procedure to take a longer time [41]. To address this, we use a modified belief update procedure. In the modified belief update procedure, instead of checking if the sample observation matches the real-world observation for every alert $z_i \in Z$, the update checks if the alerts match over a security state $z_i \in Z(s) = \cup_{e \in E} Z(e)$. After that, the particle probabilistically accepts if the condition is met. Here, the set $Z(s)$ contains the alerts that can be generated by an exploit attempts and the alerts not in $Z(s)$, *i.e.* any alert in $\bar{Z}(s) = Z \setminus Z(s)$, cannot be generated by the attempt of any exploit available in state s, as by (12). The reason for this behavior is that these are the only alerts that are informative for a change in the underlying state. So, the remaining alerts in $\bar{Z}(s)$ must have been triggered by false alerts under the current state s. The pseudocode of the defender's belief update procedure is given in Algorithm 3.

Algorithm 3 Defenders Belief Update

```
Initialize: $n_k$, $\mathfrak{B}_{t+1} = U_a$, numAdded = 0
procedure BeliefUpdate($\mathfrak{B}_t, u_r, y_r$)
    while $numAdded < n_k$ do
        $(s, \varphi) \sim \mathfrak{B}_t$
        $(s', \varphi', y, -) \sim G(s, \varphi, u_r)$
        if $y^{Z(s)} = y_r^{Z(s)}$ then          ▷ [If alerts Z(s) match]
            $\mathfrak{B}_{t+1} \leftarrow \mathfrak{B}_{t+1} \cup \{s', \varphi'\}$
            $numAdded \leftarrow numAdded + 1$
```

In Algorithm 3, we use a node utility array function as a defender's initial domain knowledge, which improves during more simulation runs. Attacker builds an array of node utility functions based on the base score metrics to exploit vulnerabilities [47]. For every exploit, attackers use the metrics to quantify the attack success probability and serves as the attacker's initial knowledge about the network and vulnerability. The attacker's node utility function is defined as follows [6]:

$$I = 10.41 \times (1 - (1 - CI) \times (1 - II) \times (1 - AI)) \quad (20)$$

$$V_i = 20 \times AC \times AI \times AV \quad (21)$$

The above terms are defined as CI = ConfImpact, II = IntegImpact, AI = AvailImpact, I = Impact, V_i = Exploitability,

AC = AccessComplexity, AI = Authentication and AV = AccessVector. The utility array function is defined below

$$U_a = I \times V_i \tag{22}$$

For any belief the defender may possess, he needs to determine an optimal action to deploy. This decision rule, which is determining the action, is called a defense policy. The optimum action for the defender while interacting with the attacker turns into a POMDP. Casting optimum action is defined as below [6],

$$\begin{aligned} V^{\pi}(\kappa_0) &= \sum_{t=0}^{\infty} \gamma^t c(\kappa_t, u_t, \varphi_t) \\ &= \sum_{t=0}^{\infty} \gamma^t E\big[c(s_t, u_t, \varphi_t)|\, \kappa_0, \pi\big] \end{aligned} \tag{23}$$

where $0 < \gamma < 1$ is the discount factor, and $c(\kappa_t, u_t)$ represents the cost for each belief state b_t when an action u_t is selected from the space of action where $c(\kappa_t, u_t) = \sum_{s_i \in S} \kappa_t^i c(s_t, u_t, \varphi_t)$. The optimal policy π^* is obtained by optimizing the long-term cost.

$$\pi^* = \arg\min_{\pi} V^{\pi}(\kappa_0) \tag{24}$$

The optimal policy defined in (24) specifies each belief state's optimal action where the expected minimum cost is calculated over the infinite time horizon.

F. ATTACKER's CAPABILITY ASSESSMENT

The concept of estimating adversary's Capability, Opportunity, and Intent (COI), which has been widely used in the military and intelligence community for threat assessment, can also be applied where network configurations and vulnerabilities are used for threat projection [48]. However, this approach does not project the attack well enough for attacks that continuously change the strategy and ignore the exposed system [49]. In this paper, we use a probabilistic approach to estimate the attacker's capability.

To assess the attacker's capability using domain knowledge, CVSS score, and the intrinsic parameter of a network, we categorized the attacker's capability into three vectors: knowledge, aggression, and stealthiness. Although it is assumed that a persistent attacker like APT is a highly skilled attacker, the attacker's capability assessment can help a network administrator to estimate the attacker's capability when deploying decoy nodes/networks. As our goal is to prevent lateral movement by deploying fake networks, the defender must understand the attacker's capability beforehand. In the following paragraph, we present how a defender can assess each skill level we defined earlier by using the defender's domain knowledge and the attacker's opportunities:

Knowledge As we defined earlier in this section, that knowledge level is defined as how the adversary changes its strategy based on the security measure imposes on the host. After the initial compromise of a system, the attackers need to move forward towards the attack goal/objective. In the lateral movement stage, the attacker tries to remain undetected in the system until they reach their goal. To remain undetected in the system, the adversary needs to understand the network well enough. Using a host-based network attack graph, the defender can correlate compromised hosts with the attacker's location in the system. As we know the available set of exploits from (12), the defender can use individual security states' likelihood in its belief matrix to assess new security information. For example, let us there is a single exploit available in state s_i and the set is $E(s_t) = e$. Now, if the exploit e is attempted, it generates the unique security alert z. No other exploit can generate the alert z here. In that case, the defender belief update allows the alert to be generated by an attempt to exploit e. The defender can then use the logical attack tree representation to see how the adversary has reached that stage. There could be multiple attack paths the attacker used, but using a log analysis defender can also identify the actual attack paths. We use attack path criticality metrics to score each attack path.

