Online Algorithms for Adaptive Cyber Defense on Bayesian Attack Graphs

Zhisheng Hu
Pennsylvania State University
University Park, PA
zxh128@psu.edu

Minghui Zhu
Pennsylvania State University
University Park, PA
muz16@psu.edu

Peng Liu
Pennsylvania State University
University Park, PA
pliu@ist.psu.edu

ABSTRACT

Emerging zero-day vulnerabilities in information and communications technology systems make cyber defenses very challenging. In particular, the defender faces uncertainties of; e.g., system states and the locations and the impacts of vulnerabilities. In this paper, we study the defense problem on a computer network that is modeled as a partially observable Markov decision process on a Bayesian attack graph. We propose online algorithms which allow the defender to identify effective defense policies when utility functions are unknown a priori. The algorithm performance is verified via numerical simulations based on real-world attacks.

1 INTRODUCTION

Zero-day vulnerabilities are extremely dangerous for information and communications technology (ICT) systems; e.g., computer networks, because they take advantage of previously unknown vulnerabilities for which no solution is currently available. A zero-day attack happens once that a software/hardware zero-day vulnerability is exploited by attackers. Traditional defenses against zero-day attacks are mainly governed by slow and deliberative processes such as testing and patching. According to [28], the top five zero-day vulnerabilities in 2014 were actively exploited by attackers for average 59 days before patches were available. During the life time of a vulnerability, severe damages can be made to ICT systems. For example, the paper [10] shows that up to 55% of websites in the Alexa Top 1 Million were initially vulnerable to Heartbleed, including GitHub, Stack Overflow, Imgur and 3% of them remained vulnerable two months after the disclosure of Heartbleed vulnerability. Moving target defense (MTD) is deemed promising to defend against zero-day attacks. Through MTD, the defender dynamically and proactively reconfigures deployed defenses so as to increase uncertainty and complexity for attackers. Existing MTD methods [31] include adaptive address space layout randomization [5], adaptive data structure layout randomization [7], dynamic platforms [20] and software diversity [16]. The questions for where, when, and how to optimally deploy heterogeneous MTD methods have arisen. Recently, adaptive cyber defense (ACD) [9] has been proposed to address the questions through integrating heuristics, machine learning, behavioral science, control theory and game theory.

In this paper, we consider the ACD problem in a computer network where an intelligent attacker exploits combinations of multiple known or zero-day vulnerabilities to compromise machines in the network. The defender faces two challenges: (1) partial observability: the defender can only observe the states of a subset of the machines due to limited detection capabilities; (2) unknown utility functions: the defender can only receive some utility values; i.e., feedbacks, after some defense actions are taken, but does not know the utility functions; i.e., the mapping from actions and environment to the feedbacks because the locations and the impacts of zero-day vulnerabilities are not available. In fact, these two challenges are ubiquitous when we design ACD schemes against zero-day attacks. For example, the system states in [7] are the security-sensitive data structures and difficult to observe because it requires huge time and human effort to find out the security-sensitive ones among all the data structures. And the segmentation faults in [11] are determined by the defender's actions; i.e., guard page locations, the attacker's actions; i.e., over-read memory, and the environment; i.e., the allocated buffers on the heap. The defender can only access the induced utility values but does not know the utility functions which involve the allocated buffers and the attacker's actions.

Partially observable Markov decision process (POMDP) [1] has been proposed to address the first challenge. A simple version of the ACD problem in the computer network has been studied in [18]. This paper uses *Bayesian attack graphs* (BAGs) [17, 21] to depict how an intelligent attacker can exploit combinations of vulnerabilities to penetrate the network and models the ACD problem on a BAG as a POMDP problem. POMDP models the decision making process where the decision-maker can interact with the environment, receive observations which partially reflect system states. By reasoning about the effects of actions and observations on the environment, the decision-maker can estimate the system state using a belief; i.e., a probability distribution over the set of possible states. And then the decision-maker can select actions on basis of the beliefs instead of system states which are not available. In fact, POMDP has been applied to a variety of real-world sequential decision-making problems, including machine maintenance, navigation problems and other applications. More details could be found in the survey paper [19]. POMDP is also suitable for security problems [13, 23, 32] where the decision-makers (defender or attacker) can only partially observe the ICT systems. The papers [13] and [23] use POMDP to model the attacks where the attacker can only observe some information of the targets; e.g. the version of operator systems. And the paper [32] uses POMDP to optimally implement diversity to balance security performance and diversity

MTD'17, October 30, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5176-8/17/10…$15.00
https://doi.org/10.1145/3140549.3140556

implementation costs where the defender can only observe whether the system is compromised or not. However, the above papers only deal with the first challenge of partial observability but assume that the defender completely knows the utility functions. As a result, the optimal defense or attack policies can be synthesized offline by resorting to; e.g., dynamic programming. This assumption may not be valid when the defender faces zero-day vulnerabilities and dynamic network environments.

