Adaptive Cyber Defense Against Multi-Stage Attacks Using Learning-Based POMDP

ZHISHENG HU, Baidu Security
MINGHUI ZHU, Pennsylvania State University
PENG LIU, Pennsylvania State University

Growing multi-stage attacks in computer networks impose significant security risks and necessitate the development of effective defense schemes that are able to autonomously respond to intrusions during vulnerability windows. However, the defender faces several real-world challenges, e.g., unknown likelihoods and unknown impacts of successful exploits. In this article, we leverage reinforcement learning to develop an innovative adaptive cyber defense to maximize the cost-effectiveness subject to the aforementioned challenges. In particular, we use Bayesian attack graphs to model the interactions between the attacker and networks. Then we formulate the defense problem of interest as a partially observable Markov decision process problem where the defender maintains belief states to estimate system states, leverages Thompson sampling to estimate transition probabilities, and utilizes reinforcement learning to choose optimal defense actions using measured utility values. The algorithm performance is verified via numerical simulations based on real-world attacks.

CCS Concepts: • **Computing methodologies** → **Reinforcement learning**; • **Security and privacy** → **Network security**;

Additional Key Words and Phrases: Reinforcement learning, adaptive cyber defense, Thompson sampling

ACM Reference format:
Zhisheng Hu, Minghui Zhu, and Peng Liu. 2020. Adaptive Cyber Defense Against Multi-Stage Attacks Using Learning-Based POMDP. *ACM Trans. Priv. Secur.* 24, 1, Article 6 (November 2020), 25 pages.
https://doi.org/10.1145/3418897

1 INTRODUCTION

With growing network threats, it is of vital importance to develop effective defenses that can protect computer networks from cyber-attacks. One-stage attacks have been widely used to compromise the hosts that the attacker can directly interact with. Malicious actions, e.g., probing, toehold, and privilege escalation, are directly taken against the target machines. Examples of one-stage attacks include SQL injection and cross-site scripting. In many scenarios, target machines could be located in the interior of a computer network. However, one-stage attacks can only compromise

Authors' addresses: Z. Hu, Baidu Security, 1195 Bordeaux Dr, Sunnyvale, CA, 94089; email: zhishenghu@baidu.com; M. Zhu, School of Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA, 16802; email: muz16@psu.edu; P. Liu, College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, 16802; email: pliu@ist.psu.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
2471-2566/2020/11-ART6 $15.00
https://doi.org/10.1145/3418897

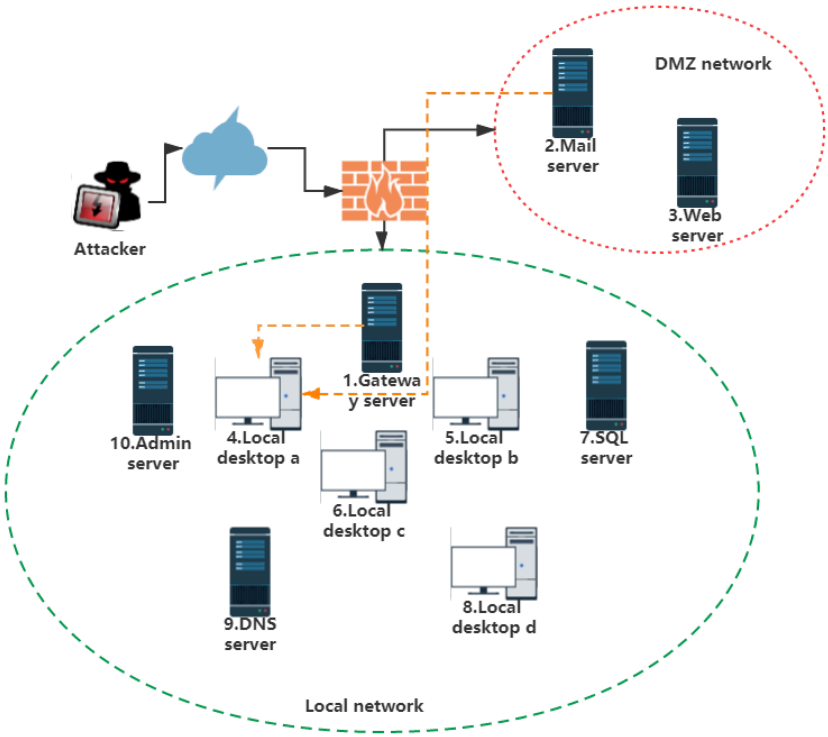

Fig. 1. An example of multi-stage attacks.

machines on the periphery of the network. In contrast, a multi-stage attack can be decomposed into a sequence of ordered actions in which each action could exploit one vulnerability to compromise a machine in the network. Once a machine is compromised, it can be used as a stepping stone to compromise its neighbor machines. The step-by-step method enables the attacker to penetrate interior layers from the periphery.

Figure 1 shows an example of multi-stage attacks. The attacker first compromises either the *Web server* by remotely exploiting IIS vulnerability in WebDAV service or the *Gateway server* by exploiting untrusted cookie in OpenSSH. Then the attacker can access local desktop *a* from the Gateway server by exploiting MS Video ActiveX stack buffer overflow or from the Web server by exploiting LICQ buffer overflow. In the real world, there are two representative classes of multi-stage attacks. One class aims to compromise as many machines in a network as possible, e.g., the example in Figure 1, WannaCry [24], etc. The other class takes several intrusion steps to reach a specific target inside a network. The attack in the DARPA 2000 dataset [52] is one example of this class. Many advanced persistent threats belong to this class as well.

A traditional defense against multi-stage attacks is to discover vulnerabilities and develop corresponding patches to remove the vulnerabilities [12]. However, the process from vulnerability discovery to patch generation is slow and deliberative. According to the Internet threat report [43], the top five vulnerabilities in 2014 were actively exploited by attackers for an average of 59 days before patches were available. The *vacancy* of effective defenses before patches are in place necessitates the development of effective defense schemes that are able to autonomously respond to intrusions during vulnerability windows, i.e., durations from when attacks are launched to when patches are released [17]. One feasible solution is adaptive cyber defense (ACD)

[8]. ACD guides the defender to identify optimal or near-optimal strategies for deploying some effective Counter-surveillance, Diversification, Detection, or Disinfection (CDDD) measures, e.g., reimage of virtual machines, in a computer network using rigorous methodologies, e.g., game theory, machine learning, and control theory. Such ACD techniques dynamically and proactively reconfigure deployed CDDD defenses so as to increase uncertainty and complexity for attackers to succeed during the vacancy. For example, Clark et al. [7] dynamically randomize the IP addresses of the machines in a computer network to prevent attacks from locating the targets.

In this article, we aim to develop ACD schemes against the first class of multi-stage attacks, i.e., those that aim to compromise as many machines as possible during the vacancy. Graphical models have been widely used in other works, e.g., [9, 19, 27], to model the causality among vulnerabilities and describe how attacks progress through a network over time in a deterministic way, i.e., the attacker definitely compromises a machine as long as the prerequisites are satisfied. Such models work well for the defender to reason all possible attack paths, but they are not suitable for the defender to synthesize effective policies as required by ACD. In reality, the probability that an exploit fails is non-trivial even if all of the prerequisites are satisfied. To better handle the reality, Bayesian attack graphs (BAGs) [20] were proposed to capture the uncertainties of successful exploits. In particular, BAGs are weighted graphs where the weights represent how likely exploits could succeed. BAGs have been used as a powerful tool to model how a multi-stage attack propagates through a network [11, 25, 48].

With BAGs, identifying optimal attack or defense strategies have been formulated as Markov decision process (MDP) problems. In many scenarios, the defender can only observe the security status of a subset of machines due to limited detection capabilities. These security problems are further formulated as partially observable Markov decision process (POMDP) [2] problems. However, the existing works of POMDP on multi-stage attacks (please refer to Section 2 for more details) neglect two real-world challenges: (1) state transition probabilities are often unknown, and (2) utility functions are unknown in many cases. The focus of this work is to develop a new ACD to address the challenges. Our rough ideas are elaborated in the following.

Unknown utility function. To quantify how "secure" the network is within a defense horizon, we need to assign a utility value to each pair of defense action and system state. After a defense action is deployed, the defender can receive a utility value to quantify the cost-effectiveness of that action. However, the defender is unaware of the utility function, i.e., the mapping from actions and system states to utility values because, e.g., the locations and the impacts of vulnerabilities are not available during vulnerability windows. We propose reinforcement learning to address the challenge. In reinforcement learning, a decision-maker selects actions on basis of its past experience (exploitation) and also by new trials (exploration). The decision-maker receives numerical utility values, which evaluate deployed actions, and guide the decision-maker to select actions that maximize accumulated utility values in the future. The decision-maker only needs to access induced utility values but is not required to know utility functions. This intriguing feature well matches our needs.

Unknown transition probabilities. In BAGs, the transition probabilities of system states are determined by how likely the exploits can succeed. However, the likelihoods usually are estimated after the comprehensive study of vulnerabilities and cannot be accessed by the defender during vulnerability windows. In this article, we consider the scenario where the transition probabilities are generated at the beginning of the attack and fixed but unknown to the defender. Some early works estimate unknown parameters solely based on the observed data in history and apply optimal actions on the basis of certainly equivalence principle, i.e., the estimated parameters are treated as the true ones. Due to lack of sufficient exploration, such approaches may lead to the convergence of the estimated parameters to incorrect values [3, 35] and result in performance degradation. To address

the challenge, the defender maintains a posterior distribution over the transition probabilities and instantiates a set of estimated transition probabilities from this posterior distribution periodically. Then the defender selects optimal defense actions according to the estimated transition probabilities by resorting to, e.g., value iteration. The preceding idea is referred to as Thompson sampling [44]. In our problem, for each state-action pair, the defender may start by assigning an arbitrary prior probability distribution over all possible transition probabilities. After an action is taken and the system evolves to the next state, the defender updates the posterior distribution. In particular, the posterior distribution of each state-action pair is a distribution over the transition probability vector where each possible successor system state is associated with one entry. In addition, the weight of each entry is updated in proportion to the count of transitions to the corresponding target state in the history (the details of posterior update will be provided in Section 4.1). Thompson sampling balances exploitation and exploration to estimate the transition probabilities. In other words, when instantiating the estimates, the defender, on one hand, will probably choose the transition probabilities that are consistent with the historical transitions and, on the other hand, can explore all possible estimates with non-zero probabilities.

Problem statement. Since the two challenges are rooted in the nature of the difficulty in effectively defending against the first class of multi-stage attacks in the real world, simultaneously addressing the two challenges in maximizing the cost-effectiveness of a set of CDDD measures (the defender chooses to deploy) is an essential cyber-defense problem.

Contribution. In this article, we utilize BAG to model the multi-stage attacks in computer networks and formulate the defense problem of interest as a POMDP problem. Then we propose a Thompson sampling–based reinforcement learning algorithm as a unified defense scheme to simultaneously address the challenges of partial observability, unknown transition probabilities, and unknown utility function. In particular, our basic algorithm proceeds in episodes. At the initialization episode, the defender starts from arbitrary estimates of system states and transition probabilities, randomly selects actions, and forms beliefs about the states based on induced utility values and observations. With the beliefs, the defender takes the most likely state (MLS) in the belief as the estimated state. At the beginning of each regular episode, the defender updates the posterior distribution of the transition probabilities based on previous actions and estimated states up to the end of the last episode, then estimates the transition probabilities by sampling from the posterior distribution. The estimated transition probabilities are used through the whole episode. The defender also uses latest received utility values to estimate the utility function at each timestep within the episode. During the regular episode, the defender performs value iterations to select optimal actions using the estimates of system states, transition probabilities, and utility function.