To calculate the attack path criticality score for a given network, we have considered attacker's opportunity metrics A_{om}, security control S_c, and pre-conditions P_{re} for that node. The path criticality score of a path p from host i to i' formulates as:

$$A_{c^p_{i,i'}} = \sum_1^j A_{om} \times S_c \times P_{re} \tag{25}$$

where the following parameters characterize the opportunity cost: available exploits, a_e, count of attack paths, c_{ap}, from host i to i', techniques used to compromise, t_c, from MITRE ATT&CK [35].

$$A_{om} = \sum_1^j a_e + {c_{ap}}^{-1} + {t_c}^{-1} \tag{26}$$

Aggression The aggression level is described by the conditional probabilities of attack and success, dictating the rate of movement through the network. The strategy attacker follows on few parameters: attacker knowledge level a_k, available opportunities in the state of action A_{om}, defenders' action d_a defined by conditional attack probability CAP,

$$P_{e_k}(s_t, u_t, \varphi_t) = \left\{ \begin{array}{l} \sum P(d_a, A_{om}|a_k) = \overline{P}_{e_k} \\ when\ e_k \in E(s_t) \backslash B(u_t) \\ \sum P(d_a, A_{om}|a_k) = \underline{P}_{e_k} \\ when\ e_k \in E(s_t) \cap B(u_t) \\ 0, when\ e_k \notin E(s_t) \end{array} \right\} \tag{27}$$

Dividing the set of available exploits into two categories helps us understand how an attacker changes the attacking strategy. In (27), $\overline{P}_{e_k}$ represents the probability of attack when there is no action, and $\underline{P}_{e_k}$ defines the attack probability when the defender's action block exploits.

Each of the attacker's attempts will succeed with a conditional probability of success. The probability of success models 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). So, for any given security state s_t, the conditional probability of success is defined by,

$$\alpha_{e_k}(s_t, u_t, \varphi_t) = \begin{Bmatrix} \overline{\alpha}_{e_k} & when\ e_k \notin B(u_t) \\ 0 & when\ e_k \in B(u_t) \end{Bmatrix} \tag{28}$$

Stealthiness Stealthiness is described by the probabilities of detection and false alarm. We have generated the probability of detection table for the assumed attacker types presented in the evaluation section. We will discuss more on this in the evaluation section.

VIII. EXPERIMENTAL EVALUATION

We effectively computed defense policies for large instances to scale our defense policy assessment algorithm using the defender's belief update procedure and the cost assignment. We did two large-scale network simulations to compute the most likely attack path and defense policies. Defense policies were computed for a problem on a graph consisting of 150 conditions (nodes), 160 exploits (hyperedges), 70 defense actions, and 43 security alerts (observation vectors over 10^9). The resulting number of security states exceeded 100 million. Our second instance on a graph consisting of 200 conditions (nodes), 250 exploits (hyperedges), 70 defense actions, and 60 security alerts (observation vectors over 10^{10}). The resulting number of security states exceeded 110 million.

1) AN ILLUSTRATIVE EXAMPLE

Figure 9 illustrates a small-scale experiment network used for an illustrative example. We synthesized a dataset of intrusion alerts due to the lack of publicly available datasets. The dataset is generated based on the 'LLDDoS1.0 DARPA' dataset. The network shown in Figure 9 consists of the firewall, intrusion detection system, and five hosts machine. The whole network is divided into two subnets based on the firewall policies. One host H1 and IDS are deployed in the DMZ, and the rest of the hosts are placed in the trusted zone. We assume that the attacker is already in the network by doing some social engineering and compromised the host H1 in the DMZ. The detailed vulnerability information was obtained from NVD public sites. There are six vulnerabilities found on our small-scale network, as presented in Table 2.

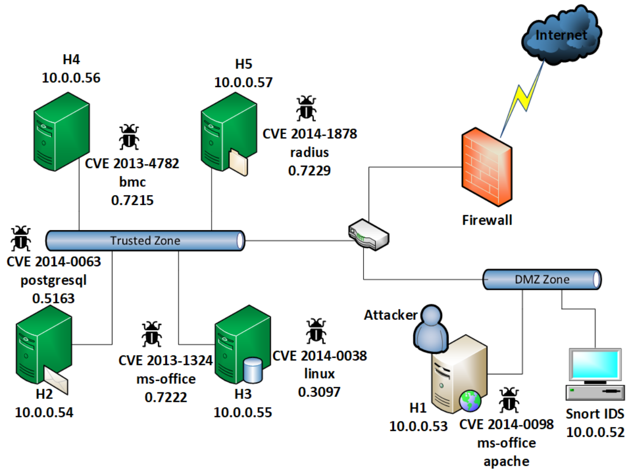


FIGURE 9. Experimental network topology.

TABLE 2. Hosts configuration and vulnerabilities information.

Host	Service	CVE ID	Severity	Weight	Impact
H1	apache	CVE 2014-0098	Mid	0.1	4.9
H2	postgresql	CVE 2014-0063	Mid	0.2	6.4
H3	Linux	CVE 2014-0038	High	0.1	10.0
H3	ms-office	CVE 2013-1324	Low	0.1	10.0
H4	bmc	CVE 2013-4782	Low	0.2	10.0
H5	radius	CVE 2014-1878	Low	0.3	2.9