Online learning or reinforcement learning has emerged as a systematic methodology to address the second challenge. In online learning, a decision-maker applies actions to a partially unknown environment and learns from the consequences of its actions on the fly, rather than from being explicitly taught offline. It selects actions on basis of its past experience (exploitation) and also by new choices (exploration). Essentially, online learning is trial-and-error learning. The decision-maker receives numerical utility values, which evaluate deployed actions, and seeks to learn how to select actions that maximize accumulated utility values over time. The decision-maker only needs to access induced utility values but is not required to know utility functions. So online learning is particularly well suited to problems where the decision-maker interacts with a partially unknown environment but can evaluate their actions via certain utility values. This intriguing feature well matches the needs of ACD to defeat zero-day attacks. For example, when defending Heartbleed attacks [8], the defender cannot predict which part of the memory is going to be over-read by the attacker. On the other hand, the defender can add guard pages, detect segmentation faults and use them to evaluate the cost-effectiveness of previously deployed defenses. The cost-effectiveness can serve as a quantitative metric to guide the defender to learn from past defenses, and adaptively choose better locations for guard pages in the near future. Online learning has been increasingly applied to address security problems. The paper [37] applies learning algorithms to solve internal denial-of-service attacks. In [36], a Q-learning algorithm is proposed to solve dynamic configuration problems of intrusion detection systems. The paper [34] proposes a reinforcement learning scheme to defend against Heartbleed attacks. The paper [12] applies the developed robust adaptive learning algorithm to dynamically configure platforms to defend against zero-day attacks. And the paper [35] proposes two novel learning-based distributed control algorithms to handle false data injection attacks in operator-vehicle networks.

Contribution. In this paper, we propose online algorithms to solve POMDP problems with unknown utility functions. In particular, given previous actions, induced utility values and observations, the defender forms a belief about the current state of the system and takes the empirical average of utility values to estimate of the utility functions. Then the agent performs value iterations with the estimates of the utility functions to approximate the optimal value function of each belief state at each time step. A limitation of this algorithm is high computational complexity. In our example, each update of estimates of the utility functions take hours when the action space has 2,240 actions and observation space has 280 observations. To mitigate the computational complexity, we introduce a Q-learning based algorithm which updates one Q value per iteration. The update time reduces to seconds with the same sizes of action space and observation space. Further, inspired by the no-false-positive feature of the observations in our ACD problem, we customize the algorithm such that the defender only reimages the machines which are known to be compromised. Finally, we conduct numerical simulations based on real-world attacks to evaluate the performance of the algorithms. The simulation results confirm that our algorithms enable the defender to identify effective defense policies when utility functions are unknown. In particular, the algorithms outperform the baseline policy, which uniformly selects defense actions, in terms of aggregate utilities and increase their leads over time.

2 RUNNING EXAMPLE: ADAPTIVE CYBER DEFENSE ON BAYESIAN ATTACK GRAPHS

In this section, we consider an ACD problem where the defender aims to defend a computer network against external intrusions launched by an attacker in real-time. The attacker exploits combinations of multiple vulnerabilities to compromise machines in the network. In particular, we first use BAGs to model the interactions between the network and attacker when the defense is absent; i.e., how the attacker attacks the network. After that, we further introduce defense and model the interactions among the attacker, network and defender.

2.1 BAGs without defense

We consider a computer network which consists of multiple machines and each machine has a vulnerability. The attacker aims to take over the network by exploiting reachable vulnerabilities. But each exploit can only succeed with a certain probability. As first introduced in [17], the interactions between the network and attacker can be modeled by a BAG, which is formally defined as follows:

Definition 1. *A BAG is defined as a tuple* $G = (\mathcal{N}, \mathcal{E}, \mathcal{P})$. $\mathcal{N} = \{1, ..., K\}$ *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 machine* i *can be compromised through machine* j. $\mathcal{P}$ *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/she compromises machine* i.

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. And the probability of leaf node $l \in \mathcal{N}_L$ is compromised is denoted by ρ_l. For non-leaf node i, we formally define their in-neighbors as $\bar{D}_i \triangleq \{j \in \mathcal{N} | (j, i) \in \mathcal{E}\}$. NIST's Common Vulnerability Scoring System (CVSS) [24] provides a way to capture the principal characteristics of a vulnerability and produces a numerical score reflecting its severity. Here we use the exploitability metrics in CVSS to calculate the value of each ρ_{ij} and ρ_l. The details of the calculation will be introduced in Section 5.

