EAI Endorsed Transactions
on Security and Safety
Research Article EAI.EU

Attacker Capability based Dynamic Deception Model for Large-Scale Networks

Md Ali Reza Al Amin[1,*], Sachin Shetty[1], Laurent Njilla[2], Deepak K. Tosh[3], and Charles Kamhoua[4]

[1]Old Dominion University, Norfolk, Virginia, USA
[2]Air Force Research Lab, Rome, New York, USA
[3]University of Texas at El Paso, El Paso, Texas, USA
[4]Army Research Lab, Adelphi, Maryland, USA

Abstract

In modern days, cyber networks need continuous monitoring to keep the network secure and available to legitimate users. Cyber attackers use reconnaissance mission 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 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.

Keywords: cyber deception, network security, POMCP, POMDP, SDN, exploit dependency graph.

Received on 30 May 2019, accepted on 13 July 2019, published on 01 August 2019

doi: 10.4108/eai.13-7-2018.162808

1. Introduction

The increasing rate of cyber networks and network devices has brought attention to make more resilient in terms of the security of the devices as well as cyber networks. The static nature of these cyber networks let the attacker perform reconnaissance activity and identify potential vulnerabilities. Using the reconnaissance mission, the attacker collects critical network information such as network topology, open ports, and services running on those ports, and unpatched vulnerabilities. Having that critical information increases the probability to penetrate the network and gaining access to critical infrastructure. As reported by the Department of Homeland Security's Industrial Control Systems Cyber Emergency Response Team (ICS-CERT), attacks on critical infrastructure sectors (such as manufacturing, energy, communication, water, and transportation systems) have remained persistent over the past few years, with 245 in 2014, 295 in 2015, 290 in 2016 and 310 in 2017 [43]. Methods and techniques have developed to detect and mitigate those attacks. Among those approaches, one technique is to patch the vulnerability upon availability of patching. It is typically time consuming and costly for a vendor to discover and develop the patch. This significant delay puts the cyber networks being operational without patching the vulnerability, which is very risky. To address this issue, one need to develop an active form of cyber defense system which not only secure the cyber networks but also maintain the system availability to the trusted users.

To develop such a system is difficult in presence of active attackers in operational system because it can inject series of exploits simultaneously. To model these complexities, researchers have proposed graph-based tool such as attack tree/graph. Unfortunately, the attack graphs

*Corresponding author. Email: malam002@odu.edu

can be enormously large for a medium-size cyber network thus makes difficult to apply in an enterprise network. To get rid of this issue, authors in [5] made an assumption on the attacker's behavior, named monotonicity, which states that the success of the previous exploit will not interfere with the success of future exploit. With the help of this assumption, the significant amount of information from the attack graph can be reduced and make a useful attack graph.

It is always beneficial for a defender if it has prior information on how an attacker can infiltrate the cyber network. This knowledge will help the defender to take appropriate actions to thwart any cyber-attack. One of the difficulties with this approach is how to quantify the attacker's progression at any given time. The attacker's status is constantly changing based on the defender's action. Also, the defender has less information about the attacker's actual strategy and actions. The defender has only access to a stream of noisy security alerts from intrusion detection system (IDS), and the security alerts suffer from a high rate of false alarm. The defender's action affects the system availability while maintaining the security of the network. So, the defender needs to make a trade-off between the availability cost and security cost.

In this paper, we aim to deceive the attacker with a fake network while maintaining the trade-off between availability and network resiliency. To do so, we use an exploit dependency graph to capture the attacker's progression throughout the network. We represent the exploit dependency graph as a hypergraph where nodes represent security conditions and directed hyperedges represent exploits. Each security conditions can be either true or false. When the condition is true that means the attacker possesses a certain capability. At any given time and certain security states, attacker uses certain capabilities to exploit the vulnerability. We incorporate these attacker capabilities in our model to capture different kinds of attacker behavior. Capturing attacker capabilities can help the defender to save or enforce more resources to prevent the attacker from reaching the goal. A TCP reset can block an attacker from further penetrating if the attacker is a novice attacker, but the action will not work for a more knowledgeable attacker. This is why it is very critical for a defender to learn the attacker's capability while the defender is making a counterattack plan. To do so, the defender maintains a belief matrix which is the joint probability distribution over attacker type and actions. In this paper, we defined attacker types based on the different level of attacker knowledge, aggression, and stealthiness. The belief matrix is constructed in such a way that it is consistent with all available defender's information such as security alerts, previously deployed action. Defender summarizes all this information make an optimal action which all balance the trade-off between security and availability. Taking optimal action casting as a partially observable Markov decision process (POMDP).

To resolve the scalability issue due to the high dimensionality of the defense problem, we use an online algorithm based on the *partially observable Monte-Carlo planning* [17], which simulates future possible state trajectories from the current belief to evaluate the effectiveness of various defender actions.

We use Software Defined Networking (SDN) to deceive the attacker with a mix of true and false information in the reconnaissance phase. These crafted information helps to change the network view perception from attackers' point of view. Defender can analyze malicious traffic and differentiate the traffic from trusted user to malicious user using SDN. In this way, a trusted user can take seamless services while the defender is brawling with the attacker.

As our main contribution in this paper, we develop a dynamic deception system (DDS) to deceive the attacker with the fake network while capturing attacker capabilities and maintaining the trade-off between availability cost and security cost. The key contributions of this paper are summarized below,

(i) To evaluate our deception goal with the fake network, we design and implement our DDS based on SDN. With the combined functionality of our deception server, online deception algorithm, and SDN controller, we achieve the cyber network deception while maintaining the network availability to the trusted users.

(ii) By incorporating attacker capabilities in our model, we show that the defender can decide when to spend resources or save resources. By simulating our model, it is evident that if the attacker's knowledge level is high where aggression and stealthiness level is moderate, the defender should spend more resources than the opposite one.

(iii) Using SDN and fake network, we show that using our approach there is less infected real network hosts than without our approach. This is because, the defender successfully steers the attacker towards the fake network by blocking vulnerabilities.

2. Related Work

Researchers have proposed cyber deception approaches that introduce fake networks by varying system characteristics [38], manipulating attacker's probes [36, 37] and introducing virtual network interface controllers and route mutation [39]. 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 [38], authors introduce systems that change the view of a cyber network by obscuring some system characteristics where [36] alters the system view by manipulating attackers' probe. Trassare et al. [37] use traceroute function to deceive the network reconnaissance attack. In Dunlop et al. [42], authors propose a mechanism where they hide IPv6 packets to achieve anonymity. They added a virtual network interface controller and shared a secret with all hosts. To defend against eavesdropping and DoS attack Duan et al. [39] presents a Random Route

Mutation Technique to change the networks data flow. Reconnaissance tools such as Nmap or Xprobe2 collects critical network information like host OS or service by analyzing the packet received after probe.

In recent publications [29], [30], and [31] authors present a system that performs dynamic address space randomization based on Software Defined Networking (SDN). These approaches turn out an effective one, but they suffer from high network overheads and also cannot detect the malicious scanning source where we achieve using our dynamic deception system.

The authors in [40], propose a defense system based on IP address randomization and placement of network decoys. In their system, they only consider the scanners from the Internet but in our system, we consider insider scanners as well as the malicious scanner from the internet.