$$A = \begin{bmatrix} 0.0092 & 0.9321 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0093 \\ 0.0093 & 0.0093 & 0.1727 & 0.4327 & 0.0093 & 0.3839 & 0.0095 \\ 0.0094 & 0.0094 & 0.0094 & 0.4405 & 0.2870 & 0.2649 & 0.0094 \\ 0.0093 & 0.0093 & 0.1435 & 0.0093 & 0.0134 & 0.4113 & 0.4317 \\ 0.0092 & 0.0092 & 0.2401 & 0.0092 & 0.0092 & 0.7103 & 0.0092 \\ 0.0096 & 0.0096 & 0.0096 & 0.0096 & 0.0096 & 0.0096 & 0.9503 \\ 0.0095 & 0.0095 & 0.0095 & 0.0095 & 0.0095 & 0.0095 & 0.0095 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.6531 & 0.3102 & 0.0267 & 0.0093 & 0.0093 & 0.0093 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \\ 0.0093 & 0.0093 & 0.0093 & 0.4932 & 0.4762 & 0.2761 & 0.0093 & 0.0093 & 0.4357 & 0.0093 & 0.3286 & 0.1502 & 0.0090 & 0.0091 \\ 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.6288 & 0.2886 & 0.0092 & 0.0092 & 0.0092 & 0.0091 & 0.0091 & 0.0091 \\ 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \\ 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0092 & 0.0091 \\ 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 \\ 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \end{bmatrix}$$

A. MOST LIKELY ATTACK PATH

We leverage [50] to generate the experimental network's corresponding attack graph, shown in Figure 10. We use an

TABLE 3. Assessments of vulnerability exploitability probability.

CVE ID	Exploitability Probability
CVE 2014-0098	0.7230
CVE 2014-0063	0.5163
CVE 2014-0038	0.3097
CVE 2013-1324	0.7222
CVE 2013-4782	0.7215
CVE 2014-1878	0.7229

automated alert analysis tool ArcSight [51], to analyze the alerts information and extract the attack sequence. We assume that the attacker is trying to obtain the root privilege of the Host H5. The training algorithm (Algorithm 1) and the prediction algorithm (Algorithm 2) are implemented as follows to predict the subsequent attack behaviors. Both algorithms, the training and prediction algorithm, are written in Python 3.

Step 1 (HMM Parameters Training): We use our alert correlation framework to generate the correlated alert dataset. For each new alert, the ACF checks all historical alerts which have been triggered over the last time window. Two alerts are correlated if they have the same src_{ip} or dst_{ip}. We use the alert correlation dataset as the historical record of alerts observation to learn and optimize the HMM parameters using the Baum-Welch algorithm as presented in Algorithm 1. We consider seven states for HMM, as shown in Table 4.

First, we initialized the HMM parameters (A, B, π) randomly. Then, the two parameters α and β from FW and BW are computed. To compute the Baum-Welch algorithm's two parameters ξ and γ, we start from state s_1 and considers all training observations sequences to update the HMM parameters. Considering 7 different attack states and 14 observations for the HMM in the attack graph presented in Figure 10, the above transition, A, and emission, B, probabilities were obtained. The values in the matrix A represent the probability that the attacker will move from one state to another. If the destination state is unreachable, the value is zero. It is important to note that both the A & B matrix should not contain any zero value element. Zero-value will produce the *NaN* error. To avoid the *NaN* error, the algorithms replace the zero value with minimal value.

Step 2 (Vulnerability Exploitability Probability): Using (21), we calculate the vulnerability exploitation probabilities presented in Table 3.

Step 3 (Attack Path Prediction): To predict the most likely attack paths, the optimized HMM parameters $((\bar{a}_{i,j}, \bar{b}_i(\bar{o}_k), \bar{\pi}_i))$ from step 1, correlated alerts, O_T, from the ACF, and LCPD from BAG are used. LCPD are calculated from the attack graph presented in Figure 10. There are 7 different attack states, and one stage is probable to another stage is called the transition probability. Table 4 presents the attack states' denotation, and all attack behaviors information is presented in Table 5.

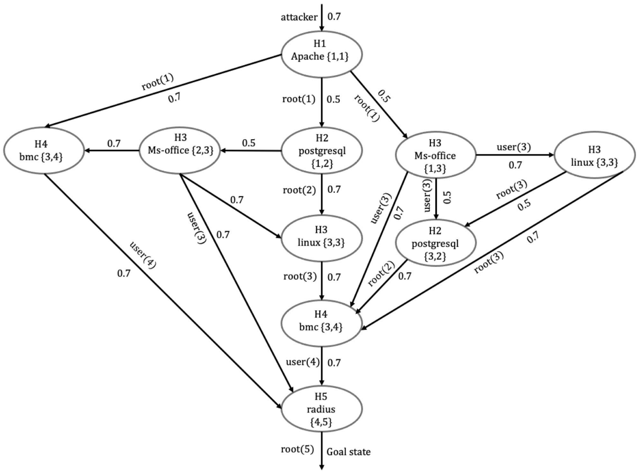


FIGURE 10. Attack graph of the experimental network.

TABLE 4. Attack states description.

State	Description
S_1	Initial State
S_2	$(H_1$,root)
S_3	$(H_2$,root)
S_4	$(H_3$,user)
S_5	$(H_3$,root)
S_6	$(H_4$,user)
S_7	$(H_5$,root)

TABLE 5. Description of attack behaviors.

State transition	CVE ID
$S_1 \rightarrow S_2$	CVE 2014-0098
$S_2 \rightarrow S_3$	CVE 2014-0063
$S_2 \rightarrow S_4$	CVE 2013-1324
$S_2 \rightarrow S_6$	CVE 2013-4782
$S_3 \rightarrow S_4$	CVE 2013-1324
$S_3 \rightarrow S_5$	CVE 2014-0038
$S_3 \rightarrow S_6$	CVE 2013-4782
$S_4 \rightarrow S_3$	CVE 2014-0063
$S_4 \rightarrow S_5$	CVE 2014-0038
$S_4 \rightarrow S_6$	CVE 2013-4782
$S_4 \rightarrow S_7$	CVE 2014-1878
$S_5 \rightarrow S_3$	CVE 2014-0063
$S_5 \rightarrow S_6$	CVE 2013-4782
$S_6 \rightarrow S_7$	CVE 2014-1878

As it is evident from Table 5 that there are 14 different state transitions for the target network. Based on the alert data from our dataset and extracted attack sequence, we introduce the state transition success probability vector T, where $T = \{0, 0.9321, 0.1727, 0, 0, 0, 0\}$. The total state transition probability matrix is presented in A, and the emission probability matrix is presented in B. After initializing parameters A&B, we use Algorithm 2 to simulate the process. The results here show that the algorithm runs five times. As it is evident from T^5 that the attack goal is S_7, and the corresponding success probability is 0.84. In Table 6, we depicted all possible attack paths.