System state. The state of machine $i \in \mathcal{N}$ at time t is either compromised (value 1) by the attacker or not (value 0); i.e., $s_t^i \in \{0, 1\}, \forall i \in \mathcal{N}$. Combing the states of the machines, we define the system state at time step t as $s_t = (s_t^1, \cdots, s_t^K) \in \mathcal{S} = \{0, 1\}^K$. Figure 1 shows

an example of BAGs. In Figure 1, the state of BAG is $s = (1, 0, 0, 0)$ where only machine 1 is compromised by the attacker.

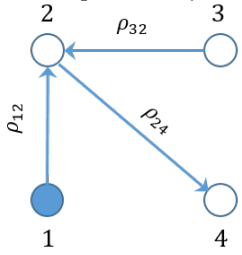

Figure 1: An example of BAGs.

Attacker's action. Initially, the attacker chooses a subset of leaf nodes to compromise. Once succeeds, the attacker can use compromised machines as stepping stones to exploit their out-neighbors. And if a machine was compromised at the previous step, it remains compromised for the current step. The attacker stops when all machines are compromised.

State transition. Since defense is not taken into account, state transitions are autonomous given attacker's initial actions. In real world, machine $i \in \mathcal{N} \setminus \mathcal{N}_L$ can be compromised at t under one of two conditions: 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 condition depends on the type of vulnerability in 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) which can access this server. The Admin server with stack overflow on MS SMV service (And-machine) can be successfully exploited when both the compromised local user (one in-neighbor) and the SQL server (another in-neighbor) [21]. The state evolution can be modeled by conditional probability $Pr(s_t|s_{t-1})$. In particular, if i is an And-machine:

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

If i is an Or-machine:

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

2.2 BAGs with defense

In this section, we consider BAGs with defense and the interactions among the attacker, network and defender are modeled as a Markov decision process (MDP). The specifics are discussed as follows.

Defender's actions. The defender's actions include detection and reimage. Formally, at time step t, the defender's action is $a_t = (a_t^r, a_t^d) \in \mathcal{A} \subset \mathcal{P}(\mathcal{N}) \times \mathcal{P}(\mathcal{N})$, 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. Intrusion Detection System (IDS) is a typical practice to implement detection. However, IDS alerts contain too many false positives [33]. In contrast, detection in this paper is implemented by labor analysis on selected machines. Hence, our detection does not include any false positive. On the other hand, the defender can only detect a subset of the machines in the network due to limited resources.

State transition. State transitions consist of a Markovian process and can be modeled by conditional probability $P(s_t|s_{t-1}, a_{t-1})$. $P(s_t^j = 1|s_{t-1}, a_{t-1})$ represents the probability that machine j 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 2.1, for an And-machine which was not compromised nor reimaged at the previous step, it can only be compromised if all of its in-neighbors are compromised. And for an Or-machine which 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 that this machine becoming 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:

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

And if $i \in \mathcal{N}$ is an Or-machine:

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

Observation. Due to limited resources, the defender can only monitor a subset of machines at each time. So the defender is only aware of partial system states, which are referred to as observations. In particular, the defender receives an observation at time step 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 paper, we simply use the states of the machines in a_{t-1}^d as observation; i.e., $o_t = (s_t^{i_1}, \cdots, s_t^{i_k})$, where $i_1, \cdots, i_k \in a_{t-1}^d$. The observation kernel $\mathcal{Z}(\cdot|s_t, a_{t-1})$ is given by:

$$\mathcal{Z}(o|s_t, a_{t-1}) = \begin{cases} 1 & \text{if } o = (s_t^{i_1}, \cdots, s_t^{i_k}) \text{ and } i_1, \cdots, i_k \in a_{t-1}^d, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

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 time step t, after taking action a_t in state s_t, the defender receives a utility value $u_t(s_t, a_t) \triangleq r_t(s_t, a_t) - c_t(a_t)$, where $r_t(s_t, a_t)$ is the reward of keeping the network secure and $c_t(a_t)$ is the cost induced by the action a_t; e.g., resources required by reimage or detection. In particular, $r_t(s_t, a_t)$ is defined according to confidentiality, integrity, and availability (CIA) triad. The utility functions are time dependent because of the existence of zero-day vulnerabilities and the dynamic environments; e.g., machine workloads, network traffic flows. Under different environments, even same pair of state-action could lead to different utilities. And the utility functions are unknown to the defender because he/she is unaware of the dynamic environment and the impact of the zero-day vulnerabilities. The details of the utility function will be discussed in Section 5.

