Summer 8-2022

Cyber Deception for Critical Infrastructure Resiliency

Md Ali Reza Al Amin
Old Dominion University

CYBER DECEPTION FOR CRITICAL INFRASTRUCTURE RESILIENCY

by

Md Ali Reza Al Amin
B.Sc. 2010, Khulna University, Bangladesh
M.Sc. 2016, Tennessee State University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

ENGINEERING - MODELING & SIMULATION

OLD DOMINION UNIVERSITY
August 2022

Approved by:

Sachin Shetty (Director)

Yuzhong Shen (Member)

Hong Yang (Member)

Chunsheng Xin (Member)

ABSTRACT

CYBER DECEPTION FOR CRITICAL INFRASTRUCTURE RESILIENCY

Md Ali Reza Al Amin
Old Dominion University, 2022
Director: Dr. Sachin Shetty

The high connectivity of modern cyber networks and devices has brought many improvements to the functionality and efficiency of networked systems. Unfortunately, these benefits have come with many new entry points for attackers, making systems much more vulnerable to intrusions. Thus, it is critically important to protect cyber infrastructure against cyber attacks. The static nature of cyber infrastructure leads to adversaries performing reconnaissance activities and identifying potential threats. Threats related to software vulnerabilities can be mitigated upon discovering a vulnerability and-, developing and releasing a patch to remove the vulnerability. Unfortunately, the period between discovering a vulnerability and applying a patch is long, often lasting five months or more. These delays pose significant risks to the organization while many cyber networks are operational. This concern necessitates the development of an active defense system capable of thwarting cyber reconnaissance missions and mitigating the progression of the attacker through the network. Thus, my research investigates how to develop an efficient defense system to address these challenges. First, we proposed the framework to show how the defender can use the network of decoys along with the real network to introduce mistrust. However, another research problem, the defender's choice of whether to save resources or spend more (number of decoys) resources in a resource-constrained system, needs to be addressed. We developed a Dynamic Deception System (DDS) that can assess various attacker types based on the attacker's knowledge, aggression, and stealthiness level to decide whether the defender should spend or save resources. In our DDS, we leveraged Software Defined Networking (SDN) to differentiate the malicious traffic from the benign traffic to deter the cyber reconnaissance mission and redirect malicious traffic to the deception server. Experiments conducted on the prototype implementation of our DDS confirmed that the defender could decide whether to spend or save resources based on the attacker types and thwarted cyber reconnaissance mission. Next, we addressed the challenge of efficiently placing network decoys by predicting the most likely attack path in Multi-Stage Attacks (MSAs). MSAs are cyber security threats where the attack campaign is performed through several attack stages and adversarial lateral movement is one of the critical stages. Adversaries can laterally move into the network without raising an alert. To prevent lateral movement, we proposed an approach that combines reactive (graph analysis) and proactive (cyber

deception technology) defense. 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. The second phase determines the 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. The evaluation results show that our approach can predict the most likely attack paths and thwart adversarial lateral movement.

To the memory of my father, Md. Liakat Ali
My beloved mother, Mrs. Rowshan Ara
My dear wife, Suborna Islam
My dear sons, Mehmet Amin & Arfaan Amin

ACKNOWLEDGEMENTS

First and foremost I am extremely grateful to my supervisor Dr.Sachin Shetty for his invaluable advice, continuous support, and patience during my Ph.D. journey. He is the person from whom I learned how to do good research in terms of reading, writing, and presentation skill. During this journey, I learned to think independently as researcher. I am truly honored that I have him as my Ph.D. supervisor.

I would like to thank my committee members Dr.Yuzhong Shen, Dr.Hong Yang, and Dr.Chunsheng Xin for their valuable comments and time during this journey. I also want to acknowledge the support and assistance of the Virginia Modeling, Analysis, and Simulation Center (VMASC) at Old Dominion University. I am thankful to my Bangladeshi community friends and family for their support during my hard time in my Ph.D. journey. I would like to express my gratitude to my wife and children. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Finally, I would like to acknowledge the support and love from my parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

This chapter provides the background and motivation for the dissertation followed by the main goals of the research. Finally, the basic outline of this dissertation is presented.

1.1 BACKGROUND AND MOTIVATION

1.1.1 BACKGROUND

Given the growing number of cyber attacks, it is imperative to design resilient cyber infrastructure. Organizations face substantial financial losses and challenges in maintaining core public services due to the increasing rate of cyber attacks rate. According to McAfee's recent report in 2018, [2], cybercrime has reached nearly $600 billion. In addition, a recent Verizon Data Breach Investigation Report (DBIR) points out that currently deployed protection mechanisms are not adequate to address current threats [3]. The report states that 66% of the breaches took months or years to discover, rising from 56% in 2012. Furthermore, 84% of these attacks only took hours or less to infiltrate computer systems [3]. Moreover, the report states that only 5% of these breaches were detected using traditional intrusion detection systems (IDSs) while 69% were detected by external parties [3]. These numbers represent the attacks that were discovered. Only 5% of the attacks are discovered using traditional security tools [4], so, it is likely that the number of undiscovered and unreported attacks is significantly higher. It is evident from these findings that the organization's current security posture is not enough to address current threats.

The static nature of any organization's IT system leads to adversaries performing reconnaissance activities and identifying potential threats. The reconnaissance phase aims to collect critical information about the network, including network topology, open ports and services running, and unpatched vulnerabilities. Having that critical information maximizes the intruder's chance of penetrating a system and gaining a foothold successfully. Patching a vulnerability will lower the possibility of being attacked where vulnerability patching depends upon discovering a vulnerability and developing a patch for that specific weakness. Unfortunately, sometimes this period (a vulnerability exposure window) often lasts a long time. This extended period puts the cyber network at higher risk. To address this, one needs an active defense system to thwart cyber

attacks while considering attack information and providing appropriate defense actions. However, there is an issue with developing such a system as an attacker considers a series of exploit to deep dive into the network. Very few targeted cyber attacks consist of only a single vulnerability. It is always beneficial for the defender to know how the intruder can infiltrate the network and capture the attacker's progression throughout the network. With the help of one of the cyber deception technologies (by introducing fake networks), we can alter the view of a system with a mix of real and fake information and make the network more secure when unpatched vulnerabilities are present in the cyber network. Figure 1 shows the life cycle of cyber deception at the conceptual level.

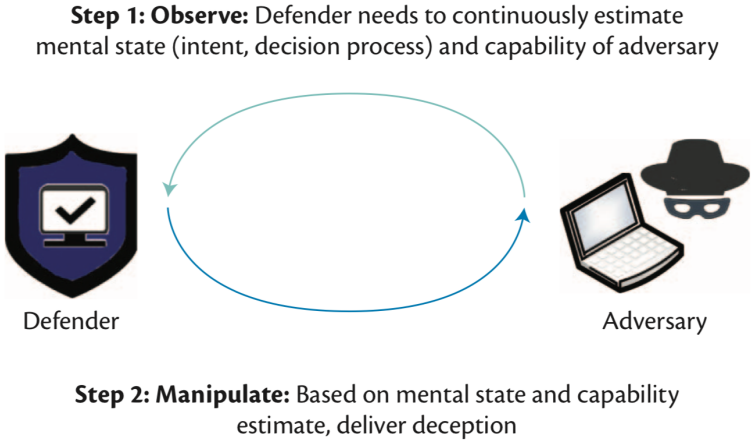


Figure 1: Life cycle of cyber deception [1].

Among all cyber attacks, Advanced Persistent Threat is the most sophisticated cyber attacks. 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 presence in the network. Early detection of adversarial lateral movement can deter an ongoing APT attack. With 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 can the attacker 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 enables the defender to react before the attacker reaches the end goal by executing proactive responses.

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. Deception-based techniques provide significant advantages over traditional security controls. Cyber-deception has attracted attention from security researchers and practitioners as an approach for designing secure cyber infrastructure. Cyber-deception can provide two advantages: a) reduce the likelihood of adversarial success and cost of cyber defense, b) provide insights into attacker's strategies, tactics, capabilities, and intent. Typically, an attacker has apriori knowledge of the infrastructure they are targeting. In a cyber-deception approach, the defender can exploit this apriori knowledge to mislead the adversary in expending their resources and time by deploying a network of decoy targets, leading to attack paths that do not lead to a successful goal. The success of a cyber-deception strategy hinges on adversaries taking on more fake attack paths than actual attack paths that would increase the adversary's resources in distinguishing between real and fake assets.

1.1.2 MOTIVATION FROM LITERATURE REVIEW

Researchers have proposed cyber deception approaches that introduce fake networks by varying system characteristics [5], manipulating attacker's probes [6, 7] and introducing virtual network interface controllers and route mutation [8] . These approaches are focused on introducing fake nodes from an attacker's point of view and assume a static environment and attacker and defender strategies. In [5], authors introduce systems that change the view of a cyber network by obscuring some system characteristics where [6] alters the system view by manipulating attackers' probe. Trassare et al. [7] use a traceroute function to deceive the network reconnaissance attack. Dunlop et al. [9], propose a mechanism where to hide IPv6 packets to achieve anonymity. They added a virtual network interface controller and shared a secret with all hosts.

All these approaches in cyber deception tend to change the network view from an attacker's point of view. However, they all failed to answer the question of what if the attacker enters the network where unpatched vulnerabilities are present, and patches are not released yet. A network administrator cannot just let the attacker compromise the system. As we mentioned earlier, the

attacker always has time-advantage over unpatched vulnerability the where rvulnerability exposure window is high. A defender has to take defensive action while making a tradeoff between availability and security. Our approach not only changes the network view but also influence the attacker to take the path toward fake networks while keeping availability and security at a satisfactory level.

The work described in [10], [11], [12], and [13] uses hidden states for characterizing risk. These approaches learn a single HMM model for any attack type. In [12], authors computed probability matrices, but they did shed details on the probability matrices' computation. Authors in [13] 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, 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.

S.Zonouz et al. [14] 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, we also use the attack graph in our model and handle the execution time by incorporating an exploit dependency graph, which reduces execution time.

Attack graphs were proposed as the first method for predicting cyber attacks [15]. 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 they do not consider the underlying different attack steps the attacker can take. Ramaki *et al*. [16] 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 can not 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 [17], 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 [18] 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 [19] 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.

The authors in [20] 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 and 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 [21], 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 can not 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 [22].

Urias *et al.* [23] 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.

Considering all of the above-discussed drawbacks in different models, in this dissertation proposal, we are proposing a cyber deception approach where the defender can push the adversary towards deployed fake networks. While the adversary in the fake networks, the defender can learn

different techniques, tools, and strategies used by the attacker. These information can help to design more secure cyber network.

1.2 PROBLEM STATEMENT

Three problem statements stem from the above-discussed literature review and motive us to solve by our develop and algorithm to make critical infrastructure more resilient. The main contributions of this dissertation are listed below:

- Given a cyber network where multiple unpatched vulnerabilities are present, how to develop a framework where the cyber administrator can secure the network while balancing the trade-off between the availability cost and security cost.
- Having the deception framework, how to incorporate attacker capabilities into the model so that the defender can decide where to spend resources and save resources.
- How to develop a system where the network defender can predict the likely attack path beforehand so that the decoy networks can be placed along the predicted path to prevent adversarial lateral movement.
- Having the predicted path framework, how to develop a deception-based system to characterize adversarial strategies in a complex networked system to decrease the attacker's win percentage and increase resource usage.

1.3 OUR CONTRIBUTION

Our research starts with developing a deception framework in which a network administrator can deploy a network of decoy assets with the aim to expend adversaries' resources and time and gather information about the adversaries' strategies, tactics, capabilities, and intent. The key challenge in this cyber deception approach is the design and placement of network decoys to ensure maximal information uncertainty for an attacker. State-of-the-art approaches to addressing this design and placement problem assume a static environment, and apriori strategies are taken by the attacker. In this paper, we propose the design and placement of network decoys considering scenarios where a defender's action influence an attacker to change their strategies and tactics during the attacker's progression while maintaining the trade-off between availability and security. The defender maintains a belief consisting of security state, and the resultant actions are cast as Partially Ob- servable Markov Decision Process (POMDP).

Table 1: Quantitative performance comparison

Key criteria	Our work	Jajodia et al.[24]	Chungang et al.[25]
State space	>100 million	<100 million	<100 million
Execution time	online solver	offline solver	offline solver
Security cost	low	high	high

In modern days, cyber networks need continuous monitoring to keep the network secure and available to legitimate users. Cyber attackers use reconnaissance missions to collect critical network information and using that information, they make an advanced level cyber-attack plan. To thwart the reconnaissance mission and counterattack plan, the cyber defender needs to come up with a state-of-the-art cyber defense strategy. In this paper, we model a dynamic deception system (DDS) which will not only thwart reconnaissance missions but also steer the attacker towards a fake network to achieve a fake goal state. In our model, we also capture the attacker's capability using a belief matrix which is a joint probability distribution over the security states and attacker types.

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 determines 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.

For the last research problem, we developed a set of metrics that will provide insights into the level of complexity a sequence of events will impose on an opponent. A set of mixed true and

false information increases an aspect of the complexity of the environment in a way that makes it more difficult for an opponent to make decisions or shape conditions in one's favor. If the attacker believes that all the information received is accurate, the probability that the attack campaign will fail is very high. On the other hand, if the attacker knows that the information is mixed with true and false information, attackers need to spend more time differentiating that information. In summary, we provided the following contributions:

- Developed a suite metrics that quantify the opportunity and capability of the adversary based on based on aggression, knowledge, and stealthiness.
- To quantify more effective evolutionary strategy to aid in selecting effective deception action by using an autonomous cyber attacker agents.

We presented the quantitative comparison of our contributions with the existing cyber deception in Table 1. Our approach can efficiently process state space for more than 100 million. Furthermore, our approach is an online solver for response time, which means the defender can engage with the attacker in real-time and make the defense decision. We also demonstrated that our model can trade-off between availability cost and security cost.

1.4 ORGANIZATION OF THE DISSERTATION REPORT

The rest of this dissertation is organized as follows: in Chapter 2 presents the preliminary concept. The implementation of the initial cyber deception framework is presented in Chapter 3. In Chapter 4, how the attacker capabilities are incorporated into the initial model to decide when the network defender should spend resources and save resources is presented. It is also shown that the model can thwart cyber reconnaissance mission using Software Defined Networking (SDN). The model of preventing adversarial lateral movement by predicting the most likely attack path is discussed in Chapter 5. Finally, how the adversarial strategies can be characterized in the complex networked system is discussed in Chapter 6.

Chapter 2

PRELIMINARIES

In this section we briefly discuss the background of different models and how technology has been leveraged.

2.1 EXPLOIT DEPENDENCY GRAPH

The concept of attack trees and graphs were developed with a goal in mind that one can study all possible sequences of exploits that an intruder can take to infiltrate a network and reach its goal(s) state. An attack graph consists of vertices (system states) and edges (transition relations) where each vertex connects each other via exploits. To generate an attack graph, one has to enumerate all system states. In this process, the attack graph quickly grows exponentially. There are several attack graph applications in network security such as vulnerability analysis, intrusion alert correlation, and attack response system. An attack graph can be applied in both penetration testing and network hardening.

Significant progress has been made in generating attack graph automatically [26], [27], [28]. Along with the network size, attack graphs grow exponentially which makes the visualization nearly impossible for a human to understand what's going on. To deal with this complexity, [29] proposed a system where one can reduce the attack graph information without loss of any generality and create a graph which grows quadratically. Authors in [30] made an assumption regarding the attacker's behavior which allows to simplify the attack graph and also reduce the attack information. The assumption named as monotonicity [30], states that the success of one exploit does not interfere with the attacker's future ability to exploit. With the help of this assumption, one does not need to enumerate all security states, rather can create exploit dependency graph describing how security conditions relate to exploit. The advantage of exploit dependency graph is that it can be easily generated for a large network where the corresponding attack graphs would be obstinately very large to generate. In [30], the authors construct a graph where nodes represent security condition, and edges represent exploit, which termed as exploit dependency graph. Security conditions are the atomic fact that they can be either true or false and exploits relate to security conditions via preconditions and postconditions. The approach taken by Ammann et al. in [30] is similar we adopt in this paper to do the modelling of attack pathways using exploit

dependency graph. The edges in exploit dependency graph relates security conditions in such a way where a single exploit might have multiple preconditions and multiple postconditions. Such edges which are connects two sets of nodes rather than a pair of nodes we called it hyperedges. The security conditions present in [30] are a mix of different attributes which is true under normal network configurations termed as initial conditions. During an attack, attributes can be made true which is attack conditions. With this issue, the initial conditions are set to be always true whether a network is subject to be an attack or not. For this reason, we take a slightly modified definition from [31] where it does not include conditions representing the normal network configuration explicitly rather assume that the set of conditions solely consists of attack conditions. This modification allows setting the conditions of a network false which has not been subject to an attack.

2.2 POMDP APPROACH

A partially observable Markov decision process (POMDP) is a process which connects unobservant system states to observations. POMDP is a combination of a Markov decision process (MDP) to model system dynamics with a hidden Markov model. The reward from the POMDP approach depends on an agent's action and sequence of system state where the agent cannot see the system state directly rather, the agent makes an observation. Based on the observation agent construct a belief state which is a probability distribution over system states. Based on the belief matrix agent call the optimal action for each belief state. The advantage of POMDP is that its general enough to model different kinds of real-world problem such as robot navigation problem, cybersecurity, machine maintenance, and planning issue with uncertainty.

2.3 POMCP FRAMEWORK

In large and fully observable domains, Monte-Carlo Tree Search (MCTS) has tremendous performance in online planning [31]. MCTS is a new approach to do online planning. It overcomes the curse of dimensionality by taking only sample states instead of taking the whole possible system states. MCTS requires a black box to simulate where the problems are too complicated or too large to represent the probability distribution. It has another advantage in terms of prior domain knowledge. In estimation the potential, MCTS uses the random simulation for long-term reward where it plans over the long horizon and often effective in estimation the potential where any prior domain knowledge or heuristics search is not present [32].

The authors in [31] extended MCTS to partially observable environments (POMDPs). Other planning algorithms, i.e., value iteration [8] suffers from two important issues referred to

as scaling and history. For example, for n-states value iteration algorithm creates n-dimensional belief state, and it must evaluate all history which is exponential in the horizon. The search algorithm in [31] constructs a search tree of histories which is online-based. The value of history is estimated by the node of the search tree using Monte-Carlo simulation. The start space in each simulation is sampled from the current belief state, and transition and observations are sampled from the black-box simulator. The authors in [31] showed that for correct belief state the planning algorithm converges to the optimal policy for any finite horizon POMDP. Monte-Carlo simulation also can be used in updating the agent belief state. The important feature of Partially Observable Monte-Carlo Planning (POMCP) algorithm is that it uses the same set of Monte- Carlo simulation for both trees search and belief state.

2.4 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.

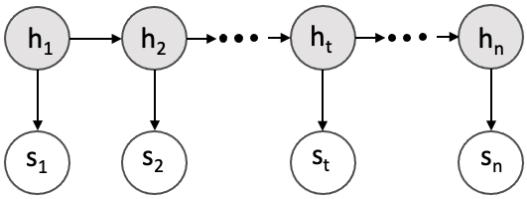


Figure 2: A Bayesian network representing a first-order HMM. The hidden states are shaded in gray.

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 can not 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 [33].

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) [34], Viterbi algorithm solves the problem [34] (2), and Baum-Welch algorithm solves the problem (3) [35].

2.5 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 [35].

2.6 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 initialization step ($t = 1$):

$$\delta_1(i) = \pi_i b_i(o_1) \tag{1}$$

$$\psi_1(i) = 0$$

- The recursion step:

$$\delta_t(j) = \max_{1 \le i \le N} [\, \delta_{t-1}(i) a_{i,j}] b_j(o_t)\,, 1 \le j \le N \tag{2}$$

$$\psi_t(j) = arg \max_{1 \le i \le N} [\, \delta_{t-1}(i) a_{i,j}]\,, 1 \le j \le N \tag{3}$$

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

$$P(T) = \max_{1 \le i \le N} \delta_T(i) \tag{4}$$

$$q_T = arg \max_{1 \leq i \leq N} \delta_T(i) \tag{5}$$

$$q_t = \psi_{t+1}(q_{t+1}) \tag{6}$$

The terms π_i and b_i in Eq. 1 defined as the initial probability and the probability of an observation, respectively. The Viterbi algorithm uses the δ parameter, where it considers only the maximum likelihood value in Eq. 2. It also uses another parameter ψ to keep track of the argument, which maximized δ for each t and j defined in Eq. 3. The term $P(T)$, q_T, and q_t in the Eq. 4, 5, and 6 are defined as maximum probability, best last state, and previous best state, respectively.

2.7 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 BW algorithm first uses the forward-backward algorithm parameters α and β. Then using Bayes' theorem and expectation-maximization [35] introduce 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:

$$\begin{aligned} \xi_t(i,j) &= P(q_t = s_i, q_{t+1} = s_j | O_T, \Lambda) \\ &= \frac{\alpha_t(i) a_{i,j} b_j(o_{t+1} \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)} \end{aligned} \tag{7}$$

Now, the marginal probability over j:

$$\begin{aligned} \gamma_t(i) &= \sum_{j=1}^{N} \xi_t(i,j) \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \end{aligned} \tag{8}$$

Using the above formulas, we can reestimate the parameters of HMM. A set of reasonable reestimation formulas for (A, B, π) are with $\bar{\pi}_i$ being the expected frequency spent in-state s_i at time 1.

$$\bar{\pi}_i = \gamma_1(i) \tag{9}$$

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,

$$\bar{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{10}$$

$\bar{b}_i(o_k)$ is defined by the number of expected transition from state i, when observation is $o_t = o_k$, over the number of expected transitions.

$$\bar{b}_i(o_k) = \frac{\sum_{t=1}^{T-1} \gamma_t(i), when\ o_t = o_k, else\ 0}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{11}$$

Chapter 3

ONLINE CYBER DECEPTION SYSTEM USING PARTIALLY OBSERVABLE MONTE-CARLO PLANNING FRAMEWORK

In this chapter, we propose the design and placement of network decoys considering scenarios where defender's action influence an attacker to change its strategies and tactics dynamically while maintaining the trade-off between availability and security. The defender maintains a belief consisting of security state and the resultant actions are modeled as Partially Observable Markov Decision Process (POMDP). Our simulation results illustrate the defender's increasing ability to influence the attacker's attack path to comprise of fake nodes and networks.

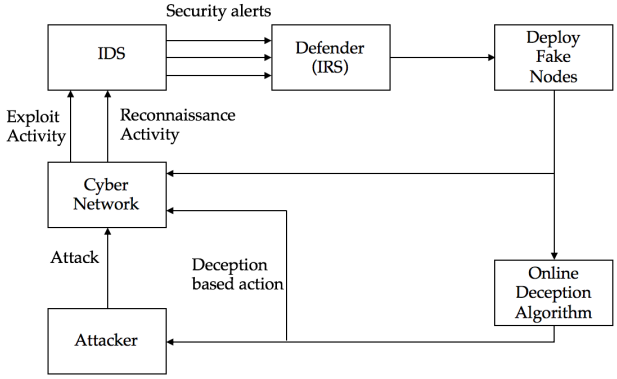


Figure 3: Security model.