$T^5 = \{1, 0.78, 0.62, 0.59, 0.76, 0.67, 0.84\}$

Here, we are looking for the most probable attack path with a length of 5. From Table 6, we can infer that only paths 5,7, and 9 have a length of 5. By matching the alert sequence in

TABLE 6. Possible attack paths.

Path Number	Attack Path
1	$S_1 \rightarrow S_2 \rightarrow S_6 \rightarrow S_7$
2	$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_7$
3	$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_6 \rightarrow S_7$
4	$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7$
5	$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7$
6	$S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow S_6 \rightarrow S_7$
7	$S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow S_3 \rightarrow S_6 \rightarrow S_7$
8	$S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow S_5 \rightarrow S_3 \rightarrow S_6 \rightarrow S_7$
9	$S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7$

the dataset, we get $S_1 \rightarrow S_2 \rightarrow S_3$ as the prior path. So that we can conclude that the future attack path will be $S_5 \rightarrow S_6 \rightarrow S_7$. For the experimental network, we get the most likely attack path for lateral movement is $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7$. In the following evaluation section, we will deploy decoy nodes along this path and show how a defender can force the attacker toward decoy nodes if the attacker does not choose the most likely attack path.

B. DEFENSE POLICY ASSESSMENT

From the previous section, we acquired the most likely attack path sequence towards a target. The defender should deploy decoy nodes along that path to keep the attacker away from the real target node. In Section C, we defined the way to estimate the attacker's capability, which is the defender's initial belief. Based on initial belief and domain knowledge, the defender will estimate attack probability and success probability for each exploit present in the system. We generated the exploit dependency graph for the experimental network using Topological Vulnerability Analysis (TVA) [42]. In Figure 11, we presented the corresponding exploit dependency graph. We use an existing POMCP solver [6] in our simulation, which is implemented in Python. In this simulation, we presented two use case scenarios to depict the attacker and the defender effort exchange to compromise the target node and prevent the target from being compromised. We assume that the defender can deploy the decoy nodes at the time of intrusion alert in the network. To alleviate the time complexity in deploying decoy nodes, the defender can design and initiate the decoy nodes without connecting with the network. The design of the decoy nodes is beyond the scope of this paper.

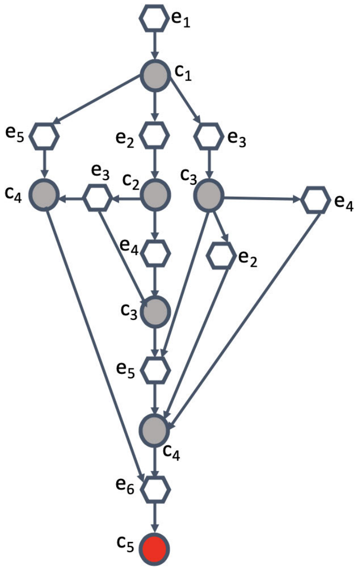


FIGURE 11. Exploit dependency graph of the experimental network.

For each of the exploits present in the network, we will now define the attack and its success probability based on the attacker's knowledge, aggression, and stealthiness defined in (25-28). Here, we estimate the attacker's knowledge, aggression, and stealthiness level are high, moderate, and high, respectively. Probabilities of attack for each exploit are as follows:

$$(\overline{P}_{e_k}, \underline{P}_{e_k}) = (0.5, 0.5) \quad for \quad e_k \in E_0$$
$$(\overline{P}_{e_k}, \underline{P}_{e_k}) = (0.7, 0.3) \quad for \quad e_k \in \{e_4, e_5, e_6\}$$
$$(\overline{P}_{e_k}, \underline{P}_{e_k}) = (0.6, 0.4) \quad for \quad e_k \in \{e_2, e_6\}$$
$$(\overline{P}_{e_k}, \underline{P}_{e_k}) = (0.9, 0.8) \quad for \quad e_k \in \{e_3, e_5, e_6\}$$

similarly, probabilities of success are as follows:

$$\alpha_{e_k} = \begin{Bmatrix} 0.7 \ when \ e_k \in E_0 \\ 0.5 \ when \ e_k \in E \backslash E_0 \end{Bmatrix}$$

In Table 7, we presented the probability of detection for each of the exploit.

TABLE 7. Probability of detection for estimated attacker's capability.

Alert	e_1	e_2	e_3	e_4	e_5	e_6
Z_1	0.3	0.4	0	0	0	0
Z_2	0	0.2	0.3	0	0	0
Z_3	0	0	0.4	0.3	0	0
Z_4	0	0	0	0.4	0.4	0
Z_5	0	0	0	0.2	0.5	0
Z_6	0	0	0	0	0.5	0.2
Z_6	0	0	0	0	0	0.6

Use Case A: In this use case scenario, we deploy decoy nodes along in the predicted attack path sequence, and the attacker chooses the decoy nodes path to move laterally in the network. Figure 12 represents the exploit dependency graph with the decoy nodes where yellow color nodes represent decoy nodes. In this simulation, we consider three actions which induce a set of blocked exploits and the actions set is as follows: $B(u^1) = \{e_2, e_4\}$, $B(u^2) = \{e_5, e_6\}$, $B(u^3) = \{e_3\}$. The discount factor for this simulation is $\gamma = 0.95$. There are total $n_s = 182$ security states and $n_z = 7$ security alerts leading to $2^8 = 256$ distinct observation vector. All simulations use particles $n_k = 1500$ to approximate the belief. The evolution of computed deception policy when $N_{sim} = 5000$ and attacker's lateral movement throughout the real and decoy nodes are presented in Figure 13.