Defender's goal. The defender knows the transition probability, observation kernel, previous actions and observations up to t. But the system state trajectory and the utility functions are unknown to the defender. We formally define the information available to the defender at time t as $I_t \triangleq (\mathcal{Z}, P, o_1, a_1, \cdots, o_{t-1}, a_{t-1}, o_t)$. The defender aims to find a defense policy over the time horizon $T = \{0, 1, \cdots, N\}$ to maximize the aggregate utility $\sum_{t \in T} u_t(s_t, a_t)$.

3 PROBLEM FORMULATION OF POMDP

In last section, we introduce an ACD problem on BAGs. Essentially, the problem is an instance of POMDP problems. In this section, we present the problem formulation of generic POMDP problems. In Section 4, we provide online algorithms to solve them.

3.1 Background of MDP

An MDP is a sequential decision making problem [4, 22] 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. The formal definition of MDP is given as follows.

Definition 2. *A (non-stationary) MDP consists of a tuple* $(T, \gamma, \mathcal{S}, \mathcal{A}, \mathcal{A}_t, P_t, u_t)$*:* $T \triangleq \{1, 2, \cdots, N\}$ *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* $\mathcal{A}_t \subseteq A$ *is the available action set at time step* t*. As a result of choosing action* a *at state* s *at time step* t*, the agent receives a utility* $u_t(s, a)$ *and the state at next time step is drawn from a transition probability* $P_t(\cdot|s, a)$*.*

A *decision rule* prescribes a procedure for action selection in each state at a specified time step. We will focus on deterministic decision rules because it is easy to implement and evaluate. A measurable mapping $d_t : \mathcal{S} \to \mathcal{A}_t$ is called a decision rule which specifies the action when the system is in state s_t at time step t. A sequence of decision rules $\pi = (d_1, d_2, \cdots, d_{N-1})$ is called a policy or strategy. Let D_t denote the set of decision rules at time step t and let $\Pi = D_1 \times D_2 \times \cdots \times D_{N-1}$ denote the set of all policies.

Let $J_1^\pi(s) \triangleq \mathbb{E}\left[\sum_{t=1}^{N} \gamma^t u_t(s_t, d_t(s_t))\right]$ represent the expected discounted total utility if policy π is used and the system's initial state is s. In the basic MDP problem [22], the agent knows the utility functions, transition probabilities for all time steps, the state trajectory and previous actions. The agent's goal is to maximize the aggregate utility over T. Then the problem is described formally as follows:

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

3.2 POMDP problem formulation

In this paper, we consider a subclass of non-stationary MDP where the agent is not aware of the system state trajectory. Instead, the agent receives an observation each time step. This class is referred to as POMDP.

Definition 3. *A POMDP consists of a tuple* $(T, \gamma, \mathcal{S}, \mathcal{A}, \mathcal{A}_t, \mathcal{O}, P_t, u_t, \mathcal{Z})$*, where* $T, \gamma, \mathcal{S}, \mathcal{A}, \mathcal{A}_t, P_t, u_t$ *are defined in Definition 2,* $\mathcal{O}$ *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*.*

Remark 1. *Note that the agent still receives* $u_t(s_t, a_t)$ *after applying an action* a_t *at state* s_t *at time step* t*. But the agent does not know* s_t*, then he only knows the utility value instead of the function value of pari* (s_t, a_t)*. To highlight that* s_t *is unknown, we denote* v_t *as the utility value received by the agent at time step* t*.*

One might simply take the observation space to be the state space and treat a POMDP as an MDP. 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 (with decision rules that map from $\mathcal{O}$ to $\mathcal{A}_t$) may not perform well. To address the challenge, the agent maintains a probability distribution over states, which is called *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. We denote the belief state update law as SE, 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. The output of the state estimator is written as $SE(b_{t-1}, a_{t-1}, o_t)$. By Bayes' rule, the updated belief state assigns the probability to state s' as follows:

$$\begin{aligned} Pr(s'|a_{t-1}, o_t, b_{t-1}) &= \frac{Pr(o_t|s', a_{t-1}, b_{t-1}) Pr(s'|a_{t-1}, b_{t-1})}{Pr(o_t|a_{t-1}, b_{t-1})} \\ &= \frac{\mathcal{Z}(o_t|s', a_{t-1}) \sum_{s \in \mathcal{S}} P_{t-1}(s'|s, a_{t-1}) b_{t-1}(s)}{\sum_{s' \in \mathcal{S}} \mathcal{Z}(o_t|s', a_{t-1}) \sum_{s \in \mathcal{S}} P_{t-1}(s'|s, a_{t-1}) b_{t-1}(s)}. \end{aligned} \tag{4}$$

The transition probability from belief state b_{t-1} to $b_t \in \mathcal{B}$ is defined as follows:

$$B_t(b_t|b_{t-1}, a_{t-1}) = \sum_{\{o \in \mathcal{O} | SE(b_{t-1}, a_{t-1}, o) = b_t\}} Pr(o|a_{t-1}, b_{t-1}), \tag{5}$$

where

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

If there is no $o \in \mathcal{O}$ such that $SE(b_{t-1}, a_{t-1}, o) = b_t$, then $B_t(b_t|b_{t-1}, a_{t-1}) = 0$. And it is shown that the process of updating the belief state is Markovian [1]; the current belief state depends only on the previous belief state and action.