A limitation of the basic algorithm is high computational complexity. To mitigate the computational complexity, we introduce a Q-learning-based algorithm, which updates one Q value per timestep within each episode. The update time reduces by around 1,000 times with the same sizes of action space and observation space. Further, we conduct numerical simulations based on real-world attacks to evaluate the performance of the algorithms. The simulation results confirm that our algorithms outperform the baseline policy, which uniformly selects defense actions, in terms of aggregate utilities and increase their leads over time.

Preliminary results of this manuscript were published in our earlier work [14]. The early version assumes that the transition probabilities of system states are fully known to the defender in advance. This assumption allows the defense problem to be formulated as a belief state MDP problem. However, as discussed, this assumption is unrealistic. To relax this key assumption, the current article proposes new algorithms that can autonomously respond to intrusions even when the

transition probabilities are completely unknown in advance. The key innovation of the algorithms is to treat the transition probabilities as unknown parameters to the defender and leverage Thompson sampling–based reinforcement learning to estimate the transition probabilities on-the-fly. Experiments are redone to evaluate the performance of the new algorithms. Additional experiments are conducted to quantify the estimation errors. Additional experiments are conducted to quantify the estimation errors.

The rest of the article is organized as follows. Section 2 presents the related works. Section 3 formulates the security problem of interest as a POMDP problem and identifies the challenges. Section 4 proposes the Thompson sampling–based reinforcement learning algorithms to solve the problem. Section 5 evaluates the performance of the algorithms. The article concludes in Section 7.

2 RELATED WORKS

CDDD. We will review recent works on CDDD measures for individual hosts and computer networks. For example, Bigelow et al. [5] randomize the memory address layout of the programs in individual hosts to make vulnerabilities more difficult to be exploited. In addition, Chen et al. [6] and Xin et al. [49] randomize the layout of data structures to prevent attacks from correctly locating target data objects and further manipulating them. For computer networks, diversity is a widely used technique that equips computers with randomized implementations of software, operating systems, or hardware platforms to force attackers to target each computer individually, substantially raising the bar on network-level threats. For example, Larsen et al. [18] generate diverse implementations of programs in the target computers to mislead the attacker, and Roeder and Schneider [34] dynamically change hardware platforms and operating system attributes to complicate attacks. Besides diversity, IP blocking [26] and machine reimaging [16] also introduce uncertainties in computer networks to mitigate attacks. Although these CDDD measures could achieve satisfactory cost-effectiveness in some real-world settings, none of them can maximize the cost-effectiveness under the four domain-specific challenges elaborated previously.

POMDP. POMDP has been widely used to launch or defeat multi-stage attacks [15, 21, 22, 36, 50]. Hughes et al. [15] and Sarraute et al. [36] use POMDP to model the attacks that can only observe partial information of the targets, e.g. the versions of operating systems. Yu and Brooks [50] leverage POMDP to optimally implement address randomization where the defender can only observe whether the computers are compromised or not. In addition, Miehling et al. [21, 22] compute optimal defense countermeasures, e.g., blocking services, through solving POMDP. Although these works address the "partial observability" challenge elaborated earlier, none of them addresses the "unknown utility function" challenge or the "unknown transition probabilities" challenge.

Reinforcement learning. Reinforcement learning, a subclass of machine learning, has been increasingly applied in ACD. Zhu et al. [56] apply learning algorithms to solve internal denial-of-service attacks. In the work of Zhu and Başar [55], a Q-learning algorithm is proposed to solve dynamic configuration problems of Intrusion Detection Systems (IDS). Our earlier work [53] propose a reinforcement learning scheme to defend against Heartbleed attacks. Hu et al. [13] apply a robust adaptive learning algorithm to dynamically reconfigure platforms to defend against zero-day attacks. In addition, Zhu and Martínez [54] propose two learning-based distributed control algorithms to handle false data injection attacks in operator-vehicle networks. A clear limitation of these works is that none of these works is able to address the aforementioned challenges.

3 PROBLEM FORMULATION

In this section, we model interactions between the attacker and a network as a BAG. Then the security problem of maximizing cost-effectiveness within the defense horizon is formulated as a

Table 1. Symbols and Notations

Notation	Meaning
$\mathcal{N}, \mathcal{E}, \mathcal{P}$	Set of machines, set of edges, set of exploit probabilities
$(i,j), \rho_{ij}$	Exploit from machine i to j, likelihood that exploit (i,j) can succeed
$\mathcal{N}_L, \mathcal{N} \setminus \mathcal{N}_L$	Nodes that have no in-neighbor, nodes that have in-neighbors
$\bar{D}_k$	In-neighbors of non-leaf node k
V_i^k	k-th vulnerability of machine i
s^i, s	State of machine i, system state
T, t	Time horizon, timestep
k_m, k_m	m-th episode, beginning of episode K_m
$s_t, \tilde{s}_t$	System state, estimated system state at timestep t
a_t, o_t	Action, observation at timestep t
$\mathcal{S}, \mathcal{A}, \mathcal{A}_s, \mathcal{O}$	State space, action space, available actions from state s observation space
$\mathcal{Z}(o\|s,a)$	Probability of observing o
$u(s,a), v_t$	Utility of taking action a at s, utility value received by the defender at timestep t
$\hat{U}(s,a)$	Estimated utility of taking action a at s
$P(s'\|s,a)$	Probability of evolving to state s' after taking action a at state s
d_t, D_t	Decision rule, set of decision rules at timestep t
π, π^*, Π	Policy, optimal policy, set of all policies
$J_t^\pi(\cdot), J_t^*(\cdot)$	Expected discounted total utility from timestep t, optimal value functions
$Q_t(s,a)$	Aggregate utility for executing action a at timestep t and following π^* thereafter
$\mu_{t+1}(P)$	Posterior distribution at $t+1$ for P
$\mathrm{N}(s,a,t)$	Observed counts of reaching all successor states from (s,a) between t and $t+1$
$Dir(\boldsymbol{\alpha}(s,a))$	Dirichlet distribution
b, BU	Belief state (probability distribution over the system states), belief state update law
$\mathcal{T}_t(s,a)$	Number of visits of state-action pair (s,a) until t

POMDP. The challenges are introduced at the end of the section. The symbols and notations used in this article are summarized in Table 1.

3.1 BAGs Without Defense

We consider a computer network that consists of multiple machines. The attacker aims to compromise as many machines as possible by exploiting reachable vulnerabilities. But each exploit can only succeed with a certain probability. In this problem, the interactions between the network and the attacker can be modeled by a BAG, which is formally defined as follows.

Definition 1. A BAG is defined as a tuple $G = (\mathcal{N}, \mathcal{E}, \mathcal{P})$. $\mathcal{N} = \{1, \ldots, K\}$ is the set of machines. $\mathcal{E}$ is the set of directed edges, where each edge is an exploit and $(i,j) \in \mathcal{E}$ if and only if machine j can be compromised through machine i. $\mathcal{P}$ is the set of exploit probabilities associated with the edges, where $\rho_{ij} \in \mathcal{P}$ represents the likelihood that exploit (i,j) can succeed, i.e., how likely the attacker can successfully compromise machine j after he or she compromises machine i.

Remark 1. In general, nodes in BAGs represent attack attributes such as attacker permission levels or compromised vulnerabilities in a machine or service. Exploits are events that allow the attackers to use their current set of attributes to obtain further attributes [21]. In this article, we slightly modify the definition of the node in the BAG, i.e., each node represents if the corresponding machine is compromised or not by attackers.

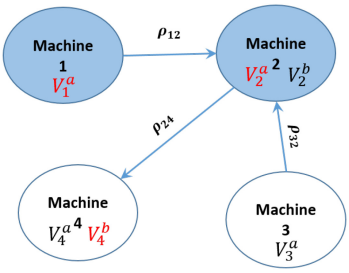

Fig. 2. A simple example of BAG.

If $(i, j) \in \mathcal{E}$, node i is referred to as an in-neighbor of node j and node j is referred to as an out-neighbor of node i. The nodes in a BAG can be classified into two categories: leaf nodes $\mathcal{N}_L \subseteq \mathcal{N}$ and non-leaf nodes, where leaf nodes do not have in-neighbors. The probability that a leaf node $l \in \mathcal{N}_L$ is compromised is denoted by ρ_l. For any non-leaf node k, we formally define its in-neighbors as $\bar{D}_k \triangleq \{i \in \mathcal{N} | (i, k) \in \mathcal{E}\}$.

NIST's Common Vulnerability Scoring System (CVSS) [37] provides a way to capture the principal characteristics of a vulnerability and produces a numerical score reflecting how easy a vulnerability can lead to a successful exploit. In particular, the likelihood that the attacker can successfully compromise machine j after he or she compromises machine i, i.e., ρ_{ij}, is quantified by the exploitability metrics of vulnerabilities in machine j. The exploitability metric of CVSS score consists of the Access Vector (AV), Access Complexity (AC), Privileges Required (PR), and User Interaction (UI). In particular, AV reflects the context by which vulnerability exploit is possible, e.g., the vulnerability can be exploited through the network, adjacent network, local access, or physical access. AC describes the conditions beyond the attacker's control that must exist to exploit the vulnerability, e.g., the target configuration settings, sequence numbers, shared secrets, etc. PR describes the level of privileges an attacker must possess before successfully exploiting the vulnerability, e.g., basic user or administrative privileges. UI describes whether a successful exploitation of this vulnerability requires a user to take some action. We will show a simple example of calculating likelihoods. Let us consider a simple example of BAG in Figure 2, where machine 2 has two vulnerabilities, denoted by V_2^a and V_2^b, and machine 4 has two vulnerabilities, denoted by V_4^a and V_4^b. In this example, machine 2 is compromised if and only if at least one of its vulnerabilities is successfully exploited, and machine 4 is compromised if and only if both of its vulnerabilities are successfully exploited. We can calculate the likelihood $\rho_{24} = \prod_{j \in \{a,b\}} 2AV(V_4^j)AC(V_4^j)PR(V_4^j)UI(V_4^j)$.

The exploitability metrics are estimated after the comprehensive study of vulnerabilities and cannot be accessed by the defender during vulnerability windows. In this work, such likelihoods are only needed to simulate real-world attacks in Section 5 and are unknown to the defender in advance.

System state. The state of machine $i \in \mathcal{N}$ at time t is either compromised (value 1) or not (value 0), i.e., $s_t^i \in \{0, 1\}, \forall i \in \mathcal{N}$. Stacking the states of the machines, we define the system state at timestep t as $s_t = (s_t^1, \ldots, s_t^K) \in \mathcal{S} = \{0, 1\}^K$.