All these approaches in cyber deception area 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 compromised the system. As we mentioned earlier, the attacker always has time-advantage over unpatched vulnerability where vulnerability 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.

In [10], the authors use the dependency graph to provide solutions for the cyber defense system. The issue with that approach is network availability to the trusted user because attacker always starts with the same static network and defender have to take actions (system modification induces blocked vulnerabilities) which will have a more significant impact on availability. We solve this issue by introducing fake networks along with real networks. Our approach will help defender maintaining the network service availability and collect critical intelligence information about the attacker.

3. Overview

3.1 Background

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. Attack graph consists of vertices (system states) and edges (transition relations) where each vertex connect 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. Attack graph can be applied in both penetration testing and network hardening.

Significant progress has been made in generating attack graph automatically [1], [2], [3]. 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, [4] 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 [5] 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 [5], 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 [5], 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 [5] 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 [5] 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 [6] 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.

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.

A discrete POMDP can be formally described as a 7-tuple ($S, A, T, R, \Omega, O, \gamma$), where

- $S = \{s_1, s_2, \ldots, s_n\}$ is a set of states,
- $A = \{a_1, a_2, \ldots, a_n\}$ is set of actions,
- T is a set of conditional transition probabilities $T(s' \mid s, a)$ for the state transition $s \rightarrow s'$,
- $R: S \times A \rightarrow \mathbb{R}$ is the reward function,
- $\Omega = \{o_1, o_2, \ldots, o_r\}$ is a set of observations,
- O is a set of conditional observation probabilities $O(o|s', a)$, and
- $\gamma \in [0,1]$ is the discount factor.

At each time step, the system is in some state $s \in S$ and for the action $a \in A$ taken by the agent the system state transitioned from state s to $s' \in S$ with probability $T(s' \mid s, a)$. While transitioning state, at the same time agent receive an observation $o \in \Omega$ with an observation probability $O(o|s', a)$. At last, the agent receives the reward $R(s, a)$. The ultimate goal is to choose an action in each belief state which will maximize the expected future discounted reward,

$$E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right].$$

POMCP Framework

In large and fully observable domains, Monte-Carlo Tree Search (MCTS) has tremendous performance in online planning [6]. 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 [7].

The authors in [6] 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 [6] 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 [6] 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 [6]. 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.

3.2 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 paper [9], we assumed attacker capabilities; however, in this paper, we incorporated attacker capabilities to do the dynamic security model which is presented in Fig. 1.

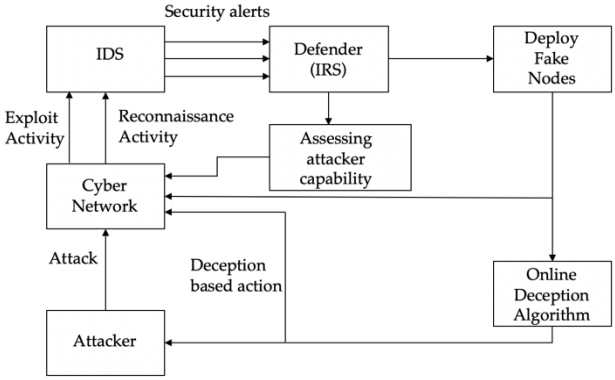


Figure 1. Dynamic security model

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 [10] which is a directed acyclic hypergraph, *H = (N, E)* consists of nodes and hyperedges. Nodes represent a set of security conditions $N = \{c_1, c_2, \ldots, c_n\}$ and hyperedges represent a set of exploits where $E = \{e_1, e_2, \ldots, e_n\}$. The security conditions in the graph can be either true or false. When the security

condition is in true state it means attacker has a particular set of capabilities whereas false value represent attacker does not possess any condition from hypergraph H. For an example, if the attacker possesses a condition that could be led to a conclusion that an attacker may build the trust relationship between two hosts or the attacker reached the goal state. To specify the goal state, we define a parameter to represent 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. This is the node defender wants to protect from an attacker. Defender's main objective is to protect the N_r^g and drive the attacker towards N_f^g.

Each exploit from hyperedges has two conditions, termed as $N_i^-(pre)$ and $N_i^+(post)$. We assume based on [10] that, to attempt an exploit e_i an attacker needs to set true all of the preconditions of that exploit termed as $j \in N_i^-$. 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 attempt to an exploit is successful, all of its postconditions become enabled and let the attacker penetrate more into the network.

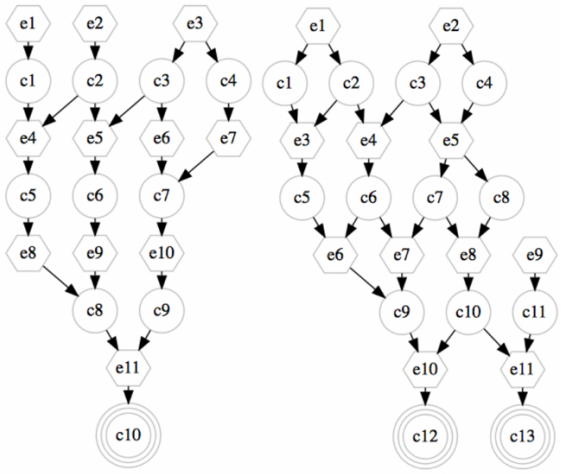


Figure 2. A sample Exploit Dependency graph with a real network (left) and a fake network (right). The above dependency graph for real & fake network *H = (N,E)* consists of $n_{c_r} = 10$ security conditions, $n_{e_r} = 11$ exploits (in the form of hyperedges), $n_{c_f} = 13$ security conditions and $n_{e_f} = 11$ exploits respectively. Triple-encircled nodes are representing as goal conditions $N_r^g = \{c_{10}\}$ *and* $N_f^g = \{c_{12}, c_{13}\}$.

In Fig. 2, we present an exploit dependency graph which is created using Topological Vulnerability Analysis (TVA) [11] tool to explain the model and the results. Whenever a condition is enabled, it means an attacker is having a particular set of capability where the current security state, s_t, describes the set of capabilities of the attacker. 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 $c_j \in N_i^+ and\ N_i^-, N_i^+ \subseteq s$ [10] and set $S = \{s_1, \dots, s_n\}$ represents the state space for this model. In this model, we assume defender will act first and taking actions which eventually interfere with the attacker's progression and reduce the attack surface. The security state evolves probabilistically as a function of defender's and attacker's action [10]. We also assume that the defender has the capability to take action in effect of blocking vulnerabilities. This action includes changing network configuration or shut down a port or any active services. But in reality, the defender is not able to block any individual vulnerability as per authors in [10], rather defender's action induces a set of blocked vulnerabilities. On the other hand, sometimes defender's action is not able to block any vulnerability. To capture this behavior, we assume that the defender has some certain set of actions. The action which will block the vulnerability and influence an attacker to choose a different network path. So, we assume that the defender can change the network configuration on the fly based on the attacker's action to prevent vertical movement. The space of defender's available action set is represented by $U = \{u^0, u^1, \dots, u^n\}$. Here, u^0 represents defender's null action which eventually means the defender will not block any exploit. The remaining actions from the set of U, signifies 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 trusted users. So, it is a goal to a defender 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 set. Based on the cost, the defender is able to choose an action which will limit the progression of the attacker throughout the network and minimizing the negative impact on the system availability.