3.1 SECURITY MODEL

The proposed security model provides a way how a defender can deceive an attacker with the fake network to prevent the real network infiltration. Throughout the paper Figure 3 will serve as the reference to describe the cyber deception approach. In this section, we demonstrate the characterization of attacker's progression using exploit dependency graph and our model framework.

3.1.1 EXPLOIT DEPENDENCY GRAPH

Exploit dependency graph have been used to model multi-stage attack scenarios that an attacker can launch to compromise services or applications. The exploit dependency graph characterizes exploits in terms of a set of preconditions and postconditions. For a given network,

exploit dependency graph can be generated using TVA (Topological Vulnerability Analysis) [36]. In an exploit dependency graph, nodes represent the security pre/post conditions and edges represent exploits.

3.1.2 DEPLOY FAKE NODES

We deploy fake nodes/networks along with the real nodes/networks whenever a reconnaissance alert is received from IDS. Attacker can certainly differentiate real nodes from fake nodes by analyzing round trip times or measured bandwidth on the link. We follow delay and bandwidth handler methods to ensure consistency of the network measurements collected during reconnaissance mission [37].

3.1.3 POMDP MODEL

State Space: An exploit dependency graph[38], a directed acyclic hypergraph (H), consists of nodes and hyperedges where nodes represent a set of security conditions (c) and hyperedges render a set of exploits (e).The security condition of the hypergraph have two options either true or false where true means attacker have certain capability and opposite. The node which defender wants to protect termed as goal node denoted by $N_r^g \subseteq N$, $N_f^g \subseteq N$ where N_r^g and N_f^g are real network goal node and fake network goal node respectively. Defender's goal is to protect the goal nodes (real nodes) and drive the attacker towards the fake nodes. Each hyperedge has two conditions in terms of exploits (e_i) termed as pre (N_i^-) and post (N_i^+). It is assumed that attacker can exploit e_i if all preconditions $j \in N_i^-$ are enabled [38]. There will be entry points for the attacker to penetrate the network without having the prior capabilities termed as initial exploits. An attacker can increase the capability set by exploiting more vulnerabilities. Whenever an exploit's attempt is successful all the postconditions of the exploit become successful.

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 $N_i^-, N_i^+ \subseteq s$ [38] and set $S = \{s_1,...,s_n\}$ represents the state space for this model.

Action: Defender's action influence an attacker to choose a network path. So, we assume that the defender can change its network configuration on the fly based on attacker's action to prevent vertical movement. A simple mean of network configuration can be blocking a port from further communication, which will inhibit the attacker to progress. Another way to prevent attacker's progression is to apply countermeasure of any discovered exploit.

The space of defender's available action set is represented by U = $\{u^0, u^1,...,u^n\}$. Here, u^0 represents defender's null action which eventually means defender will not block any exploit.

The remaining actions from the set U, signifies the network changes which will induce a set of blocked exploits. Each action associated with the set of blocked exploit influence the attacker to seek the available paths. Again, defender can not block any individual vulnerability rather it must select a defense action which will induces a set of blocked exploits.

Threat Model: We construct the model based on a single attacker who is attempting to penetrate the network. An attacker can only increase its capability by exploiting more vulnerabilities, on the other hand, it also increases the chance of being detected. Defender's goal is to prevent the exploitation of a vulnerability on the real network by allowing the exploitation on the fake network. The set of available exploits for real and fake network at a given state s can be defined as [38]:

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

$$E(s_t = s) = \{ef_i = (N_i^-, N_i^+) \in |N_i^- \subset s, N_i^+ \not\subseteq s\} \tag{13}$$

There are 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 [38].

The attack probability which defines attacker will attempt each real network exploits while security state s_t and defense action u_t for a given exploit $er_k \in E$ is given by,

$$P_{er_k}(s_t, u_t) = \left\{ \begin{array}{ll} \overline{P}_{er_k} & when\ er_k \in E(s_t) \backslash B(u_t) \\ \underline{P}_{er_k} & when\ er_k \in E(s_t) \cap B(u_t) \\ 0 & when\ er_k \notin E(s_t) \end{array} \right\} \tag{14}$$

similarly for fake network,

$$P_{ef_k}(s_t, u_t) = \left\{ \begin{array}{ll} \overline{P}_{ef_k} & when\ ef_k \in E(s_t) \backslash B(u_t) \\ \underline{P}_{ef_k} & when\ ef_k \in E(s_t) \cap B(u_t) \\ 0 & when\ ef_k \notin E(s_t) \end{array} \right\} \tag{15}$$

In above equations, $\overline{P}_{er_k}$ & $\overline{P}_{ef_k}$ represents the probability of attack when there is no action and $\underline{P}_{er_k}$ & $\underline{P}_{ef_k}$ defines the attack probability when defender's action block exploits. To block any vulnerabilities defender will choose the action, $u \in U$, from the set of accessible defense actions represented by U for any given iteration. Attacker always try to create a set of available initial exploits from reconnaissance state to penetrate the network. There will be a conditional probability of success for each the exploit attacker attempted So, for any given exploits, er_k & ef_k, the

probability of success is given by,

$$\alpha_{er_k}(s_t, u_t) = \left\{ \begin{array}{ll} \overline{\alpha}_{er_k} & when\ er_k \notin B(u_t) \\ 0 & when\ er_k \in B(u_t) \end{array} \right\} \tag{16}$$

similarly for the fake network,

$$\alpha_{ef_k}(s_t, u_t) = \left\{ \begin{array}{ll} \overline{\alpha}_{ef_k} & when\ ef_k \notin B(u_t) \\ \underline{\alpha}_{ef_k} & when\ ef_k \in B(u_t) \end{array} \right\} \tag{17}$$

As soon as, the exploit attempts are successful it enables all the postconditions which eventually form the updated security state as shown in Figure 4.

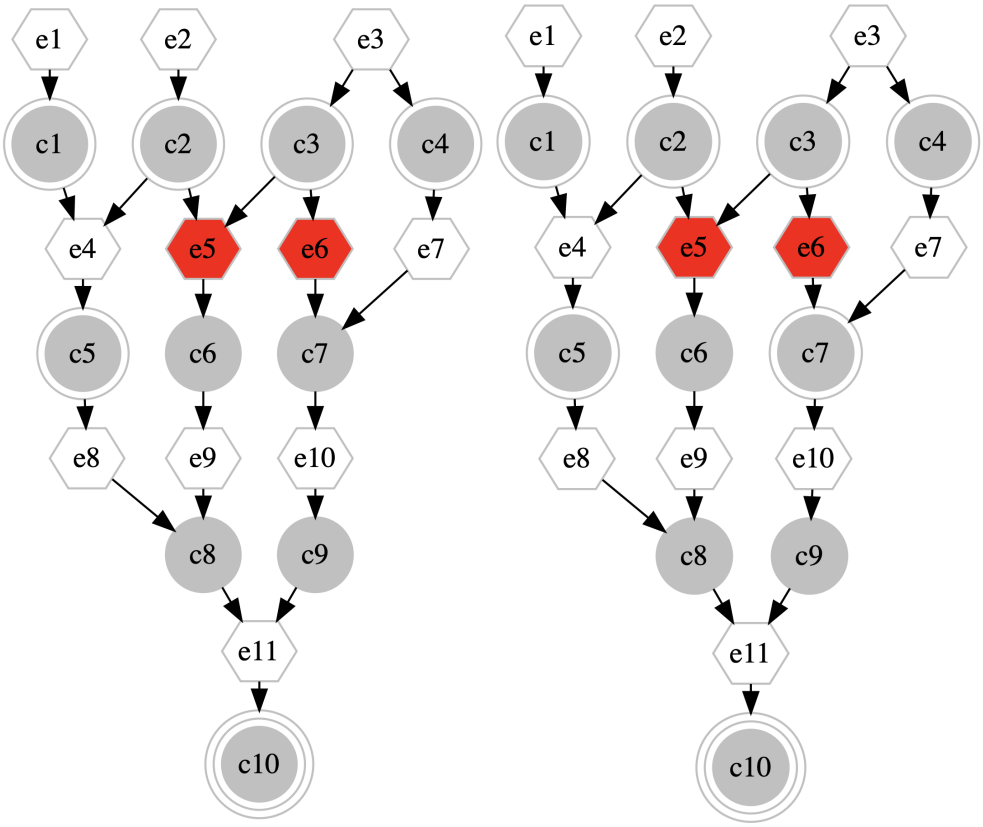


Figure 4: Sample evolution (real network) of the security state.

Defender's lack of information regarding the current security state and attacker true strategy which can be learned from noisy security alerts. In the next section, we describe how defender uses those information to construct the belief by getting security alerts from Intrusion Detection System (IDS). These security alerts are mixed of false positive and false negative alerts. For defender, it is important to differentiate those mixed alerts for better defense actions.

Defender's Observation: Defender's efficiency in terms of choosing actions to limit the attacker progression can be improved by correlating the alert with exploit activity. On the other hand, defender's efficiency can be degraded in the presence of a high rate of false alarm. To get rid of false alarm and precisely quantify attacker progression from the dependency graph we used state-based approach. From the security state, defender can create an available set of exploits to the attacker using Eq. 12 where defender can update the belief state with new security information by weighing the likelihood of individual security state. If we consider an example, for the belief matrices $b, b^{'} \varepsilon \Delta S$, some security state, $s_t \varepsilon S$ and assumed attacker type let one single exploit is available $E(s_t) = \{e\}$ and assumed that if the attacker attempt to exploit e it will create a unique security alert z where no other alert will match with alert z. So, if the defender posses belief b and see the alert z, then the belief update allows for the possibility of generated alert is coming from exploit e. Otherwise, if the defender's belief reflects $b^{'}$ and get the alert z it will immediately discard the alert as a false alarm.

Let $Z = \{z_1, z_2, ..., z_n\}$ and $Z^{'} = \{z_1^{'}, z_2^{'}, ..., z_n^{'}\}$ represent the finite set of security alerts, real and fake network respectively, generated by the IDS which is eventually the observation set for the defender. Each of the alert from real nodes set and fake nodes set can be generated by the IDS, given by the set $Z(e_{ri}) = \{z_{A_i(1)}, z_{A_i(2)}, ..., z_{A_i(ai)}\} \in P(Z)$ and $Z(e_{fi}) = \{z_{D_i(1)}, z_{D_i(2)}, ..., z_{D_i(di)}\} \in P^{'}(Z)$ where $P(Z)$ and $P^{'}(Z^{'})$ are the power set of Z and $Z^{'}$. Using this security alerts defender constructs a belief, $b_t \in \Delta S$, where ΔS is the space of probability distribution over security state. The belief state specifies the probability of being in each state given the history of action and observation experienced so far, starting from an initial belief b_0 and belief update procedure is given in the next section.

Defender's Belief Update: For any defense action $u_t = u$ and observation $y_{t+1} = y_k$, the belief update is defined as $b_{t+1} = [T_j(b_t, y_k, u)]_{s_j \in S}$ where $(j)^{'}th$ is the update function, $T_j(b_t, y_k, u) = P(S_{t+1} = s_j \mid U_t = u, Y_{t+1} = y_k, B_t = b_t)$ is given by [38],

$$b_{t+1}^{j} = T_j(b_t, y_k, u) = \frac{p_j^u(b_t) r_{jk}^u(b_t)}{\rho(b_t, y_k, u)} \tag{18}$$

The above terms are defined below,

$$p_j^u(b_t) = P(S_{t+1} = s_j \mid U_t, B_t) = \sum_{s_i \in S} b_t^i \; p_{ij}^u \tag{19}$$

$$r_{jk}^u(b_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t, B_t) = \sum_{s_i \in S} b_t^i \; r_{ijk}^u \tag{20}$$

$$\rho(b_t, y_k, u) = P(Y_{t+1} \mid U_t, B_t) = \sum_{s_j \in S} r_{jk}^u(b_t) \; p_j^u(b_t) \tag{21}$$

where p_{ij}^u is the transition probability from state s_i to s_j under defense action u, and $r_{jk}^u(b_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t = u, B_t = b_t)$ is the probability that IDS will generate observation vector y_k when transitioning from state s_i to state s_j under a defense action u. Eq. 19 defines the trajectory of beliefs based on security alerts termed as observations and series of actions. 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 Eq. 21, each event in the set of exploit is in binary form (successful and unsuccessful).

The belief update procedure is a controlled Markov Chain where control is defender's action [38]. The majority of POMDP planning methods operate under Bayes theorem [39]. For a large scale cyber network, a single Bayes update procedure could be computationally infeasible. To plan efficiently for large-scale POMDP, we adopted the model described in [31] for the approximation of the belief state.

3.2 DEFENDER'S ACTIONS

As soon as the attacker progress through the network defender will take action to limit the attacker's progression. Selection of action step can be improved if defender have some domain knowledge beforehand. To aid with the domain knowledge, we introduce the utility function. Before taking any defensive action it is also necessary to measure the impact on availability and security cost.

3.2.1 UTILITY FUNCTION

Attacker build an array of node utility function based on the base score metrics for exploiting vulnerabilities [40]. For every exploit, the attacker uses the metrics to justify the attack success probability which is illustrated in Eq. 24 and serves as the attacker's initial knowledge about the network and vulnerability. Defender also create the same utility array. From [40], we

borrow the impact (I), and exploitability (V) metrics to define the defender's utility.

$$I = a * (1 - (1 - CI) * (1 - II) * (1 - AI)) \tag{22}$$

$$V_i = b * AC * AI * AV \tag{23}$$

The above terms are defined as CI = ConfImpact, II= IntegImpact, AI = AvailImpact, I = Impact, V_i = Exploitability, AC = AccessComplexity, AI = Authentication and AV = Accessvector, a, b are the constant parameters and the values are 10.41 and 20 respectively. The range of the Eq. 22 and Eq. 23 value is 0 to 10 where 0 represents low and 10 represents high. Similarly, the range for the utility array function, Eq. 24 is 0 to 100 where 100 means high score. The utility array function is defined below

$$U_{a(r,f)} = I * V_i \tag{24}$$

Example 1: Consider a scenario where there are five nodes and attacker sends scan queries to the neighbors of node 1. Defender needs to respond the scan queries deceptively by mixing of true/false information at random. Here, 2, 3 are real nodes and 4, 5 are fake nodes having following vulnerabilities $vul(n_2)$, $vul(n_3)$, $vul(n_4)$ and $vul(n_5)$. Defender wants to drive the attacker towards node 4 and 5. We are assuming that using above utility array equation defender come up with following values $U_a(n_2) = 15$, $U_a(n_3) = 5$, $U_a(n_4) = 30$, and $U_a(n_5) = 50$. A true rational attacker will go after node 5.

3.2.2 COST FUNCTION

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 as intelligence information on the adversary which helps to derive the attacker's capability, intentions, and targets etc., 2) defender can maximize the network availability to the trusted user during a cyber attack. An availability cost c_a for each action defender could take to drive the adversary towards the fake network. For some defense actions there will be no impact on the availability, and sometimes there will be a greater 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 for the security cost $c_s : S \times U \rightarrow \mathbb{R}$ to depict the cost while the system is in various security state under defense action u. Here, we are considering the availability of a node regarding end-to-end packet delay (considering IT system). If the delay exceeds the limit, the node will still available but legitimate

users could not able take services. For instance, the delay in practice can be the time it takes a user to begin to interacting with the page, or the time it takes to completely load the whole content of the page, which defines the availability factor.

End-to-End Packet Delay

Packet starts journey from a host (source), passes through a series of routers and ends it journey in another host (destination) [41]. Let's assume that, d_E and N represents total delay and number devices between a source and destination. The end-to-end delay defined in [42] as

$$d_E = N(d_{proc} + d_{trans} + d_{prop} + d_{queue}) + d_{proco} \tag{25}$$

The above equation's terms are 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 source and 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 [41]. So that, Eq. 25 can be reduced to

$$d_E = N \times d_{trans} \tag{26}$$

where $d_{trans} = L/R$, L = packet size and R = transmission rate.

For every defense action 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 defender's goal is to keep away the attacker from achieving the goal. The total cost in terms of a security state and defense action is given below

$$c(s_t, u_t) = (1 - f)c_s(s_t, u_t) + f * d_E(u_t) \tag{27}$$

Here, f, is 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). The proposed online deception algorithm, is based on an existing online solver [31], computes optimal action from deception standpoint to deceive attacker with fake network while balancing availability and security cost.

Algorithm 1 Defender's Belief Update Algorithm

Initialize: n_k, $\mathfrak{B}_{t+1} = U_{a(r,f)}$, numAdded = 0

procedure BELIEFUPDATE($\mathfrak{B}_t, u_r, y_r$)
  while $numAdded < n_k$ **do**
    $(s)\sim \mathfrak{B}_t$
    $(s^{'} y, -) \sim G(s, u_r)$
    if $y^{Z(s)} = y_r^{Z(s)}$ **then** [If alerts Z(s) match]
      $\mathfrak{B}_{t+1} \leftarrow \mathfrak{B}_{t+1} \cup \{s^{'}\}$
      $numAdded \leftarrow numAdded + 1$
    end if
  end while
end procedure

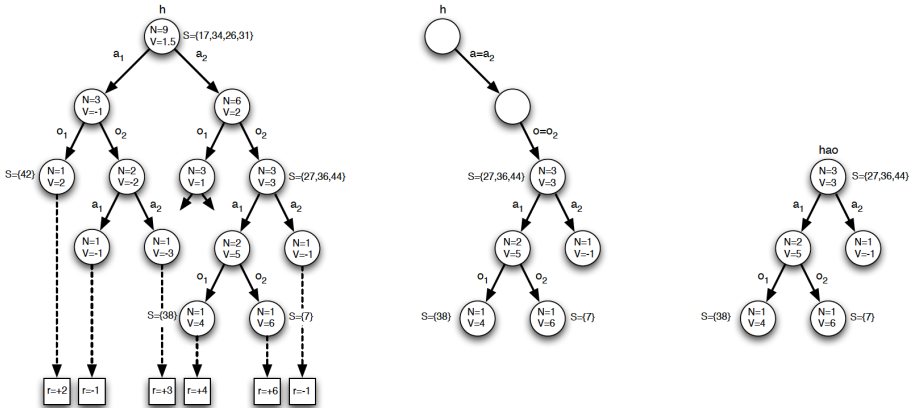


Figure 5: An illustration of POMCP in an environment.

3.3 ONLINE DECEPTION ALGORITHM

Although embedding state space on the dependency graph allows us to accurately quantify the level of progression of the attacker but still computing the optimal deceptive action is a challenge where high diemsionality [43] of deception actions are present while interacting with the attacker. Offline POMDP solver is a way to compute the optimal action for each belief state before runtime. Although such solvers have improved their efficiency, capturing the optimal action can be intractable for large networks. To resolve this issue, Silver and Veness [31] developed an online algorithm termed as Partially Observable Monte-Carlo Planning (POMCP) to handle large-scale network while computing optimal action. Online methods interleave the computation and execution (runtime) phases of a policy [38], yielding a much more scalable approach than

offline methods.

POMCP algorithm is based on and makes use of POMDP [44]. There are two types of nodes in POMCP: belief nodes and action nodes where belief nodes represent a belief state and action nodes are the children nodes of belief nodes that can be reached by doing an action. In this work, action selection procedure is as same as POMCP algorithm described in [31] and belief update procedure is modified based on [38] where it solves the large observation space problem. Modified belief update procedure is given in Algorithm 1, where $\mathfrak{B}_t$ is a state-action pair named particles. The action selection step involves Monte-Carlo simulation from the current belief state to assess the quality of various deception action. An agent begins the simulation by calling a generative model provides a sample successor state, observation and cost given a state and action, $(s^{'}, y, c) \sim G(s, u)$. Calling generative model and successive sampling from the current belief creates histories of search tree presented in Figure 5. *Monte-Carlo Tree Search (MCTS)* uses Monte-Carlo simulation for assessing search tree nodes [45]. In the search tree nodes represent histories and branches from the node in forwarding direction represents the possible future histories because of having partial observability of the fundamental process. A simpler version of MCTS uses greedy tree policy in the very beginning of the simulation, where it selects the action with the highest value. To improve the greedy action selection, UCT algorithm [46] is used. In the search tree, each action selection is done using UCB1 [47] and state is being viewed as multi-armed bandit rule to balance the exploration and exploitation. In the UCT algorithm, there is an option to use the domain knowledge [46] to initialize the new nodes. We use the utility array function U_{ar} as our initial domain knowledge which is improved during more simulation runs. The optimum action for the defender while interacting with the attacker turns into a POMDP. Casting optimum action is defined as below,

$$\begin{aligned} V^{\pi}(b_0) &= \sum_{t=0}^{\infty} \gamma^t c(b_t, u_t) \\ &= \sum_{t=0}^{\infty} \gamma^t E[c(s_t, u_t) |\, b_0, \pi] \end{aligned} \tag{28}$$

where $0 < \gamma < 1$ is the discount factor and $c(b_t, u_t)$ represents the cost for each belief state b_t when an action u_t is selected from the space of action where $c(b_t, u_t) = \sum_{s_i \in S} b_t^i c(s_t, u_t)$. For each belief state, defense action generates according to the policy function and belief update must follow the

procedure defined in Eq. 28. The optimal policy π^* is obtained by optimizing the long-term cost.

$$\pi^* = \arg\min_{\pi} V^{\pi}(b_0) \tag{29}$$

The optimal policy defined in Eq. 29 specifies the optimal action for each belief state $b_t \in \Delta S$ where the expected minimum expected cost calculated over the infinite time horizon. The defender will chose the action where the cost makes trade-off between availability and security cost.