Attacker's action. Here we consider an aggressive attacker who always tries to compromise all available machines (there are many real-world attacks, e.g., WannaCry Code red [57], that behave in this way). Upon success, the attacker can use compromised machines as stepping stones to exploit their out-neighbors. In addition, if a machine was compromised at the previous step, it remains compromised for the current step. The attacker stops when all machines are compromised. The attacker will restart compromising the machines from the periphery of the network when there is no exploitable out-neighbor.

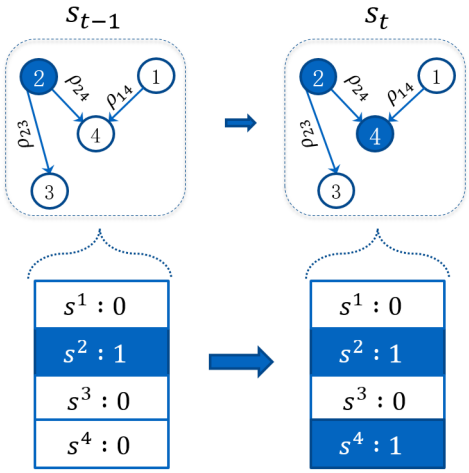

Fig. 3. An example of state transition.

State transition. Since defenses are not taken into account, state transitions are autonomous given the attacker's initial actions. In the real world, machine $i \in \mathcal{N} \setminus \mathcal{N}_L$ can be compromised at t under one of two conditions: either all of the machines in $\bar{D}_i$ are compromised or at least one of machines in $\bar{D}_i$ is compromised at $t-1$. The choice of the conditions depends on the type of machine i. We call machine i an *And-machine* if it can be compromised only all of the machines in $\bar{D}_i$ are compromised; otherwise, we call it an *Or-machine*. For example, the machine with input validation error in the SQL server (Or-machine) can be exploited from any machine (in-neighbor) that can access this server. The Admin server with stack overflow on MS SMV service (And-machine) can be successfully exploited when both the local user (one in-neighbor) and the SQL server (the other in-neighbor) are compromised [31]. Figure 3 shows an example of state transition from $s_{t-1} = (0,1,0,0)$ to $s_t = (0,1,0,1)$. The state evolution can be modeled by conditional probability $Pr(s_t|s_{t-1})$. In particular, if i is an And-machine,

$$Pr\left(s_t^i = 1|s_{t-1}\right) = \begin{cases} \prod\limits_{j\in\bar{D}_i} \rho_{ji} & \text{if } s_{t-1}^i = 0 \text{ and } s_{t-1}^j = 1, \quad \forall j \in \bar{D}_i, \\ 1 & \text{if } s_{t-1}^i = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

If i is an Or-machine,

$$Pr\left(s_t^i = 1|s_{t-1}\right) = \begin{cases} 1 - \prod\limits_{\{j\in\bar{D}_i|s_{t-1}^j=1\}} (1-\rho_{ji}) & \text{if } s_{t-1}^i = 0 \text{ and } \exists s_{t-1}^j = 1, \\ 1 & \text{if } s_{t-1}^i = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

3.2 Defense Problem: POMDP

In this section, we formulate the defense problem as a POMDP. The specifics are discussed as follows.

POMDP, as a subclass of MDP, is a sequential decision-making problem where outcomes are partially under the control of an agent. The agent's goal is to deploy a sequence of actions that enable the system to perform optimally with respect to some predetermined criterion. We give the formal definition of POMDP as follows.

Definition 2. A POMDP consists of a tuple $(T, \gamma, \mathcal{S}, \mathcal{A}, \mathcal{A}_s, \mathcal{O}, P, u, \mathcal{Z})$: $T \triangleq \{1, 2, \ldots, N\}$ is the time horizon with $N \leq \infty$, $\gamma \in (0, 1]$ is the discount factor, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $\mathcal{A}_s \subseteq A$ is the finite set of actions available from state s. $\mathcal{O}$ is the observation space, and $\mathcal{Z}(o|s, a)$ is the conditional probability of observing o after the agent takes action a and the system evolves to state s. As a result of choosing action a at state s at a timestep, the agent receives a utility $u(s, a)$ and the state at the next timestep is drawn from a transition probability $P(\cdot|s, a)$.

Defender's actions. In this work, the agent is referred to as the defender. The defender's actions include detection and reimage. Formally, at timestep t, the defender's action is $a_t = (a_t^r, a_t^d) \in \mathcal{A}_{s_t} \subset \mathcal{P}(\mathcal{N}) \times \mathcal{P}(\mathcal{N})$, where a_t^r are the machines that are reimaged, a_t^d are the machines that are monitored, and $\mathcal{P}(\mathcal{N})$ is the power set of $\mathcal{N}$.

Concern about high false positives. The IDS is a typical practice to implement detection. However, IDS alerts contain too many false positives [51]. In contrast, we assume that the detection in this work is implemented by labor analysis instead of IDS. However, labor analysis requires many more resources like domain knowledge and analyzing time than IDS. The defender can only detect a (small) subset of the machines in the network. We further assume that our detection on the selected machines does not include any false positive.

State transition. State transitions consist of a Markovian process and can be modeled by conditional probability $P(s_t|s_{t-1}, a_{t-1})$. $P(s_t^i = 1|s_{t-1}, a_{t-1})$ represents the probability that machine i is compromised at step t when the previous state is s_{t-1} and the previous defense is a_{t-1}. As mentioned in Section 3.1, for an And-machine that was not compromised nor reimaged at the previous step, it can only be compromised if all of its in-neighbors are compromised. In addition, for an Or-machine that was not compromised nor reimaged at the previous step, it can be compromised if at least one of its in-neighbors is compromised. If the machine was compromised at the previous step, it remains compromised if it was not reimaged. In other cases, the probability for this machine to be compromised is 0. With a slight change of Equations (1) and (2), the probability is given as follows. If $i \in \mathcal{N}$ is an And-machine,

$$P\left(s_t^i = 1|s_{t-1}, a_{t-1}\right) = \begin{cases} \prod\limits_{j \in \bar{D}_i} \rho_{ji} & \text{if } s_{t-1}^i = 0, s_{t-1}^j = 1, \forall j \in \bar{D}_i \text{ and } i \notin a_{t-1}^r, \\ 1 & \text{if } s_{t-1}^i = 1 \text{ and } i \notin a_{t-1}^r, \\ 0 & \text{otherwise.} \end{cases}$$

If $i \in \mathcal{N}$ is an Or-machine,

$$P\left(s_t^i = 1|s_{t-1}, a_{t-1}\right) = \begin{cases} 1 - \prod\limits_{\{j \in \bar{D}_i | s_{t-1}^j = 1\}} (1 - \rho_{ji}) & \text{if } s_{t-1}^i = 0 \text{ and } i \notin a_{t-1}^r, \\ 1 & \text{if } s_{t-1}^i = 1 \text{ and } i \notin a_{t-1}^r, \\ 0 & \text{otherwise.} \end{cases}$$

Observation. Due to limited resources, the defender can only monitor a subset of machines at each time. Thus, the defender is only aware of partial system states, which are referred to as observations. In particular, the defender receives an observation at timestep t, denoted by o_t. Observation generation can be modeled by an observation kernel $\mathcal{Z}(\cdot|s_t, a_{t-1})$, which presents the probability that the defender receives observation o when the previous action is a_{t-1} and the system state evolves to s_t. In this article, we simply use the states of the machines in a_{t-1}^d as observation, i.e., $o_t = (s_t^{i_1}, \ldots, s_t^{i_k})$, where $i_1, \ldots, i_k \in a_{t-1}^d$. The observation kernel $\mathcal{Z}(\cdot|s_t, a_{t-1})$ is given by

$$\mathcal{Z}(o|s_t, a_{t-1}) = \begin{cases} 1 & \text{if } o = (s_t^{i_1}, \ldots, s_t^{i_k}) \text{ and } i_1, \ldots, i_k \in a_{t-1}^d, \\ 0 & \text{otherwise.} \end{cases}$$

Utility function. The defender aims to find a defense policy to keep the network "secure." Utility functions are introduced to quantify security levels of the network. At timestep t, after taking action a_t in state s_t, the defender receives a utility value $u(s_t, a_t) \triangleq r(s_t, a_t) - c(a_t)$, where $r(s_t, a_t)$ is the reward of keeping the network secure and $c(a_t)$ is the cost induced by the action a_t, e.g., resources required by reimage or detection. In particular, $r(s, a)$ is defined according to the measurable confidentiality, integrity, and availability (CIA) impacts, $r(s, a) \triangleq R - \sum_{\{s^i \in s | s^i = 1, i \notin a^r\}} [I_C(i) + I_I(i) + I_A(i)]$, and $c(a)$ is defined according to the availability impact, $c(a) \triangleq \sum_{\{i \in a^r | s^i = 0\}} I_A(i)$. Here, R represents the base security level of the clean network, e.g., the number of clean machines. $I_C(i), I_I(i)$, and $I_A(i)$ are the CIA impacts induced by the successful exploitation of the vulnerabilities on machine i. For example, $I_C(i)$ could be the number of suspicious read operations in machine i, $I_I(i)$ could be the number of modified files stored in machine i, and $I_A(i)$ could be the number of lost new connections or the remaining disk space of machine i [29, 37]. We choose CIA impacts because they are commonly considered as the three most crucial components of network security.

Recall that we aim to design effective defense schemes during vulnerability windows when the defender has not fully studied the vulnerabilities and does not know their impacts. Therefore, in this work, we assume that the defender is unaware of the utility function in advance. Instead, IDS can measure the utility values according to the measurements in the compromised machines or the network in real time.

Defender's knowledge and goal. The defender knows the observation kernel, previous actions, and observations up to t. But the system state trajectory, transition probabilities, and utility function are unknown to the defender. We formally define the information available to the defender at time t as $I_t \triangleq (\mathcal{Z}, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t)$. The defender aims to find a defense policy to maximize the aggregate utility.

A *decision rule* prescribes a procedure for action selection in each state at a specified timestep. We will focus on deterministic decision rules because they are easy to implement and evaluate. A measurable mapping $d_t : \mathcal{S} \to \mathcal{A}_{s_t}$ is called a *decision rule* that specifies the action when the system is in state s_t at timestep t. A sequence of decision rules $\pi = (d_1, d_2, \ldots, d_N)$ is called a *policy* or *strategy*. Let D_t denote the set of decision rules at timestep t, and let $\Pi = D_1 \times D_2 \times \cdots \times D_N$ denote the set of all policies.

Let $J_1^\pi(s) \triangleq \mathbb{E}[\sum_{t=1}^N \gamma^t u(s_t, d_t(s_t))]$ represent the expected discounted total utility if policy π is used and the system's initial state is s. Here, $\gamma \in (0, 1]$ is the discounted factor. The defender's goal is to maximize the aggregate utility over T as follows:

$$\begin{aligned} &\max_{\pi \in \Pi} J_1^\pi(s) = \mathbb{E}\left[\sum_{t=1}^N \gamma^t u(s_t, a_t)\right] \\ &\text{subject to } a_t = d_t(s_t) \\ &\qquad s_t \leftarrow P(\cdot | s_{t-1}, a_{t-1}). \end{aligned} \tag{PA}$$

3.3 Challenges

In this article, we consider a special case of POMDP where the system dynamics are unknown. In other words, the actual transition probabilities P are generated at the beginning and then fixed but unknown to the defender. In addition, the defender is not aware of the system state trajectory. Instead, the defender receives an observation each timestep. In addition, the defender does not know the utility function u and can only receive a utility value at timestep t. To highlight that u is unknown, we denote $v_t = u(s_t, a_t)$ as the utility value received by the defender at timestep t.