Given the fact that the state trajectory is unknown and the belief states provide sufficient statistics of the history [27], the agent would select actions on the basis of belief states. Here we slightly abuse the definition of the decision rule and the policy. In POMDP, $d_t : \mathcal{B} \to \mathcal{A}_t$ is the decision rule which specifies the action choice when the belief state is b_t at time step t. Let $\Pi = D_1 \times D_2 \times \cdots \times D_{N-1}$ denote the set of all policies. Therefore, a POMDP can be cast as a completely observable MDP where the state space is the belief state space $\mathcal{B}$ and the action space is $\mathcal{A}$. We formally define this MDP as a belief-state MDP.

Definition 4. *A belief-state MDP consists of a tuple $(T, \gamma, \mathcal{B}, \mathcal{A}, \mathcal{A}_t, B_t, U_t)$: $T, \gamma, \mathcal{A}, \mathcal{A}_t$ are defined in Definition 2; $\mathcal{B}$ is belief state space; $B_t(\cdot|b, a)$ is the belief transition probability at t, $U_t(b, a) = \sum_{s \in \mathcal{S}} b(s) u_t(s, a)$ is the expected immediate utility from executing action a at state s given the belief state b and U_t is referred to as the expected immediate utility function at time step t.*

By introducing the belief-state MDP, we formulate the agent's problem as follows:

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

where $J_1^{\pi}(b)$ is the expected discounted total utility from $t = 1$ to $t = N$ if policy π is used and the belief state at $t = 1$ is b.

3.3 Offline algorithms V.S online algorithms for POMDP

POMDP problem and its solutions [1, 6, 23] have been studied for several decades. In existing solutions, a key assumption is that the utility functions and transition probabilities for all time steps are known to the agent in advance. As a result, optimal controllers can be synthesized offline. This class of solutions of POMDP is referred to as offline algorithms. The ACD problem on BAGs mentioned in Section 2.2 is essentially a POMDP problem, but its utility functions are unknown to the defender. As a result, existing offline algorithms cannot solve the ACD problem and other POMDP problems where utility functions are unknown. In next section, we propose online algorithms which can handle unknown utility functions. Note that the agent needs transition probabilities P_t and observation kernel Z to perform belief update mentioned in Section 3.2. One of future directions is to relax such assumption.

4 ONLINE ALGORITHMS FOR POMDP

In this section, we propose two online algorithms to solve POMDP problems with unknown utility functions. In what follows, the main idea of the algorithm is first introduced. And then the informal statement of the algorithm is presented followed by the discussion. After that, the computational complexity is analyzed and a Q-learning based algorithm is proposed to handle the computational complexity.

4.1 Main idea

As mentioned in Section 3.2, a POMDP can be cast to a belief-state MDP and the goal of the agent is to solve Problem (PB). For this problem, the agent aims to compute the optimal value functions $J_t^*(b) \triangleq \max_{\pi \in \Pi} J_t^{\pi}(b), \forall t \in T$. By the principle of optimality [2], the optimal value of belief state b is equal to the maximum sum achieved by a particular action, where the sum consists of instant utility induced by the action from b and the expected optimal value of next belief state which is discounted by γ. With this property, the optimal value function at t can be calculated backward from the optimal value function at $t + 1$ as follows:

$$J_t^*(b) = \max_{a \in \mathcal{A}_t} [U_t(b, a) + \gamma \sum_{b' \in \mathcal{B}} B_t(b'|b, a) J_{t+1}^*(b')]. \tag{6}$$

And the optimal action $d_t^*(b)$ is the one which achieves the optimal value of $J_t^*(b)$; i.e.,

$$d_t^*(b) = \arg\max_{a \in \mathcal{A}_t} [U_t(b, a) + \gamma \sum_{b' \in \mathcal{B}} B_t(b'|b, a) J_{t+1}^*(b')].$$

With the knowledge of utility functions u_t, transition probabilities P_t for all $t \in T$, and observation kernel $\mathcal{Z}$, $J_t^*(b)$ can be calculated offline based on equation (6) for all $t \in T$ and $b \in \mathcal{B}$.