Based on the single attacker who is trying to infiltrate the system 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 and to allow the exploitation on 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, attacker will attempt exploits based on his capabilities. Available set of exploits is defined by Eq. (1) which is given below for real and fake network,

$$E(s_t = s) = \{er_i = (N_i^-, N_i^+) \in |N_i^- \subset s, N_i^+ \not\subseteq s\} \quad (1)$$
$$E(s_t = s) = \{ef_i = (N_i^-, N_i^+) \in |N_i^- \subset s, N_i^+ \not\subseteq s\} \quad (2)$$

Two essential requirements 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 [10].

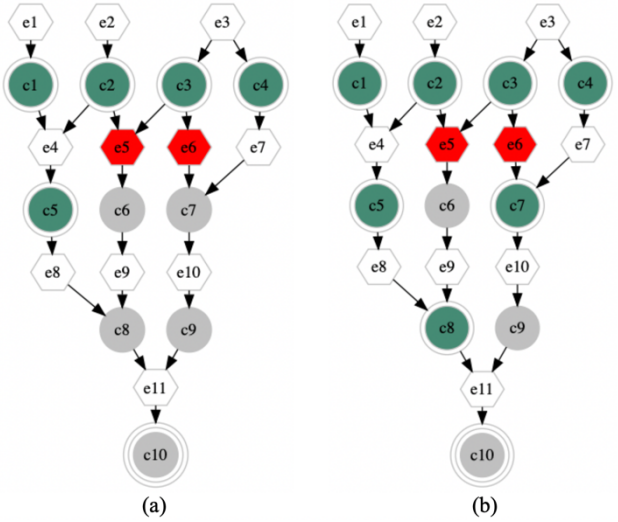


Figure 3. Sample evolution (real network) of the security state for a given state-action-type (s_t, u_t, φ_t): (a) Consider the security state $s_t = \{c_1 c_2 c_3 c_4 c_5\}$ (green circle) and defense action $u_t = u$ where $B(u) = \{e_5, e_6\}$ (here blocked exploits are shown with red shaped hyperedge). So, the available set of exploits using Eq. (1) 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 succeed which is defined by Eq. (3,5). 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). In the above figure, doubled circle shaded shape represents the security state.

Fig 3. describes the set of available exploits for the security state s_t. Fig. 3 is produced from the exploit dependency graph presented in Fig. 2.

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 $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$. 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 attacker capabilities a_e. For a given security state s_t and under a defense action u_t the CAP over the real network exploit $er_k \in E$ is given by,

$$P_{er_k}(s_t, u_t, \varphi_t) = \begin{cases} \sum P(d_a, a_e | a_c) = \bar{P}_{er_k}(\varphi_i), when\ er_k \in E(s_t) \setminus B(u_t) \\ \sum P(d_a, a_e | a_c) = \underline{P}_{er_k}(\varphi_i), when\ er_k \in E(s_t) \cap B(u_t) \\ 0 \qquad\qquad when\ er_k \notin E(s_t) \end{cases} \tag{3}$$

Similarly, for the fake network,

$$P_{ef_k}(s_t, u_t, \varphi_t) = \begin{cases} \sum P(d_a, a_e | a_c) = \bar{P}_{ef_k}(\varphi_i), when\ ef_k \in E(s_t) \setminus B(u_t) \\ \sum P(d_a, a_e | a_c) = \underline{P}_{ef_k}(\varphi_i), when\ ef_k \in E(s_t) \cap B(u_t) \\ 0 \qquad\qquad when\ ef_k \notin E(s_t) \end{cases} \tag{4}$$

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 $\bar{P}_{er_k}(\varphi_i) \& \bar{P}_{ef_k}(\varphi_i)$. 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 $\bar{P}_{er_k}(\varphi_i) = \underline{P}_{er_k}(\varphi_i)$. On the other hand, if the attacker identifies that exploits are blocked in this security state, the attacker would not attempt it, $\underline{P}_{er_k}(\varphi_i) = 0$.

Exploits that are attempted with a probability depends on a certain parameter succeed, which is called attack success probability (ASP). To block vulnerabilities defender will choose the action from the action set $u \in U$. Attacker always tries to create a set of available initial exploits from reconnaissance state to penetrate the network. So, for any given exploit, er_k and ef_k, there is a probability of success,

$$\alpha_{er_k}(s_t, u_t, \varphi_t) = \begin{cases} \bar{\alpha}_{er_k} & when\ er_k \notin B(u_t) \\ 0 & when\ er_k \in B(u_t) \end{cases} \tag{5}$$

Similarly, for the fake network,

$$\alpha_{ef_k}(s_t, u_t, \varphi_t) = \begin{cases} \bar{\alpha}_{ef_k} & when\ ef_k \notin B(u_t) \\ 0 & when\ ef_k \in B(u_t) \end{cases} \tag{6}$$

As soon as, the exploit attempts are successful, it enables all the postconditions, which eventually form the updated security state, as shown in Fig. 3. 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 the defender uses that information to construct the belief by getting security alerts from the Intrusion Detection System (IDS). These security alerts are mixed with false positive and false negative alerts. For a defender, it is important to differentiate those mixed alerts for better defense actions. To do the modeling defender's observation with the security state, we take the approach from our previous paper [9] which is described below.

Intrusion Detection System (IDS) is a major component in this model because the defender's certainty over the security state depends on security alert. IDS generate security alerts in a sequential form when an attacker attempts to exploit and progress through the network. Those security alerts are not free-form noise terms false positive and false negative. Even sometimes there will be no alert for exploit activity which solely depends on

attacker capability (stealthiness) termed as a false negative. Similarly, it generates alert for legitimate user activity termed as false positive. It is critically important for the defender to know the exploit activity is going on. Based on the alert, the defender will choose his defensive action to drive the attacker towards deployed fake networks. Filtering out the noisy alert from true alert is an important factor to improve the efficiency of the defender when it turns in real-time. In this work, we are considering only known vulnerabilities. There are several alert correlations with exploit activity techniques out there [12], [13], [14]. In this work, we are not focusing on alert correlation; rather, we are assuming that defender can do the alert correlation. Let $Z = \{z_1, z_2, \dots, z_n\}$ and $Z' = \{z'_1, z'_2, \dots, z'_n\}$ 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 $Z(e_{ri}) = \{z_{A_i(1)}, z_{A_i(2)}, \dots, z_{A_i(ai)}\} \in P(Z)$ and $Z(e_{fi}) = \{z_{D_i(1)}, z_{D_i(2)}, \dots, z_{D_i(di)}\} \in P'(Z')$ where $P(Z)$ and $P'(Z')$ are the power set of Z and Z'. The vector of security alerts received by the defender at time $t+1$, denoted by $y_{t+1} \in Y = \{0,1\}^{n_z}$, consists of all security alerts triggered during the given iteration [10].