4 LEARNING-BASED ALGORITHMS FOR THE PROBLEM

In this section, we propose two learning-based algorithms to solve problem (PA) subject to the aforementioned challenges.

4.1 Thompson Sampling

To deal with unknown transition probabilities, we treat P as an unknown parameter of the MDP and focus on a Bayesian framework on this parameter [28]. In other words, we propose a learning process to estimate the parameter P by computing the posterior distribution of P using observed information and a prior distribution. In particular, we partition the time horizon T into M relatively long episodes $k_1, \ldots k_M$, and at the beginning of each episode k_m, the parameter P_{k_m} is randomly sampled from the posterior distribution, then actions are selected based on the sampled parameter during the episode. This idea is called *Thompson sampling* or the *posterior sampling method* [44].

Formally, let μ_1 be the prior distribution for P. At the end of timestep $t \geq 1$, the defender takes an action a_t and observes the new state s_{t+1}. Then the posterior distribution at $t+1$ for P is updated based on Bayes' rule:

$$\mu_{t+1}(P) = \frac{\mu_t(P)P(s_{t+1}|s_t, a_t)}{\int_{P'} \mu_t(P')P'(s_{t+1}|s_t, a_t)}.$$

However, the update law needs to compute the full posterior joint distribution over states. Unfortunately, this often requires calculating intractable integrals. One way to achieve algebraic simplicity is called *conjugate prior*. If all of the possible posterior distributions are in the same probability distribution family as the prior probability distribution, the prior and posterior are then called *conjugate distributions*. Then the prior distribution is called a *conjugate prior for the likelihood function*. Thus, if a suitable distribution family is chosen as the prior, then the posterior distribution lives in the same family no matter what observations are received. In the MDP problem with finite states, one common type of prior distribution is the Dirichlet distribution [28, 35, 42], which is a distribution over multinomial distributions and a conjugate prior for multinomial distribution. Since the MDP has the Markov property, the transition probabilities associated with a state-action pair are independent of the sequence of states and actions that lead to that state. Hence, the posterior distribution for P is represented independently for each state-action pair (s, a) as a distribution over the transition probability vector $\mathbf{p}_{s,a} \triangleq [p_{s,a}^{s_1}, \ldots, p_{s,a}^{s_n}] = [P(s_1|s, a), \ldots, P(s_n|s, a)]$, where each element in the vector is the probability of transiting into a possible successor state and $\mathbf{p}_{s,a}$ is a multinomial distribution. Therefore, we choose Dirichlet distribution as the posterior distribution for $\mathbf{p}_{s,a}$. Formally, the Dirichlet distribution for $\mathbf{p}_{s,a}$ is defined as $Dir(\alpha(s_1, s, a), \ldots, \alpha(s_n, s, a))$ with following probability density function:

$$f(P(s_1|s, a), \ldots, P(s_n|s, a); \alpha(s_1, s, a), \ldots, \alpha(s_n, s, a)) = \frac{\Gamma\left(\sum_{i=1}^n \alpha(s_i, s, a)\right)}{\prod_{i=1}^n \Gamma(\alpha(s_i, s, a))} \prod_{i=1}^n P(s_i|s, a)^{\alpha(s_i,s,a)-1},$$

where $\boldsymbol{\alpha}(s, a) = (\alpha(s_1, s, a), \ldots, \alpha(s_n, s, a))$ are positive-valued parameters and Γ is the gamma function. Intuitively, $\alpha(s_i, s, a)$ represents the number of reaching the successor state of s_i from state-action pair (s, a) [42].

It is proven in the work of Friedman and Singer [10] that Dirichlet distribution can be used as an incremental parametric posterior for multinomial distributions. At timestep t, we can learn the parameters based on historical transitions up to t. In particular, let us denote the learned parameters at t by $\boldsymbol{\alpha}(s, a, t) = (\alpha(s_1, s, a, t), \ldots, \alpha(s_n, s, a, t))$ and the posterior distribution of $\mathbf{p}_{s,a}$ at t by $\mu_t(\mathbf{p}_{s,a}) = Dir(\boldsymbol{\alpha}(s, a, t))$. Let $\mathbf{N}(s, a, t) = (N_{s_1}(s, a, t), \ldots, N_{s_n}(s, a, t))$ be the observed counts of reaching all successor states from (s, a) between t and $t+1$, then the posterior distribution is

updated as follows

$$\mu_{t+1}(\mathbf{p}_{s,a}) = Dir(\boldsymbol{\alpha}(s,a,t) + \mathbf{N}(s,a,t)).$$

The update rule avoids the calculation of intractable integral in Bayes' rule. Note that the update of the posterior distribution requires the information of system states, which are unknown in our problem. In what follows, we introduce how to estimate the states.

4.2 State Estimation

One might simply take an observation as the state and then come up with decision rules that map from $\mathcal{O}$ to $\mathcal{A}_s$. However, the transitions of the observations might not necessarily be Markovian, because multiple states could be mapped to the same observation under different actions. As a result, an optimal policy directly based on observations may not perform well. To address the challenge, the defender maintains a probability distribution over the states, which is called the *belief state*. Let $\mathcal{B}$ be the belief state space (the set of all possible probability distributions over $\mathcal{S}$) and $b(s)$ be the probability assigned to state s when the belief state is b. The belief state at t assigns probability to each state s given all available information in history, i.e., $b_t(s) = Pr(s|a_0, o_1, b_0, \ldots, a_{t-1}, o_t, b_{t-1})$. In addition, the belief state update law, denoted as BU, incorporates the latest action and observation. The output of the update law is written as $BU(b_{t-1}, a_{t-1}, o_t)$, which takes the last belief state b_{t-1}, action a_{t-1}, and the current observation o_t as inputs and generates the updated belief state b_t as follows:

$$\begin{aligned} Pr(s'|a_{t-1}, o_t, b_{t-1}) &= \frac{Pr(o_t|s', a_{t-1}, b_{t-1}) Pr(s'|a_{t-1}, b_{t-1})}{Pr(o_t|a_{t-1}, b_{t-1})} \\ &= \frac{\mathcal{Z}(o_t|s', a_{t-1}) \sum_{s\in\mathcal{S}} P(s'|s, a_{t-1}) b_{t-1}(s)}{\sum_{s'\in\mathcal{S}} \mathcal{Z}(o_t|s', a_{t-1}) \sum_{s\in\mathcal{S}} P(s'|s, a_{t-1}) b_{t-1}(s)}. \end{aligned} \tag{3}$$

With the belief state, we propose a simple state estimation rule according to the idea of the MLS [38]. In other words, we take the state with the highest probability in the current belief state as the estimated state:

$$\tilde{s}_t = \arg\max_{s\in\mathcal{S}} b_t(s).$$

4.3 Utility Function Estimation

To address the challenge of unknown utility function, we use the latest received utility value to estimate the value of $u(s,a)$ for any state-action pair (s,a). In particular, we maintain an estimated function $\hat{U}(s,a) \triangleq u(\tilde{s}_\tau, a_\tau)$, where $\tau \le t$ is the last time when $\tilde{s}_\tau = s$ and $a_\tau = a$. The reasons for using only the latest received utility value are as follows. First, the utility function is deterministic in this problem. Second, we optimistically assume that the estimated state $\tilde{s}_t$ will approach the real state s_t as time goes by. In Section 5, we experimentally show that the estimated $\hat{U}(s,a)$ can be close to $u(s,a)$.

4.4 Main Idea

Given the fact that the belief states provide sufficient statistics of the history [40], the defender would select actions on the basis of belief states [2, 39]. With a slightly abused definition of the decision rule and the policy, i.e., $d_t : \mathcal{B} \to \mathcal{A}$, the defender might want to solve the following

problem:

$$\begin{aligned} \max_{\pi\in\Pi} J_1^{\pi}(b) &= \mathbb{E}\left[\sum_{t=1}^{N} \gamma^t U(b_t, a_t)\right] \\ \text{subject to } a_t &= d_t(b_t) \\ b_t &\leftarrow B_t(\cdot|b_{t-1}, a_{t-1}), \end{aligned} \tag{PB}$$

where $B_t(\cdot|b,a)$ is the belief transition probability at t and $U(b,a) = \sum_{s\in\mathcal{S}} b(s)u(s,a)$ is the expected immediate utility from executing action a at state s given the belief state b. The transition probability from belief state b_{t-1} to $b_t \in \mathcal{B}$ is defined as follows:

$$B_t(b_t|b_{t-1}, a_{t-1}) = \sum_{\{o\in O|BU(b_{t-1},a_{t-1},o)=b_t\}} Pr(o|a_{t-1}, b_{t-1}),$$

where

$$\begin{aligned} &Pr(o|a_{t-1}, b_{t-1}) \\ &= \sum_{s'\in\mathcal{S}} \mathcal{Z}(o|s', a_{t-1}) \sum_{s\in\mathcal{S}} P(s'|s, a_{t-1}) b_{t-1}(s). \end{aligned}$$

If there is no $o \in O$ such that $BU(b_{t-1}, a_{t-1}, o) = b_t$, then $B_t(b_t|b_{t-1}, a_{t-1}) = 0$. In addition, it is shown that the process of updating the belief state is Markovian [2], i.e., the current belief state depends only on the latest belief state and action.

The update of belief state requires the knowledge of transition probability P. As mentioned before, we can use sampled transition probability P_{k_m} and treat it as the actual transition probability in episode k_m. With P_{k_m}, we can solve (PB) within one episode. Here, $J_t^{\pi}(b)$ is the expected discounted total utility from any timestep $t \in k_m$ to the end of k_m if policy π is used and the belief state at t is b. Then the defender aims to compute the optimal value functions $J_t^*(b) \triangleq \max_{\pi\in\Pi} J_t^{\pi}(b), \forall b \in \mathcal{B}$, and $\forall t \in k_m$. By the principle of optimality [4], the optimal value function at t can be calculated backward from the optimal value function at $t+1$ as follows:

$$J_t^*(b) = \max_{a\in\mathcal{A}} \left[U(b,a) + \gamma \sum_{o\in O} Pr(o|a,b) J_{t+1}^*(BU(b,a,o))\right].$$

We can use dynamic programming (DP) to get the estimates of the optimal value functions J_t^*. Let us consider the special case where the episode length is infinite. The optimal value functions can be rewritten as $J^*(b) \triangleq \max_{\pi\in\Pi} \mathbb{E}[\sum_{t=1}^{\infty} \gamma^t U(b_t, d_t(b_t))]$, where $\pi = (d_1, d_2, \ldots)$. We begin with any initial estimated value function $J^0(b)$, then the n-th estimated value function is constructed from the $(n-1)$-th by the recursive equation:

$$J^n(b) = \max_{a\in\mathcal{A}} \left[U(b,a) + \gamma \sum_{o\in O} Pr(o|a,b) J^{n-1}(BU(b,a,o))\right]. \tag{4}$$

By applying (4) repeatedly over n, J^n will converge to the fixed point J^* [4].