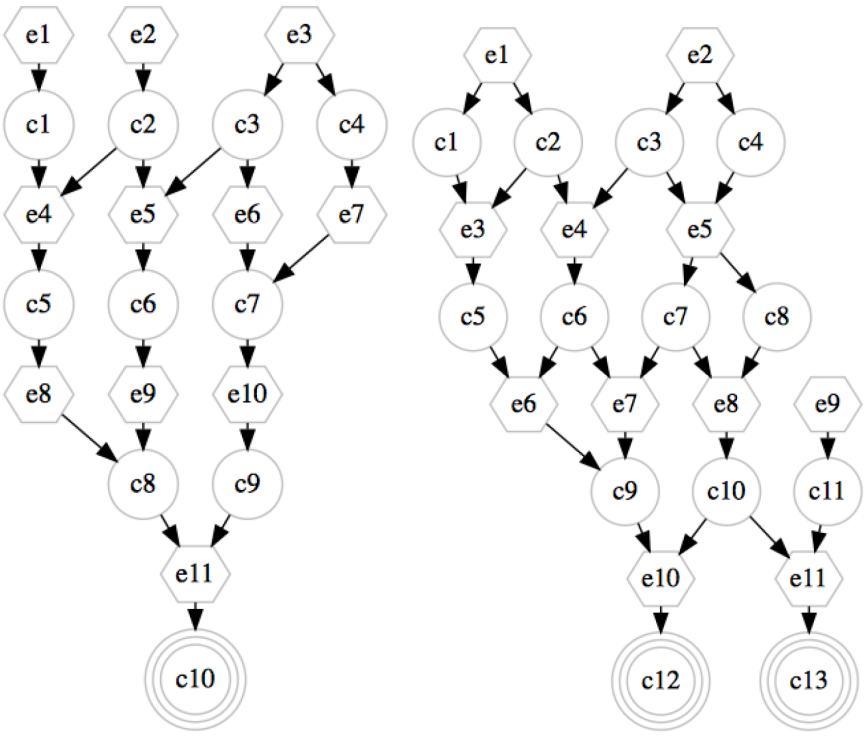
In POMCP, a belief state updates when a sample observation matches with real-world observation, but for large observation space, it barely matches with real-world observation. In the modified belief update procedure presented in Algorithm 1, check a statement whether each incoming alert $z_i \in Z$ match with over a security state, $Z(s) = Z(e)$. The alerts are generated whenever an attacker attempts an exploit. Alerts not in $Z(s)$ cannot be generated by exploit activity for that security state. We are referring those alerts are false alarms for the defender.

To evaluate the scalability of our approach, we experimented our online deception algorithm on a graph consisting 160 conditions (nodes), 150 exploits (hyperedges), 60 defense actions, 35 security alerts resulting more than 10^9 observation vectors. The resulting security states from this example exceed 100 million.

3.4 EXPERIMENTAL RESULTS AND DISCUSSION

In this simulation, we assume that attacker is a true rational type where his aggression, knowledge, and stealthiness are moderate, high and high respectively. Using the state-based alert correlation we creates a probability table for alert detection with assumed attacker type where column represents exploit activity and rows are triggered alert. The probability of detection table is not presented here due to high volume of dataset.

Under null defender's action, the probability of attacking real nodes and fake nodes are same. For this simulation, we assume that the exploit dependency graph is already generated using TVA (Topological Vulnerability Analysis) [38]. We use the [48] software package to use the POMCP solver in our simulation and use python and MatLAB to implement our model. Attacker's progression depends on the defender's action and we assume that defender moves first with null action and wait for the attacker to proceed. In this simulation, we use the sample exploit dependency graph presented in Figure 6 to evaluate our approach and present our simulation results. Attack probabilities for each of the exploit under assumed true rational attacker type,



(a) Real Network (b) Fake Network

Figure 6: A sample Exploit Dependency graph with real network (left) and fake network (right).

$$
\begin{aligned}
(\overline{P}_{er_k}, \underline{P}_{er_k} &= (0.8, 0.3) \quad for \ er_k \in E_0 \\
(\overline{P}_{ef_k}, \underline{P}_{ef_k} &= (0.8, 0.3) \quad for \ ef_k \in E_0 \\
(\overline{P}_{er_k}, \underline{P}_{er_k} &= (0.7, 0.3) \quad for \ er_k \in \{e_4, e_5, e_6, e_8, e_9\} \\
(\overline{P}_{ef_k}, \underline{P}_{ef_k} &= (0.9, 0.7) \quad for \ ef_k \in \{e_5, e_7\} \\
(\overline{P}_{er_k}, \underline{P}_{er_k} &= (0.6, 0.4) \quad for \ er_k \in \{e_7, e_{10}, e_{11}\} \\
(\overline{P}_{ef_k}, \underline{P}_{ef_k} &= (0.9, 0.8) \quad for \ ef_k \in \{e_7, e_8\}
\end{aligned}
$$

similarly, probability of success are,

$$
\alpha_{er_k} = \left\{ \begin{array}{ll} 0.7 & when \ er_k \in E_0 \\ 0.5 & when \ er_k \in E \backslash E_0 \end{array} \right\}
$$

$$\alpha_{ef_k} = \left\{ \begin{array}{ll} 0.85 & when\ ef_k \in E_0 \\ 0.7 & when\ ef_k \in E \backslash E_0 \end{array} \right\}$$

As we defined earlier, the space of actions is the power set of each defense action. In this simulation, we consider three actions for real network which induce a set of block exploits defined as, $B(u^1) = \{e_1, e_2, e_3\}$, $B(u^2) = \{e_4, e_5, e_6, e_7, e_8\}$, $B(u^3) = \{e_9, e_{10}, e_{11}\}$. Similarly for fake network two actions, $B(u^1) = \{e_5, e_7\}$, $B(u^2) = \{e_7, e_8\}$ where the cost of each action is 0.30. The weight cost in Eq. 27 is 0.5 and the discount factor γ = 0.95. In total (real & fake) there are $n_s = 356$ security states and $n_z = 12$ security alerts leading to $2^{12} = 4096$ distinct observation vector. To approximate the belief, all simulations use particles n_k = 1500. The sample evolution of computed deception policy when $N_{Sim} = 5000$ is given in Figure 7, 8. The computed deception policy is intuitive.

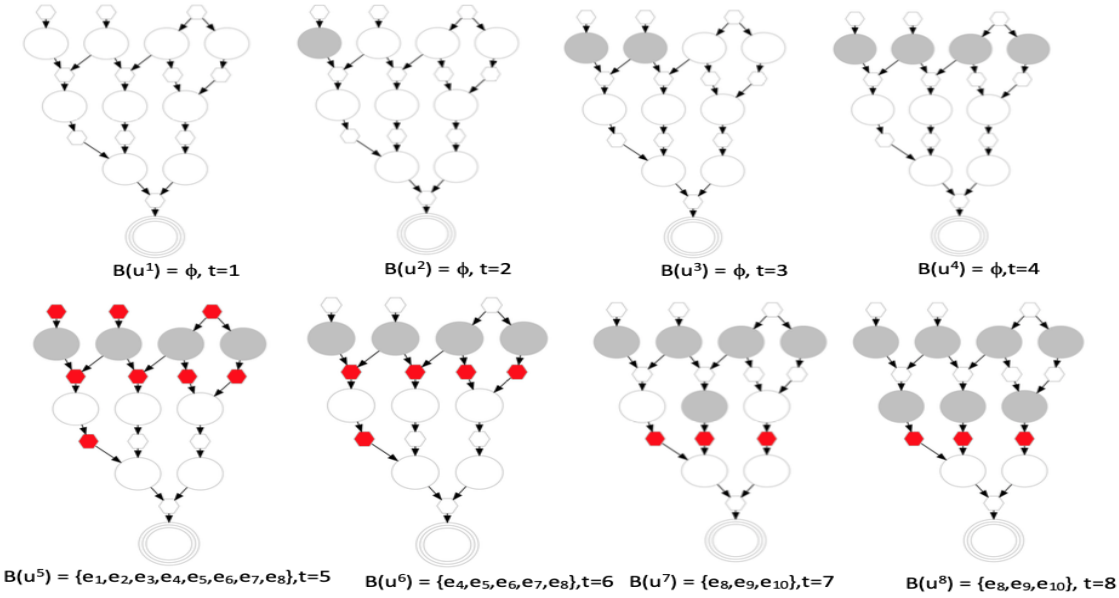


Figure 7: Sample evolution of deception policy when attacker is in real network.

It is assumed that the security state starts from the empty state defined as, $s_0 = \phi$. The defender uses utility array function to construct the initial belief which is defined in Eq. 24. We run the simulation 5000 times. The defender initially (from t=1 to t=4) does not take any action to save the availability cost. As the attacker progress and enable more conditions, defender's belief gradually updates based on the received security alerts.

Then defender begins to deploy actions (t=5) to block exploits. As we know from monotonicity assumption, once a security condition enabled it remains enable all the time. Whenever

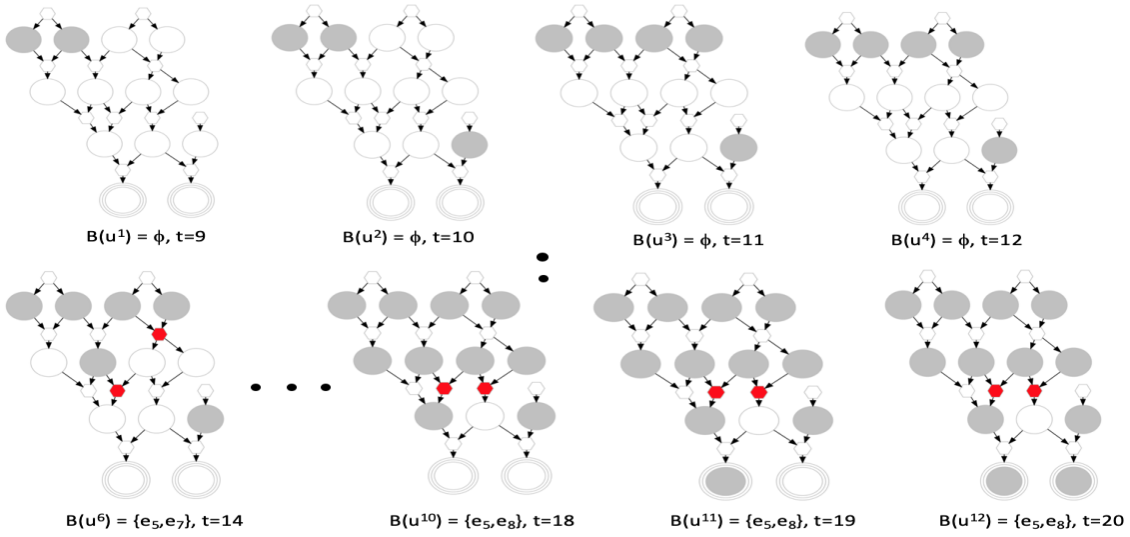


Figure 8: Sample evolution of deception policy when attacker is in fake network.

Table 2: Deception's system performance evaluation table for real network

Simulation Runs	Attacker Starts with Real Node	Attacker Ends on Real Node	Attacker Ends on Fake Node
500	15	13	2
1000	13	10	3
1500	11	7	4
2000	10	6	4
3000	8	3	5
4000	7	1	6
5000	6	0	6

defender's belief reflects that attacker is close to goal conditions, it will block the exploits to prevent the attacker reach the goal. As we can see from Figure 7 at time step t=8, defender blocks exploits $\{ e_8, e_9, e_{10}\}$ which prevents attacker to move forward. From this point, the attacker will try to progress from another point as he received the response from the defender in the reconnaissance stage with a mix of true and false information. Then he moves toward the fake network, Figure 8, based on his available set of exploits dictated by Eq. 21. At this stage defender let the attacker move forward. From time step t=9 to 13, defender action is null. As it (fake) is same as the real network from the attacker perspective, the defender will take action only when attacker have an alternative way to reach the next security state (see time steps t=14-20 in Figure 8). This

Table 3: Deception's system performance evaluation table for decoy network

Simulation Runs	Attacker Starts with Fake Node	Attacker Ends on Fake Node	Attacker Ends on Real Node
500	10	10	0
1000	12	12	0
1500	14	14	0
2000	15	15	0
3000	17	17	0
4000	18	18	0
5000	19	19	0

way attacker will have more confidence that he is in the right track and ultimately he gains nothing. On the other hand, the defender can save more availability cost and learn the attacker which will increase defender's certainty about attacker.

In Table 2, we presented our deception system's performance evaluation data while attacker start to exploit real initial nodes vulnerability and ended up with real to real network end state and real to fake end state. The numerical numbers in the 2nd column represent how many times out of 25 sample runs attacker start with real network initial nodes and 3rd column represents how many times attacker ended up with real network end state without transition to the fake network and 4th column represents how many times attacker make transition from real network to fake network and end up with fake goal state.

In Table 3, we presents the same statistics for the fake network.

From Table 3, we can see that up to 76% of the time attacker start with the fake initial nodes and carry out the series of exploit to achieve the fake goal state. When the N_{Sim} = 500, out of 25 sample runs 15 times attacker start with the real network (Table 2) and 13 times ended up with real network goal state because of poor quality of possible future histories estimation. When the number of simulation increases and more possible future histories are taken into account, the action estimation quality increased as well as policy function (e.g. N_{sim} = 5000, 19 times out of 25 times attacker start and ended up with fake goal state).

In Figure 9, we plot the discounted cost against each time step for 25 sample runs while attacker in real network state. When N_{sim} = 500, 15 times attacker starts with the real network where out of 15 times attacker reached the real goal state (node) 13 times. Trajectories which ended up with red circle represents the path where attacker reached the goal. Initially, for low simulation counts e.g., $N_{sim} = 500$ defender does not have much information about attacker's

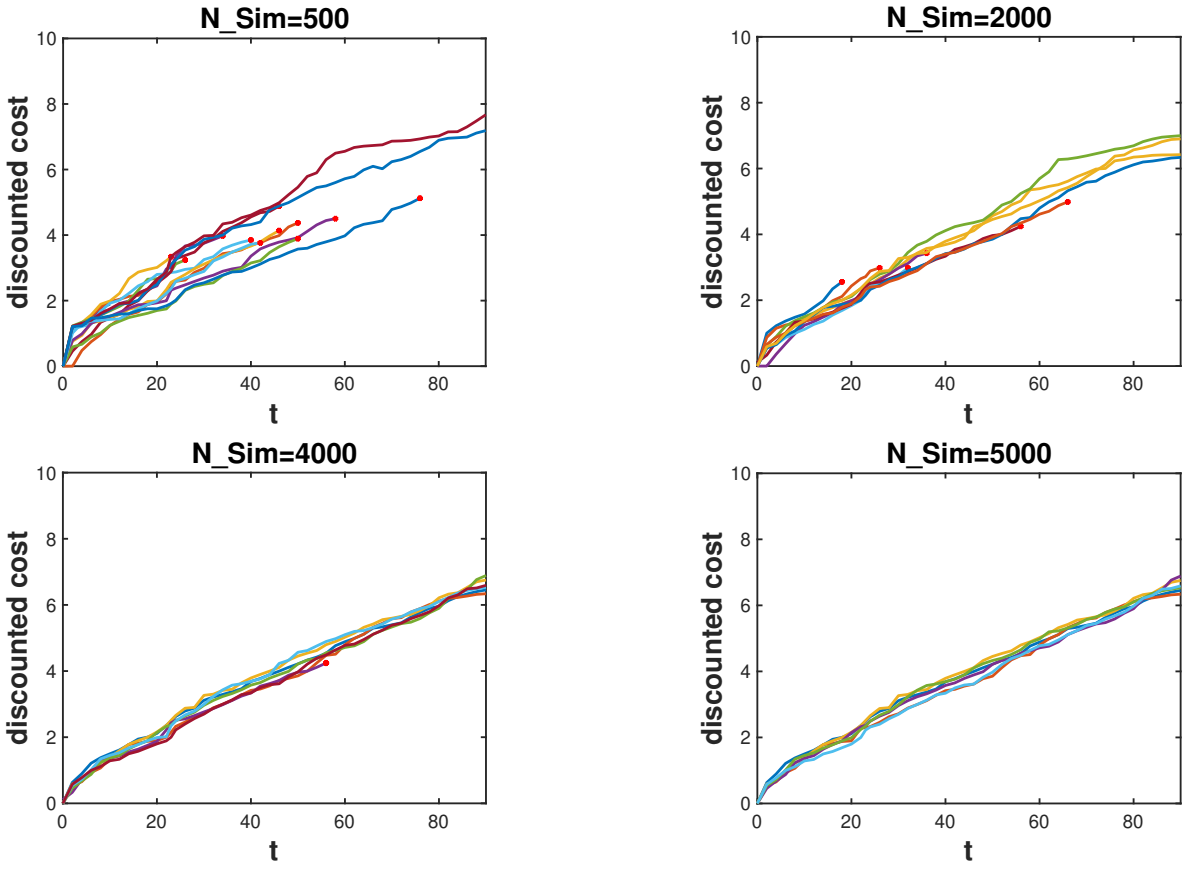


Figure 9: Discounted cost

strategy, capability. Because of this, defender aggressively blocks exploit from the very beginning ($t = 0$) which eventually produces low quality of estimation and ended up with less availability. For poor estimation, attacker also reaches into the goal node several times as shown in Figure 9 upper left corner. As soon as, simulation count increases more possible future histories are included which results high quality of estimation (which set of exploits to be blocked). As it is evidenced from Figure 9 lower right corner, though attacker start with real network for 5000 trials but could not able to reach any goal state.

3.5 CONCLUSION

In this chapter, we develop a technique to alter the view of the system with a mix of real and fake information and a cyber deception approach where defender's action influence an attacker to take different attack path while maintaining availability cost and security cost. To do so, we use an exploit dependency graph which describes attacker progression throughout the network. For every cyber defense system, there is a goal to maintain the availability to the trusted user while security is at a satisfactory level. Scalability is achieved via online deception algorithm where it samples from large-scale cyber domain instead of creating the entire state space. Using our approach, a defender not only saves availability cost but also can learn the attacker while the attacker is in fake networks. This knowledge will help the defender to make better security planning in the future. One future research direction concerns modeling attacker type so that defender can precisely identify the attacker type based on an attacker knowledge, stealthiness and aggression level. Having the capability of identifying an attacker type will improve the efficiency of a defender to choose the deception algorithm. For example, if the attacker's knowledge, stealthiness and aggression level is low, low and low respectively and the defender is deploying the deception algorithm for high, high and moderate attacker, then the defender is wasting the available resource which is not an efficient way to deal with the attacker.

Chapter 4

ATTACKER CAPABILITY BASED DYNAMIC DECEPTION MODEL FOR LARGE-SCALE NETWORKS

In this chapter, we model a dynamic deception system (DDS) which not only thwart reconnaissance mission but also steer the attacker towards fake network to achieve a fake goal state. In our model, we also capture the attacker's capability using a belief matrix which is a joint probability distribution over the security states and attacker types. Experiments conducted on the prototype implementation of our DDS confirm that the defender can make the decision whether to spend more resources or save resources based on attacker types and thwart reconnaissance mission.

4.1 THREAT MODEL AND ASSUMPTIONS

The model is based on a single attacker who is trying to penetrate the network where we are going to capture the attacker's capability. Without considering the attacker's capability, a security model is a waste of resource or lack of resource. Based on the attacker's capability, the defender is going to block vulnerabilities to thwart the attacker and drive the attacker towards the fake network. The defender is able to be blocking exploits by doing system modification. Those system modifications have an effect on normal system operation. This is why the defender needs to estimate the true attacker's capability. For a novice attacker, might be it is sufficient to apply some countermeasure rather than blocking a vulnerability. In our previous chapter, we assumed attacker capabilities; however, in this paper, we incorporated attacker capabilities to do the dynamic security model which is presented in Figure 10

There are two main primary objectives of our dynamic security model i.e., 1) quantify the security state and, 2) taking the optimum deception action based on the attacker capabilities. To quantify the security, we define the security state as a current level of attacker progression. To capture the attacker progression, we use an exploit dependency graph based approach which is described in the earlier chapter.

The strategy attacker will take solely depends on attacker capability. To model attacker types we assume an attacker will be one of n types which are represented by the set $\Phi = \{\varphi_1, \varphi_2, ..., \varphi_n\}$. Each type of attacker $\varphi_i \in \Phi$ will have the conditional attack probabilities (CAP) over the exploits. CAP depends on the parameters such as defender's action d_a the available set of exploits a_e and

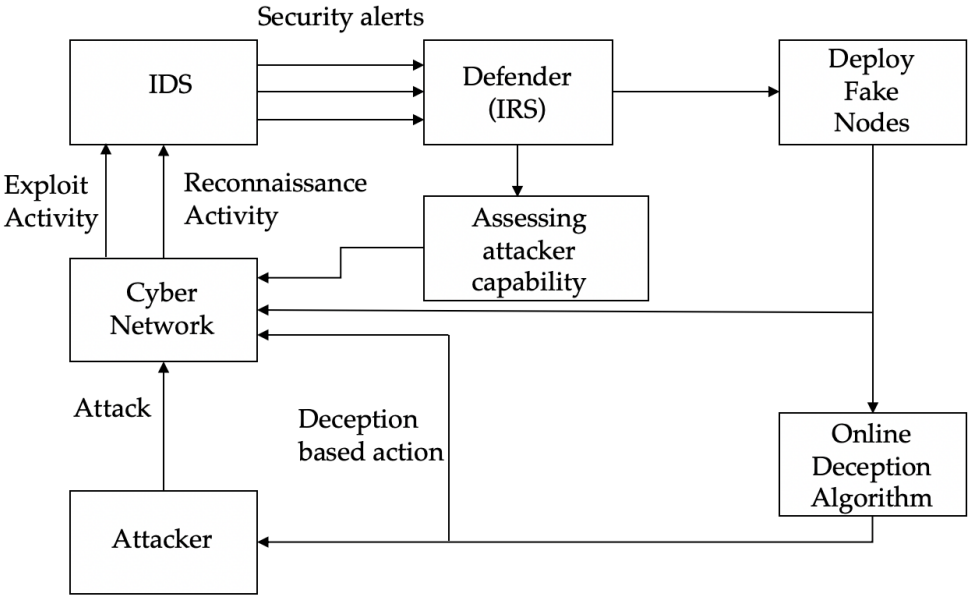


Figure 10: Dynamic security Model.

attacker capabilities a_c. For a given security state s_t and under a defense action u_t the CAP over the real network exploit $er_k \in E$ given by,

$$P_{er_k}(s_t, u_t, \varphi_t) = \left\{ \begin{array}{l} \sum P(d_a, a_e | a_c) = \overline{P}_{er_k}(\varphi_i), when\ er_k \in E(s_t) \backslash B(u_t) \\ \sum P(d_a, a_e | a_c) = \underline{P}_{er_k}(\varphi_i), when\ er_k \in E(s_t) \cap B(u_t) \\ 0, when\ er_k \notin E(s_t) \end{array} \right\} \tag{30}$$

Similarly, for the fake network,

$$P_{ef_k}(s_t, u_t, \varphi_t) = \left\{ \begin{array}{l} \sum P(d_a, a_e | a_c) = \overline{P}_{ef_k}(\varphi_i), when\ ef_k \in E(s_t) \backslash B(u_t) \\ \sum P(d_a, a_e | a_c) = \underline{P}_{ef_k}(\varphi_i), when\ ef_k \in E(s_t) \cap B(u_t) \\ 0, when\ ef_k \notin E(s_t) \end{array} \right\} \tag{31}$$

By dividing the set of available exploits into two categories helps us to understand how an attacker change the attacking strategy. When defender does not block any exploits, attacker attempt with a probability which is defined by the term $\overline{P}_{er_k}(\varphi_i)$ & $\overline{P}_{ef_k}(\varphi_i)$. On the other hand, attacker attempt with a probability $\underline{P}_{er_k}(\varphi_i)$ & $\underline{P}_{ef_k}(\varphi_i)$, when defender blocks exploit. The value 0 means that there are no available set of exploits to be attempted. When the attacker is not able

to identify blocked exploits in a security state for action u that means $\overline{P}_{er_k}(\varphi_i) = \underline{P}_{er_k}(\varphi_i)$ the other hand, if the attacker identifies that exploits are blocked in this security state, the attacker would not attempt it, $\underline{P}_{er_k}(\varphi_i) = 0$.