To capture the uncertainty over the security state and attacker type we construct a belief matrix denoted by β_t. This belief matrix is also called information state [15]. 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 $\zeta_t = (\beta_0, u_0, y_0, \dots, u_{t-1}, y_t)$. The belief matrix represents joint probability distribution over security states and attacker types [10], is given below as a matrix form,

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

The space $\in \Delta(S \times \Phi)$ represents the probability distribution over state and type space $(S \times \Phi)$. In the matrix, β_t presented in the double-stochastic matrix for each t. Each row in the matrix probability mass function over the type space for a given state, and each column represents a probability mass function over the space of security states for a given type [10].

Defender update the matrix whenever any information reflects consisting of current defense action u_t and observation vector y_{t+1}. 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 [8],

$$b_{t+1}^j = T_j(b_t, y_k, u) = \frac{p_j^u(b_t) r_{jk}^u(b_t)}{\rho(b_t, y_k, u)} \quad (7)$$

The above terms are defined below,

$$p_j^u(b_t) = P(S_{t+1} = s_j \mid U_t, B_t) = \sum_{s_i \in S} b_t^i p_{ij}^u \quad (8)$$

$$r_{jk}^u(b_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t, B_t) = \sum_{s_i \in S} b_t^i r_{ijk}^u \quad (9)$$

$$\rho(b_t, y_k, u) = P(Y_{t+1} \mid U_t, B_t) = \sum_{s_j \in S} r_{jk}^u(b_t) p_j^u(b_t) \quad (10)$$

where p_j^u is the transition probability from state s_i to s_j under defense action u, and $r_{jk}^u(b_t) = P(Y_{t+1} \mid S_{t+1} = s_t, U_t, B_t)$ is the probability that IDS will generate observation vector y_k when transitioning from state s_i to s_j under a defense action u. Eq. (8) 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. (1), each event in the set of exploits in the binary form (successful and unsuccessful). The belief update procedure is a controlled Markov Chain where control is defender action [10]. The majority of POMDP planning methods operate under Bayes theorem [16]. 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 [17] for the approximation of the belief state.

As it is mentioned earlier in this section that this model is based upon a single attacker who is trying to penetrate into the network. However, from multiple attackers' perspective the model needs to be updated. As an example, if we think that there are two attackers in the network and defender is trying to deceive those two attackers, using our model defender can deceive one attacker at a time. Two attackers may appear at different locations in the network at the same time. As our model is state based so that to work with two or more attackers at a time, we need to improve our model. Another important factor, we need to consider that defender cannot block single vulnerability rather than defender's action induce a set of blocked vulnerabilities. This is another reason why multiple attacker concept will not work with our model.

4. Defender's Action

As soon as the attacker progress through the network defender will take action in real-time to limit the attacker progression. Selection of action step can be improved if the

defender has 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.

4.1 Utility Function

Attacker builds an array of node utility function based on the base score metrics for exploiting vulnerabilities [18]. For every exploit, the attacker uses the metrics to justify the attack success probability which is illustrated in Eq. (13) and serves as the attacker's initial knowledge about the network and vulnerability. The defender also creates the same utility array. From [18], we borrow the impact (I), and exploitability (V) metrics to define the defender's utility.

$$I = 10.41 * (1 - (1 - CI) * (1 - II) * (1 - AI) \quad (11)$$

$$V_i = 20 * AC * AI * AV \quad (12)$$

The above terms are defined as CI = ConfImpact, II= IntegImpact, AI = AvailImpact, I = Impact, Vi = Exploitability, AC = AccessComplexity, AI = Authentication and AV = Accessvector. The utility array function is defined below,

$$U_{a(r,f)} = I * V_i \quad (13)$$

Example 1: Consider a scenario where there are five nodes and attacker send scan queries to the neighbors of node 1. The defender needs to respond to 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 the following values $U_a(n_2) = 15$, $U_a(n_3) = 5$, $U_a(n_4) = 30$, and Ua(n5) = 50. A true rational attacker will go after node 5.

4.2 Cost Function

In cyber-deception, there is a possibility where you can 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 take to drive the adversary towards the fake network. For some defense action, there will be no impact on the availability, and sometimes there will be a more significant impact. To formalize this notion, we represent the availability cost $c_a: U \rightarrow \mathbb{R}$ for each defense action taken by the defender similarly 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).

End-to-End Packet Delay

Let's assume that, d_E and N represent total delay and number devices between a source and destination. The end-to-end delay defined in [19] as,

$$d_E = N\left(d_{proc} + d_{trans} + d_{prop} + d_{queue}\right) + d_{proco} \quad (14)$$

The above equation's terms are defined as follows 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 a 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 [24]. So that, Eq. (14) can be reduced to,

$$d_E = N \times d_{trans} \quad (15)$$

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 follows $c_u = d_E$. 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 regarding a security state and defense action is given below,

$$c(s_t, u_t, \varphi_t) = (1 - f)c_s(s_t, \varphi_t) + f * d_E(u_t) \quad (16)$$

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). The proposed online deception algorithm is based on an existing online solver [9], computes optimal action from deception standpoint to deceive attacker with the fake network while balancing availability and security cost.

5. Dynamic Deception System

In our dynamic deception system (DDS), we deploy fake networks along with the real networks to deceive the attacker and drive the attacker towards the fake network while the attacker is in real network. In this approach, defender can save more availability cost in terms of securing the cyber network. To deploy the fake network and make it as looks like the real network we use software defined networking (SDN). The core part of our dynamic deception system consists of SDN flow rules generated by

our SDN controller which working with a deception server and make the network traffic in the way to looks like different than it actually is.

5.1 System Architecture Overview

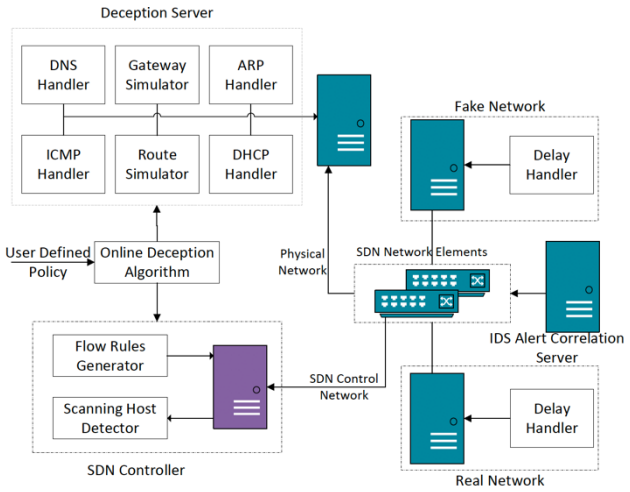


Figure 4. DDS system architecture

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 [22] to implement the SDN controller and Scapy framework [23] to implement our deception server. We use mininet [21], which is the current state-of-art SDN network emulator to test our implementation. In Fig. 4 we presented a systematic architectural overview of our DDS system. In the next couple of sections, we briefly describe our DDS system.

5.2 Online Deception Algorithm

For online deception algorithm, we took the approach described in our previous paper [9] which is described below.