However, the value iteration (4) cannot be carried out because $\mathcal{B}$ is infinite. An approximation idea called *point based* [30, 39] was proposed. In other words, perform the value iteration only on a finite and incrementally expanded subset of belief states instead of the whole belief state space $\mathcal{B}$. There are several ways to construct the set. For example, Pineau et al. [30] start from the initial belief state, compute all possible successors of each belief state, and then the furthest successor of each belief state is added. Such expansion doubles the size of the set at each timestep and attempts to cover the space of reachable belief states as best as possible. But this method requires significant computational effort because computing all successors for each belief state requires $O(|O||\mathcal{A}|)$

belief state update operations described in (3), where each belief state update operation requires $O(|\mathcal{S}|^2)$ products. This method did not show the ability to handle a large number of states, actions, and observations. In our evaluation, the update requires more than 10^{10} products and takes more than 2 hours to finish one timestep. Another method [41] also starts from the initial belief state but expands the set by adding the new belief states that appear over time. The estimated value function J^n can converge only when the value of each $J^n(b)$ gets updated infinitely often. This method only adds one successor into the set at each timestep. That means this method needs a (relatively) large number of timesteps to visit enough belief states. Thus, if no belief states are revisited during an episode, $J^n(b)$ does not work well. In addition, for each belief state, it could have $O(|\mathcal{O}||\mathcal{A}|)$ successors. In our evaluation, it could take more than 600,000 timesteps to revisit a belief state.

Therefore, in this work, we do not select actions on the basis of belief states. Instead, we select actions based on the estimated states. As mentioned before, we perform Thompson sampling to deal with unknown transition probabilities. In particular, we make no major assumptions and start with a uniform distribution (a special case of Dirichlet distribution with all α equal to 1) for P and then maintain a posterior distribution for P according to the estimated system states. At the beginning of each episode, we randomly sample $P_{k_m} \triangleq [\mathbf{p}^{k_m}_{s_1,a_1}, \ldots, \mathbf{p}^{k_m}_{s_{|\mathcal{S}|},a_{|\mathcal{A}|}}]^T$ from the posterior distribution. The details of how to maintain the posterior distribution is provided in Section 4.1. With a slightly abused definition of estimated value function J^n_m, we perform the following value iteration repeatedly in episode k_m: $J^n_m(s) = \max_{a\in\mathcal{A}_s}[\hat{U}(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{k_m}(s'|s,a) J^{n-1}_m(s')]$. Then we select an action that maximizes J^n_m at the n-th timestep in episode k_m.

4.5 Algorithm Statement

Let t_m be the first step of the episode k_m and $\mathbf{N}(s,a,m) = (N_{s_1}(s,a,m), \ldots, N_{s_n}(s,a,m))$ be the observed counts of reaching all successor states from (s,a) during episode k_m. Algorithm 1 starts from an arbitrary belief state b_1 (line 1) and draws a transition probability $\mathbf{p}^{k_1}_{s,a}$ from uniform distribution (line 5). For the initial episode (lines 7 to 15), the defender uniformly selects an action for each timestep (line 8), updates the belief state (line 12), and estimates the successor state (line 13). The counts of reaching the estimated successor states from current state-action pairs are recorded (lines 14 and 35). Then the posterior distribution for $pp_{s,a}$ is updated accordingly (Line 18). After an action is selected, the value of state-action pair $(\tilde{s}_t, a_t)$ in $\hat{U}$ is updated by the received utility value of v_t (lines 10 and 29). During each episode k_m ($m \geq 2$), the initial values of J^0_m are set based on $\hat{U}$ for all states $s \in \mathbb{S}$ (line 21). In addition, the estimated value function J^{n+1}_m is updated with latest $\hat{U}$ (line 30).

The defender might choose the action greedily, i.e., choosing the action that maximizes the estimate of value function $(\hat{U}(\tilde{s}_t, a) + \gamma \sum_{s'\in\mathcal{S}} P_{k_m}(s'|\tilde{s}_t, a) J^n_m(s')$ in line 27). However, the estimated value function J^n_m can deviate from the optimal value function J^*_t greatly because greedy policy only visits a portion of state-action pairs. To address this issue, Algorithm 1 follows the idea of "ϵ-greedy exploratory policy" [1, 23, 45]. In particular, the defender chooses a random action with a diminishing probability $\epsilon(t) \in [0,1]$ (the exploration phase) and chooses the greedy action $\arg\max_{a\in\mathcal{A}_{\tilde{s}_t}}[\hat{U}(\tilde{s}_t, a) + \gamma \sum_{s'\in\mathcal{S}} P_{k_m}(s'|\tilde{s}_t, a) J^n_m(s')]$ otherwise (the exploitation phase). The exploration ensures that all state-action pairs are visited infinitely often. With the exploration, the algorithm avoids being trapped in the sub-optimal actions. By decreasing $\epsilon(t)$ over time, the defender will rely more on the exploitation.

4.6 Computational Complexity

We proceed to discuss the computational complexity of the update of the estimated value functions J^{n+1}_m (line 30 of Algorithm 1). The update of $J^{n+1}_m(s)$ for all $s \in \mathcal{S}$ needs $O(|\mathcal{S}||\mathcal{A}_s|)$ products. The

ALGORITHM 1: Thompson sampling with unknown utility function and states

Assign initial belief b_1 and arbitrarily choose a_1;
Assign prior distribution $\mu_1(\mathbf{p}_{s,a}) = Dir(\alpha(s_1, s, a, 1) = 1, \ldots, \alpha(s_n, s, a, 1) = 1))$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$;
Estimate initial utility values $\hat{U}(\tilde{s}, a) = 0, \forall (s, a) \in \mathcal{S} \times \mathcal{A}$;
Estimate initial state $\tilde{s}_1 = \arg\max_{s \in \mathcal{S}} b_1(s)$;
Sample $\mathbf{p}_{s,a}^{k_1} \sim \mu_1(\mathbf{p}_{s,a})$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$;
$\mathbf{N}(s, a, 1) = (0, \ldots, 0)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$;
for $t \in k_1$ **do**
    Uniformly select an action $a_t \in \mathcal{A}_{\tilde{s}_t}$ and execute a_t;
    Receive utility value $v_t = u(s_t, a_t)$;
    Update $\hat{U}(\tilde{s}_t, a_t) = v_t$;
    Receive observation o_{t+1};
    Update belief state $b_{t+1} = BU(b_t, a_t, o_{t+1})$;
    Estimate successor state $\tilde{s}_{t+1} = \arg\max_{s \in \mathcal{S}} b_{t+1}(s)$;
    $N_{\tilde{s}_{t+1}}(\tilde{s}_t, a_t, 1) = N_{\tilde{s}_{t+1}}(\tilde{s}_t, a_t, 1) + 1$;
end for
$m = 2$;
while $m \leq M$ **do**
    $\mu_{t_m}(\mathbf{p}_{s,a}) = Dir(\boldsymbol{\alpha}(s, a, m-1) + \mathbf{N}(s, a, m-1))$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$;
    $\mathbf{N}(s, a, m) = (0, \ldots, 0)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$;
    $n = 0$;
    $J_m^n(s) = \max_{a \in \mathcal{A}_s} \hat{U}(s, a)$ for all $s \in \mathcal{S}$;
    Sample $\mathbf{p}_{s,a}^{k_m} \sim \mu_{t_m}(\mathbf{p}_{s,a})$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$;
    for $t \in k_m$ **do**
        With probability $\epsilon(t)$:
            Uniformly select an action $a_t \in \mathcal{A}_{\tilde{s}_t}$;
        With probability $1 - \epsilon(t)$:
            $a_t \in \arg\max_{a \in \mathcal{A}_{\tilde{s}_t}} [\hat{U}(\tilde{s}_t, a) + \gamma \sum_{s' \in \mathcal{S}} P_{k_m}(s' | \tilde{s}_t, a) J_m^n(s')]$;
        Receive utility value v_t;
        Update $\hat{U}(\tilde{s}_t, a_t) = v_t$;
        $J_m^{n+1}(s) = \max_{a \in \mathcal{A}_s} [\hat{U}(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{k_m}(s' | s, a) J_m^n(s')]$, for all $s \in \mathcal{S}$;
        $n = n + 1$;
        Receive observation o_{t+1};
        Update belief state $b_{t+1} = BU(b_t, a_t, o_{t+1})$;
        Estimate successor state $\tilde{s}_{t+1} = \arg\max_{s \in \mathcal{S}} b_{t+1}(s)$;
        $N_{\tilde{s}_{t+1}}(\tilde{s}_t, a_t, m) = N_{\tilde{s}_{t+1}}(\tilde{s}_t, a_t, m) + 1$;
    end for
    $m = m + 1$;
end while

computational complexity could be very high when the state space is large. We propose a Q-learning version of Algorithm 1 that can reduce the computational complexity significantly.

4.7 Q-Learning-Based Algorithm

We introduce an intermediate state-action value function called the *Q-function*:

$$Q_t(s, a) = u(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) J_{t+1}^*(s').$$

$Q_t(s, a)$ is the aggregate utility for executing action a at timestep t and following the optimal policy π^* thereafter. The relation between $Q_t(s, a)$ and $J_t^*(s)$ is given by $J_t^*(s) = \max_{a \in \mathcal{A}_{\tilde{s}_t}} Q_t(s, a)$.

ALGORITHM 2: Q-Thompson sampling with unknown utility function and states

...
20: $n = 0$;
21: Let $\hat{Q}_m^n(s,a) = \hat{U}(s,a)$ for all $(s,a) \in \mathbb{S} \times \mathcal{A}$;
22: **for** $t \in k_m$ **do**
23: With probability $\epsilon(t)$:
24: Uniformly select an action $a_t \in \mathcal{A}_{\tilde{s}_t}$;
25: With probability $1 - \epsilon(t)$:
26: $a_t \in \arg\max_{a \in \mathcal{A}_{\tilde{s}_t}} [\hat{Q}_m^n(\tilde{s}_t, a)]$;
27: Receive a utility value v_t;
28: Update $\hat{U}(\tilde{s}_t, a_t) = v_t$;
29: Observe o_{t+1};
30: Update belief state $b_{t+1} = BU(b_t, a_t, o_{t+1})$;
31: Estimate successor state $\tilde{s}_{t+1} = \arg\max_{s \in \mathcal{S}} b_{t+1}(s)$;
32: $\hat{Q}_m^{n+1}(\tilde{s}_t, a_t) = (1 - \beta_m)\hat{Q}_m^n(\tilde{s}_t, a_t) + \beta_m \left[v_t + \max_{a' \in \mathcal{A}_{\tilde{s}_{t+1}}} \hat{Q}_m^n(\tilde{s}_{t+1}, a')\right]$;
33: $n = n + 1$;
34: **end for**
...

In addition, we estimate the Q-functions Q_t instead of the optimal value functions J_t^* by applying Q-learning [47] in Algorithm 2. In particular, let $\hat{Q}_m^n$ be the n-th estimate of Q-function in episode k_m, then the $(n+1)$-th estimate is updated from $\hat{Q}_m^n$ by the recursive equation shown in line 32 of Algorithm 2. The recursive equation makes a correction of $\hat{Q}_m^n(\tilde{s}_t, a_t)$, i.e., the value of state-action pair $(\tilde{s}_t, a_t)$, based on the newly received utility value v_t.