4.2 SYSTEM ARCHITECTURE OVERVIEW

Our dynamic deception system consists of five components such as a) a SDN controller which generates the flow rules dynamically and control the network traffic, b) deception server which manipulates network traffic, imitate some virtual network resources based on the user policy, and perform the online deception algorithm, c) delay handler which keeps the bandwidth balance between real and fake network, so that attacker couldn't distinguish the real and fake network, d) IDS alert correlation server is responsible for correlating the alert with the exploit activity, e) SDN network elements are responsible to controlling and analyzing the network traffic after getting the flow rules from SDN controller. When packet arrives at SDN switch, which is connected to our system, the SDN controller generates flow rule in accordance with our fake network. The packet either sends to the deception server or send to the destination after tagging each packet. When the packet sent to the deception server, the packet is crafted in accordance with the fake network when reply back to the sender by adding artificial delay to make consistency. If the packet is sent to the real network, an artificial delay is added when reply back to the sender to make consistency between real network and fake network. For a very large network, the deception server could be a bottleneck because of a large number of requests can come to the server. To handle this issue, our deception server can be replicated so that each of the deception servers can handle a certain number of requests. Our system is implemented using in Python. We use POX framework [49] to implement the SDN controller and Scapy framework [50] to implement our deception server. We use mininet [51], which is the current state-of-art SDN network emulator to test our implementation. In Figure 11 we presented a systematic architectural overview of our DDS system. In the next couple of sections, we briefly describe our DDS system.

4.2.1 ONLINE DECEPTION ALGORITHM

For deception we use the same algorithm described in earlier chapter.

4.2.2 SOFTWARE DEFINED NETWORK CONTROLLER

In our DDS, the primary objective of the SDN controller is to generate network flow rules based upon the arrival of network packets. The generated flow rules later forward to SDN switch

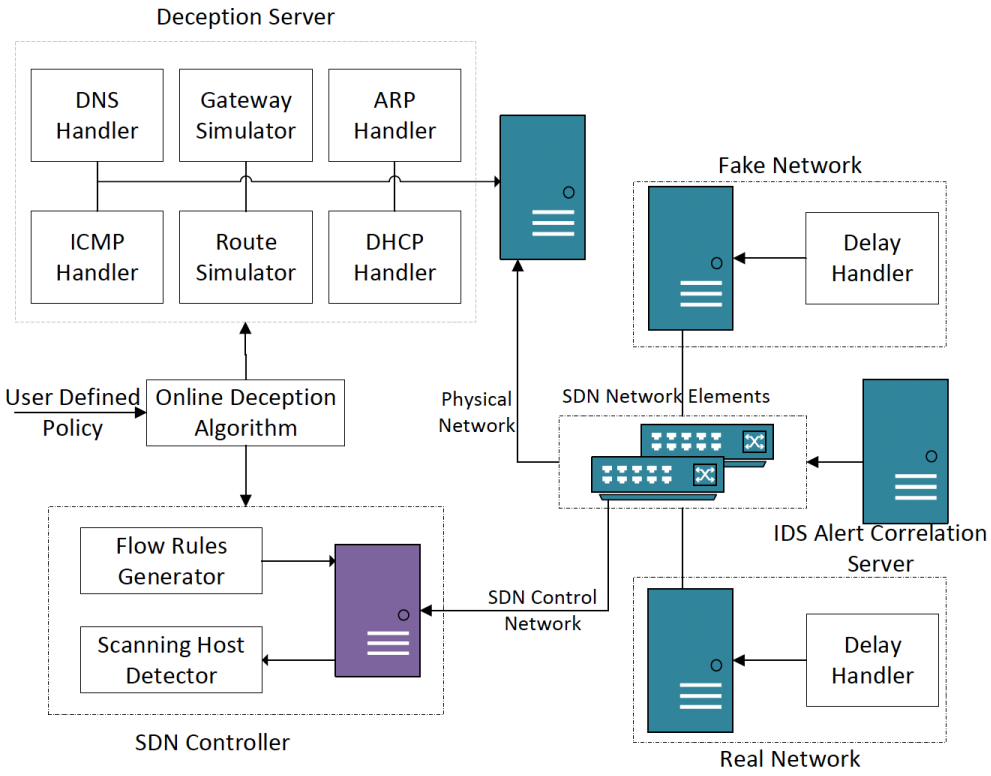


Figure 11: System architecture.

to control and analyse the network traffic. For our deception model, we use the following flow rules based on our fake network needs,

Routing Packets to or from the Fake Network: The use of fake network makes our model dynamic in the sense that it changes the attacker's perception about network structure from the real one. With the mix of real and fake information make the network significantly larger than the actual one. The network flows from and to the fake network are monitored and analyzed by the SDN controller to identify the infected host.

Dynamic Address Translation: To send the fake network information along with the real network information, our deception system rewrites packet headers on-the-fly based.

ARP Request Forwarding: In our system, ARP request forwarding the most important part as all the requests are handled by our deception server. Usually, a network is flooded by ARP request to discover a host and match the IP address with MAC address. Deception server receives ARP request and responds with an appropriate response.

Routing of DHCP Packets: As fake networks associated with DHCP lease, our deception server serves as a DHCP server. It leases IP to the fake network's host when any host from the

fake networks trying to connect with the network.

Routing of DNS Packets: To make sure the reachability to the legitimate services, DNS requests are handled by our deception server. To forward the DNS packets appropriate flow rules between host and the deception server are generated.

4.2.3 DECEPTION SERVER

In our deception server, there are six components to deceive the cyber adversary and handle the packets coming from hosts connected with the network and crafted the packet based on the fake networks. Below we briefly discuss the six components,

DHCP Handler: The DHCP handler acts as a DHCP server in our deception server and responsible for assigning DHCP lease to hosts which are trying to connect with the network.

ARP Handler: All ARP requests are forwarded by appropriate flow rules to our deception server. Based on our fake network specifications, our deception server modified the request and sent back to the requesting host.

ICMP Handler: ICMP error messages are forwarded by the specific rules to our deception server. Packets with the message like destination host unreachable contain nested packet. Such a nested packet cannot be updated automatically in the SDN switches. We forward such packets to our deception server and crafted accordingly and send back to the destination.

DNS Handler: To make sure the reachability to the legitimate services, DNS requests are handled by our deception server and creates appropriate responses.

Gateway Simulator: Gateway simulator is using to make the fake network more realistic as some of the components from the fake network does not have any endpoints. Such endpoints are like routers or gateway. If our deception server receives any probing request, it sends back an appropriate response to the destination.

Route Simulator: Route simulator is using in our deception server to reply packets with mapping functions like traceroute. If the probing request to any node has lower TTL value than specified in our fake network, our deception server handles those packets on behalf of router/gateway between the scanning source node and destination node.

4.2.4 DELAY HANDLER

Besides the traditional scanning method, advance level attackers can analyze the statistics of round-trip time and measured bandwidth on links to find the inconsistency.By adding artificial delay to certain packets, we change the link bandwidth and host delays. To make the consistency, firstly, we collect measurement data from real network nodes and use those data as the basis for

Table 4: Four attacker types

Attacker Types	Knowledge	Aggression	Stealthiness
Type-I	High	Moderate	High
Type-II	Moderate	High	High
Type-III	Moderate	Moderate	Moderate
Type-IV	Low	Low	Low

our fake network.

4.3 EVALUATION AND RESULTS

4.3.1 EXPERIMENTAL SETUP AND METRICS

Now we will investigate an illustrative example using the sample exploit dependency graph presented in Figure 12. For this example, we assume an attacker will $n_a = 4$ types by varying attacker knowledge, aggression, and stealthiness level presented in Table 4. We will present four use cases, how defender deceives the attacker with a fake network for four attacker types. Aggression level is defined by the conditional attack probabilities and success, which in terms called the rate of movement of the attacker throughout the real network. Knowledge level is defined by the Eq. 30, 31 where the separation of two parameters $\overline{P}_{er_k}(\varphi_i)$ & $\underline{P}_{er_k}(\varphi_i)$ dictate the knowledge level of the attacker.

The weight cost is 0.5 and the discount factor $\gamma = 0.95$. In total (real & fake) there are $n_s = 356$ security states and $n_z = 12$ security alerts leading to $2^{12} = 4096$ distinct observation vectors. To approximate the belief, all simulations use particles $n_k = 1500$. For this simulation, we assume that the exploit dependency graph is already generated using TVA (Topological Vulnerability Analysis) [46]. We use the [52] software package to use the POMCP solver in our simulation and use Python and Matlab to implement our model. In the section, we are going to present our simulation results for each of the attacker types defined in Table 4. In Table 5. we presented probabilities of detection for real networks for each of the four attacker types.

In Table 5 columns represent attempted exploit, and rows present the triggered alert. Each entry from the table represents the probability of detection under each of the attack types.

4.3.2 RESULTS

Table 5: Probability of detection for each of the attacker types

Alert	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}
Z_1	0.3	0.4	0	0	0	0	0	0	0	0	0
	0.5	0.5	0	0	0	0	0	0	0	0	0
	0.4	0.4	0	0	0	0	0	0	0	0	0
	0.8	0.2	0	0	0	0	0	0	0	0	0
Z_2	0	0.2	0.3	0	0	0	0	0	0	0	0
	0	0.4	0.2	0	0	0	0	0	0	0	0
	0	0.4	0.3	0	0	0	0	0	0	0	0
	0	0.6	0.8	0	0	0	0	0	0	0	0
Z_3	0	0	0.4	0.3	0	0	0	0	0	0	0
	0	0	0.3	0.4	0	0	0	0	0	0	0
	0	0	0.3	0	0	0	0	0	0	0	0
	0	0	0.6	0.7	0	0	0	0	0	0	0
Z_4	0	0	0	0.4	0.4	0	0	0	0	0	0
	0	0	0	0.2	0.3	0	0	0	0	0	0
	0	0	0	0.5	0.4	0	0	0	0	0	0
	0	0	0	0.6	0.7	0	0	0	0	0	0
Z_5	0	0	0	0.2	0.5	0	0	0	0	0	0
	0	0	0	0	0.3	0	0	0	0	0	0
	0	0	0	0.4	0.4	0	0	0	0	0	0
	0	0	0	0.7	0.8	0	0	0	0	0	0
Z_6	0	0	0	0	0.5	0.2	0	0	0	0	0
	0	0	0	0	0.3	0.4	0	0	0	0	0
	0	0	0	0	0.4	0.3	0	0	0	0	0
	0	0	0	0	0.6	0.7	0	0	0	0	0
Z_7	0	0	0	0	0	0.6	0.2	0	0	0	0
	0	0	0	0	0	0.3	0.5	0	0	0	0
	0	0	0	0	0	0.2	0.3	0	0	0	0
	0	0	0	0	0	0.8	0.7	0	0	0	0
Z_8	0	0	0	0	0	0.2	0.3	0.3	0	0	0
	0	0	0	0	0	0.5	0.5	0.2	0	0	0
	0	0	0	0	0	0.3	0.4	0.4	0	0	0
	0	0	0	0	0	0.7	0.6	0.7	0	0	0
Z_9	0	0	0	0	0	0	0	0	0.4	0.3	0
	0	0	0	0	0	0	0	0	0.5	0.2	0
	0	0	0	0	0	0	0	0	0.2	0.5	0
	0	0	0	0	0	0	0	0	0.7	0.7	0
Z_{10}	0	0	0	0	0	0	0	0.5	0.2	0.2	0
	0	0	0	0	0	0	0	0.4	0.3	0.4	0
	0	0	0	0	0	0	0	0.5	0.5	0.6	0
	0	0	0	0	0	0	0	0.6	0.7	0.8	0

Z_{11}	0	0	0	0	0	0	0	0	0.3	0.2	0
	0	0	0	0	0	0	0	0	0.5	0.4	0
	0	0	0	0	0	0	0	0	0.6	0.3	0
	0	0	0	0	0	0	0	0	0.8	0.7	0
Z_{12}	0	0	0	0	0	0	0	0	0	0.3	0
	0	0	0	0	0	0	0	0	0	0.4	0
	0	0	0	0	0	0	0	0	0	0.4	0
	0	0	0	0	0	0	0	0	0	0.8	0
Z_{13}	0	0	0	0	0	0	0	0	0	0	0.3
	0	0	0	0	0	0	0	0	0	0	0.3
	0	0	0	0	0	0	0	0	0	0	0.4
	0	0	0	0	0	0	0	0	0	0	0.8

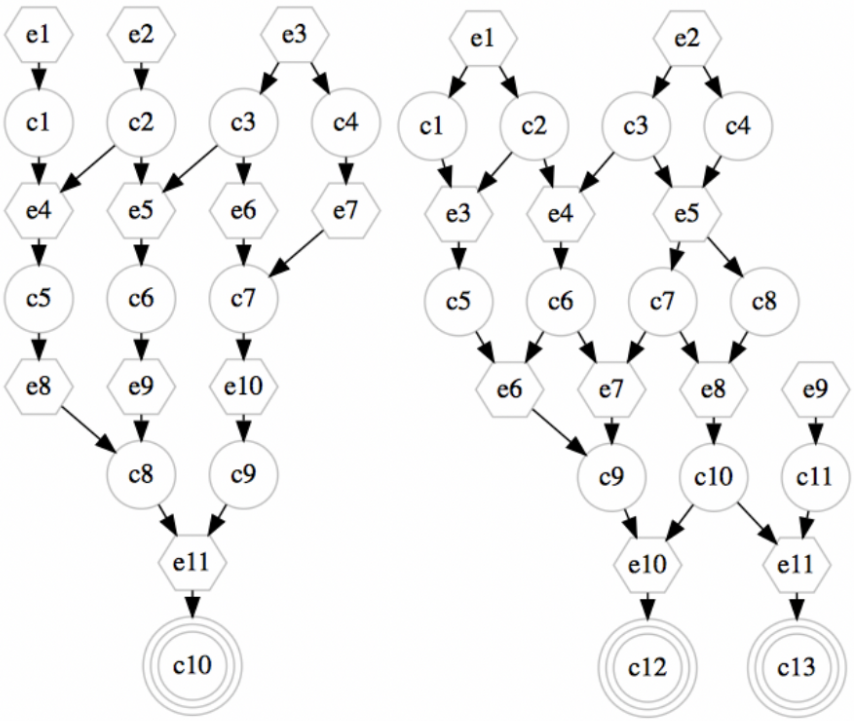


Figure 12: Experimental exploit dependency graph.

Use Case I

For this use case, we use attacker Type-I (φ_1) from Table 4. We calculated the conditional attack probabilities for real and fake networks using Eq. 30 & 31 which is presented below.

$$
\begin{aligned}
(\overline{P}_{er_k}(\varphi_1),\, \underline{P}_{er_k}(\varphi_1)) &= (0.8, 0.3) \quad for\ er_k \in E_0 \\
(\overline{P}_{ef_k}(\varphi_1),\, \underline{P}_{ef_k}(\varphi_1)) &= (0.8, 0.3) \quad for\ ef_k \in E_0 \\
(\overline{P}_{er_k}(\varphi_1),\, \underline{P}_{er_k}(\varphi_1)) &= (0.7, 0.3) \quad for\ er_k \in \{e_4, e_5, e_6, e_8, e_9\} \\
(\overline{P}_{ef_k}(\varphi_1),\, \underline{P}_{ef_k}(\varphi_1)) &= (0.9, 0.7) \quad for\ ef_k \in \{e_5, e_7\} \\
(\overline{P}_{er_k}(\varphi_1),\, \underline{P}_{er_k}(\varphi_1)) &= (0.6, 0.4) \quad for\ er_k \in \{e_7, e_{10}, e_{11}\} \\
(\overline{P}_{ef_k}(\varphi_1),\, \underline{P}_{ef_k}(\varphi_1)) &= (0.9, 0.8) \quad for\ ef_k \in \{e_7, e_8\}
\end{aligned}
$$

similarly, probability of success are,

$$
\alpha_{er_k}(\varphi_1) = \left\{ \begin{array}{ll} 0.7 & when\ er_k \in E_0 \\ 0.5 & when\ er_k \in E \backslash E_0 \end{array} \right\}
$$

$$
\alpha_{ef_k}(\varphi_1) = \left\{ \begin{array}{ll} 0.85 & when\ ef_k \in E_0 \\ 0.7 & when\ ef_k \in E \backslash E_0 \end{array} \right\}
$$

As we defined earlier, the space of actions is the power set of each defense action. In this simulation, we consider three actions for real network which induce a set of block exploits defined as, $B(u^1) = \{e_1, e_2, e_3\}$, $B(u^2) = \{e_4, e_5, e_6, e_7, e_8\}$, $B(u^3) = \{e_9, e_{10}, e_{11}\}$. Similarly for fake network two actions, $B(u^1) = \{e_5, e_7\}$, $B(u^2) = \{e_7, e_8\}$ where the cost of each action is 0.30. The weight cost in Eq. 27 is 0.5 and the discount factor γ = 0.95. In total (real & fake) there are $n_s = 356$ security states and $n_z = 12$ security alerts leading to $2^{12} = 4096$ distinct observation vector. To approximate the belief, all simulations use particles n_k = 1500. The sample evolution of computed deception policy when $N_{Sim} = 5000$ is given in Figure 7, 8. The computed deception policy is intuitive. It is assumed that the security state starts from the empty state defined as, $s_0 = \emptyset$. The defender uses utility array function to construct the initial belief which is defined in Eq. 24. We run the simulation 5000 times.

In Table 6, we present our deception system performance evaluation data while attacker start to exploit real initial nodes vulnerability and ended up with real to real network end state and real to fake end state. The numerical numbers in the 2nd column represent how many times out of 25 sample runs attacker start with real network initial nodes and 3rd column represents how many times attacker ended up with real network end state without transition to the fake network and 4th column represents how many times attacker make transition from real network to fake network

Table 6: Use Case I: Deception system performance evaluation table for real network

Simulation Runs	Attacker Starts with Real Node	Attacker Ends on Real Node	Attacker Ends on Fake Node
500	15	13	2
1000	13	10	3
1500	11	7	4
2000	10	6	4
3000	8	3	5
4000	7	1	6
5000	6	0	6

Table 7: Use Case I: Deception system performance evaluation table for decoy network

Simulation Runs	No. of Times Attacker Starts with Fake Node	No. of Times Attacker Ends on Fake Node	No. of Times Attacker Ends on Real Node
500	10	10	0
1000	12	12	0
1500	14	14	0
2000	15	15	0
3000	17	17	0
4000	18	18	0
5000	19	19	0

and end up with fake goal state.

In Table 7, we presents the same statistics for the fake network.

From Table 7, we can see that up to 76% $((19 \times 25) \div 100)$ of the time attacker start with the fake initial nodes and carry out the series of exploit to achieve the fake goal state. When the N_{Sim} = 500, out of 25 sample runs 15 times attacker start with the real network (Table 6) and 13 times ended up with real network goal state because of poor quality of possible future histories estimation. When the number of simulation increases and more possible future histories are taken into account, the action estimation quality increased as well as policy function (e.g. N_{sim} = 5000, 19 times out of 25 times attacker start and ended up with fake goal state).

Use Case II

For use case II, we use attacker type II (φ_2) from the Table 4 where attacker knowledge, aggression and stealthiness level as follows moderate, high, and high respectively. The conditional attack probabilities and success probabilities are given below for this use case,

$$
\begin{aligned}
(\overline{P}_{er_k}(\varphi_2),\, \underline{P}_{er_k}(\varphi_2)) &= (0.7, 0.7) \quad for\ er_k \in E_0 \\
(\overline{P}_{ef_k}(\varphi_2),\, \underline{P}_{ef_k}(\varphi_2)) &= (0.7, 0.7) \quad for\ ef_k \in E_0 \\
(\overline{P}_{er_k}(\varphi_2),\, \underline{P}_{er_k}(\varphi_2)) &= (0.8, 0.4) \quad for\ er_k \in \{e_4, e_5, e_7\} \\
(\overline{P}_{ef_k}(\varphi_2),\, \underline{P}_{ef_k}(\varphi_2)) &= (0.9, 0.6) \quad for\ ef_k \in \{e_5, e_7\} \\
(\overline{P}_{er_k}(\varphi_2),\, \underline{P}_{er_k}(\varphi_2)) &= (0.7, 0.5) \quad for\ er_k \in \{e_8, e_9, e_{11}\} \\
(\overline{P}_{ef_k}(\varphi_2),\, \underline{P}_{ef_k}(\varphi_2)) &= (0.9, 0.7) \quad for\ ef_k \in \{e_7, e_8\}
\end{aligned}
$$

similarly, probability of success are,

$$
\alpha_{er_k}(\varphi_2) = \left\{ \begin{array}{ll} 0.8 & when\ er_k \in E_0 \\ 0.5 & when\ er_k \in E \backslash E_0 \end{array} \right\}
$$

$$
\alpha_{ef_k}(\varphi_2) = \left\{ \begin{array}{ll} 0.85 & when\ ef_k \in E_0 \\ 0.7 & when\ ef_k \in E \backslash E_0 \end{array} \right\}
$$

We kept other simulation parameters same as for use case I as we are evaluating use case II for the same exploit dependency graph, we presented in Figure 12. In this simulation, we present the performance evaluation table to capture the attacker progression from real to real and real to fake network.

From Table 8 we can see that, the number times attacker starts with the real node less than the use case I because attacker has less knowledge level than previous use case. The results are reasonable because attacker hardly distinguishes the real and fake network. Also, it is difficult for the attacker to discover which exploits are not blocked by the defender in a security state. In Table 9, we present the same simulation results for the fake network.

Table 9 represents the statistics on how many times attacker go back to real node from the fake node. As we stated earlier that as soon as attacker enters the fake network, attacker cannot go back to the real node. We can conclude based on this simulation that up to 84% of the time attacker starts with the fake initial nodes and carry out the series of exploit to achieve the fake goal state because of moderate level of knowledge skill.