Online defense algorithm is a heuristic search algorithm for determining defense actions in real-time as the attacker progresses through the network and security alerts are generated where scalability is achieved via a sample-based, online defense algorithm that takes advantage of the structure of the security model to enable computation in large-scale domains. After employing defense actions (e.g., blocking vulnerability) defender can evaluate the improvements by assessing the attacker's attacking path. For a scalable 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 [24], capturing the optimal action can be intractable for large networks. To resolve this issue, Silver and Veness [6] 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 policy, yielding a much more scalable approach than offline methods.

POMCP algorithm is based on and makes use of POMDP [24]. There are two types of nodes in POMCP: belief nodes which represent a belief state and action nodes which are their children nodes that can reach by doing an action. In this work, action selection procedure as same as POMCP algorithm described in [6] and belief update procedure is based on [10] where it solves the large observation space problem. 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 alerts for defender.

An agent begins the simulation by calling a generative model provides a sample successor state, observation and cost given a state and action, $(s', y, c) \sim G(s, u)$. The modified belief update procedure is given in Algorithm 1, where $\mathfrak{B}_t$ is a state-action pair named particles. History of search tree as shown in Fig. 5,

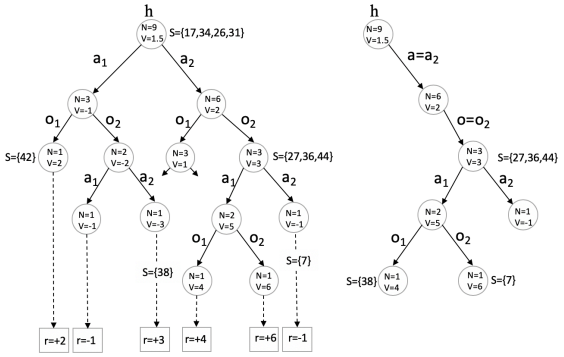


Figure 5. An illustration of POMCP in an environment with 2 actions, 2 observations, 50 states, and no intermediate rewards. The agent constructs a search tree from multiple simulations and evaluates each history by its mean return (left). The agent uses the search tree to select a real action a and observes a real observation o (middle) [9].

is constructed by calling the generative model and successive sampling from current belief. Monte-Carlo Tree Search (MCTS) uses Monte-Carlo simulation for assessing search tree nodes [25]. 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 at the very beginning of the simulation, where it selects the action with the highest value. UCT algorithm [26] is used to improve the greedy action selection stage. In the search tree, each action selection is made using UCB1 [27], and the 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 [26] to initialize the new nodes. We use the utility array function $U_{a(r,f)}$ 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,

$$\begin{aligned} V^{\pi}(b_0) &= \sum_{t=0}^{\infty} \gamma^t c(b_t, u_t, \varphi_t) \\ &= \sum_{t=0}^{\infty} \gamma^t E\left[c\left(s_t, u_t, \varphi_t\right) \mid b_0, \pi\right] \end{aligned} \tag{17}$$

where $0 < \gamma < 1$ is the discount factor, and $c(b_t, u_t, \varphi_t)$ represents the cost under attacker types φ_t for each belief state b_t when an action u_t is selected from the space of action where $c(b_t, u_t) = \sum_{s_i \in S} b_t^i c\left(s_t, u_t, \varphi_t\right)$. For each belief state, defense action generates according to the policy function and belief update must follow the procedure defined in Eq. (14.7). The optimal policy π^* is obtained by optimizing the long-term cost, which is given below,

$$\pi^* = arg \min_{\pi} V^{\pi}(b_0) \tag{18}$$

The optimal policy defined in Eq. (18) specifies the optimal action for each belief state $b_t \in \Delta(S \times \varphi)$ where the expected minimum expected cost calculated over the infinite time horizon. The defender will choose the action where the cost makes the 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 refer 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. The pseudocode for modified belief update is given below,

Algorithm 1 Defender's Belief Update Algorithm

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

```
procedure BELIEFUPDATE(𝔅_t, u_r, y_r)
    while numAdded < n_k do
        (s, φ) ~ 𝔅_t
        (s', φ', y, −) ~ G(s, φ, u_r)
        if y^{Z(s)} = y_r^{Z(s)} then [If alerts Z(s) match]
            𝔅_{t+1} ← 𝔅_{t+1} ∪ {s', φ'}
            numAdded ← numAdded + 1
```

5.3 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 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.

5.4 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.

5.5 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 [44]. To make the real and fake network indistinguishable, we take a similar approach described in [44]. 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 our fake network.

6. Evaluation

6.1 Experimental Setup and Metrics

Now we will investigate an illustrative example using the sample exploit dependency graph presented in Fig. 2. For this example, we assume an attacker will $n_a = 4$ types by varying attacker knowledge, aggression, and stealthiness level. 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 (3), (4) where the separation of two parameters $\bar{P}_{er_k}(\varphi_i)$ & $\underline{P}_{er_k}(\varphi_i)$ dictate the knowledge level of the attacker. Stealthiness is described by the false alarm and the probabilities of detection. In the below table we presented the four attacker types $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$ with their knowledge, aggression and stealthiness level.

Table 1. 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

The weight cost in Eq. (16) 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 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) [11]. We use the [28] 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 1. In Table 2. we presented probabilities of detection for real networks for each of the four attacker types.

Table 2. Probability of detection for each of the attacker types

Alert	Exploit										
	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 0.3 0.4 0.8	0.4 0.5 0.4 0.2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
z_2	0 0 0 0	0.2 0.4 0.4 0.6	0.3 0.2 0.3 0.8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
z_3	0 0 0 0	0 0 0 0	0.4 0.3 0.3 0.6	0.3 0.4 0 0.7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
z_4	0 0 0 0	0 0 0 0	0 0 0 0	0.4 0.2 0.5 0.6	0.4 0.3 0.4 0.7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
z_5	0 0 0 0	0 0 0 0	0 0 0 0	0.2 0 0.4 0.7	0.5 0.3 0.4 0.8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
z_6	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.5 0.3 0.4 0.6	0.2 0.4 0.3 0.7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
z_7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.6 0.3 0.2 0.8	0.2 0.5 0.3 0.7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
z_8	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.2 0.5 0.3 0.7	0.3 0.5 0.4 0.6	0.3 0.2 0.4 0.7	0 0 0 0	0 0 0 0	0 0 0 0
z_9	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.4 0.5 0.2 0.7	0.3 0.2 0.5 0.7	0 0 0 0
z_{10}	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.5 0.4 0.5 0.6	0.2 0.3 0.5 0.7	0.2 0.4 0.6 0.8	0 0 0 0
z_{11}	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.3 0.5 0.6 0.8	0.2 0.4 0.3 0.7	0 0 0 0
z_{12}	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.3 0.4 0.4 0.8	0 0 0 0
z_{13}	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0.3 0.3 0.4 0.8

In Table 2. 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.