There are two challenges to perform (6). First, the utility functions are unknown in our problem. Second, the belief space $\mathcal{B}$, where J_t^* is defined, is infinite. To address the first challenge, we use dynamic programming (DP) to get the estimates of the optimal value functions J_t^*. Let us consider the special case where U_t, B_t and $\mathcal{A}_t$ are time independent and the time horizon is infinite. The optimal value functions can be rewritten as $J^*(b) \triangleq \max_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t U_t(b_t, d_t(b_t))\right]$, where $\pi = (d_1, d_2, \cdots)$. 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:

$$J^n(b) = \max_{a \in \mathcal{A}_t} [U_t(b, a) + \gamma \sum_{b' \in \mathcal{B}} B_t(b'|b, a) J^{n-1}(b')]. \tag{7}$$

By applying (7) repeatedly over n, J^n will converge to the fixed point J^* [3]. In this paper, the time horizon could be finite. Besides, the utility functions and the transition probabilities are time dependent. Therefore, given fixed U_t, B_t and $\mathcal{A}_t$, J^n may not converge to J_t^*. Further, to perform the value iteration (7), the expected immediate utility function U_t is needed. However, by Definition 4, without the knowledge of utility function u_t, we cannot directly calculate U_t. To address above two issues, we partition the time horizon T into $M + 1$ relatively long intervals $k_0, k_1, \cdots k_M$, and we use constant empirical average utility values $\hat{U}_m(b, a)$ to represent the values of $U_t(b, a)$ for all belief-action pairs $(b, a) \in \mathcal{B} \times \mathcal{A}_t$ when $t \in k_m$. And we perform the slightly changed value iteration repeatedly in interval k_m as follows:

$$J_m^n(b) = \max_{a \in \mathcal{A}_t} [\hat{U}_m(b, a) + \gamma \sum_{b' \in \mathcal{B}} B_t(b'|b, a) J_m^{n-1}(b')].$$

Then we select an action that maximizes J_m^n at the n-th time step in interval k_m.

To address the challenge of computation tractability, we restrict the recursion on a sequence of incrementally expanded subsets of

Algorithm 1 Point-based DP-FPL

1: Assign initial belief b_0 and arbitrarily choose a_0;
2: $\mathbb{B}_0 = \{b_0\}$;
3: Initialize the number of visits of each belief-action pair: $\mathcal{T}(b,a) = 0, \forall (b,a) \in \mathcal{B} \times \mathcal{A}$;
4: **for** $t \in k_0$ **do**
5: Receive observation o_1;
6: Update belief state $b_t = SE(b_{t-1}, a_{t-1}, o_t)$;
7: Uniformly select an action $a_t \in \mathcal{A}_t$ and execute a_t;
8: Receive utility value $v_t = u_t(s_t, a_t)$;
9: $\mathcal{T}(b_t, a_t) = \mathcal{T}(b_t, a_t) + 1$;
10: **end for**
11: $\mathbb{B}_1 = \mathbb{B}_0 \cup \bigcup_{t \in k_0} \{b_t\}$;
12: **while** $m \geq 1$ **do**
13: $\hat{U}_m(b,a) = \begin{cases} \frac{1}{\mathcal{T}(b,a)} \sum_{t=1}^{t_m - 1} v_t \mathbf{1}_{\{b_t = b \text{ and } a_t = a\}} & \text{if } \mathcal{T}(b,a) > 0 \\ -\infty & \text{if } \mathcal{T}(b,a) = 0 \end{cases}$, for all $(b,a) \in \mathbb{B}_m \times \mathcal{A}_t$;
14: $n = 0$;
15: $J_m^n(b) = \max_{a \in \mathcal{A}_t} \hat{U}_m(b,a)$ for all $b \in \mathbb{B}_m$;
16: **for** $t \in k_m$ **do**
17: Observe o_t;
18: Update belief state $b_t = SE(b_{t-1}, a_{t-1}, o_t)$;
19: **if** $b_t \in \mathbb{B}_m$ **then**
20: $a_t \in \arg\max_{a \in \mathcal{A}_t} [\hat{U}_m(b_t, a) + \gamma \sum_{b' \in \mathbb{B}_m} B_t(b'|b_t, a) J_m^n(b') + \epsilon_t(a)]$;
21: **else**
22: Uniformly select an action $a_t \in \mathcal{A}_t$ and execute a_t;
23: **end if**
24: Receive utility value $v_t = u_t(s_t, a_t)$;
25: $J_m^{n+1}(b) = \max_{a \in \mathcal{A}_t} [\hat{U}_m(b,a) + \gamma \sum_{b' \in \mathbb{B}_m} B_t(b'|b,a) J_m^n(b')]$, for all $b \in \mathbb{B}_m$;
26: $n = n + 1$;
27: $\mathcal{T}(b_t, a_t) = \mathcal{T}(b_t, a_t) + 1$;
28: **end for**
29: $\mathbb{B}_{m+1} = \mathbb{B}_m \cup \bigcup_{t \in k_m} \{b_t\}$;
30: $m = m + 1$;
31: **end while**

belief space $\mathcal{B}$; i.e., $\{\mathbb{B}_0, \cdots, \mathbb{B}_m, \cdots, \mathbb{B}_M\}$. In particular, we start from the initial belief state and add the new belief states which appear over time. In the rest of the paper, the subset of $\mathcal{B}$ is referred to as belief set.