Table 8: Use Case II: Deception system performance evaluation table for real network

Simulation Runs	No. of Times Attacker Starts with Real Node	No. of Times Attacker Ends on Real Node	No. of Times Attacker Ends on Fake Node
500	14	9	5
1000	12	8	4
1500	12	7	5
2000	9	5	4
3000	8	4	4
4000	7	2	5
5000	4	0	4

Table 9: Use Case II: Deception system performance evaluation table for decoy network

Simulation Runs	No. of Times Attacker Starts with Fake Node	No. of Times Attacker Ends on Fake Node	No. of Times Attacker Ends on Real Node
500	11	11	0
1000	13	13	0
1500	13	13	0
2000	16	16	0
3000	17	17	0
4000	18	18	0
5000	21	21	0

Use Case III

For use case III, we use attacker type III (φ_3) from the Table 4 where attacker knowledge, aggression and stealthiness level as follows moderate, moderate, and moderate respectively. The conditional attack probabilities and success probabilities are given below for this use case,

Table 10: Use Case III: Deception system performance evaluation table for real network

Simulation Runs	No. of Times Attacker Starts with Real Node	No. of Times Attacker Ends on Real Node	No. of Times Attacker Ends on Fake Node
500	11	7	4
1000	11	8	3
1500	10	8	2
2000	8	5	3
3000	7	2	5
4000	7	2	5
5000	3	0	3

$$(\overline{P}_{er_k}(\varphi_3),\, \underline{P}_{er_k}(\varphi_3)) = (0.8, 0.2) \quad for\ er_k \in E_0$$
$$(\overline{P}_{ef_k}(\varphi_3),\, \underline{P}_{ef_k}(\varphi_3)) = (0.8, 0.2) \quad for\ ef_k \in E_0$$
$$(\overline{P}_{er_k}(\varphi_3),\, \underline{P}_{er_k}(\varphi_3)) = (0.7, 0.3) \quad for\ er_k \in \{e_4, e_5, e_7\}$$
$$(\overline{P}_{ef_k}(\varphi_3),\, \underline{P}_{ef_k}(\varphi_3)) = (0.7, 0.3) \quad for\ ef_k \in \{e_5, e_7\}$$
$$(\overline{P}_{er_k}(\varphi_3),\, \underline{P}_{er_k}(\varphi_3)) = (0.7, 0.3) \quad for\ er_k \in \{e_8, e_9, e_{11}\}$$
$$(\overline{P}_{ef_k}(\varphi_3),\, \underline{P}_{ef_k}(\varphi_3)) = (0.8, 0.2) \quad for\ ef_k \in \{e_7, e_8\}$$

similarly, probability of success are,

$$\alpha_{er_k}(\varphi_3) = \left\{ \begin{array}{ll} 0.8 & when\ er_k \in E_0 \\ 0.5 & when\ er_k \in E \backslash E_0 \end{array} \right\}$$

$$\alpha_{ef_k}(\varphi_3) = \left\{ \begin{array}{ll} 0.85 & when\ ef_k \in E_0 \\ 0.7 & when\ ef_k \in E \backslash E_0 \end{array} \right\}$$

Our deception system performance evaluation table for this simulation is presented in Table 10.

The number of times attacker starts with the real node is increased in this simulation. As defender beliefs reflect that attacker is more knowledgeable, the conditional attack probabilities

are higher than the previous case. In fact, in this simulation, the numbers are higher than previous two use cases. This is because defender possesses a high knowledge level. Because of his high knowledge level, he has the ability to find out the blocked exploits before he moves. As soon as the attacker identifies the blocked exploits, he will not attempt it unlit defender changed her action. In this case, up to 88% of the time attacker starts with the fake initial nodes.

Use Case IV

For use case IV, we use attacker type IV (φ_3) from the Table 4 where attacker knowledge, aggression and stealthiness level as follows low, low, and low respectively. The conditional attack probabilities for attacker type IV are given below,

$$
\begin{aligned}
(\overline{P}_{er_k}(\varphi_3),\, \underline{P}_{er_k}(\varphi_3)) &= (0.9, 0.7) \quad for\ er_k \in E_0 \\
(\overline{P}_{ef_k}(\varphi_3),\, \underline{P}_{ef_k}(\varphi_3)) &= (0.9, 0.7) \quad for\ ef_k \in E_0 \\
(\overline{P}_{er_k}(\varphi_3),\, \underline{P}_{er_k}(\varphi_3)) &= (0.8, 0.7) \quad for\ er_k \in \{e_4, e_5, e_7\} \\
(\overline{P}_{ef_k}(\varphi_3),\, \underline{P}_{ef_k}(\varphi_3)) &= (0.7, 0.7) \quad for\ ef_k \in \{e_5, e_7\} \\
(\overline{P}_{er_k}(\varphi_3),\, \underline{P}_{er_k}(\varphi_3)) &= (0.8, 0.6) \quad for\ er_k \in \{e_8, e_9, e_{11}\} \\
(\overline{P}_{ef_k}(\varphi_3),\, \underline{P}_{ef_k}(\varphi_3)) &= (0.8, 0.7) \quad for\ ef_k \in \{e_7, e_8\}
\end{aligned}
$$

similarly, probability of success are,

$$
\alpha_{er_k}(\varphi_3) = \left\{ \begin{array}{ll} 0.8 & when\ er_k \in E_0 \\ 0.5 & when\ er_k \in E \backslash E_0 \end{array} \right\}
$$

$$
\alpha_{ef_k}(\varphi_3) = \left\{ \begin{array}{ll} 0.85 & when\ ef_k \in E_0 \\ 0.7 & when\ ef_k \in E \backslash E_0 \end{array} \right\}
$$

The performance evaluation table for this simulation is presented in Table 11. From Table 11, we can see that though attacker starts with the real node few times but end up into the real network goal node very few times. The number times attacker ended up on fake goal node is higher than any of the previous three use cases. This is because of the attacker skill set (knowledge, aggression, and stealthiness) reflects as a novice attacker. From the statistics, we can infer that up to 92% of time attacker starts with the fake node and ended up with fake goal state. In this case, defender did not use many resources to block this attacker. As defender's belief reflects that it is a novice attacker. This is why defender saved a lot of resources in terms of availability and security

Table 11: Use Case IV: Deception system performance evaluation table for real network

Simulation Runs	No. of Times Attacker Starts with Real Node	No. of Times Attacker Ends on Real Node	No. of Times Attacker Ends on Fake Node
500	12	1	11
1000	10	0	10
1500	10	0	10
2000	9	2	7
3000	9	1	8
4000	5	0	5
5000	2	0	2

Table 12: Overall deception system performance statistics for all attacker types

Attacker Types	Performance Statistics
Type-I	76%
Type-II	84%
Type-III	88%
Type-IV	92%

cost.

We also investigate the host infection rate with and without our DDS based on network scanning techniques. To do this, we implemented some previous common scanning techniques [53], [54], and [55] which is also discussed in the related work section. To implement these scanning techniques, we use a python library name *libnmap* [56] which provides an API to Nmap [57] as well as python scapy framework. Based on the discussion [58], an adversarial scanner first selects the scanning space which is denoted by Ω. In the scanning space, attacker selects the IP addresses to probe. Also, the address distance denoted by λ specifies the numerical differences between IP address of scanner and scanning target [58].

Local preference scanning discussed in [53], is a kind of biased scanning technique. In this technique, based on the localhost information some specific regions of a network are chosen. But there is an issue, for the current state-of-the-art computer networks, hosts are not uniformly distributed within the address apace. The attacker can increase the speed to detect vulnerable host by scanning IP address where it densely populated [58].

Preference sequential scanning probes the IP address sequentially. In preference scanning technique, attacker use local preference and selects start IP address with small address distance $\lambda(h)$ to the host IP address.

Non-preference sequential scanning is the same as preference sequential scanning, but it selects the starting IP address in a random manner within the scanning space Ω.

Preference parallel using parallelism to increase the scanning performance with a drawback of causing a large amount of network traffic. For our simulation, we use 10 parallel probing messages.

In Figure 13 we presented the performance of dynamic deception system. We deployed 20 subnets, and in each there are 45 hosts are present. The fake network nodes are evenly distributed throughout the subnet. From the performance figure, we can see that with our DDS the infected host detection rate is less than without DDS. Here infected host means attacker successfully exploit the vulnerabilities in that host. From the Figure 13 it can be inferred that defender successfully drive the attacker towards fake network by blocking vulnerabilities in the real network.

From Table 12, it is clearly evident that as soon as attacker knowledge level is decreasing, defender can save more resources in terms of network availability to legitimate users. Based on our simulation results, it is evident that the defender can decide when and where to spend more resources or save resources.

4.4 CONCLUSION

In this paper, we show that with our dynamic defense system defender can save resource in terms of availability cost and security cost. By introducing fake networks, we also alter the perception of network view to the attacker, and defender's action influence an attacker to take fake network attack path towards fake goal state. Using SDN, the defender can analyze the malicious traffic and reply back to the attacker with a mix of true and false information. After adding attacker capabilities in the model, we learned that if the attacker's knowledge level is high and aggression and stealthiness level are moderate, the defender needs to spend more resources than the opposite case.

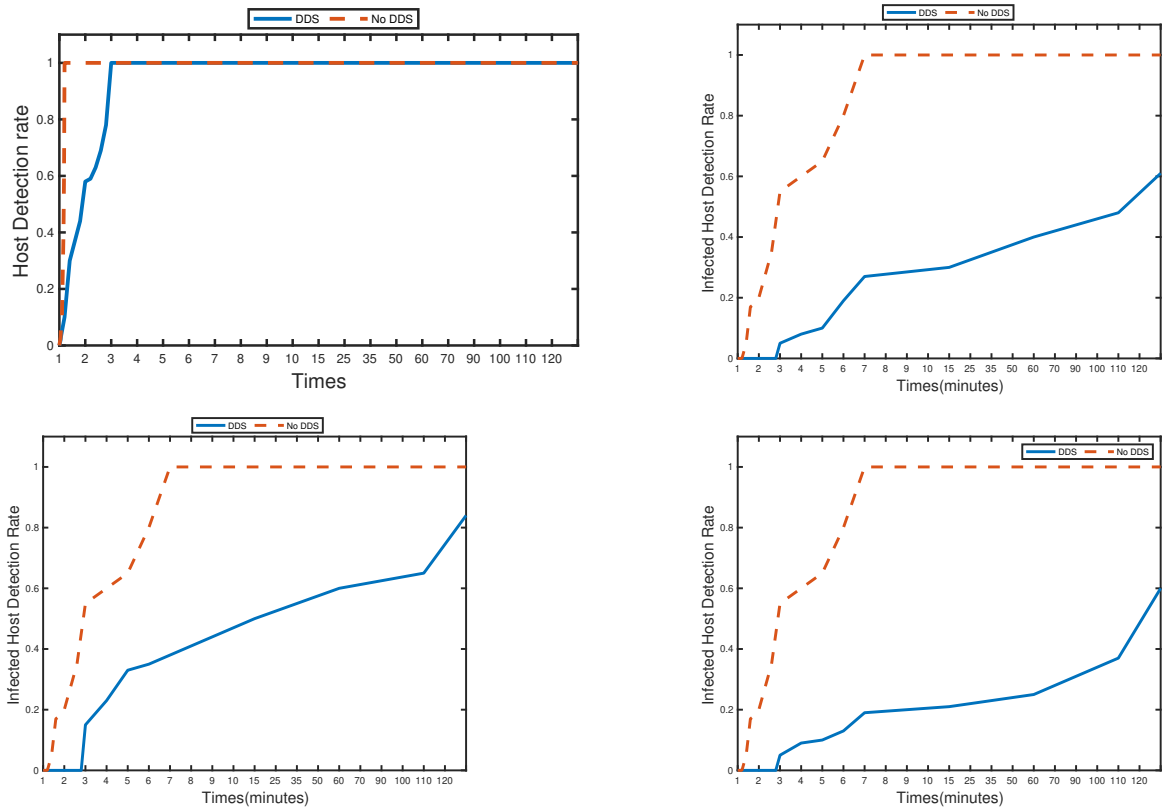


Figure 13: Average vulnerable host detection rate in minutes for the scanning strategies Preference Parallel, Local Preference, Preference Sequential, Non- Preference Sequential with and without our DDS system.

Chapter 5

HIDDEN MARKOV MODEL AND CYBER DECEPTION FOR THE PREVENTION OF ADVERSARIAL LATERAL MOVEMENT

This chapter 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.

5.1 PRELIMINARIES AND ASSUMPTIONS

This section provides an overview of APT life-cycle.

5.1.1 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 successfully carried out the attack. In Figure 14, we depict the different phases of an APT attack[59].

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

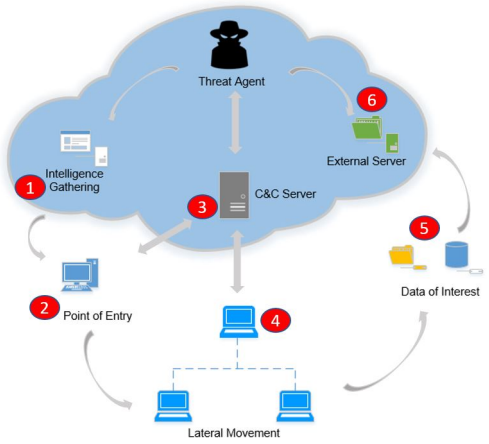


Figure 14: Typical stages of APT attack.

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 [59].

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 [60]. 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.

Table 13: 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

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 [61].

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 [62] 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. [63] 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 the literature [64]. However, studies showed that this is the common life cycle followed by most of the APT attacks.

The list of assumptions that are made throughout the chapter 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.

5.2 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 [65]. 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 15, 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.

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

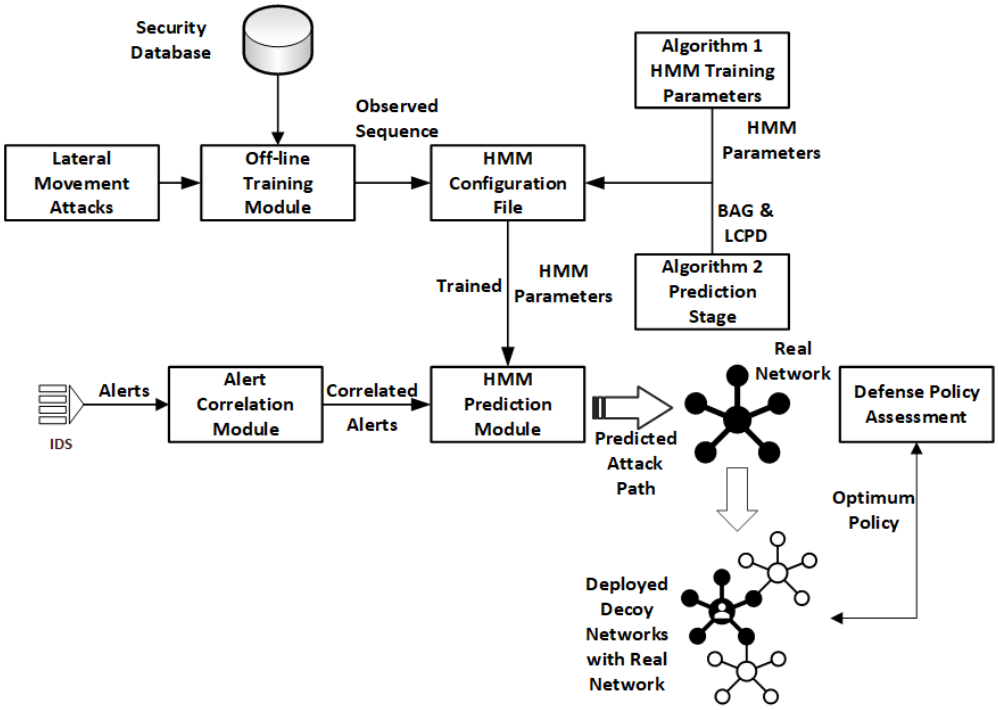


Figure 15: System model architecture.

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).

5.3 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 [66]. 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 sub-section, 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 can not 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.

5.4 PREDICTION VALUES

Our HMM-based state estimation model is inspired by the work presented in [67], 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, $S = (s_1, s_2, s_3, ..., s_N)$ *and discrete observation symbols,* $\bar{O}_M = \bar{o}_1, \bar{o}_2, ..., \bar{o}_m$*, the state transition matrix,* $A = \{a_{i,j}\}$*, the observation emission matrix,* $B = \{b_i(\bar{o}_k)\}$*, and initial matrix,* π_i*, where* $i, j \varepsilon [1, ..., N]$ *and* $k \varepsilon [1, ..., M]$ *[67]. 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:

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

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:

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

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.

5.4.1 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 [35]. 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 [35] 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.

Algorithm 2 HMM Parameters Training

Input: Correlated alert O_T sequence
Output: Optimized A, B, π
Initialization: Random(A, B, π)

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: $\quad \xi_t(i,j) = \frac{\alpha_t(i) a_{i,j} b_j(o_{t+1} \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}$
6: $\quad \gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i) \beta_t(i)}$
7: **end for**
8: **while** iterate until convergence **do**
9: $\quad \bar{\pi}_i = \gamma_1(i)$
10: $\quad \bar{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$
11: $\quad \bar{b}_i(\bar{o}_k) = \frac{\sum_{t=1}^{T-1} \gamma_t(i), when\ o_t = \bar{o}_k, else\ 0}{\sum_{t=1}^{T-1} \gamma_t(i)}$
12: **end while**

In Algorithm 2, 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.

5.4.2 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[35]:

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

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

$$\psi_1(i) = 0$$

- The recursion step:

$$\delta_t(j) = \max_{1 \le i \le N} [\, \delta_{t-1}(i) a_{i,j}] b_j(o_t)\,, 1 \le j \le N \tag{32}$$

$$\psi_t(j) = arg \max_{1 \le i \le N} [\, \delta_{t-1}(i) a_{i,j}]\,, 1 \le j \le N \tag{33}$$

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

$$P(T) = \max_{1 \le i \le N} \delta_T(i) \tag{34}$$

$$q_T = arg \max_{1 \le i \le N} \delta_T(i) \tag{35}$$

$$q_t = \psi_{t+1}(q_{t+1}) \tag{36}$$

The term $P(T)$, q_T, and q_t in the Eq. 34, 35, and 36 defines as maximum probability, best last state, and previous best state, respectively.

State probability can be expressed in terms of forward-backward variables [68]:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{37}$$

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

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

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 [35]:

$$\beta_T(i) = P(o_{t+1}, o_{t+2}, ..., o_T|q_t = s_i, \lambda) \tag{38}$$

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,

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

5.4.3 ATTACK PATHS PREDICTION

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*).

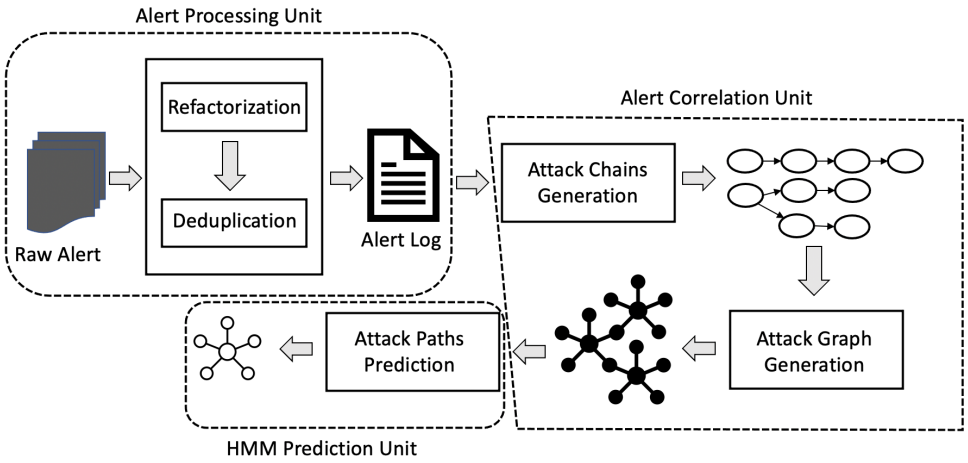


Figure 16: Attack path prediction framework.

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.

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.

HMM Prediction Unit

To predict the next state of the attacker, we use Bayesian Attack Graph (BAG) [69], and Common Vulnerability Scoring System [70]. A Bayesian Attack Graph is a four tuple $BAG = (S, \tau, \varepsilon, P)$ 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 [69]:

$$P_r(S_j|P_a[S_j]) = \left\{ \begin{array}{l} 0, \exists S_i \in P_a[S_j] | S_i = 0, \\ P_r\left(\cap_{S_i=1} e_i\right), otherwise \end{array} \right\} \tag{39}$$

For OR,

$$P_r(S_j|P_a[S_j]) = \left\{ \begin{array}{l} 0, \forall S_i \in P_a[S_j] | S_i = 0, \\ P_r\left(\cup_{S_i=1} e_i\right), otherwise \end{array} \right\} \tag{40}$$

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

$$P_r(\cap_{S_i=1} e_i) = \prod_{S_i=1} P_r(e_i) \tag{41}$$

For OR decomposition,

$$P_r(\cup_{S_i=1} e_i) = 1 - \prod_{S_i=1} [1 - P_r(e_i)] \tag{42}$$

To compute the LCPD, network administrator needs to estimate the success probability of a known exploit given in Eq. 52. The procedure of incorporating LCPD in our prediction algorithm is described in Algorithm 3.

Algorithm 3 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 $j = 1, 2, ..., N$ **do**
2: **for** AND decomposition **do**
3: $P_r(\cap_{S_i=1} e_i) = \prod_{S_i=1} P_r(e_i)$
4: **end for**
5: **for** OR decomposition **do**
6: $P_r(\cup_{S_i=1} e_i) = 1 - \prod_{S_i=1} [1 - P_r(e_i)]$
7: **end for**
8: **for** each intermediate state $i = 1, 2, ..., N$ **do**
9: $\alpha_t(i) = [\sum_{r=1}^{N} \alpha_{t-1}(i) a(r,i)] b_i(o_t)$ ▷ r=index of all possible prior states
10: **end for**
11: $P_{q_{t+1}=s_j} = \sum_{i=1}^{N} \alpha_t(i) P_r a_{i,j}$
12: **end for**

This algorithm's inputs are as follows: optimized HMM parameters from Algorithm 2, 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.

5.5 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.

5.5.1 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 [30], 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 [30]. In [30], 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 [30], 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*. In this dissertation, hyperedge is defined as an edge connecting two *sets* of nodes rather than a pair of nodes.

To capture the attacker progression, we use an exploit dependency graph [38], a directed

acyclic hypergraph, $H = (N, E)$, where, $N = \{c_1, c_2, ..., c_n\}$ is the set of security conditions and $E = \{e_1, e_2, ..., e_n\}$ 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.