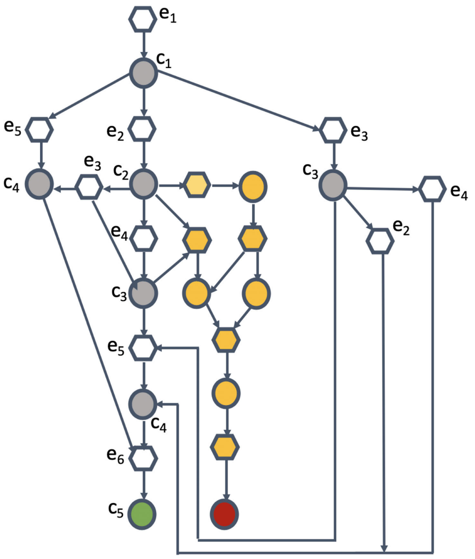
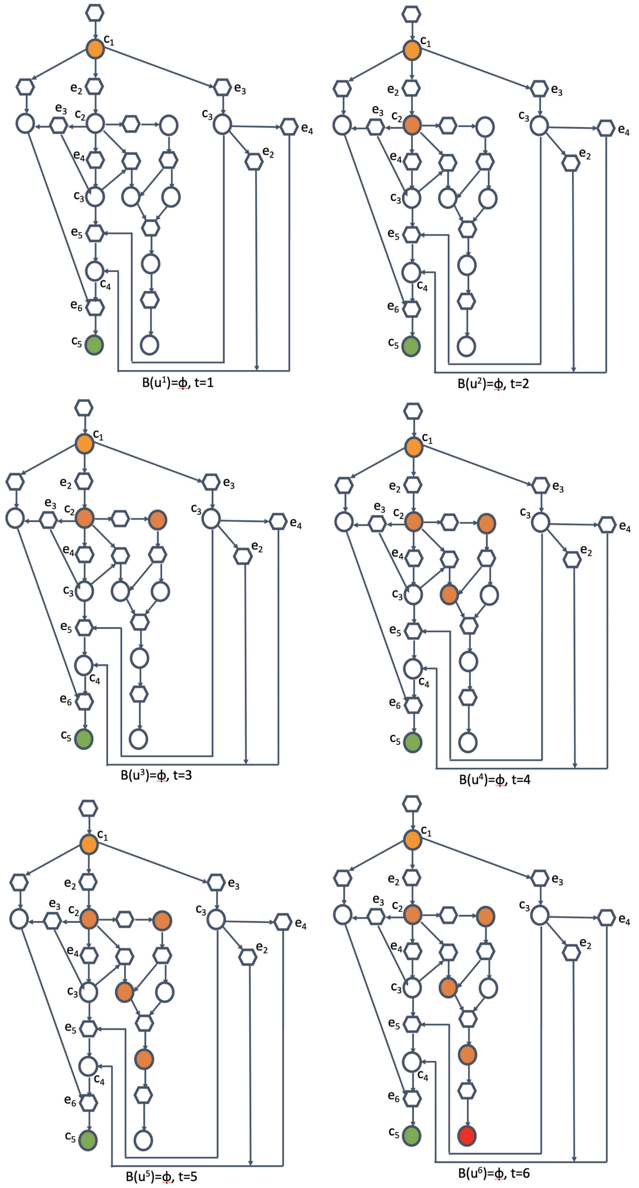


FIGURE 12. Exploit dependency graph of the experimental network with decoy nodes.

Here, it is assumed that the attacker moves first, and the security state starts from the empty state $s_0 = \phi$. As we already stated that attackers already penetrate the network by using social engineering. The attacker's starting position in the network is the host H_1, represented by an orange color (C_1) in the top left corner of Figure 13. To make the service available to the legitimate users, the defender does not block any exploits in advance; rather, the defender's belief matrix gradually improves on the security state. As in this use case scenario, it is assumed that the attacker would take the decoy nodes path towards the fake goal state. It is evident from Figure 13 that at time $t = 1$ attacker is in C_1 then gradually moves laterally by exploiting more vulnerability ($t = 2$ to $t = 6$). The fake goal state is marked by red color at the most right bottom of Figure 13.

Use Case B: In this use case, we will demonstrate how a defender can push the attacker towards deployed decoy nodes when the attacker does not take the decoy nodes path. In this case, the defender will block exploits to prevent the attacker from compromising the real goal state. Figure 14 demonstrates the graphical representation of the defender's actions observing the attacker's lateral movement. We use the same simulation parameters used in Use Case A. The evolution of computed deception policy is presented in Figure 14 when $N_{sim} = 5000$.

Initially, the defender does not take any actions (from t=1 to t=2) rather gradually updates the belief based on the received security alerts. Then defender begins to deploy defense actions (t=3) when the defender belief reflects that the attacker is not taking the predicted path. It is evident from Figure 14 when t=3 that the attacker exploited vulnerability e_3 and reached c_3, which is not in the predicted paths. Defender takes an action that induces a set of blocked exploits, in this case, e_2, e_4 marked as a red hexagon in Figure 14. Because of the blocked exploits, the attacker cannot move laterally to exploit vulnerabilities e_5, e_6. These are the ultimate two vulnerabilities that need to be exploited to reach the real network goal state c_5. In this situation, the attacker tries to find another way to move forward. At t=4, the attacker reached c_2 (orange circle) by exploiting exploits e_2. At t=5, it is evident that the defender's belief reflects that attacker is in the predicted real network attack path and towards the real goal state. In this case, the defender's action block vulnerabilities e_5, e_6, and the attacker is forced to take the decoy nodes path to move forward. The red circle in Figure 14 represents the fake goal state.