6.2 Experimental Results

Use Case I

For this use case, we use attacker Type-I (φ_1) from Table 1. We calculated the conditional attack probabilities for real and fake networks using Eq. (3) & (4) which is presented below.

$$\left(\overline{P}_{er_k}(\varphi_1), \underline{P}_{er_k}(\varphi_1)\right) = (0.8,0.3) \;\; for\; er_k \in E_0$$

$$\left(\overline{P}_{ef_k}(\varphi_1), \underline{P}_{ef_k}(\varphi_1)\right) = (0.8,0.3) \;\; for\; ef_k \in E_0$$

$$\left(\overline{P}_{er_k}(\varphi_1), \underline{P}_{er_k}(\varphi_1)\right) = (0.7,0.3) \;\; for\; er_k \in \{e_4, e_5, e_6, e_8, e_9\}$$

$$\left(\overline{P}_{ef_k}(\varphi_1), \underline{P}_{ef_k}(\varphi_1)\right) = (0.9,0.7) \;\; for\; er_k \in \{e_5, e_7\}$$

$$\left(\overline{P}_{er_k}(\varphi_1), \underline{P}_{er_k}(\varphi_1)\right) = (0.7,0.3) \;\; for\; er_k \in \{e_7, e_{10}, e_{11}\}$$

$$\left(\overline{P}_{ef_k}(\varphi_1), \underline{P}_{ef_k}(\varphi_1)\right) = (0.9,0.7) \;\; for\; er_k \in \{e_7, e_8\}$$

Similarly, for attack success probability we use Eq. (5) & (6) which is given below,

$$\alpha_{er_k}(\varphi_1) = \begin{cases} 0.7 & when\; er_k \in E_0 \\ 0.5 & when\; er_k \in E \backslash E_0 \end{cases}$$

$$\alpha_{ef_k}(\varphi_1) = \begin{cases} 0.85 & when\; ef_k \in E_0 \\ 0.7 & when\; ef_k \in E \backslash E_0 \end{cases}$$

As we defined earlier, the space of actions is the power set of each defense action. In this simulation, we consider there are 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 the fake network, $B(u^1) = \{e_5, e_7\}$, $B(u^1) = \{e_7, e_8\}$ where the cost of each action is 0.30. The sample evolution of computed deception policy when N_{sim} = 5000 is given in Fig. 7 & 8.

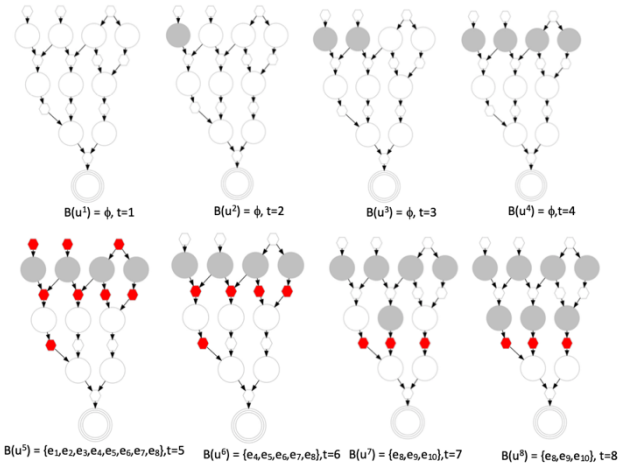


Figure 6. Sample evolution of deception policy when attacker is in real network.
Security state is represented by shaded node and blocked exploits are represented

by red shaped hyperedge.

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. (13). We run the simulation 5000 times.

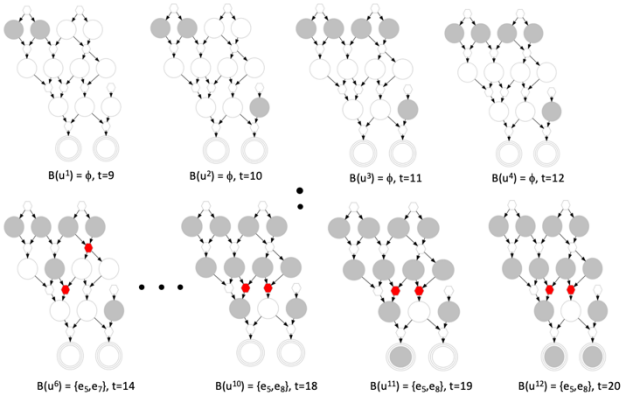


Figure 7. Sample evolution of deception policy when the attacker is in the fake network.
Security state is represented by shaded node, and blocked exploits are represented
by red shaped hyperedge.

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 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 to enable all the time. Whenever defender belief reflects that attacker is close to goal conditions will block the exploits to prevent the attacker from reaching his goal. As we can see from Fig. 6 at time step t=8, defender blocks exploits $\{e_8, e_9, e_{10}\}$ which prevents the attacker from moving 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, Fig. 7, based on his available set of exploits dictated by Eq. (1). 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 has an alternative way to reach the next security state (see time steps t=14-20 in Fig. 7).

Table 3. Performance evaluation table for real to real and real to fake

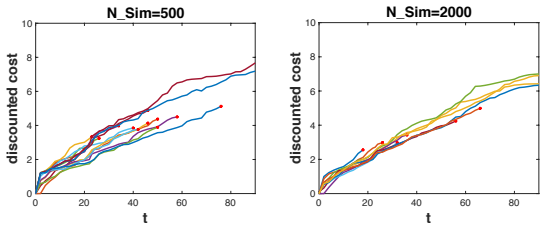
Simulation Runs	No. of Times Attacker Starts with Real Node	No. of Times Attacker Ends on Real Node	No. of Times Attacker Ends on Fake Node
500	15	13	2
1000	13	10	3
1500	11	7	4
2000	10	6	4
3000	8	3	5
4000	7	1	6
5000	6	0	6

In Table 3, we present our 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 4, we present the same statistics for the fake network.

Table 4. Performance evaluation table for fake to fake and fake to real

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

From Table 4, we can see that up to 76% of the time attacker starts 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 3) and 13 times ended up with real network goal state because of poor quality of possible future histories estimation. When the number of simulations 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).



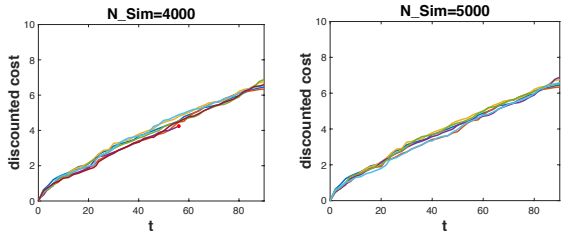


Figure 8. Discounted cost

In Fig. 8, 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 the 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 strategy, capability. Because of this, defender aggressively blocks exploit from the very beginning (t = 0), which eventually produces a 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 Fig. 8 upper left corner. As soon as, simulation count increases more possible future histories are included which results in high quality of estimation (which set of exploits to be blocked). As it is evident from Fig. 8 bottom right corner, though attacker starts with real network for 5000 trials but could not reach any goal state.