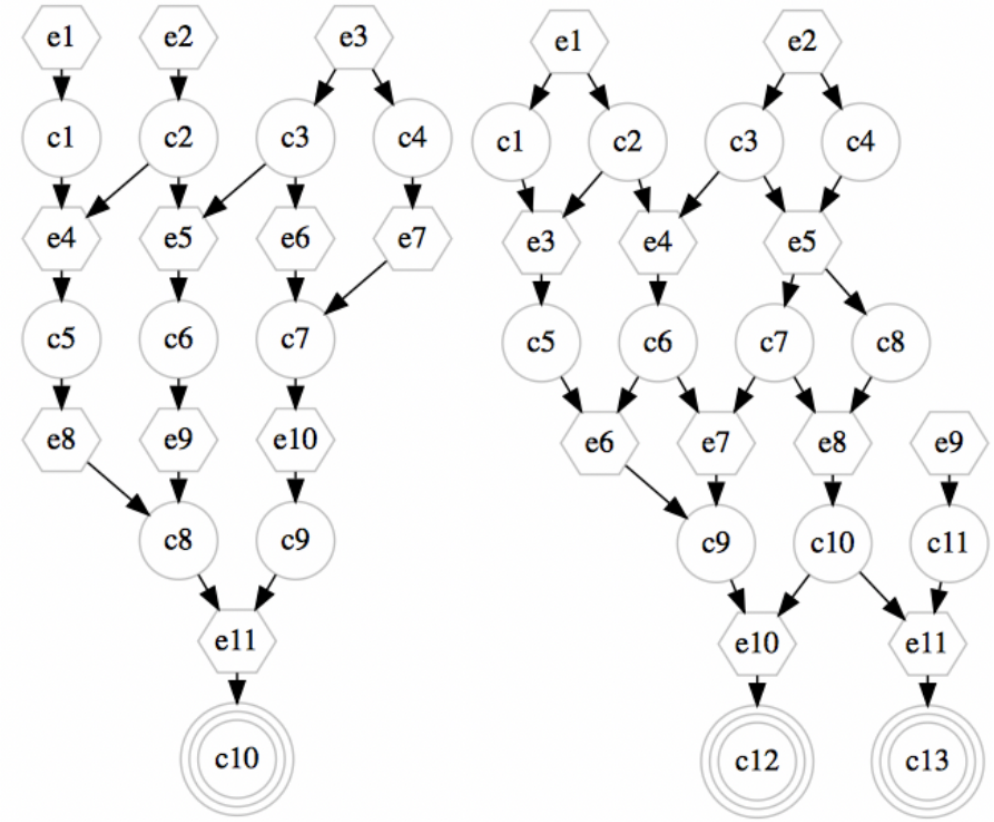


Figure 17: A sample Exploit Dependency Graph with a real network (left) and a fake network (right).

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 $j \in N_i^-$ [38]. 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 17, we present an exploit dependency graph generated using Topological Vulnerability Analysis (TVA) [71] 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 $S = \{s_1, ..., s_n\}$ 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.

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 [38]. 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 $U = \{u^0, u^1, ..., u^n\}$. 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 [38],

$$E(s_t = s) = \{e_i = (N_i^-, N_i^+) \in |N_i^- \subset s, N_i^+ \not\subseteq s\} \tag{43}$$

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 [38]. 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 18 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 18. 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).

5.5.2 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

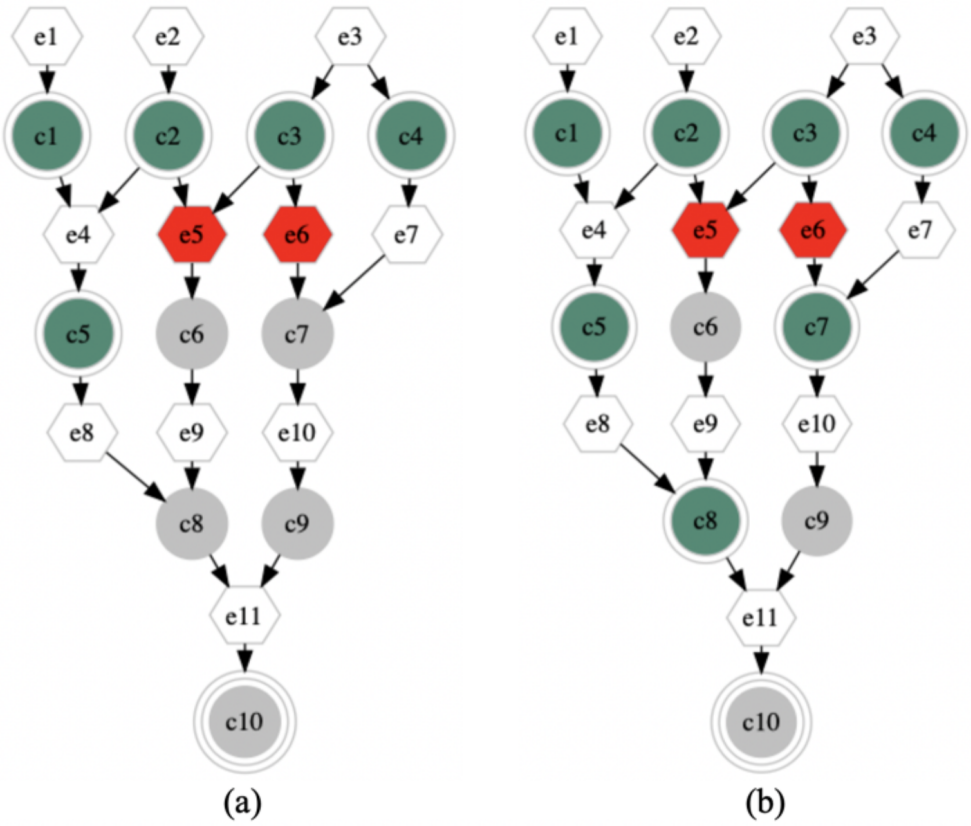


Figure 18: Sample evolution of the security state.

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 $Z = \{z_1, z_2, ..., z_n\}$ 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 $Z(e_i) = \{z_{A_i(1)}, z_{A_i(2)}, ..., z_{A_i(ai)}\} \in P(Z)$ where $P(Z)$ is the power set of Z [38]. 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 $h_t = (\kappa_0, u_0, y_0, ..., u_{t-1}, y_t)$. The belief matrix represents the joint probability distribution over security states, and the attacker types [38], is given below as a matrix form,

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

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 [38],

$$\kappa_{t+1}^j = T_j(\kappa_t, y_k, u) = \frac{p_j^u(\kappa_t) r_{jk}^u(\kappa_t)}{\rho(\kappa_t, y_k, u)} \tag{44}$$

The above terms are defined below,

$$p_j^u(\kappa_t) = P(S_{t+1} = s_j \mid U_t, K_t) = \sum_{s_i \in S} \kappa_t^i \, p_{ij}^u \tag{45}$$

$$r_{jk}^u(\kappa_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t, K_t) = \sum_{s_i \in S} \kappa_t^i \, r_{ijk}^u \tag{46}$$

$$\rho(\kappa_t, y_k, u) = P(Y_{t+1} \mid U_t, B_t) = \sum_{s_j \in S} r_{jk}^u(\kappa_t) \, p_j^u(\kappa_t) \tag{47}$$

where p_{ij}^u is the transition probability from state s_i to s_j under defense action u, and $r_{jk}^u(\kappa_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t = u, K_t = \kappa_t)$ 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 Eq. 44. 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 Eq. 43, each event in the set of exploit is in binary form (successful and unsuccessful).

5.5.3 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 $c_a : U \rightarrow$ for each defense action taken by the defender. Similarly, the security cost $c_s : S \times U \rightarrow$ 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).

5.5.4 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 [72] as

$$d_E = N(d_{proc} + d_{trans} + d_{prop} + d_{queue}) + d_{proco} \tag{48}$$

The terms are in Eq. 48 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 [72]. So that, Eq. 48 can be reduced to

$$d_E = N \times d_{trans} \tag{49}$$

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

$$c(s_t, u_t, \varphi_t) = (1-f)c_s(s_t, \varphi_t) + f * d_E(u_t) \tag{50}$$

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).

5.5.5 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 [73], capturing the optimal action can be intractable for large networks. To resolve this issue, Silver and Veness [31] 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 [31], 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 20. A generative model makes predictions of all future events [74]. 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 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

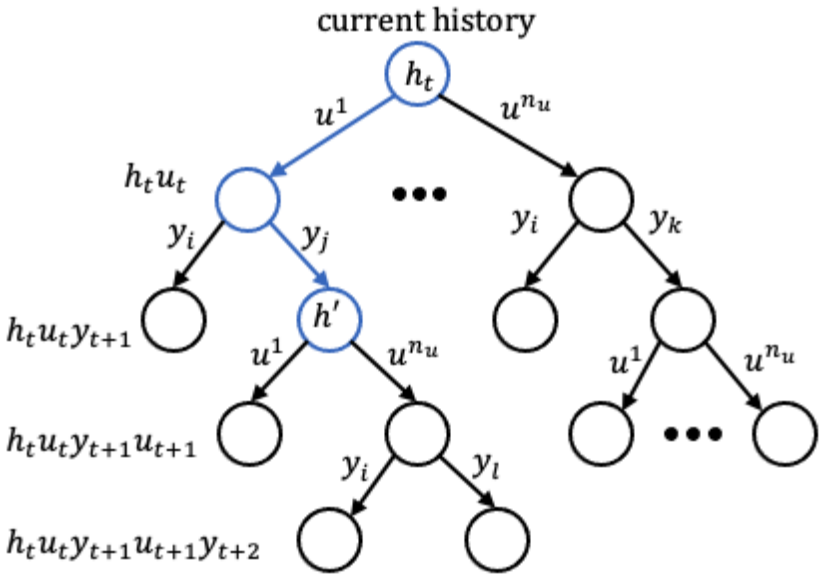


Figure 19: An illustration of the search tree.

in Figure 19.

As the process is partially observable, the search tree in Figure 19 consists of nodes representing histories, and branches from the original tree represent possible future histories. The multi-armed bandit rule, termed as UCB1 [47], 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

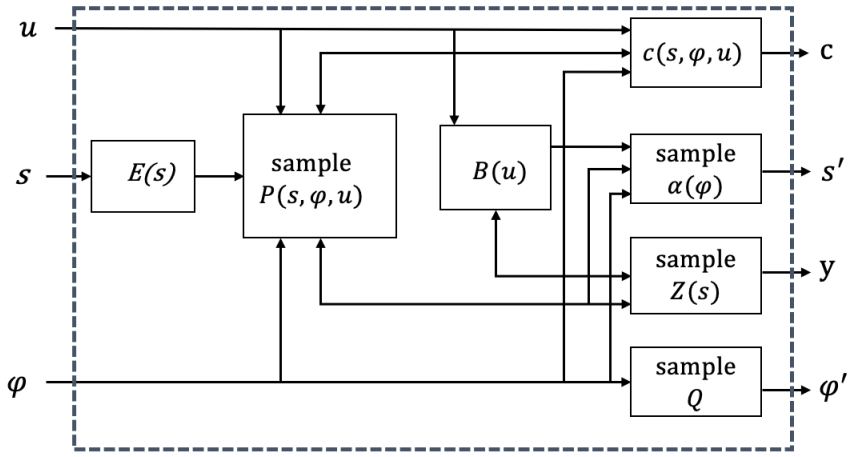


Figure 20: The generative model.

observation, causes the belief update procedure to take a longer time [38]. 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 $\bar{Z}(s) = Z \setminus Z(s)$, can not 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 4 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]
      $\mathfrak{B}_{t+1} \leftarrow \mathfrak{B}_{t+1} \cup \{s', \varphi'\}$
      $numAdded \leftarrow numAdded + 1$
    end if
  end while
end procedure

In Algorithm 4, 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 [40]. 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 [72]:

$$I = 10.41 \times (1 - (1 - CI) \times (1 - II) \times (1 - AI)) \tag{51}$$

$$V_i = 20 \times AC \times AI \times AV \tag{52}$$

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

$$U_a = I \times V_i \tag{53}$$

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 [72],

$$\begin{aligned} V^{\pi}(\kappa_0) &= \sum_{t=0}^{\infty} \gamma^t c(\kappa_t, u_t, \varphi_t) \\ &= \sum_{t=0}^{\infty} \gamma^t E[c(s_t, u_t, \varphi_t) |\ \kappa_0, \pi] \end{aligned} \tag{54}$$

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.

$$\pi^* = \arg\min_{\pi} V^{\pi}(\kappa_0) \tag{55}$$

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.

5.5.6 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 [75]. However, this approach does not project the attack well enough for attacks that continuously change the strategy and ignore the exposed system [76]. 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 Eq. 43, 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:

$$A_{c^{p}_{i,i^{'}}} = \sum_{1}^{j} A_{om} \times S_c \times P_{re} \tag{56}$$

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 [65].

$$A_{om} = \sum_{1}^{j} a_e + {c_{ap}}^{-1} + {t_c}^{-1} \tag{57}$$

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,

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,

$$\alpha_{e_k}(s_t, u_t, \varphi_t) = \left\{ \begin{array}{ll} \overline{\alpha}_{e_k} & when\ e_k \notin B(u_t) \\ 0 & when\ e_k \in B(u_t) \end{array} \right\} \tag{58}$$

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.

5.6 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

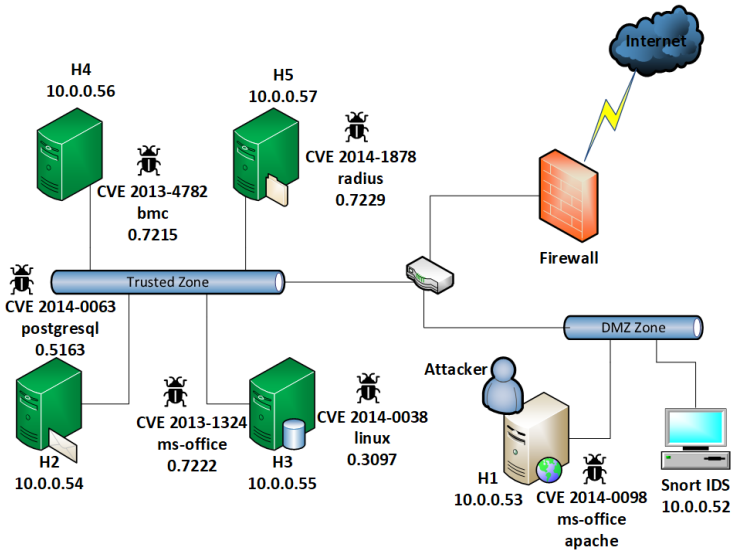


Figure 21: Experimental network topology.

Table 14: 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

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.

An Illustrative Example

Figure 21 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 21 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 14.

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=

$$\begin{bmatrix} 0.6531 & 0.3102 & 0.0267 & 0.0093 & 0.0093 & 0.0093 \\ 0.0093 & 0.0093 & 0.0093 & 0.4932 & 0.4762 & 0.2761 \\ 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \\ 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 \\ 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 \\ 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 \\ 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \end{bmatrix}$$

$$\begin{bmatrix} 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \\ 0.0093 & 0.0093 & 0.4357 & 0.0093 & 0.3286 & 0.1502 & 0.0090 & 0.0091 \\ 0.6288 & 0.2886 & 0.0092 & 0.0092 & 0.0092 & 0.0091 & 0.0091 & 0.0091 \\ 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \\ 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0092 & 0.0091 \\ 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 & 0.0092 \\ 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 & 0.0091 \end{bmatrix}$$

5.6.1 MOST LIKELY ATTACK PATH

We leverage [77] to generate the experimental network's corresponding attack graph, shown in Figure 22. We use an automated alert analysis tool ArcSight [78], 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 2) and the prediction algorithm (Algorithm 4) 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 2. We consider seven states for HMM, as shown in Table 16.

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 22, 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 Eq. 52, we calculate the vulnerability exploitation probabilities presented in Table 15.

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 22. There are 7 different attack states, and one stage is probable to another stage is called the transition probability. Table 16 presents the attack states' denotation, and all attack behaviors information is presented in Table 17.

As it is evident from Table 17 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 3 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 18, we depicted all possible attack paths.

Figure 22: Attack graph of the experimental network.

Table 15: 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

Table 16: 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 17: 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

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

Here, we are looking for the most probable attack path with a length of 5. From Table 18, we can infer that only paths 5,7, and 9 have a length of 5. By matching the alert sequence in 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.

5.6.2 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) [71]. In Figure 23, we presented the corresponding exploit dependency graph. We use an existing POMCP solver [72] 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

Table 18: 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	$S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7$

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.

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:

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

similarly , probabilities of success are as follows:

$$
\alpha_{e_k} = \left\{ \begin{array}{ll} 0.7 & when \; e_k \in E_0 \\ 0.5 & when \; e_k \in E \backslash E_0 \end{array} \right\}
$$

In Table 19, we presented the probability of detection for each of the exploit.

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

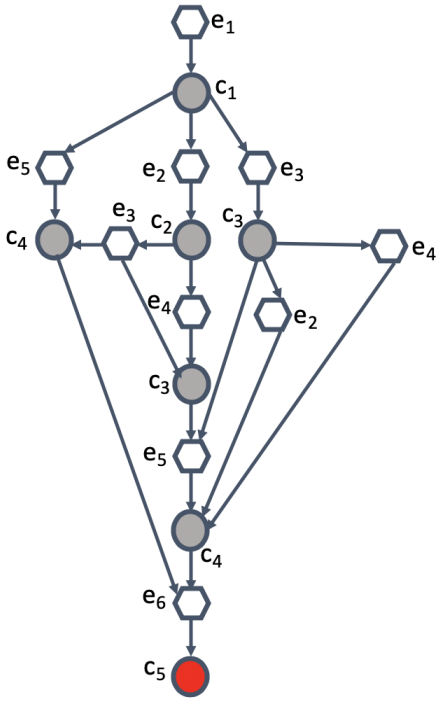


Figure 23: Exploit dependency graph of the experimental network.

Table 19: 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

network. Figure 24 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

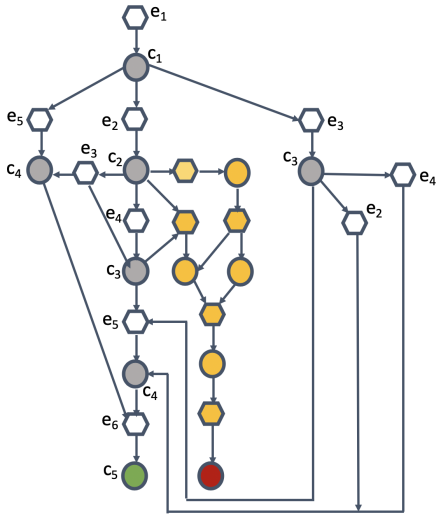


Figure 24: Exploit dependency graph of the experimental network with decoy nodes.

in Figure 25.

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 25. 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 25 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 25.

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 26 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 26 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

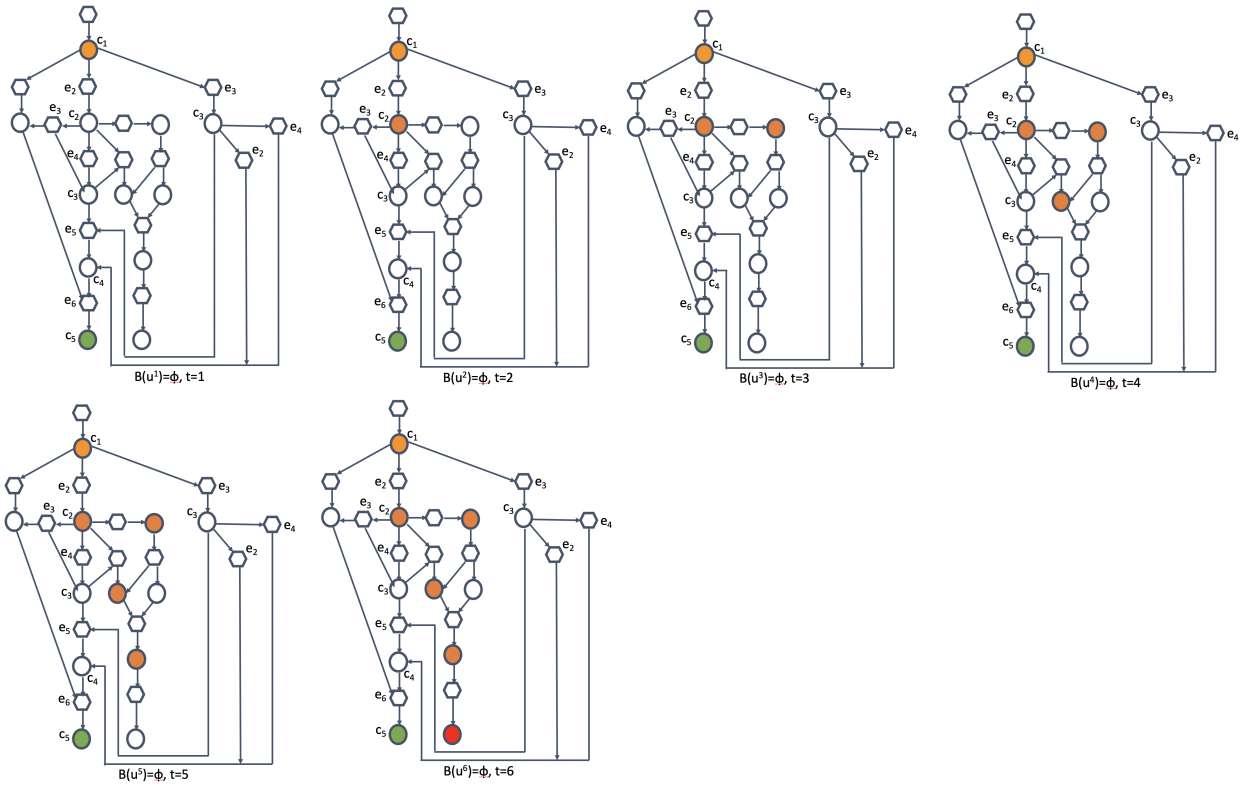


Figure 25: Sample evolution of deception policy and attacker's lateral movement for use case A.

from Figure 26 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 26. Because of the blocked exploits, the attacker can not 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 26 represents the fake goal state.

5.7 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

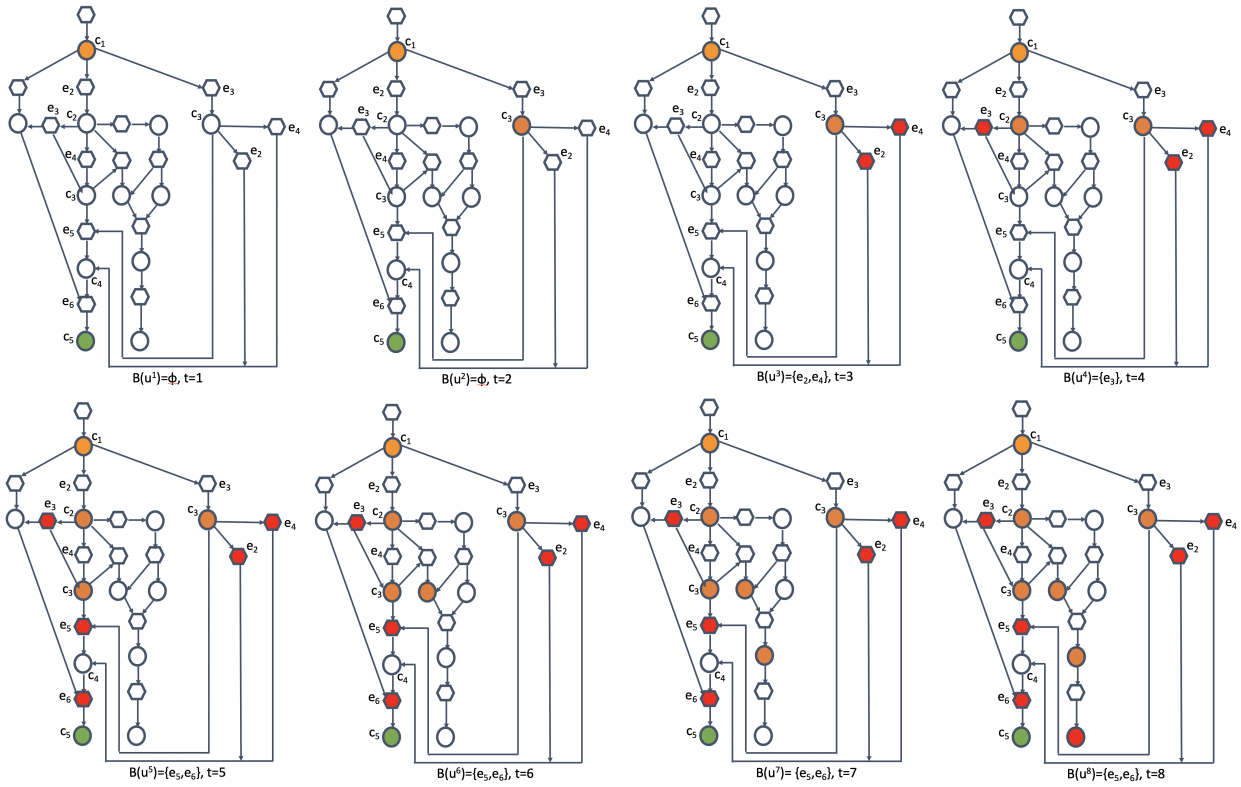


Figure 26: Sample evolution of deception policy and attacker's lateral movement for use case B.