The defender chooses the action that maximizes the estimate of Q-function by also following ϵ-greedy exploratory policy. Algorithm 2 updates $\hat{Q}_m^{n+1}$ asynchronously. In other words, it updates the value of $\hat{Q}_m^{n+1}$ for a single state-action pair at each timestep.

Compared with Algorithm 1, Algorithm 2 only updates one value of $\hat{Q}_m^{n+1}$. Therefore, Algorithm 2 only requires at most $|\mathcal{A}_{\tilde{s}_{t+1}}|$ comparisons for the update at timestep t.

5 EVALUATION

In this section, we conduct numerical simulations to evaluate the performance of Algorithms 1 and 2. The simulations are based on the following real-world settings of the ACD problem on BAGs.

5.1 Evaluation Setup

We setup a test network similar to the one in the work of Poolsappasit et al. [31], which is shown in Figure 1. The network consists of 10 machines located in two subnets. The Web server and Mail server are located in the DMZ network while the local desktops, the Gateway server, the SQL server, the DNS server, and the Admin server are located in the local network. A firewall is installed to prevent remote access to the internal hosts. All communications to external parties are delivered through the Gateway server. The vulnerabilities are chosen based on the work of Poolsappasit et al. [31] and listed in Table 2. These vulnerabilities can produce multiple attack scenarios. In the evaluation, we use a BAG to simulate one attack scenario where the attacker starts from the Gateway server, the Mail server, or the Web server and tries to compromise the whole network. From one of the three entrances, local desktop a can be compromised by exploiting MS Video ActiveX buffer overflow. From the Web server, local desktop b can be compromised by exploiting LICQ buffer overflow. With local user privilege, local desktop c can be compromised

Table 2. Vulnerabilities in the Test Network

Machine	Vulnerability	CVE#
Gateway server	Untrusted cookie in OpenSSH	2007-4752
Mail server	Error message information leakage	2008-3060
Web server	IIS vulnerability in WebDAV service	2009-1535
Local desktop a	MS Video ActiveX stack buffer overflow	2009-0015
Local desktop b	LICQ buffer overflow	2001-0439
Local desktop c	Remote login	2008-3610
Local desktop d	Remote code execution	2008-0840
DSN Server	DNS cache poisoning	2008-1447
SQL server	SQL injection	2008-5416
Admin server	MS SMV service Stack buffer overflow	2008-4050

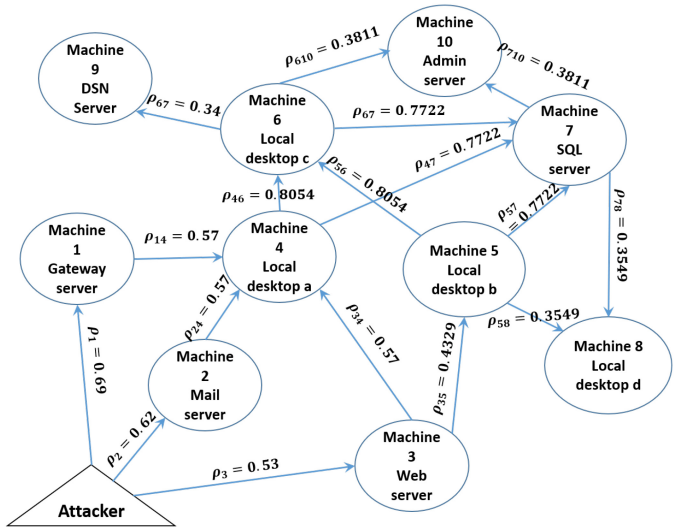

Fig. 4. BAG of the test network.

by exploiting remote login. The SQL server can be compromised through any of the three local desktops by exploiting SQL injection. Finally, with information in local desktop b and the SQL server, local desktop d can be compromised with root privilege; with local user privilege of desktop c, the DSN server can be compromised by exploiting DNS cache poisoning, and with information in local desktop c and the SQL server, the Admin server can be compromised by exploiting MS SMV service Stack buffer overflow.

BAG. We simulate the preceding attack scenario with the BAG shown in Figure 4. The Web server and the Gateway server are leaf nodes and accessible to the remote attacker. The remaining machines are non-leaf nodes. The edges show the possible exploits in the network. For example, the local desktop b can only be attacked after the Web server is compromised. As mentioned in Section 3.1, the exploit probabilities are calculated based on the exploitability metric of CVSS scores. For each $(i, j) \in \mathcal{E}$, we calculate the exploit probability as follows:

$$\rho_{ij} = 2 \times AV(j) \times AC(j) \times PR(j) \times UI(j),$$

where $AV(j)$, $AC(j)$, $PR(j)$, and $UI(j)$ are the corresponding scores of the exploitability components of the vulnerability on machine j.

System state. The system state space has $|\mathcal{S}| = 2^{10} = 1{,}024$ states. Each state reflects which machines are compromised.

Attacker's knowledge and action. The attacker wants to compromise as many machines as possible. The attacker stops when the whole network is compromised. In the evaluation, the attacker chooses one of the leaf nodes to start the attack and he or she knows the states of the machines. If the attacker has no non-leaf nodes to exploit for the next timestep, e.g., all machines are recovered, he or she will restart the attack from the leaf nodes again.

Defender's action. Recall that we use labor analysis to implement detection. Therefore, the defender can only detect a subset of the machines in the network due to limited resources. In the simulations, each defender's action is to detect 3 out of 10 machines and to reimage at most 3 out of 10 machines. There are a total of $|\mathcal{O}| = 2^3 \times \binom{10}{3} = 960$ observations and $|\mathcal{A}| = \binom{10}{3} \times (\binom{10}{0} + \binom{10}{1} + \binom{10}{2} + \binom{10}{3}) = 21{,}000$ actions.

Utility. As mentioned in Section 3.2, utilities are introduced to quantify security levels of the network. In the evaluation, we use the utility function $u(s, a) = r(s, a) - c(a)$ defined in Section 3.2, which consists of a reward part and a cost part. In the simulation, we do not have real measurements in the machines or network, e.g., the disk space, modified files. Here we use the impact scores of CVSS to simulate the CIA impacts. Each impact score is a real number scaling from 0 to 10, and a higher score means that the network is less secure. Recall that we use a constant R to represent the base security level where all of the machines are clean. In this network, $R = 100$. To represent the reward of keeping the network secure, we subtract the total impact score of compromised machines in the network from R. Therefore, a higher value of r represents that the network is more secure and vice versa.

5.2 Simulation Setup

Based on the evaluation setup, we simulate the interactions among the attacker, network, and defender in Python. All simulations are conducted on an Intel Core i7 machine with 16 GB of memory running OS X 10.15.5.

5.3 Results

Let $\mathbf{N}(s, a, t) = (N_{s_1}(s, a, t), \ldots, N_{s_n}(s, a, t))$ be the observed counts of reaching all successor states from (s, a) between t and $t + 1$.

To evaluate the performance of our algorithms, we introduce a baseline policy. The baseline policy uniformly chooses one action at each timestep and is referred to as the uniform selection policy. We also compare the performance of our algorithms with two policies: the optimal policy and the "solely history" policy. The optimal policy is the solution of (PA) if the defender fully knows the system state, utility function, and transition probabilities. The solely history policy is the same as Algorithm 2 except for the transition probability estimation part. The solely history policy estimates the transition probabilities based solely on the observed transitions in history. In other words, up to timestep t, the defender observes that the aggregate count of reaching successor state s_i from the state-action pair (s, a) is $\sum_{\tau=1}^{t} N_{s_i}(s, a, \tau)$, then the transition probability is estimated as $P(s_{t+1} = s_i | s, a) = \frac{\sum_{\tau=1}^{t} N_{s_i}(s, a, \tau)}{\sum_{s' \in \mathcal{S}} \sum_{\tau=1}^{t} N_{s'}(s, a, \tau)}$. In general, there is no restriction on the action space, i.e., $\mathcal{A}_s = \mathcal{A}$ for all s. However, in this particular ACD problem, the defender has no incentive to reimage clean machines. Thus, we restrict the available actions to those that only

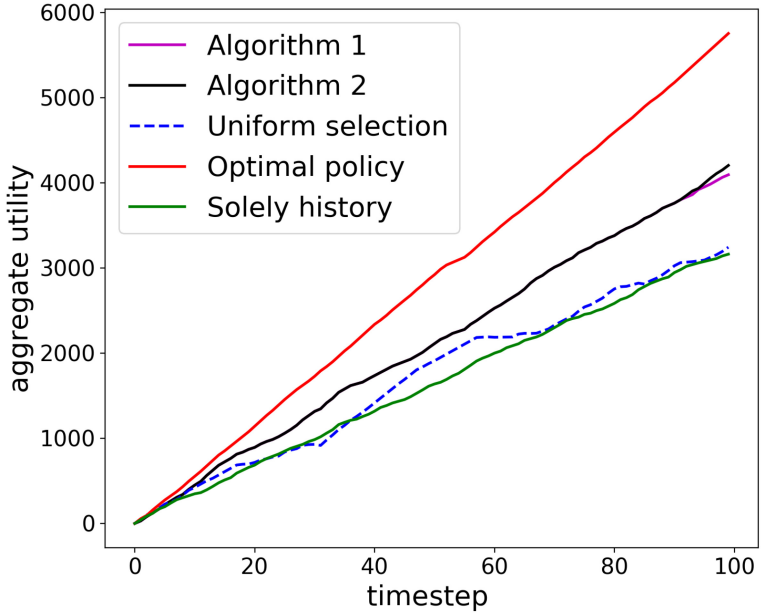

Fig. 5. Aggregate utilities of short-duration simulations.

Table 3. Time Consumption

Algorithm	Time Consumption per Step (s)
Algorithm 1 (Value iteration)	563.98292
Algorithm 2 (Q-learning)	5.55842
The uniform selection policy	0.00067

reimage the compromised machines based on the estimated state. In particular, for an estimated state $s = (s^1, \ldots, s^K)$, $\mathcal{A}_s \triangleq \mathcal{A} \setminus \{a \in \mathcal{A} | \exists i \in a^r s.t. s^i = 0\}$.

5.3.1 Preliminary Results. We first present preliminary results of short-duration simulation. For instance, the duration of the simulation (from the time the attack begins until the time the attack ends) is 100 timesteps and is partitioned into 10 equally long episodes. Figure 5 shows that Algorithm 1 and Algorithm 2 have similar aggregate utilities. As mentioned in Section 4.6, the computational complexity of Algorithm 1 could be very high when the state space is large. To experimentally show the computational complexity of different algorithms, we compare the time consumptions of Algorithm 1, Algorithm 2, and the uniform selection policy. The comparison results shown in Table 3 validate that Algorithm 2 can reduce the computational complexity significantly. Notice that Algorithm 2 requires much more time to execute than the uniform selection because it still needs to update the estimated Q-function $\hat{Q}_m^{n+1}$.