$B(u^1)=\phi$, t=1

$B(u^2)=\phi$, t=2

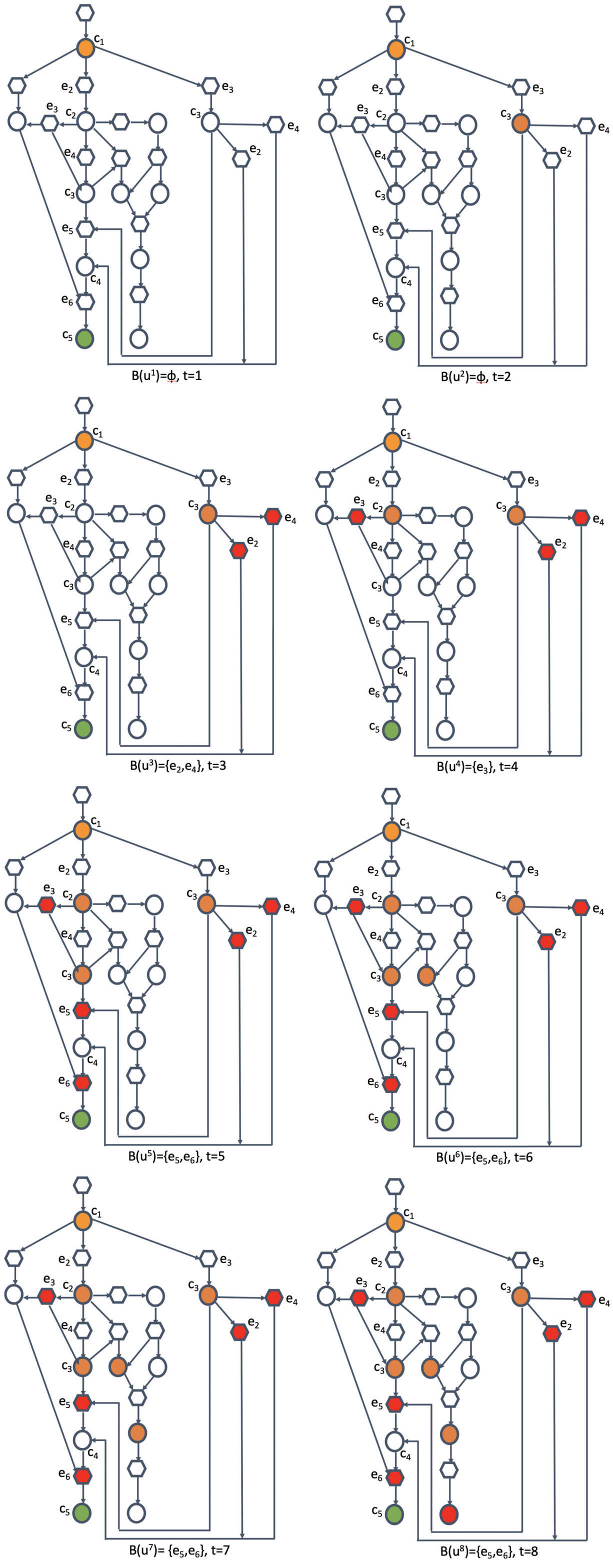
$B(u^3)=\phi$, t=3

$B(u^4)=\phi$, t=4

$B(u^5)=\phi$, t=5

$B(u^6)=\phi$, t=6

FIGURE 13. Sample evolution of deception policy and attacker's lateral movement.



$B(u^1)=\phi$, t=1

$B(u^2)=\phi$, t=2

$B(u^3)=\{e_2,e_4\}$, t=3

$B(u^4)=\{e_3\}$, t=4

$B(u^5)=\{e_5,e_6\}$, t=5

$B(u^6)=\{e_5,e_6\}$, t=6

$B(u^7)=\{e_5,e_6\}$, t=7

$B(u^8)=\{e_5,e_6\}$, t=8

FIGURE 14. Sample evolution of deception policy and attacker's lateral movement.

IX. CONCLUSION

This paper proposes an adversarial lateral movement prevention technique by incorporating reactive (graph analysis) and proactive (cyber deception technology) methods. In our proposed system, the approach undergoes two main phases. The first phase predicts the most likely attack path based on Intrusion Detection System (IDS) alerts and *pcap* packet capture traces. The second phase is deploying decoy nodes along the predicted path. To predict the path, we use transition probabilities and present and past observations of the HMM. In the second phase, we utilize the predicted attack path to deploy decoy nodes. The Hidden Markov based model has been developed to predict the most likely attack path from the lateral movement stage. Forecasting the next sequence of attack paths helps the defender deploy decoy nodes and save time and cost in a resource-constrained environment. It also allows us to prevent the attack from reaching the final stage of data exfiltration. This prediction module uses the Viterbi and forward-backward algorithm to determine the most likely attack path sequences by correlating the sequence of alert and packet trace analysis. For future work, we plan to incorporate MITRE ATT&CK post-compromise framework and additional context from the target system in our model.

REFERENCES

[1] Mcaafee-Report. (2018). *The Economic Impact of Cybercrime no Slowing Down*. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/executive-summaries/es-economic-impact-cybercrime.pdf

[2] I. Ghafir, K. G. Kyriakopoulos, S. Lambotharan, F. J. Aparicio-Navarro, B. Assadhan, H. Binsalleeh, and D. M. Diab, "Hidden Markov models and alert correlations for the prediction of advanced persistent threats," *IEEE Access*, vol. 7, pp. 99508–99520, 2019.