4.2 Algorithm statement

Formally, we define the information set of the agent in this kind of POMDP problems at time t as $I_t \triangleq (\mathcal{Z}, o_1, b_1, a_1, v_1, P_1, \cdots, o_{t-1}, b_{t-1}, a_{t-1}, v_{t-1}, P_{t-1}, o_t, b_t, P_t)$. That is, the agent knows the observation kernel, transition probabilities, belief states, and observations up to t. Additionally, the agent knows the actions and the received utility values up to $t-1$. But the visited states and utility functions are unknown. The pseudo-code of the algorithm is listed in Algorithm 1 and the details are shown as follows.

Let t_m denote the first step of the interval k_m. For the initial interval (Lines 4 to 10), the agent uniformly selects an action for each time step (Line 7), records the belief states that evolve from the last belief (Line 6) and expands the belief set for interval k_1 as $\mathbb{B}_1$ (Line 11). During each interval k_m ($m \geq 1$), $\hat{U}_m$ is updated at the beginning and kept constant afterwards (Line 13), where $\mathbf{1}_{\{condition\}} = 1$ when *condition* is true and $\mathbf{1}_{\{condition\}} = 0$ otherwise. As mentioned before, it is infeasible to get the values of function $\hat{U}_m$ for all pairs of belief-action because $\mathcal{B}$ is infinite. We only consider the values of $\hat{U}_m$ for belief-action pairs $(b,a) \in \mathbb{B}_m \times \mathcal{A}_t$. The initial values of J_m^0 are set for all of belief states $b \in \mathbb{B}_m$ (Line 15). If $b_t \in \mathbb{B}_m$, then a_t is chosen to maximize the sum of J_m^n and $\epsilon_t(a)$ at b_t (Line 20). Otherwise, a_t is uniformly chosen from $\mathcal{A}_t$ (Line 22). The action is selected according to the concept of "following the perturbed leader (FPL)" [14, 15], where a diminishing random perturbation for each action $\epsilon_t(a)$ is added. Then the agent receives utility value v_t (Line 24). And estimated value function J_m^{n+1} is updated (Line 25). At the end of interval k_m, the belief set for next interval $\mathbb{B}_{m+1}$ is expanded by adding all new belief states which appear in interval k_m into the previous belief set $\mathbb{B}_m$ (Line 29).

4.3 Discussion

Because the utility functions are unknown to the agent, it is infeasible to compute the values of $U_t(b,a)$ for all pairs of belief-action. In fact, the agent can only receive the utility value v_t at time step t. One key insight of Algorithm 1 is to use the empirical average utility values $\hat{U}_m(b,a)$ to estimate the expected immediate utility values $U_t(b,a)$ for all belief-action pairs $(b,a) \in \mathbb{B}_m \times \mathcal{A}_t$. The reasons of using empirical averages are as follows. First, the agent can only receive a collection of utility values in this problem. Second, empirical averages are most commonly used mathematical measure of central tendency [30]. By using the empirical average $\hat{U}_m(b,a)$, we can estimate the typical value of $U_t(b,a)$ based on the history. Then $\hat{U}_m(b,a)$ is used in DP to get the estimate of the value functions. Algorithm 1 chooses the most successful action which maximizes the estimate of value function ($\hat{U}_m(b_t,a) + \gamma \sum_{b' \in \mathbb{B}_m} B_t(b'|b_t,a) J_m^n(b')$ in Line 19). It represents the exploitation phase. However, the empirical averages $\hat{U}_m(b,a)$ can deviate from the true values $U_t(b,a)$ greatly. To address this issue, Algorithm 1 introduces perturbation $\epsilon_n(a)$ using the idea of FPL. In particular, Algorithm 1 chooses arguments of the maxima of the summation of $\epsilon_n(a)$ and $\hat{U}_m(b_t,a) + \gamma \sum_{b' \in \mathbb{B}_m} B_t(b'|b_t,a) J_m^n(b')$. It represents the exploration phase. The perturbation term $\epsilon_n(a)$ compensates the scenario where action a is optimal but the value of $\hat{U}_m(b_t,a) + \gamma \sum_{b' \in \mathbb{B}_m} B_t(b'|b_t,a) J_m^n(b')$ is not maximum. With this perturbation, the algorithm avoids being trapped in the suboptimal actions. The perturbation is a diminishing random vector. As time goes by, the effect of the perturbation will decrease so that Algorithm 1 will rely more on the exploitation. In order to enforce computational tractability, the value function computations are

restricted to a sequence of incrementally expanded belief sets. This idea is called *point-based* [26].