5.3.2 Cost-Effectiveness. We evaluate the cost-effectiveness of our algorithms with long-duration simulation. In what follows, the duration is 10,000 timesteps and partitioned into 100 equally long episodes. Since Algorithm 1 and Algorithm 2 have similar cost-effectiveness but Algorithm 2 is much more computationally efficient than Algorithm 1, we will not consider Algorithm 1 for the rest of the evaluation. Figure 6 compares the aggregate utilities of Algorithm 2,

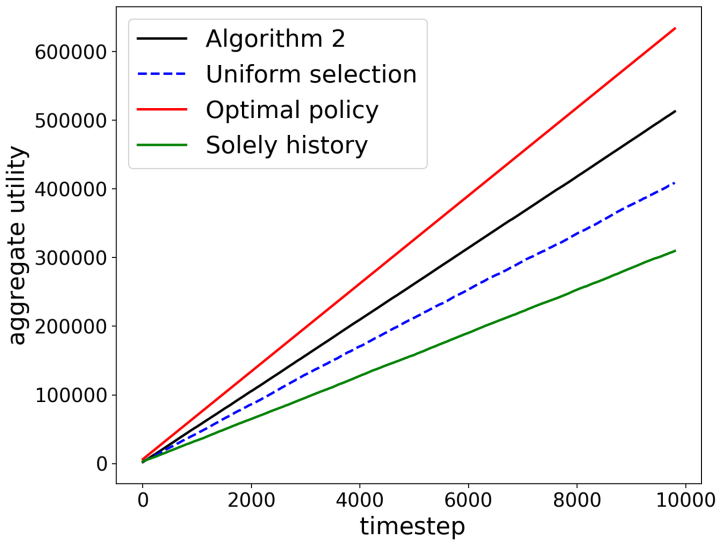

Fig. 6. Aggregate utilities of long-duration simulation.

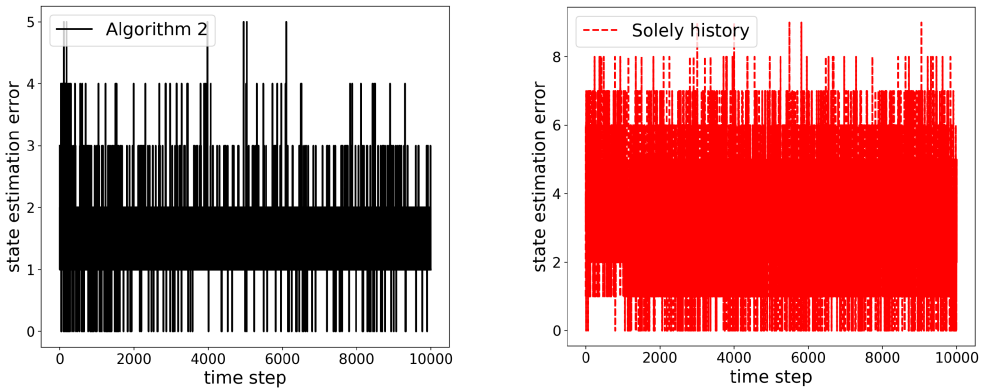

Fig. 7. State estimation errors.

the uniform selection, the solely history policy, and the optimal policy. As expected, Algorithm 2 cannot compete with the optimal policy, which has all of the information needed. However, Algorithm 2 outperforms the uniform policy and the solely history policy. Further, the aggregate utility of Algorithm 2 is well approximated by a linear function with a slope around 50, whereas the counterpart of the solely history policy is around 30 and the counterpart of the uniform policy is around 40. Thus, the lead of Algorithm 2 increases over time. The results indicate that the lack of sufficient exploration in the solely history policy may lead to the convergence of the estimated parameters to incorrect values. We will validate this conjecture in the next part.

5.3.3 Estimation Errors. Next we validate the estimation performance of Algorithm 2.

State estimation errors. As mentioned in Section 4.2, the defender estimates the system state based on its belief over all states. We use Manhattan distance to define the estimation errors, i.e., $\mathbb{D}(s, \tilde{s}) \triangleq (\sum_{i=1}^{K} |s^i - \tilde{s}^i|x)$. Under our evaluation setup, the maximum state estimation error is 10. Figure 7 shows that the state estimation errors of Algorithm 2 oscillate at the beginning and reduce

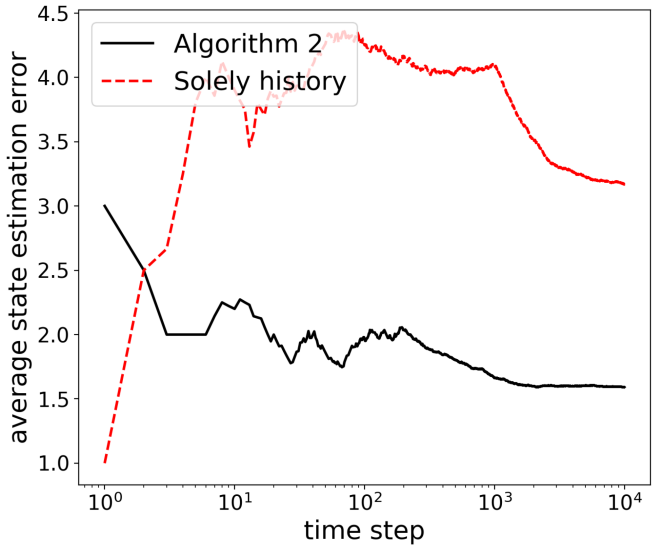

Fig. 8. Average state estimation errors.

to a relatively low level as time goes by. The possible reason for the phenomenon could be that the algorithms randomly collect information at the early stage (exploration) and then lean to the best estimates in history (exploitation). As a comparison, the solely history policy keeps oscillating at a relatively high level. In addition, we evaluate the average error over time to focus on the central tendency of errors in history (shown in Figure 8), i.e., $\frac{1}{t}\sum_{\tau=0}^{t}\mathbb{D}\left(s_\tau, \tilde{s}_\tau\right)$. The average distance of Algorithm 2 oscillates at the beginning and then reaches the steady state around 1.5, whereas that of the solely history policy keeps oscillating and is above 3 for most of the time.

Transition probability estimation errors. We also quantify the estimation error of the sampled transition probability P_{k_m} at each episode k_m by using the distance between P_{k_m} and P. In particular, $P_{k_m} = [\mathbf{p}^{k_m}_{s_1,a_1}, \ldots, \mathbf{p}^{k_m}_{s_{|\mathcal{S}|},a_{|\mathcal{A}|}}]^T$ contains $|\mathcal{S}\times\mathcal{A}|$ row vectors. Similarly, the actual transition probability P contains $|\mathcal{S}\times\mathcal{A}|$ row vectors. In the evaluation, we compute the Manhattan distance between each pair of row vectors (indexed by (s,a)) that has been visited in the history and use the average of the vector-wise distances to quantify the estimation error of P_{k_m} as follows:

$$\mathbb{D}(P_{k_m}, P) \triangleq \frac{\sum_{s,a}(\sum_{i=1}^{|\mathcal{S}|}|P_{k_m}(s_i|s,a) - P(s_i|s,a)|)\mathbf{1}_{\{\mathcal{T}_t(s,a)>0\}}}{\sum_{s,a}\mathbf{1}_{\{\mathcal{T}_t(s,a)>0\}}},$$

where $\mathcal{T}_t(s,a)$ is the number of visits of state-action pair (s,a) until t, and $\mathbf{1}_{\{condition\}} = 1$ when *condition* is true and $\mathbf{1}_{\{condition\}} = 0$ otherwise. Under our evaluation setup, the maximum distance between a pair of row vectors in P_{k_m} and P is 2, because $\sum_{i=1}^{|\mathcal{S}|}|P_{k_m}(s_i|s,a) - P(s_i|s,a)| \leq \sum_{i=1}^{|\mathcal{S}|}(|P_{k_m}(s_i|s,a)| + |P(s_i|s,a)|)$ with $\sum_{i=1}^{|\mathcal{S}|}|P_{k_m}(s_i|s,a)| = 1$ and $\sum_{i=1}^{|\mathcal{S}|}|P(s_i|s,a)| = 1$. Figure 9 shows that the estimation error of Algorithm 2 exponentially decreases and reduces by around 9.52% after 100 episodes, i.e., 10,000 timesteps, whereas that of the solely history policy decreases at the beginning and then keeps increasing as simulation goes on. This comparison result validates that Thompson sampling can balance exploitation and exploration to estimate the transition probabilities and avoid converging to incorrect parameters by solely exploiting the observed data in history.

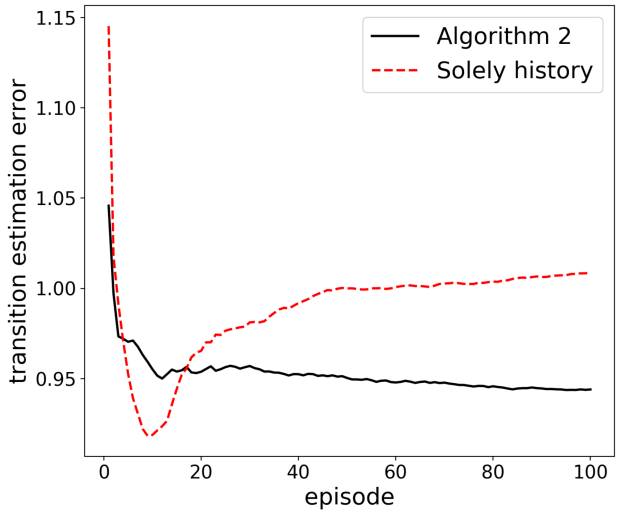

Fig. 9. Transition probability estimation errors.

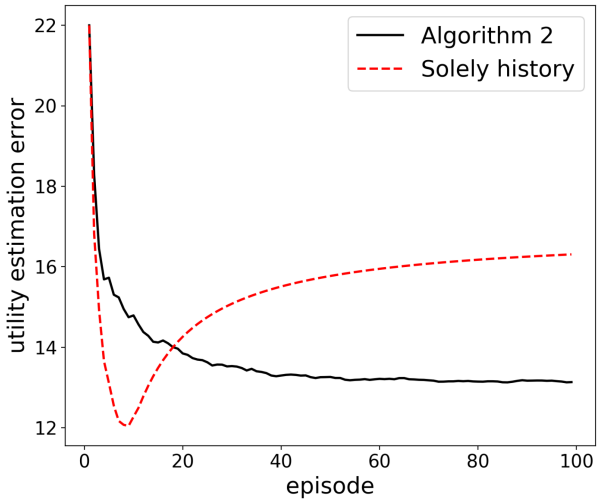

Fig. 10. Utility estimation errors.

Utility estimation errors. As mentioned in Section 4.3, we use the latest received utility values $\hat{U}(s,a)$ to estimate the values of $u(s,a)$ for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$. Here we quantify the estimation error of the utility values. In particular, we only consider the elements of visited state-action pairs in $\hat{U}$ and u. Here we use the average of the element-wise distance to quantify the estimation error as follows:

$$\mathbb{D}(\hat{U}, u, t) \triangleq \frac{\sum_{s,a} |\hat{U}(s,a) - u(s,a)| \mathbf{1}_{\{\mathcal{T}_t(s,a)>0\}}}{\sum_{s,a} \mathbf{1}_{\{\mathcal{T}_t(s,a)>0\}}}.$$

Figure 10 shows that the steady-state estimation error of Algorithm 2 is around half of the peak estimation error, whereas the estimation error of the solely history policy decreases at the beginning and then keeps increasing as simulation goes on.