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.

Chapter 6

DECEPTION FOR CHARACTERIZING ADVERSARIAL STRATEGIES IN COMPLEX NETWORKED SYSTEMS

In this chapter, we model the need for systematic characterization of complex networked systems involving friendly forces and opponent forces to understand the adversary opportunities and capabilities to cause harm and develop counter-strategies that would minimize the adversarial impact. Specifically, we need the ability to quantify the incurred cost (cost of protecting friendly forces) and induced cost (opponent cost to cause damage to friendly systems). Cyber attackers' evolving skills cause it challenging to secure the network. Thus it is paramount to characterize adversarial strategies and estimate the attacker's capability. Furthermore, estimating the adversarial capability can aid the cyber defender when deciding to place deceptive elements in the network. Thus, we develop a suite of metrics that quantify the opportunity and capability of the adversary. Using these metrics, the cyber defender can estimate the attacker's capability. In our simulation, we incorporated the developed metrics to estimate adversary capabilities based on the attacker's aggression, knowledge, and stealthiness level. To minimize the adversarial impact, we consider placing decoy nodes as deceptive elements in the network and measure the effectiveness of having decoy nodes. Our experimental evaluation suggests that placing decoy nodes in the network can effectively increase the attacker's resource usage and decrease the win percentage.

6.1 INTRODUCTION

Cyber deception has also emerged as a defense approach to secure our cyber infrastructure from opponent forces. The cyber deception technique allows deploying a network of decoy assets in a complex networked system environment, aiming to exhaust the opponent's resources and time, gather information about their strategies, tactics, capabilities, and intent, and redirect the opponent towards less desirable states. The cyber deception approach will mitigate the information asymmetry that exists between an opponent and friendly systems by converting the friendly's disadvantageous position to a position of strength. However, in order to deploy cyber-deception to realize a resilient cyber infrastructure, several research challenges need to be addressed: a) Design of network decoys, b) Placement of network decoys, c) Maximizing information uncertainty for an opponent, d) Anticipating opponent attack strategy and nodes targeted, e) Quantifying the

performance of deception mechanism to assess the level of asymmetry between an opponent and friendly systems.

Practical deployment of cyber deception relies on friendly forces' ability to optimally place decoy nodes along the networked paths. We have developed a cyber deception approach focused on predicting the most likely sequence of attack paths and deploying decoy nodes along the predicted path [79]. 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. The second phase determines the 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 opponent when it deviates from the predicted path.

In this paper we address the following research challenge, a set of metrics that will provide insights into the level of complexity a sequence of events will impose on an opponent. A set of mixed true and false information increases an aspect of the complexity of the environment in a way that makes it more difficult for an opponent to make decisions or shape conditions in one's favor. If the attacker believes that all the information, he is receiving is accurate, the probability that the attack campaign will fail is very high. On the other hand, if the attacker knows that the information is mixed with true and false information, attackers need to spend more time differentiating that information. To define the metrics that will provide insights into the level of complexity a sequence of events will impose on an opponent, we consider two dimensions of evolution in terms of the opponent's progression: timeliness and effectiveness. Timeliness metrics measures the time it takes to generate new strategies while effectiveness reflects of these generations and impact measure the causeeffect relationship whenever the attacker or defender takes an action. Timeliness suite of metrics measures how quickly the attacker or defender evolves its strategies with or without considering the resulting effectiveness. On the other hand, effectiveness suite of metrics measures the effectiveness of generations over the course of the evolution. In summary, we provide the following contributions:

- Developed a suite metrics that quantify the opportunity and capability of the adversary based on based on aggression, knowledge, and stealthiness.

- To quantify more effective evolutionary strategy to aid in selecting effective deception action by using an autonomous cyber attacker agents.

6.2 RELATED WORK

The importance of security metrics to capture the attacker-defender evolution strategy and related challenges in developing those metrics have been recognized by the security communities [80].

Some literature has been proposed dynamic security metrics [81] where it mentioned metrics for measuring the strength of the preventive defense. The authors in [82] proposed metrics for measuring the reactive defense, and metrics to measure the overall defense are proposed in [83]. A few kinds of literature [84] have proposed some time-related metrics to forecast cyber threats and incidents. er to characterizing adversarial strategies in the cyber deception domain.

The authors in [85] proposed two metrics, Reconnaissance Surface measure (RSM) and Attacker's Belief Error (ABE), to quantify the effectiveness of network deception. Furthermore, the authors model an attacker's evolving knowledge during their interaction with the target system as a belief system. These two metrics are from the network perspective to disrupt the reconnaissance effort. At the same time, we measure the effectiveness of the deception strategy from the system perspective to disrupt the adversarial lateral propagation. To the best of our knowledge, we are the first to study the dynamic security metrics in the cyber deception domain to measure the deception strategy evolution and incorporating it with the POMCP framework to push the adversary towards the decoy network.

6.3 METRICS TO CAPTURE ATTACKER'S CAPABILITY

Cyber attackers and defenders are continuously evolving their strategies to maximize the effectiveness of cyber-attacks or defense. Sometimes cyber attackers are more agile than defenders because cyber defenders take reactive responses against new cyber attacks. The proposing framework aims to define metrics to capture the insights of the complexity a sequence of actions will impose on an opponent's decision calculus. The imposition of complexity is to take actions that increase an aspect of the complexity of the environment in a way that makes it more difficult for an opponent to make decisions or shapes conditions in one's favor. For example, in a game of chess, a player receives a notification of a "win' or "lose" after a long sequence of moves. Strategic moves are complex ones that present an opponent with a multitude of dilemmas, which obviate any simple or singular response.

The framework is designed to consider the evolution of both attack and defense strategies. The defenders need to improve the strategy over time. The metrics will try to understand and improve the defenders' evolution. A single metric can not adequately measure the effectiveness of attack and defense generation over time; we consider a set of metrics. Then these metrics can be aggregated using a weighted average.

6.3.1 OVERVIEW OF THE METRICS

Here, we consider two sets of metrics in terms of the attacker-defender evolution: timeliness and effectiveness. Timeliness measures the time to evolve new strategy generation from attackers' and defenders' perspectives. Effectiveness measures how effective the strategy is over each other (i.e., attacker and defender). Thus, we use timeliness as a reference of effectiveness and vice versa.

1. **Timeliness-Oriented Metrics:** This suite of metrics measures how an attacker or defender emerges its strategies with or without taking into account the resulting effectiveness. Timeliness-oriented metrics contains 4 metrics for both the attacker and defender as follows:

    - *Generation-Time (GT)* measures the time between two consecutive generations of strategies that are observed by the measuring party (i.e., an attacker or defender).
    - *Effective-Generation-Time (EGT)* measures the time it takes for a party to evolve a generation which indeed increases the effectiveness against the opponent.

2. **Effectiveness-Oriented Metrics:** This suite of metrics measures the effectiveness of generations over the course of the evolution. This suite contains 2 metrics, which are equally applicable to both an attacker and defender, leading to 4 metrics in total. The 2 metrics are as follows:

    - *Evolutionary-Effectiveness (EE)* measures the overall effectiveness of generations with respect to the opponent's generation. This is a random variable over $t \in [0, T]$.
    - Aggregated-Generation (AG) measures the gain in the effectiveness of all generations $t \in [0, T]$.

6.4 ATTACK DEFENSE STRATEGY EVOLUTION

We now describe the graph-theoretic representation of the Bayesian attack graph and metrics mathematical model. Here, we restricted the attention to directed acyclic graphs based on the

assumption *monotonicity* [30] on the attacker's behavior. The monotonicity assumption states that the success of the previous exploit will not interfere with the success of the future exploit. The nodes in a Bayesian attack graph represent *attributes* whereas edges represent *exploits*. Attacker capabilities, vulnerabilities of a service or system, interpreted as attributes and exploits, allow the attacker to obtain further capabilities using their current capabilities.

DEFINITION 5.1 [69] *A Bayesian attack graph, G, is defined as the tuple* $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \mathscr{E}, \mathscr{P})$ where

- $\mathscr{N}$ = {1,...,N} is the set of nodes.
- $\mathscr{T}$ is the set of node types. The node type can one of two types, $\mathscr{T}_{\setminus} \in \{\wedge(AND), \vee(OR)\}$.
- $\mathscr{E}$ is the set of directed edges.
- $\mathscr{P}$ is the set of exploit probabilities associated with edges.

Here, each of the nodes $i \in \mathscr{N}$ (attributes) can be either enabled or disabled, which means the attacker possesses a certain capability or not. So, at time t, the network state denoted by $X_t = (X_t^1, ..., X_t^N)$ where X_t^i is the state of attribute i at time t. We have AND and OR configurations for different attack scenarios for a complex networked system (i.e., power grid) from the Bayesian attack graph. The metrics we defined earlier help us to estimate the attackers' opportunity and capability. It also creates a path to assess the effectiveness of different strategies an attacker or defender might take. Our very first suite of metrics, Strategy Evolution - Timeliness, captures the opportunity for the adversary to make a successful attack and the defender to make a successful defense. The second suite of metrics, Strategy Evolution - Effectiveness, captures the capability of the defender and attacker.

6.4.1 STRATEGY EVOLUTION - TIMELINESS

Generation-Time (GT)

Strategic evolution of attack-defense in terms of timeliness measures the time it takes for the attacker or defender to evolve its strategy. The first metric is Generation-Time (GT), a random variable because the strategy generation is often a stochastic process. We assume that the defender will not evolve its strategy before the attacker's movement. So that at time $t = 0$ there will be no defender's impact on the initial attack.

Defenders' Generation-Time: Suppose the defense is evolved at $t_0 = 0, t_1, ..., t_n \leq T$, namely $t_0, t_1, ..., t_n \subseteq [0, T]$. The defender's GT, namely random variable $GT(D)$, is sampled by $GT(D, i)$ [86]

$$G_T(D, i) = t_{i+1} - t_i \; for \; i = 0, 1, ..., n-1 \tag{59}$$

This implies that $D_{t_i+\Delta t} = D_{t_i}$ for any $\Delta t < t_{i+1} - t_i$ and $D_{t_i+GT(D,i)} = D_{t_{i+1}} \neq D_{t_i}$ because the defense is not evolved until time t_{i+1}. At time t_{i+1}, the defender's strategy evolve and will have an impact on the attacker's success probability. So, the defender's success probability in terms of blocking the attack,

$$P(X^i_{t+1}|X_t) = \begin{cases} \prod\limits_{y \in p_x} \alpha_{yx} \times (1 \setminus G_T(D, i), \; if \; X^y_t = 1 \forall j \in p_x \\ 0, \quad otherwise \end{cases} \tag{60}$$

Attackers' Generation-Time: Suppose the attack evolves at $t^{'}_0 = 0, t^{'}_1, ..., t^{'}_k \leq T$, namely $t^{'}_0, t^{'}_1, ..., t^{'}_k \subseteq [0, T]$. Here, the notation $t^{'}$ is used to further highlight the perspective of the attacker's. So, the random variable $GT(A)$ is sampled by $GT(A, j)$, [86]

$$G_T(A, j) = t^{'}_{j+1} - t^{'}_i \; for \; j = 0, 1, ..., k-1 \tag{61}$$

This means that $A_{t^{'}_j+\Delta t} = A_{t^{'}_j}$ for any $\Delta t < t^{'}_{j+1} - t^{'}_j$ and $A_{t^{'}_j+GT(A,j)} = A_{t^{'}_{i+1}} \neq A^{'}_{t_j}$. The attacker's attack success probability defined as follows,

$$P(X^i_{t+1}|X_t) = \begin{cases} \prod\limits_{y \in p_x} \beta_{yx} \times (1 \setminus G_T(A, j), \; if \; X^y_t = 1 \forall j \in p_x \\ 0, \quad otherwise \end{cases} \tag{62}$$

Effective-Generation-Time (EGT)

Effective Generation Time (EGT) measures the time to make an effective strategy generation as a whole attack-defense generation. The earlier metric GT, only measure the generation time without considering the effectiveness of the action from both attackers and defenders perspective. This is why we may not get an relationship with respect to the opponent's action. In that case EGT helps to devise the relationship between the attackers and defenders action.

EGT Defenders': Suppose defense generations evolve at $t_0 = 0, t_1, ..., t_n \leq T$, namely $t_0, t_1, ..., t_n \subseteq [0, T]$. The defender's EGT is a *random variable*, denoted by $EGT(D)$, because the evolution of defense generations are stochastic in nature. The random variable $EGT(D)$ is sampled by $EGT(D, i)$ for $i = 0, ..., n-1$ such that $D_{t_i + EGT(d,i)}$ is the nearest future generation that leads to a higher than $D_{t_i}(A_{t_i}, M)$ defense effectiveness. Formally [86],

$$EGT(D, i) = t_{i^*} - t_i \tag{63}$$

when there exists some $t_{i^*} \in t_{i+1}, ..., t_n$ such that

$$D_{t_i + \Delta t}(A_{t_i}, M) \leq D_{t_i}(A_{t_i}, M) \tag{64}$$

and

$$D_{t_i + EGT(d,i)}(A_{t_i}, M) = D_{t_i^*}(A_{t_i}, M) > D_{t_i}(A_{t_i}, M) \tag{65}$$

EGT Attackers': Suppose the attack generations evolve at $t_0^{'} = 0, t_1^{'}, ..., t_k^{'} \leq T$, namely $t_0^{'}, t_1^{'}, ..., t_k^{'} \subseteq [0, T]$. The attacker's EGT is a *random variable*, denoted by $EGT(A)$, because the evolution of attack generations are stochastic in nature. The random variable $EGT(A)$ is sampled by $EGT(A, j)$ for $j = 0, ..., n-1$ such that $D_{t_j^{'} + EGT(d,j)}$ is the nearest future generation that leads to a smaller than $D_{t_j^{'}}(A_{t_j^{'}}, M)$ defense effectiveness. Formally [86],

$$EGT(A, j) = t_{j^*}^{'} - t_i^{'} \tag{66}$$

when there exists some $t_{j^*}^{'} \in t_{j+1}^{'}, ..., t_k^{'}$ such that

$$D_{t_j^{'} + \Delta t}(A_{t_i^{'}}, M) \geq D_{t_i^{'}}(A_{t_j^{'}}, M) \tag{67}$$

and

$$D_{t_j^{'} + EGT(A,j)}(A_{t_j^{'}}, M) = D_{t_{j^*}^{'}}(A_{t_{j^*}^{'}}, M) < D_{t_j^{'}}(A_{t_j^{'}}, M) \tag{68}$$

6.4.2 STRATEGY EVOLUTION - EFFECTIVENESS

Evolutionary Effectiveness

Now, we need to compare each generation with respect to a reference generation. Evolutionary Effectiveness metrics measures the difference between two generations. The measurement

of generation is a random variable sampled by the opponent's generations. Suppose defense generations are evolved at time $t_0 = 0, t_1, ..., t_l$ and attack generation are evolved at time $t_0^{'} = 0, t_1^1, ..., t_k^{'}$. Here we assume that, the defender's generation as the reference generation and the attacker's generation with respect to the defender's generation [86],

$$EE(\mathscr{A}, i) = \frac{1}{T+1} \sum_{t'=0}^{T} [D_{ti}(A_{t'}, M)] \tag{69}$$

where the defender's EE is defined by a random variable,

$$EE(\mathscr{D}, i) = \frac{1}{T+1} \sum_{t=0}^{T} [D_t(A_{t'_j}, M)] \tag{70}$$

Now, we can define the attacker's capability in terms of the defender's EE($\mathscr{D}$,j) random variable, available exploits a_e, and pre-conditions of a node $\mathscr{P}_{re}$

$$\mathscr{C}_a = EE(\mathscr{D}, i) \times \sum_{t=1}^{T} a_e \times P_{re} \tag{71}$$

Aggregated-Generation

We need to estimate the overall security gained by the defender when the defender is successful in blocking an attack. Aggregated-Generation (AG) metrics measure the security gained over time horizon $[0, T]$. The gain is represented by $\mathscr{G}(i)$ where $i = 1, ..., T$ [86],

$$AG(\mathscr{D}) = \frac{1}{T} \sum_{i=1}^{T} \mathscr{G}(i) \tag{72}$$

As of now, we have characterized how to capture defender and attacker evolving strategies, we need to assess defense action to achieve higher security gain. To assess various defense action, we employ Partially Observable Monte-Carlo Planning (POMCP) framework that simulates future possible state trajectories from the current security state. The POMCP framework maps the current security state and attacker strategy to a defense action. 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 objectives is to quantify the level of security of the system as an attacker progresses. To capture the security level, we define the security state as a current level of the network's attacker progression. In the following section we describe the framework.

6.5 CONTROL OVER ATTACKER'S DECISION-MAKING PROCESS

6.5.1 DECISION CALCULUS

We begin our analysis by laying out an adversary's decision calculus as it applies to the attack context. This represents the base state of adversarial decision making and defender operations, both traditional and complexity focused, that attempt to shape the outcome of this decision flow to achieve a desired set of operational effects. In some cases, the outcome might be more robust deterrence as the attacker sees that the likelihood of arriving at a desired end state is diminished. In other cases, the outcome might be an inability of the attacker to act quickly as necessary information is denied and risk-reward trade-offs are more complex. We focused on decision making as the target of these attacks because, as we explored possible analytical frames, this provides the clearest articulation of ways that complexity exerts its influence in operational practice. In this dissertation, we assume that the defender can collectively formulate the attacker's decision calculus. Rather, we focused on how the defender can control the attackers' decision-making process if the decision calculus is given. In the previous section we identified and formulated metrics to provide insight into the level of complexity a sequence of events will impose on an opponent. In the following section, we describe how the metrics will help in capturing and controlling the attacker's decision-making process.

6.5.2 DECISION MAKING PROCESS

Most cyber security algorithms assume that attackers make rational decisions. However, human decision making processes are only boundedly rational and based on the similarity of the present contextual features to past experience. An attackers' decision-making is a complex process. To capture the attacker's decision making process, we categorize the attacker based on the attacker's i)Knowledge, ii) Stealthiness, and iii) Aggression level. *Aggression* level is defined by the conditional attack probabilities and success. From the reconnaissance stage of the attack phase, an attacker collects critical information from the network and make the attack plan. Identifying the available set of exploits defined the level of the attacker's *knowledge*. *Stealthiness* is described by the false alarm and the probabilities of detection. The attacker types we consider is presented in Table 20. To capture the attacker types, we consider numerical values from 0 to 10 range for different levels of attacker's *knowledge*, *aggression*, and *stealthiness* presented in Table 20. The defender needs to make the decision based on the attacker types which is sometimes difficult. However, an autonomous agent can interact with the defender and make decision by

automatically. We need a model-free technique that can find the best course of action given its current state. Q-Learning is a reinforcement learning algorithm where it can come up with rules of its own, or it may operate outside the policy given it to follow.

Table 20: Attacker types

Attacker Types	Knowledge	Aggression	Stealthiness
Type-I	High (9)	Moderate (7)	High (7)
Type-II	Moderate (7)	High (9)	High (9)

6.6 EXPERIMENTAL EVALUATION

In this section, we demonstrated that using reinforcement-based learning algorithm i.e., Q-Learning we can gain control over the attacker's decision-making process. We placed decoy nodes along with the real nodes in the network and measure the effectiveness when the attacker is an autonomous agent.

6.6.1 SIMULATION SETUP

We adopted the simulation environment from the toy capture the flag (ToyCTF) example provided in the CyberBattleSim code [87]. CyberBattleSim is an experimentation research platform to investigate the interaction of automated agents operating in a simulated abstract enterprise network environment. The network topology in this simulation platform is fixed and a set of vulnerabilities are present which can be used by the attacker to move laterally.

Action Space

We consider three exploits which includes local, remote, and connect and control for the attacker as the attacker action space. *Local exploits* are executable on a node the attacker controls, *remote exploits* the attacker can exploits from the local node to remote node. *Connect and control* exploits can be executed with a matching credential object on a node which is visible to the attacker.

State Space and State Transitions

The scenario presented in Figure 27 involves a small network with machine running different OSes, software. We assume that a single attacker (agent) is present in the network and the goal of the attacker is to maximize the reward by discovering and exploiting nodes in the network.

Here, the environment from the attacker's perspective is partially observable. The attacker can not

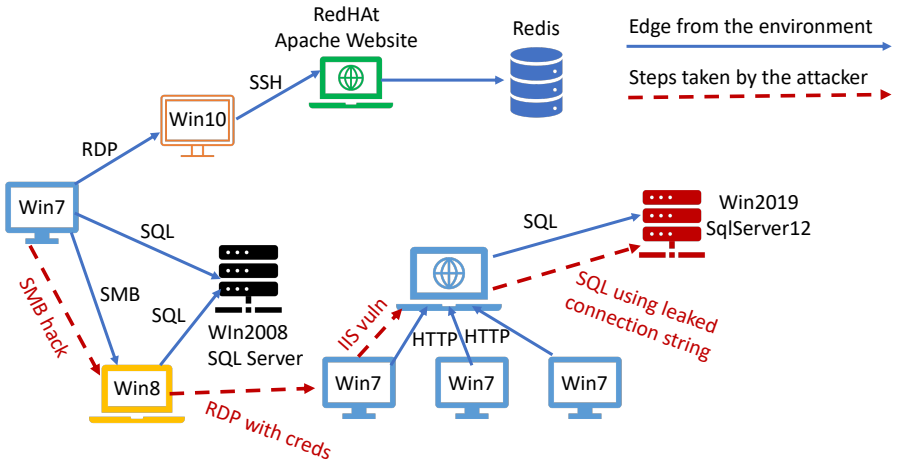


Figure 27: A toy example of network with machines running different OSes, software without deception.

see all the nodes and edges of the network graph in advance. Instead, the attacker takes actions to laterally move and observe the environment. For the simulation, we consider three types of actions the attacker can take to offer a mix of exploitation and exploration. The *local exploits* reveal credentials to exploit the local node and it can be used later with a *connect and control exploit* to gain control of the specific node. *Local exploit* can also discover the path to neighboring nodes. *Remote exploits* are used to form the local node to exploit the remote node. The reward represents the the intrinsic value of a node. The attacker breaches the network as follows (represented by the red arrow in Figure 27):

$$Win7 \rightarrow Win8 \rightarrow Win7 \rightarrow IIS \rightarrow SQLDB$$

Decoy Nodes

We consider decoy nodes [88] to be deceptive elements. Decoy nodes appear as real nodes and the attacker can exploit and control the decoy node. An attacker can find the credentials for the decoy node in the real node and leading to the decoy node. Any new connection established to the decoy node generates a defined penalty of -100 for the attacker. We do not increase the penalty when the attacker make repeats actions. Instead, the repeating action incurs a penalty of -1.