Use Case II

For use case II, we use attacker type II (φ_2) from the Table 1 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,

$$\left(\overline{P}_{er_k}(\varphi_2), \underline{P}_{er_k}(\varphi_2)\right) = (0.7,0.7) \;\; for\; er_k \in E_0$$

$$\left(\overline{P}_{ef_k}(\varphi_2), \underline{P}_{ef_k}(\varphi_2)\right) = (0.7,0.7) \;\; for\; ef_k \in E_0$$

$$\left(\overline{P}_{er_k}(\varphi_2), \underline{P}_{er_k}(\varphi_2)\right) = (0.8,0.4) \;\; for\; er_k \in \{e_4, e_5, e_7\}$$

$$\left(\overline{P}_{ef_k}(\varphi_2), \underline{P}_{ef_k}(\varphi_2)\right) = (0.9,0.6) \;\; for\; er_k \in \{e_5, e_7\}$$

$$\left(\overline{P}_{er_k}(\varphi_2), \underline{P}_{er_k}(\varphi_2)\right) = (0.7,0.5) \;\; for\; er_k \in \{e_9, e_9, e_{11}\}$$

$$\left(\overline{P}_{ef_k}(\varphi_2), \underline{P}_{ef_k}(\varphi_2)\right) = (0.9,0.7) \;\; for\; er_k \in \{e_7, e_8\}$$

attack success probability,

$$\alpha_{er_k}(\varphi_2) = \begin{cases} 0.8 & when\; er_k \in E_0 \\ 0.5 & when\; er_k \in E \backslash E_0 \end{cases}$$

$$\alpha_{ef_k}(\varphi_2) = \begin{cases} 0.85 & when\; ef_k \in E_0 \\ 0.7 & when\; ef_k \in E \backslash E_0 \end{cases}$$

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 Fig 1. In this simulation, we present the performance evaluation table to capture the attacker progression from real to real and real to fake network.

Table 5. Performance evaluation table for real to real and real to fake

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

From Table 5 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 6, we present the same simulation results for the fake network.

Table 6. Performance evaluation table for fake to fake and fake to real

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

Table 6 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.

Use Case III

For use case III, we use attacker type III (φ_3) from the Table 1 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,

$$\left(\overline{P}_{er_k}(\varphi_3), \underline{P}_{er_k}(\varphi_3)\right) = (0.8,0.2) \;\; for\; er_k \in E_0$$

$$\left(\overline{P}_{ef_k}(\varphi_3), \underline{P}_{ef_k}(\varphi_3)\right) = (0.8,0.2) \;\; for\; ef_k \in E_0$$

$$\left(\overline{P}_{er_k}(\varphi_3), \underline{P}_{er_k}(\varphi_3)\right) = (0.7,0.3) \;\; for\; er_k \in \{e_4, e_5, e_7\}$$

$$\left(\overline{P}_{ef_k}(\varphi_3), \underline{P}_{ef_k}(\varphi_3)\right) = (0.7,0.3) \;\; for\; er_k \in \{e_5, e_7\}$$

$$\left(\overline{P}_{er_k}(\varphi_3), \underline{P}_{er_k}(\varphi_3)\right) = (0.7,0.3) \;\; for\; er_k \in \{e_9, e_9, e_{11}\}$$

$$\left(\overline{P}_{ef_k}(\varphi_3), \underline{P}_{ef_k}(\varphi_3)\right) = (0.8,0.2) \;\; for\; er_k \in \{e_7, e_8\}$$

attack success probability,

$$\alpha_{er_k}(\varphi_3) = \begin{cases} 0.8 & when\; er_k \in E_0 \\ 0.5 & when\; er_k \in E \backslash E_0 \end{cases}$$

$$\alpha_{ef_k}(\varphi_3) = \begin{cases} 0.85 & when\; ef_k \in E_0 \\ 0.7 & when\; ef_k \in E \backslash E_0 \end{cases}$$

The performance evaluation table for this simulation is presented in below,

Table 7. Performance evaluation table for real to real and real to fake

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

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_4) from the Table 1 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,

$$\left(\overline{P}_{er_k}(\varphi_4), \underline{P}_{er_k}(\varphi_4)\right) = (0.9,0.7) \;\; for\; er_k \in E_0$$

$$\left(\overline{P}_{ef_k}(\varphi_4), \underline{P}_{ef_k}(\varphi_4)\right) = (0.9,0.7) \;\; for\; ef_k \in E_0$$

$$\left(\overline{P}_{er_k}(\varphi_4), \underline{P}_{er_k}(\varphi_4)\right) = (0.8,0.7) \;\; for\; er_k \in \{e_4, e_5, e_7\}$$

$$\left(\overline{P}_{ef_k}(\varphi_4), \underline{P}_{ef_k}(\varphi_4)\right) = (0.7,0.7) \;\; for\; er_k \in \{e_5, e_7\}$$

$$\left(\overline{P}_{er_k}(\varphi_4), \underline{P}_{er_k}(\varphi_4)\right) = (0.8,0.6) \;\; for\; er_k \in \{e_9, e_9, e_{11}\}$$

$$\left(\overline{P}_{ef_k}(\varphi_4), \underline{P}_{ef_k}(\varphi_4)\right) = (0.8,0.7) \;\; for\; er_k \in \{e_7, e_8\}$$

attack success probabilities,

$$\alpha_{er_k}(\varphi_4) = \begin{cases} 0.8 & when\; er_k \in E_0 \\ 0.5 & when\; er_k \in E \backslash E_0 \end{cases}$$

$$\alpha_{ef_k}(\varphi_4) = \begin{cases} 0.85 & when\; ef_k \in E_0 \\ 0.7 & when\; ef_k \in E \backslash E_0 \end{cases}$$

The performance evaluation table for this simulation is presented below,

Table 8. Performance evaluation table for real to real and real to fake

Simulation Runs	No. of Times Attacker	No. of Times Attacker	No. of Times Attacker

	Starts with Real Node	Ends on Real Node	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

From Table 8, 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 skillset (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 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 [29], [30], [31], and [32] which is also discussed in the related work section. To implement these scanning techniques, we use a python library name *libnmap* [33] which provides an API to Nmap [34] as well as python scapy framework. Based on the discussion [35], 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 [35].

Table 9. The performance statistics for all attacker types

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

Local Preference Scanning discussed in [29], 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 [35].

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.

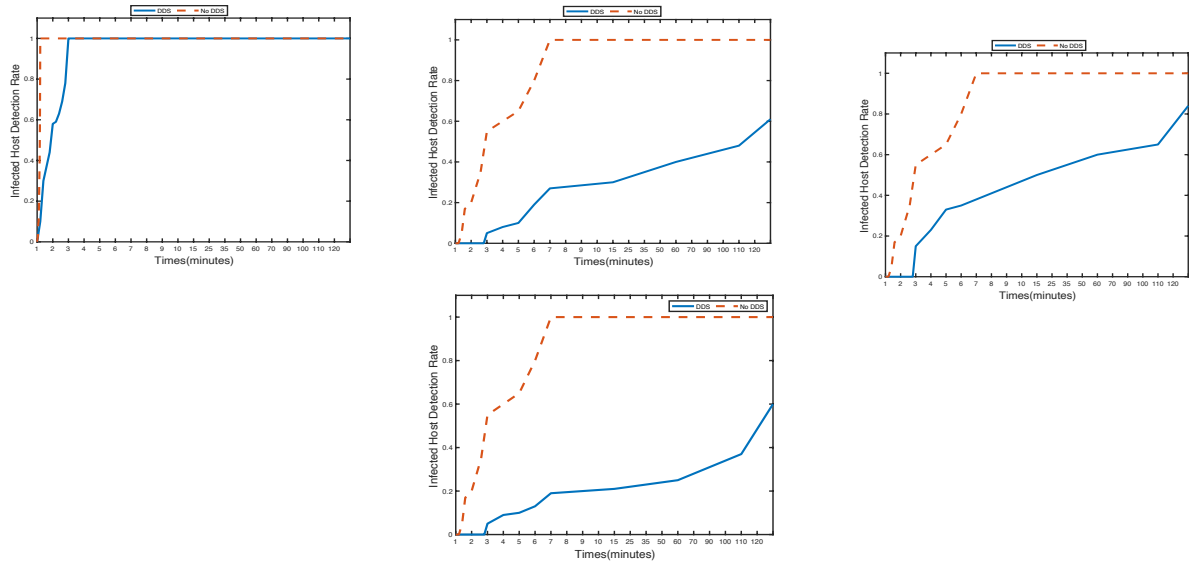


Figure 9. Average vulnerable host detection rate in minutes for the scanning strategies Preference Parallel, Local Preference, Preference Sequential, Non- Preference Sequential with and without our DDS system