[3] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1851–1877, 2nd Quart., 2019.

[4] Z. Anming and J. Chunfu, "Study on the applications of hidden Markov models to computer intrusion detection," in *Proc. 5th World Congr. Intell. Control Autom.*, vol. 5, 2004, pp. 4352–4356.

[5] S. Shin, S. Lee, H. Kim, and S. Kim, "Advanced probabilistic approach for network intrusion forecasting and detection," *Expert Syst. Appl.*, vol. 40, no. 1, pp. 315–322, Jan. 2013.

[6] M. Al Amin, S. Shetty, L. Njilla, D. Tosh, and C. Kamouha, "Attacker capability based dynamic deception model for large-scale networks," *EAI Endorsed Trans. Secur. Saf.*, vol. 6, no. 21, Aug. 2019, Art. no. 162808.

[7] A. Schlenker, O. Thakoor, H. Xu, M. Tambe, P. Vayanos, F. Fang, L. Tran-Thanh, and Y. Vorobeychik, "Deceiving cyber adversaries: A game theoretic approach," in *Proc. Int. Conf. Auto. Agents Multiagent Syst.*, 2018.

[8] M. Albanese, E. Battista, S. Jajodia, and V. Casola, "Manipulating the attacker's view of a system's attack surface," in *Proc. IEEE Conf. Commun. Netw. Secur.*, Oct. 2014, pp. 472–480.

[9] S. T. Trassare, R. Beverly, and D. Alderson, "A technique for network topology deception," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Nov. 2013, pp. 1795–1800.

[10] Q. Duan, E. Al-Shaer, and H. Jafarian, "Efficient random route mutation considering flow and network constraints," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2013, pp. 260–268.

[11] G. Zhao, K. Xu, L. Xu, and B. Wu, "Detecting APT malware infections based on malicious DNS and traffic analysis," *IEEE Access*, vol. 3, pp. 1132–1142, 2015.

[12] B. C. M. Cappers and J. J. van Wijk, "SNAPS: Semantic network traffic analysis through projection and selection," in *Proc. IEEE Symp. Visualizat. Cyber Secur. (VizSec)*, Oct. 2015, pp. 1–8.

[13] M. Marchetti, F. Pierazzi, A. Guido, and M. Colajanni, "Countering advanced persistent threats through security intelligence and big data analytics," in *Proc. 8th Int. Conf. Cyber Conflict (CyCon)*, May 2016, pp. 243–261.

[14] K. Haslum, A. Abraham, and S. Knapskog, "DIPS: A framework for distributed intrusion prediction and prevention using hidden Markov models and online fuzzy risk assessment," in *Proc. 3rd Int. Symp. Inf. Assurance Secur.*, Aug. 2007, pp. 183–190.

[15] K. Haslum, M. E. G. Moe, and S. J. Knapskog, "Real-time intrusion prevention and security analysis of networks using HMMs," in *Proc. 33rd IEEE Conf. Local Comput. Netw. (LCN)*, Oct. 2008, pp. 927–934.

[16] A. S. Sendi, M. Dagenais, M. Jabbarifar, and M. Couture, "Real time intrusion prediction based on optimized alerts with hidden Markov model," *J. Netw.*, vol. 7, no. 2, p. 311, Feb. 2012.

[17] H. A. Kholidy, A. Erradi, S. Abdelwahed, and A. Azab, "A finite state hidden Markov model for predicting multistage attacks in cloud systems," in *Proc. IEEE 12th Int. Conf. Dependable, Autonomic Secure Comput.*, Aug. 2014, pp. 14–19.

[18] S. Zonouz, K. M. Rogers, R. Berthier, R. B. Bobba, W. H. Sanders, and T. J. Overbye, "SCPSE: Security-oriented cyber-physical state estimation for power grid critical infrastructures," *IEEE Trans. Smart Grid*, vol. 3, no. 4, pp. 1790–1799, Dec. 2012.

[19] T. Hughes and O. Sheyner, "Attack scenario graphs for computer network threat analysis and prediction," *Complexity*, vol. 9, no. 2, pp. 15–18, Nov. 2003.

[20] A. A. Ramaki, M. Amini, and R. E. Atani, "RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection," *Comput. Secur.*, vol. 49, pp. 206–219, Mar. 2015.

[21] D. Yu and D. Frincke, "Improving the quality of alerts and predicting intruder's next goal with hidden colored Petri-net," *Comput. Netw.*, vol. 51, no. 3, pp. 632–654, Feb. 2007.

[22] A. Shameli-Sendi, J. Desfossez, M. Dagenais, and M. Jabbarifar, "A retroactive-burst framework for automated intrusion response system," *J. Comput. Netw. Commun.*, vol. 2013, pp. 1–8, Jan. 2013.

[23] S. Fayyad and C. Meinel, "Attack scenario prediction methodology," in *Proc. 10th Int. Conf. Inf. Technol., New Generat.*, Apr. 2013, pp. 53–59.

[24] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Berlin, Germany: Springer, 2009, pp. 51–70.

[25] V. Conitzer and T. Sandholm, "New complexity results about Nash equilibria," *Games Econ. Behav.*, vol. 63, no. 2, pp. 621–641, Jul. 2008.

[26] V. E. Urias, W. M. S. Stout, and H. W. Lin, "Gathering threat intelligence through computer network deception," in *Proc. IEEE Symp. Technol. Homeland Secur. (HST)*, May 2016, pp. 1–6.