Reward Function

To facilitate penalties with the deception the original code was modified per the following reward function:

- Decoy connection: -100
- Exploit worked: +50
- Exploit use: -1
- Control of node: +1000
- Win condition: +5000
- Repeated mistake: -1

Learning Algorithms

We consider the attacker agents provided by CyberBattleSim which includes the following algorithms:

Deep Q-Learning: Deep Q Networks (DQN) are neural networks that utilize deep Q learning to provide models. DQL uses a Neural Network as the Q-value function approximator. The agent interacts with the environment and applies rewards based on the current state and action. It also could be defined as a value-based RL agent. The following parameter is used in the simulation $\varepsilon = 0.9$, epsilon-min= 0.1, epochs= 300, steps= 5000.

Tabular Q-Learning: Q-learning is a sample-based version of Q-value iteration. This method attempts to directly find optimal Q-values, instead of computing Q-values of a given policy. The value for $\gamma = 0.025$ is selected for the simulation.

6.6.2 RESULTS

In our first simulation, we consider attacker Type-I and the toy example network presented in Figure 27. We deployed the decoy nodes along the real nodes. The goal of the RL agent is to learn a policy so that the expected cumulative reward can be maximized. We consider two metrics to assess the deception framework. Firstly, we calculate the percentage of the attacker wins for defined attacker types in Table 20. Secondly, we measure the resources the attacker needs to spend because of the deception is present.

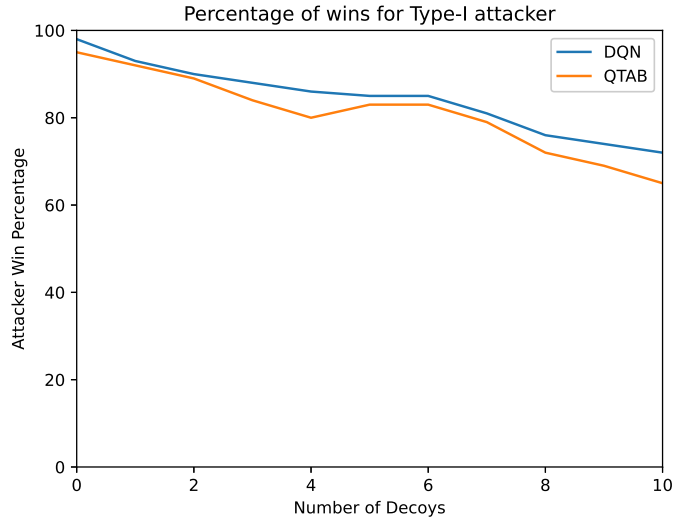


Figure 28: Attacker's win percentage when the number of decoy increases.

An attacker's win is the percentage of an episode where the attacker meets the win criteria. The win condition for the attacker is defined when the attacker takes control of the real nodes. Figure 28 illustrates that type-I attacker agents using different algorithms that can vary overall success on the cyber attack goal. When the number of decoy nodes is increasing, the win

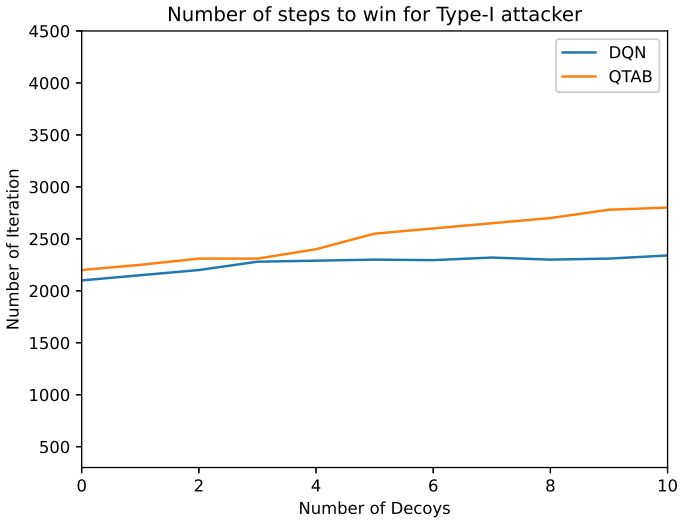


Figure 29: The number of steps required for the attacker to win as a function of the number of decoys.

percentage is also decreasing. The Deep Q-network (DQN) outperforms the QTAB algorithm. It

is evident from the Figure 29 that the attacker attacker is spending more resources (number of iteration increases) when the number of decoy nodes increases.

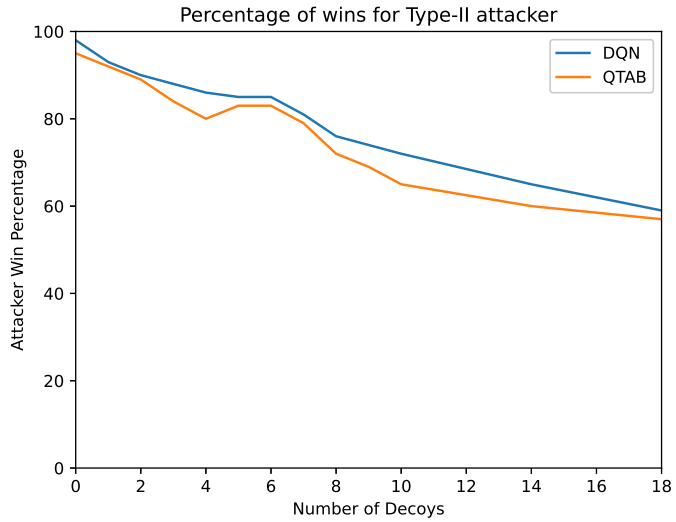


Figure 30: Attacker's win percentage when the number of decoy increases.

For type-II attacker, we find that the attacker win percentage increases compared to type-I as attacker demonstrated in Figure 30. In type-II, we increased the attacker's aggression and stealthiness values. Because of high aggression and stealthiness, the number of iterations to enter the win state for the attacker also decreased as shown in Figure 31. Our findings suggest that even though the attacker is highly skilled, adding decoy nodes can significantly improve the overall network security. Although the defender agent is not employed in this paper, our results can provide insight for an autonomous cyber defender. Early placement of decoy nodes in the network can effectively block the attacker from lateral movement. Our next steps include addition of more deceptive elements with an autonomous deceptive cyber defender.

6.7 CONCLUSION

In this paper, we propose the framework for characterizing adversarial strategies in complex networked systems by developing a set of metrics and provides evaluation to measure the effectiveness of deception. We formulated two sets of metrics to capture the defender and attacker strategy evolution in terms of time and effectiveness and attacker's progression based on the Bayesian attack graph. While the defender's strategy evolution not employing in the simulation but or findings measure the effectiveness of employing decoy nodes in the network. We categorize two types of attacker based on attacker aggression, knowledge, and stealthiness. Increasing decoy

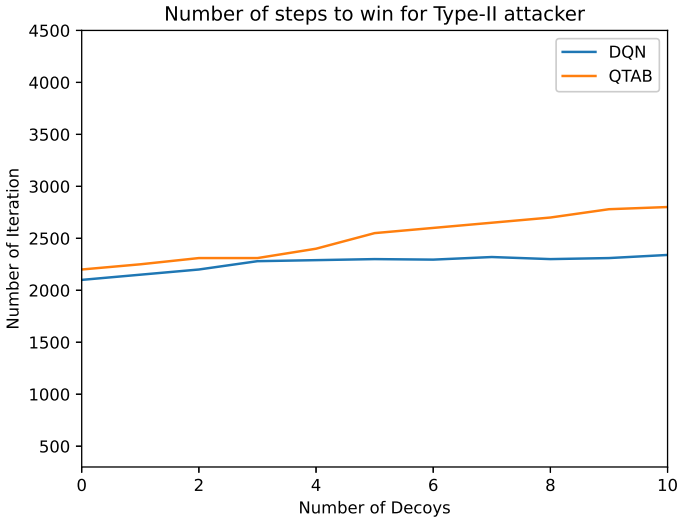


Figure 31: The number of steps required for the attacker to win as a function of the number of decoys.

nodes in the network can effectively reduce a highly skilled attacker's win percentage. Thus, in future work we will consider adding more deceptive elements i.e., honeypots and honey-tokens. We will also incorporate an autonomous cyber deceptive agent in the simulation.

Bibliography

[1] C. Wang and Z. Lu, "Cyber deception: Overview and the road ahead," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 80–85, 2018.

[2] "Mcaafee-report. (2018). the economic impact of cybercrime no slow- ing down.." Available at `https://www.mcafee.com/enterprise/en-us/assets/executive-summaries/es-economic-impact-cybercrime.pdf`.

[3] Verizon, "Threats on the horizon – the rise of the advanced persistent threat." Available at `http://www.verizonenterprise.com/DBIR/`.

[4] S. Jajodia, V. Subrahmanian, V. Swarup, and C. Wang, *Cyber deception*, vol. 6. Springer, 2016.

[5] A. Schlenker, O. Thakoor, H. Xu, F. Fang, M. Tambe, L. Tran-Thanh, P. Vayanos, and Y. Vorobeychik, "Deceiving cyber adversaries: A game theoretic approach," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 892–900, International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[6] M. Albanese, E. Battista, S. Jajodia, and V. Casola, "Manipulating the attacker's view of a system's attack surface," in *Communications and Network Security (CNS), 2014 IEEE Conference on*, pp. 472–480, IEEE, 2014.

[7] S. T. Trassare, R. Beverly, and D. Alderson, "A technique for network topology deception," in *Military Communications Conference, MILCOM 2013-2013 IEEE*, pp. 1795–1800, IEEE, 2013.

[8] Q. Duan, E. Al-Shaer, and H. Jafarian, "Efficient random route mutation considering flow and network constraints," in *Communications and Network Security (CNS), 2013 IEEE Conference on*, pp. 260–268, IEEE, 2013.

[9] M. Dunlop, S. Groat, R. C. Marchany, and J. G. Tront, "Implementing an ipv6 moving target defense on a live network," in *Moving Target Research Symposium 2012*, Cyber-Physical Systems Virtual Organization, 2012.

[10] 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 *Third International Symposium on Information Assurance and Security*, pp. 183–190, IEEE, 2007.

[11] K. Haslum, M. E. Moe, and S. J. Knapskog, "Real-time intrusion prevention and security analysis of networks using hmms," in *2008 33rd IEEE Conference on Local Computer Networks (LCN)*, pp. 927–934, IEEE, 2008.

[12] A. S. Sendi, M. Dagenais, M. Jabbarifar, and M. Couture, "Real time intrusion prediction based on optimized alerts with hidden markov model," *Journal of networks*, vol. 7, no. 2, p. 311, 2012.

[13] H. A. Kholidy, A. Erradi, S. Abdelwahed, and A. Azab, "A finite state hidden markov model for predicting multistage attacks in cloud systems," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pp. 14–19, IEEE, 2014.

[14] 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 Transactions on Smart Grid*, vol. 3, no. 4, pp. 1790–1799, 2012.

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

[16] A. A. Ramaki, M. Amini, and R. E. Atani, "Rteca: Real time episode correlation algorithm for multi-step attack scenarios detection," *computers & security*, vol. 49, pp. 206–219, 2015.

[17] D. Yu and D. Frincke, "Improving the quality of alerts and predicting intruder's next goal with hidden colored petri-net," *Computer Networks*, vol. 51, no. 3, pp. 632–654, 2007.

[18] A. Shameli-Sendi, J. Desfossez, M. Dagenais, and M. Jabbarifar, "A retroactive-burst framework for automated intrusion response system," *Journal of Computer Networks and communications*, vol. 2013, 2013.

[19] S. Fayyad and C. Meinel, "Attack scenario prediction methodology," in *2013 10th International Conference on Information Technology: New Generations*, pp. 53–59, IEEE, 2013.

[20] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *International Conference on Security and Privacy in Communication Systems*, pp. 51–70, Springer, 2009.

[21] 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 *International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[22] V. Conitzer and T. Sandholm, "New complexity results about nash equilibria," *Games and Economic Behavior*, vol. 63, no. 2, pp. 621–641, 2008.

[23] V. E. Urias, W. M. Stout, and H. W. Lin, "Gathering threat intelligence through computer network deception," in *2016 IEEE Symposium on Technologies for Homeland Security (HST)*, pp. 1–6, IEEE, 2016.

[24] S. Jajodia, N. Park, F. Pierazzi, A. Pugliese, E. Serra, G. I. Simari, and V. Subrahmanian, "A probabilistic logic of cyber deception," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2532–2544, 2017.

[25] C. Gao, Y. Wang, and X. Xiong, "A cyber deception defense method based on signal game to deal with network intrusion," *Security and Communication Networks*, vol. 2022, 2022.

[26] C. Ramakrishnan and R. Sekar, "Model-based analysis of configuration vulnerabilities 1," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 189–209, 2002.

[27] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pp. 156–165, IEEE, 2000.

[28] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings 2002 IEEE Symposium on Security and Privacy*, pp. 273–284, IEEE, 2002.

[29] S. Noel and S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," in *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 109–118, 2004.

[30] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 217–224, 2002.

[31] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," in *Advances in neural information processing systems*, pp. 2164–2172, 2010.

[32] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[33] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[34] B. Bauer and K. Karl-Friedrich, "Towards an automatic sign language recognition system using subunits," in *International Gesture Workshop*, pp. 64–75, Springer, 2001.

[35] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[36] S. Jajodia and S. Noel, "Topological vulnerability analysis," in *Cyber situational awareness*, pp. 139–154, Springer, 2010.

[37] S. Achleitner, T. F. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Deceiving network reconnaissance using sdn-based virtual topologies," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1098–1112, 2017.

[38] E. Miehling, M. Rasouli, and D. Teneketzis, "A pomdp approach to the dynamic defense of large-scale cyber networks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2490–2505, 2018.

[39] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, "Online planning algorithms for pomdps," *Journal of Artificial Intelligence Research*, vol. 32, pp. 663–704, 2008.

[40] P. Mell, K. Scarfone, and S. Romanosky, "A complete guide to the common vulnerability scoring system version 2.0," in *Published by FIRST-forum of incident response and security teams*, vol. 1, p. 23, 2007.

[41] J. Kurose and W. Keith, "K. ross computer networking: a top down approach," 2007.

[42] K. Hasan, S. Shetty, A. Hassanzadeh, M. B. Salem, and J. Chen, "Modeling cost of countermeasures in software defined networking-enabled energy delivery systems," in *2018 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, IEEE, 2018.

[43] S. Ullah, S. Shetty, and A. Hassanzadeh, "Towards modeling attacker's opportunity for improving cyber resilience in energy delivery systems," in *2018 Resilience Week (RWS)*, pp. 100–107, Aug 2018.

[44] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces.," in *Robotics: Science and systems*, vol. 2008, Zurich, Switzerland., 2008.

[45] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*, pp. 72–83, Springer, 2006.

[46] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[47] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[48] P. Emami, A. J. Hamlet, and C. Crane, "Pomdpy: An extensible framework for implementing pomdps in python," 2015.

[49] "Pox—python sdn controller."

[50] "Scapy."

[51] M. Team, "Mininet-realistic virtual sdn network emulator," 2018.

[52] P. Emami, A. Hamlet, and C. Crane, "Pomdpy: An extensible framework for implementing pomdps in python," 2015.

[53] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *International Conference on Security and Privacy in Communication Systems*, pp. 310–327, Springer, 2012.

[54] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Adversary-aware ip address randomization for proactive agility against sophisticated attackers," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 738–746, IEEE, 2015.

[55] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," in *Proceedings of the 2003 ACM workshop on Rapid Malcode*, pp. 11–18, 2003.

[56] "Python api for nmap." Available at `https://libnmap.readthedocs.org/`.

[57] "Nmap network scanner." Available at `https://nmap.org/`.

[58] C. C. Zou, D. Towsley, and W. Gong, "On the performance of internet worm scanning strategies," *Performance Evaluation*, vol. 63, no. 7, pp. 700–723, 2006.

[59] T. Micro, "The custom defense against targeted attacks." Available at `http://www.trendmicro.fr/media/wp/custom-defense-against-targeted-attacks-whitepaper-en.pdf` (2020/03/28).

[60] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 48–61, 2010.

[61] A. K. Kaushik, E. S. Pilli, and R. Joshi, "Network forensic system for port scanning attack," in *2010 IEEE 2nd International Advance Computing Conference (IACC)*, pp. 310–315, IEEE, 2010.

[62] M. I. Center, "Apt1: Exposing one of china's cyber espionage units," *Mandian. com*, 2013.

[63] M. Ussath, D. Jaeger, F. Cheng, and C. Meinel, "Advanced persistent threats: Behind the scenes," in *2016 Annual Conference on Information Science and Systems (CISS)*, pp. 181–186, IEEE, 2016.

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

[65] "Mitre adversarial tactics, techniues, and common knowledge, 2020." Available at `https://attack.mitre.org/techniques/enterprise`.

[66] "The evolving face of cyber threats whitepaper, ibm, 2017."

[67] 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.

[68] S.-Z. Yu and H. Kobayashi, "An efficient forward-backward algorithm for an explicit-duration hidden markov model," *IEEE signal processing letters*, vol. 10, no. 1, pp. 11–14, 2003.

[69] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2011.

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

[71] S. Jajodia and S. Noel, "Topological vulnerability analysis," in *Cyber situational awareness*, pp. 139–154, Springer, 2010.

[72] M. A. R. A. Amin, S. Shetty, L. Njilla, D. K. Tosh, and C. Kamouha, "Attacker capability based dynamic deception model for large-scale networks," *EAI Endorsed Transactions on Security and Safety*, vol. 6, 8 2019.

[73] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces.," in *Robotics: Science and systems*, vol. 2008, Zurich, Switzerland., 2008.

[74] E. Talvitie and S. Singh, "Learning to make predictions in partially observable environments without a generative model," *Journal of Artificial Intelligence Research*, vol. 42, pp. 353–392, 2011.

[75] A. Steinberg, "Open interaction network model for recognizing and predicting threat events," in *2007 Information, Decision and Control*, pp. 285–290, IEEE, 2007.

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

[77] X. Ou, S. Govindavajhala, and A. W. Appel, "Mulval: A logic-based network security analyzer.," in *USENIX security symposium*, vol. 8, pp. 113–128, Baltimore, MD, 2005.

[78] "Arcsight, "esm: Enterprise security manager [ol],"." Available at `http://cn.linkedin.com/topic/enterprise-security-manager`.

[79] M. A. R. Al Amin, S. Shetty, L. Njilla, D. K. Tosh, and C. Kamhoua, "Hidden markov model and cyber deception for the prevention of adversarial lateral movement," *IEEE Access*, vol. 9, pp. 49662–49682, 2021.

[80] T. Cyberspace, "Strategic plan for the federal cybersecurity research and development program," *Executive Office of the President National Science and Technology Council*, 2011.

[81] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–35, 2016.

[82] M. Kührer, C. Rossow, and T. Holz, "Paint it black: Evaluating the effectiveness of malware blacklists," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 1–21, Springer, 2014.

[83] D. Levin, "Lessons learned in using live red teams in ia experiments," in *Proceedings DARPA Information Survivability Conference and Exposition*, vol. 1, pp. 110–119, IEEE, 2003.

[84] Z. Zhan, M. Xu, and S. Xu, "Predicting cyber attack rates with extreme values," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1666–1677, 2015.

[85] S. Sugrim, S. Venkatesan, J. A. Youzwak, C.-Y. J. Chiang, R. Chadha, M. Albanese, and H. Cam, "Measuring the effectiveness of network deception," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 142–147, IEEE, 2018.

[86] J. D. Mireles, E. Ficke, J.-H. Cho, P. Hurley, and S. Xu, "Metrics towards measuring cyber agility," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 12, pp. 3217–3232, 2019.

[87] W. B. J. B. K. F. E. G. J. G. K. H. B. M. J. N. N. N. J. P. H. W. Christian Seifert, Michael Betser, "Microsoft defender research team.." `https://github.com/microsoft/cyberbattlesim`, 2021.

[88] M. Al Amin, S. Shetty, L. Njilla, D. Tosh, and C. Kamouha, "Attacker capability based dynamic deception model for large-scale networks," *EAI Endorsed Transactions on Security and Safety*, vol. 6, no. 21, 2019.

VITA

Md Ali Reza Al Amin
Department of Engineering - Modeling & Simulation
Old Dominion University
Norfolk, VA 23529

Publications

- Md Ali Reza Al Amin, Peter Foytik, Sachin Shetty, Jessica Dorismond, and Marco Gamarra. "Deception for Characterizing Adversarial Strategies in Complex Networked Systems." In 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), pp. 429-432. IEEE, 2022.

- Md Ali Reza Al Amin, Sachin Shetty, Laurent L. Njilla, Deepak K. Tosh, and Charles A. Kamhoua, "Hidden Markov Model and Cyber Deception for the Prevention of Adversarial Lateral Movement." in IEEE Access, 2021.

- Md Ali Reza Al Amin, Sachin Shetty, Laurent L. Njilla, Deepak K. Tosh, and Charles A. Kamhoua, "Dynamic Cyber Deception Using Partially Observable Monte-Carlo Planning Framework." in Modeling and Design of Secure Internet of Things.

- Md Ali Reza Al Amin, Sachin Shetty, Laurent L. Njilla, Deepak K. Tosh, and Charles A. Kamhoua, "Online cyber deception system using partially observable monte-carlo planning framework.", in International Conference on Security and Privacy in Communication Systems (SecureComm) 2019, Orlando, USA, October 23-25, 2019.

- Md Ali Reza Al Amin, Sachin Shetty, Laurent L. Njilla, Deepak K. Tosh, and Charles A. Kamhoua, "Attacker capability based dynamic deception model for large-scale networks." in EAI Endorsed Transactions on Security and Safety, 2019.

Typeset using LaTeX.