In Fig. 9 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 Fig. 9 it can be inferred that defender successfully drive the attacker towards fake network by blocking vulnerabilities in the real network.

From Table 9, 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.

7. 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.

Acknowledgements.

This work is supported by the Office of the Assistant Secretary of Defense for Research and Engineering (OASD (R & E)) agreement FA8750-15-2-0120.

References

[1] Ramakrishnan CR, Sekar R. Model-based analysis of configuration vulnerabilities 1. Journal of Computer Security. 2002 Jan 1;10(1-2):189-209.

[2] Ritchey, R.W. and Ammann, P., 2000, May. Using model checking to analyze network vulnerabilities. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*(pp. 156-165). IEEE.

[3] Sheyner, O., Haines, J., Jha, S., Lippmann, R. and Wing, J.M., 2002, May. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy* (pp. 273-284). IEEE.

[4] Noel, S. and Jajodia, S., 2004, October. 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). ACM.

[5] Ammann, P., Wijesekera, D. and Kaushik, S., 2002, November. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (pp. 217-224). ACM.

[6] Silver, D. and Veness, J., 2010. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems* (pp. 2164-2172).

[7] Kocsis, L. and Szepesvári, C., 2006, September. Bandit based monte-carlo planning. In *European conference on machine learning* (pp. 282-293). Springer, Berlin, Heidelberg.

[8] Kaelbling, L.P., Littman, M.L. and Cassandra, A.R., 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, *101*(1-2), pp.99-134.

[9] Md Ali Reza Al Amin, Sachin Shetty, Laurent Njilla, Deepak Tosh and Charles Kamhoua "Online Cyber Deception System using PartiallyObservable Monte-Carlo Planning Framework", Securecomm, 2019

[10] 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.

[11] Jajodia, S., Noel, S. and O'berry, B., 2005. Topological analysis of network attack vulnerability. In *Managing Cyber Threats* (pp. 247-266). Springer, Boston, MA.

[12] B. Morin, L. M´e, H. Debar, and M. Ducass´e, "M2d2: A formal data model for ids alert correlation," in International Workshop on Recent Advances in Intrusion Detection. Springer, 2002, pp. 115–137.

[13] R. Gula, "Correlating ids alerts with vulnerability information," 2002.

[14] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," IEEE Transactions on dependable and secure computing, vol. 1, no. 3, pp. 146–169, 2004.

[15] Astrom, K.J., 1965. Optimal control of Markov processes with incomplete state information. *Journal of mathematical analysis and applications*, *10*(1), pp.174-205.

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

[17] Silver, D. and Veness, J., 2010. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems* (pp. 2164-2172).

[18] Mell, P., Scarfone, K. and Romanosky, S., 2007, June. 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).

[19] Hasan, K., Shetty, S., Hassanzadeh, A., Salem, M.B. and Chen, J., 2018, May. 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.

[20] Kurose, J.F., 2005. Computer networking: A top-down approach featuring the internet, 3/E. Pearson Education India.

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

[22] Pox—Python SDN Controller. Accessed on Mar. 21, 2016. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[23] Scapy. Accessed on Mar. 21, 2016. [Online]. Available: http://www.secdev.org/projects/scapy/

[24] Kurniawati, H., Hsu, D. and Lee, W.S., 2008, June. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In Robotics: Science and systems (Vol. 2008).

[25] Coulom, R., 2006, May. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games* (pp. 72-83). Springer, Berlin, Heidelberg.

[26] Kocsis, L. and Szepesvári, C., 2006, September. Bandit based monte-carlo planning. In *European conference on machine learning* (pp. 282-293). Springer, Berlin, Heidelberg.

[27] Auer, P., Cesa-Bianchi, N. and Fischer, P., 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*(2-3), pp.235-256.

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

[29] Al-Shaer, E., Duan, Q. and Jafarian, J.H., 2012, September. Random host mutation for moving target defense. In *International Conference on Security and Privacy in Communication Systems* (pp. 310-327). Springer, Berlin, Heidelberg.

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

[31] Jafarian, J.H., Al-Shaer, E. and Duan, Q., 2012, August. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 127-132). ACM.

[32] Weaver, N., Paxson, V., Staniford, S. and Cunningham, R., 2003, October. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid malcode*(pp. 11-18). ACM.

[33] Python API for nMap. Accessed on Mar. 21, 2016. [Online]. Available: https://libnmap.readthedocs.org/

[34] Nmap Network Scanner. Accessed on Jun. 25, 2016. [Online]. Available: https://nmap.org/

[35] Zou, C.C., Towsley, D. and Gong, W., 2006. On the performance of Internet worm scanning strategies. *Performance Evaluation*, *63*(7), pp.700-723.

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

[37] Trassare, S.T., Beverly, R. and Alderson, D., 2013, November. A technique for network topology deception. In *MILCOM 2013-2013 IEEE Military Communications Conference* (pp. 1795-1800). IEEE.

[38] Schlenker, A., Thakoor, O., Xu, H., Fang, F., Tambe, M., Tran-Thanh, L., Vayanos, P. and Vorobeychik, Y., 2018, July. 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.

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

[40] Sun, J. and Sun, K., 2016, April. DESIR: Decoy-enhanced seamless IP randomization. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications* (pp. 1-9). IEEE.

[41] Zonouz, S.A., Khurana, H., Sanders, W.H. and Yardley, T.M., 2013. RRE: A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems*, *25*(2), pp.395-406.

[42] Dunlop, M., Groat, S., Marchany, R. and Tront, J., 2012. Implementing an IPv6 moving target defense on a live network. In *Moving Target Research Symposium 2012*. Cyber-Physical Systems Virtual Organization.

[43] Department of Homeland Security. Industrial Control Systems Cyber Emergency Response Team (ICS-CERT). Accessed: Sep. 22, 2019. [Online]. Available: https://ics-cert.us-cert.gov/

[44] Achleitner, S., La Porta, T.F., McDaniel, P., Sugrim, S., Krishnamurthy, S.V. and Chadha, R., 2017. Deceiving network reconnaissance using SDN-based virtual topologies. *IEEE Transactions on Network and Service Management*, *14*(4), pp.1098-1112.