6 DISCUSSION AND LIMITATIONS

In this work, we propose a Thompson sampling–based reinforcement learning algorithm to balance exploitation and exploration. The exploitation requires that the transition probabilities are generated at the beginning of the attack and fixed all the time. But in a real network, how likely an exploit can succeed might change in dynamic environments. For example, dynamic network traffic flows might change the attacker's AC. One of our future works is to estimate dynamic transition probabilities.

We test our algorithm in a 10-machine network. Although the 10-machine network does not seem very large, its state-action space is huge. In fact, the simulations of the 10-machine network have reached the memory limit of our computer. In addition, curse-of-dimensionality and curse-of-history are well-known challenges for POMDP. For larger networks, one can use approximate solutions [30, 39, 41] or compression techniques [32, 33, 46]. For example, Spaan and Vlassis [41] perform the value iteration only on a finite and incrementally expanded subset of belief states instead of the whole belief state space. Virin et al. [46] compress the state space by clustering the system states based on their optimal Q-values, i.e., states with similar Q-values are grouped together. However, such solutions only work for the scenarios where transition probabilities and reward functions are known. It is highly non-trivial to extend the aforementioned schemes to deal with unknown transition probabilities and reward functions. We leave this as a future work.

7 CONCLUSION

This article proposes Thompson sampling–based reinforcement learning algorithms to effectively defend against a class of multi-stage attacks during vulnerability windows with partial observability, unknown transition probabilities, and unknown utility function. The cost-effectiveness of the algorithms is verified in numerical simulations based on real-world attacks on a computer network.

REFERENCES

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.

[2] Karl J. Åström. 1965. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications* 10, 1 (1965), 174–205.

[3] A. Becker, P. Kumar, and Ching Zong Wei. 1985. Adaptive control with the stochastic approximation algorithm: Geometry and convergence. *IEEE Transactions on Automatic Control* 30, 4 (1985), 330–338.

[4] Richard E. Bellman and Stuart E. Dreyfus. 1962. *Applied Dynamic Programming*. Princeton University Press.

[5] David Bigelow, Thomas Hobson, Robert Rudd, William Streilein, and Hamed Okhravi. 2015. Timely rerandomization for mitigating memory disclosures. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. 268–279.

[6] P. Chen, Z. Hu, J. Xu, M. Zhu, and P. Liu. 2018. Feedback control can make data structure layout randomization more cost-effective under zero-day attacks. *Cybersecurity* 1 (2018), 3.

[7] Andrew Clark, Kun Sun, Linda Bushnell, and Radha Poovendran. 2015. A game-theoretic approach to IP address randomization in decoy-based cyber defense. In *Proceedings of the 6th International Conference on Decision and Game Theory for Security (GameSec'15)*. 3–21.

[8] George Cybenko, Sushil Jajodia, Michael P. Wellman, and Peng Liu. 2014. Adversarial and uncertain reasoning for adaptive cyber defense: Building the scientific foundation. In *Proceedings of the International Conference on Information Systems Security (ICISS'14)*. 1–8.

[9] F. Dai, Y. Hu, K. Zheng, and B. Wu. 2015. Exploring risk flow attack graph for security risk assessment. *IET Information Security* 9, 6 (2015), 344–353.

[10] Nir Friedman and Yoram Singer. 1999. Efficient Bayesian parameter estimation in large discrete domains. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II (NIPS'98)*. 417–423.

[11] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. 2008. Measuring network security using dynamic Bayesian network. In *Proceedings of the 4th ACM Workshop on Quality of Protection (QoP'08)*. 23–30.

[12] Brian Gorenc and Fritz Sands. 2017. *Hacker Machine Interface: The State of SCADA HMI Vulnerabilities.* Technical Report. Trend Micro Zero Day Initiative Team.

[13] Z. Hu, M. Zhu, P. Chen, and P. Liu. 2019. On convergence rates of game theoretic reinforcement learning algorithms. *Automatica* 104, 6 (2019), 90–101.

[14] Zhisheng Hu, Minghui Zhu, and Peng Liu. 2017. Online algorithms for adaptive cyber defense on Bayesian attack graphs. In *Proceedings of the 4th ACM Workshop on Moving Target Defense in Association with the 2017 ACM Conference on Computer and Communications Security (MTD'17).* 99–109.

[15] Jeff Hughes, Lawrence Carin, and George Cybenko. 2008. Cybersecurity strategies: The QuERIES methodology. *Computer* 41, 8 (2008), 20–26.

[16] S. Iannucci, Q. Chen, and S. Abdelwahed. 2016. High-performance intrusion response planning on many-core architectures. In *Proceedings of the 2016 25th International Conference on Computer Communication and Networks (ICCCN'16).* 1–6.

[17] Håvard Johansen, Dag Johansen, and Robbert van Renesse. 2007. *FirePatch: Secure and Time-Critical Dissemination of Software Patches.* Springer US, 373–384.

[18] Per Larsen, Andrei Homescu, Stefan Brunthaler, and Michael Franz. 2014. SoK: Automated software diversity. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP'14).*

[19] R. Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, and R. Cunningham. 2006. Validating and restoring defense in depth using attack graphs. In *Proceedings of the 2006 IEEE Military Communications Conference (MILCOM'06).* 1–10.

[20] Yu Liu and Hong Man. 2005. Network vulnerability assessment using Bayesian networks. In *Proceedings of the 2005 Conference on Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security.* 61–71.

[21] Erik Miehling, Mohammad Rasouli, and Demosthenis Teneketzis. 2015. Optimal defense policies for partially observable spreading processes on Bayesian attack graphs. In *Proceedings of the 2nd ACM Workshop on Moving Target Defense (MTD'15).* 67–76.

[22] E. Miehling, M. Rasouli, and D. Teneketzis. 2018. A POMDP approach to the dynamic defense of large-scale cyber networks. *IEEE Transactions on Information Forensics and Security* 13, 10 (2018), 2490–2505.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[24] Savita Mohurle and Manisha Patil. 2017. A brief study of wannacry threat: Ransomware attack 2017. *International Journal of Advanced Research in Computer Science* 8, 5 (2017), 1938–1940.

[25] Thanh H. Nguyen, Mason Wright, Michael P. Wellman, and Satinder Baveja. 2017. Multi-stage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. In *Proceedings of the 2017 Workshop on Moving Target Defense (MTD'17).* 87–97.

[26] S. Ossenbuhl, J. Steinberger, and H. Baier. 2015. Towards automated incident handling: How to select an appropriate response against a network-based attack? In *Proceedings of the 2015 9th International Conference on IT Security Incident Management IT Forensics (IMF'15).* 51–67.

[27] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. 2006. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06).* 336–345.

[28] Yi Ouyang, Mukul Gagrani, Ashutosh Nayyar, and Rahul Jain. 2017. Learning unknown Markov decision processes: A Thompson sampling approach. In *Advances in Neural Information Processing Systems 30 (NIPS'17).* 1333–1342.

[29] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman. 2018. SoK: Security and privacy in machine learning. In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroSP'18).* 399–414.

[30] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03).* 1025–1030.

[31] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. 2012. Dynamic security risk management using Bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (2012), 61–74.

[32] Pascal Poupart and Craig Boutilier. 2003. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems (NIPS'02).* 1579–1586.

[33] Pascal Poupart and Craig Boutilier. 2004. VDCBPI: An approximate scalable algorithm for large POMDPs. In *Advances in Neural Information Processing Systems (NIPS'04).* 1081–1088.

[34] Tom Roeder and Fred B. Schneider. 2010. Proactive obfuscation. *ACM Transactions on Computer Systems* 28, 2 (2010), Article 4, 54 pages.

[35] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, and Ian Osband. 2017. A tutorial on Thompson sampling. arXiv:1707.02038

[36] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. 2012. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12).* 1816–1824.

[37] Mike Schiffman. 2017. Common Vulnerability Scoring System v3.0: Specification Document. Retrieved September 19, 2020 from https://www.first.org/cvss/v3.0/specification-document.
[38] Guy Shani. 2007. *Learning and Solving Partially Observable Markov Decision Processes*. Ben Gurion University.
[39] Guy Shani, Joelle Pineau, and Robert Kaplow. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27, 1 (2013), 1–51.
[40] Edward J. Sondik. 1978. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research* 26, 2 (1978), 282–304.
[41] Matthijs T. J. Spaan and Nikos Vlassis. 2005. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24, 1 (2005), 195–220.
[42] Malcolm J. A. Strens. 2000. A Bayesian framework for reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML'00)*. 943–950.
[43] Symantec. 2015. Internet Security Threat Report. Retrieved September 19, 2020 from https://library.cyentia.com/report/report_002191.html.
[44] W. R. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 2 (1933), 285–294.
[45] Michel Tokic. 2010. Adaptive ϵ-greedy exploration in reinforcement learning based on value differences. In *KI 2010: Advances in Artificial Intelligence*. Lecture Notes in Computer Science, Vol. 6359. Springer, 203–210.
[46] Yan Virin, Guy Shani, Solomon Eyal Shimony, and Ronen I. Brafman. 2007. Scaling up: Solving POMDPs through value based clustering. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'07)*, Vol. 22. 1290–1295.
[47] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (1992), 279–292.
[48] Peng Xie, J. H. Li, Xinming Ou, Peng Liu, and R. Levy. 2010. Using Bayesian networks for cyber security analysis. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN'10)*. 211–220.
[49] Zhi Xin, Huiyu Chen, Hao Han, Bing Mao, and Li Xie. 2010. Misleading malware similarities analysis by automatic data structure obfuscation. In *Proceedings of the 13th International Conference on Information Security (ISC'10)*.
[50] Lu Yu and Richard R. Brooks. 2013. Applying POMDP to moving target optimization. In *Proceedings of the 8th Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW'13)*. Article 49, 4 pages.
[51] Emmanuele Zambon and Damiano Bolzoni. 2006. Network Intrusion Detection Systems: False Positive Reduction Through Anomaly Detection. Retrieved September 19, 2020 from http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zambon.pdf.
[52] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. 2010. A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security* 29, 1 (2010), 124–140.
[53] Minghui Zhu, Zhisheng Hu, and Peng Liu. 2014. Reinforcement learning algorithms for adaptive cyber defense against Heartbleed. In *Proceedings of the 1st ACM Workshop on Moving Target Defense (MTD'14)*. 51–58.
[54] Minghui Zhu and Sonia Martínez. 2014. On attack-resilient distributed formation control in operator-vehicle networks. *SIAM Journal on Control and Optimization* 52, 5 (2014), 3176–3202.
[55] Quanyan Zhu and Tamer Başar. 2009. Dynamic policy-based IDS configuration. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC'09) Held Jointly with the 2009 28th Chinese Control Conference*. 8600–8605.
[56] Quanyan Zhu, Hamidou Tembine, and Tamer Başar. 2013. Hybrid learning in stochastic games and its applications in network security. *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control* 17, 14 (2013), 305–329.
[57] Cliff Changchun Zou, Weibo Gong, and Don Towsley. 2002. Code red worm propagation modeling and analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*. 138–147.

Received October 2019; revised June 2020; accepted August 2020