[27] T. Micro. *The Custom Defense Against Targeted Attacks*. (Mar. 28, 2020). [Online]. Available: http://www.trendmicro.fr/media/wp/custom-defense-against-targeted-attacks-whitepaper-en.pdf

[28] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, 2010, pp. 48–61.

[29] A. K. Kaushik, E. S. Pilli, and R. C. Joshi, "Network forensic system for port scanning attack," in *Proc. IEEE 2nd Int. Advance Comput. Conf. (IACC)*, Feb. 2010, pp. 310–315.

[30] M. I. Center, "APT1: Exposing one of China's cyber espionage units," Mandian, FireEye, Milpitas, CA, USA, Tech. Rep., Feb. 2013.

[31] M. Ussath, D. Jaeger, F. Cheng, and C. Meinel, "Advanced persistent threats: Behind the scenes," in *Proc. Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2016, pp. 181–186.

[32] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[33] B. Bauer and K. Karl-Friedrich, "Towards an automatic sign language recognition system using subunits," in *Proc. Int. Gesture Workshop*. Berlin, Germany: Springer, 2001, pp. 64–75.

[34] L. Rabiner and B. Juang, "An introduction to hidden Markov models," *IEEE ASSP Mag.*, vol. 3, no. 1, pp. 4–16, Jan. 1986.

[35] (2020). *Mitre Adversarial Tactics, Techniues, and Common Knowledge*. [Online]. Available: https://attack.mitre.org/techniques/enterprise

[36] *The Evolving Face of Cyber Threats Whitepaper*, IBM, Endicott, NY, USA, 2017.

[37] S.-Z. Yu and H. Kobayashi, "An efficient forward-backward algorithm for an explicit-duration hidden Markov model," *IEEE Signal Process. Lett.*, vol. 10, no. 1, pp. 11–14, Jan. 2003.

[38] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using Bayesian attack graphs," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 1, pp. 61–74, Jan. 2012.

[39] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Secur. Privacy Mag.*, vol. 4, no. 6, pp. 85–89, Nov. 2006.

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

[41] E. Miehling, M. Rasouli, and D. Teneketzis, "A POMDP approach to the dynamic defense of large-scale cyber networks," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 10, pp. 2490–2505, Oct. 2018.

[42] S. Jajodia and S. Noel, "Topological vulnerability analysis," in *Cyber Situational Awareness*. Boston, MA, USA: Springer, 2010, pp. 139–154.

[43] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proc. 4th Robot., Sci. Syst.*, Jun. 2008.

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

[45] E. Talvitie and S. Singh, "Learning to make predictions in partially observable environments without a generative model," *J. Artif. Intell. Res.*, vol. 42, pp. 353–392, Nov. 2011.

[46] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, nos. 2–3, pp. 235–256, 2002.

[47] P. Mell, K. Scarfone, and S. Romanosky, "A complete guide to the common vulnerability scoring system version 2.0," in *Proc. 1st-Forum Incident Response Secur. Teams*, vol. 1, 2007, p. 23.

[48] A. Steinberg, "Open interaction network model for recognizing and predicting threat events," in *Proc. IEEE Inf., Decis. Control*, Feb. 2007, pp. 285–290.

[49] S. J. Yang, H. Du, J. Holsopple, and M. Sudit, "Attack projection," in *Cyber Defense and Situational Awareness*. Cham, Switzerland: Springer, 2014, pp. 239–261.

[50] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A logic-based network security analyzer," in *Proc. USENIX Secur. Symp.*, Baltimore, MD, USA, vol. 8, 2005, pp. 113–128.

[51] Arcsight. *ESM: Enterprise Security Manager [OL]*. Accessed: Jun. 2020. [Online]. Available: http://cn.linkedin.com/topic/enterprise-security-manager

MD ALI REZA AL AMIN received the M.S. degree in computer information and system engineering from Tennessee State University, Nashville, USA, in 2016. He is currently pursuing the Ph.D. degree in computational modeling and simulation engineering with Old Dominion University, Norfolk, USA. His research interests include cyber security, cyber deception technology, and AI in cyber security.

SACHIN SHETTY (Senior Member, IEEE) received the Ph.D. degree in modeling and simulation from Old Dominion University, in 2007. He is currently the Associate Director of the Virginia Modeling, Analysis, and Simulation Center, Old Dominion University. He holds a joint appointment as an Associate Professor with the Department of Computational, Modeling, and Simulation Engineering. His research interests include the intersection of computer networking, network security, and machine learning.

LAURENT NJILLA received the B.S. degree in computer science from the University of Yaoundé 1, Cameroon, the M.S. degree in computer engineering from the University of Central Florida (UCF), in 2005, and the Ph.D. degree in electrical engineering from Florida International University (FIU), in 2015. He joined the Cyber Assurance Branch, U.S. Air Force Research Laboratory (AFRL), Rome, NY, as a Research Electronics Engineer, in 2015.

DEEPAK K. TOSH is currently an Assistant Professor in computer science with The University of Texas at El Paso. His research interests include blockchain technology, cybersecurity, data provenance mechanisms, the security of the Internet of Battlefield Things (IoBT) environments. He has been actively working with researchers from the Air Force Research Laboratory (AFRL), Rome, NY, and the Army Research Laboratory (ARL), Adelphi, MD, on the cybersecurity and blockchain research.

CHARLES KAMHOUA (Senior Member, IEEE) received the B.S. degree in electronics from the University of Douala (ENSET), Cameroon, in 1999, and the M.S. degree in telecommunication and networking and the Ph.D. degree in electrical engineering from Florida International University (FIU), in 2008 and 2011, respectively. He is currently a Researcher with the Network Security Branch, U.S. Army Research Laboratory (ARL), Adelphi, MD, where he is responsible for conducting and directing basic research in the area of game theory applied to cyber security.