4.4 Computational complexity

We proceed to discuss the computational complexity of the update of the estimated value functions J_m^{n+1} (Line 25 in Algorithm 1). In what follows, we use $|S|$ to represent the cardinality of set $|S|$. To update $J_m^{n+1}(b)$ for all $b \in \mathcal{B}_m$, the agent needs to compute $B_t(b'|b,a)$ for all $b', b \in \mathcal{B}_m$ and all $a \in \mathcal{A}_t$. According to equations (4) and (5), getting $B_t(b'|b,a)$ for a given b, a and b' needs $O(|\mathcal{O}||\mathcal{S}|^2)$ products. Then, getting $B_t(b'|b,a)$ for all $b', b \in \mathcal{B}_m$ and all $a \in \mathcal{A}_t$ needs $O(|\mathcal{O}||\mathcal{S}|^2|\mathcal{B}_m|^2|\mathcal{A}_t|)$ products. Even though the update only takes polynomial time, it could be very time-consuming in practice. In Section 5, we will show that why the update is infeasible in real-world based simulations. To address the computational complexity, we propose a Q-learning version of Algorithm 1.

Algorithm 2 Point-based Q-FPL

...

14: $n = 0$;

15: Let $\hat{Q}_m^n(b,a) = \hat{U}_m(b,a)$ for all $(b,a) \in \mathbb{B}_m \times \mathcal{A}_t$;

16: **for** $t \in k_m$ **do**

17: Update belief state $b_t = SE(b_{t-1}, a_{t-1}, o_t)$;

18: **if** $b_t \in \mathbb{B}_m$ **then**

19: $a_t \in \arg\max_{a \in \mathcal{A}_t} [\hat{Q}_m^n(b_t,a) + \epsilon_t(a)]$;

20: **else**

21: Uniformly select an action $a_t \in \mathcal{A}_t$ and execute a_t;

22: **end if**

23: Receive a utility value v_t;

24: Observe o_{t+1};

25: Update the subsequent belief state $b_{t+1} = SE(b_t, a_t, o_{t+1})$;

26: $\hat{Q}_m^{n+1}(b_t,a_t) = (1-\beta_m)\hat{Q}_m^n(b_t,a_t) + \beta_m \left[\hat{U}_m(b_t,a_t) + \max_{a' \in \mathcal{A}_t} \hat{Q}_m^n(b_{t+1},a')\right]$;

27: $n = n + 1$;

28: **end for**

...

4.5 Q-learning based algorithm

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

$$Q_t(b,a) = U_t(b,a) + \gamma \sum_{b \in \mathcal{B}} B_t(b'|b,a) J_{t+1}^*(b').$$

$Q_t(b,a)$ is the aggregate utility for executing action a at time step t and following the optimal policy π^* thereafter. The relation between $Q_t(b,a)$ and $J_t^*(b)$ is given by $J_t^*(b) = \max_{a \in \mathcal{A}_t} Q_t(b,a)$. And we estimate the Q-functions Q_t instead of the optimal value functions J_t^* by applying Q-learning [29] in Algorithm 2. In particular, let $\hat{Q}_m^n$ be the n-th estimate of Q-function in interval k_m, then the $(n+1)$-th estimate is updated from $\hat{Q}_m^n$ by the recursive equation shown in Line 26 of Algorithm 2. the recursive equation makes a correction of $\hat{Q}_m^n(b_t,a_t)$; i.e., the value of belief-action pair (b_t,a_t), based on the new empirical average utility values $\hat{U}_m(b_t,a_t)$.

The agents choose the action which maximizes the estimate of Q-function plus the perturbation if the belief state b_t is in $\mathbb{B}_m$ (Line 19) and uniformly selects an action if the belief state is new (Line 21). Algorithm 2 updates $\hat{Q}_m^{n+1}$ asynchronously (Line 26). That is, it updates the value of $\hat{Q}_m^{n+1}$ for a single belief-action pair each time step.

Compared with Algorithm 1, Algorithm 2 does not need to compute $B_t(b'|b,a)$ and it only updates one value of $\hat{Q}_m^{n+1}$. Therefore, Algorithm 2 only requires at most $|\mathcal{A}_t|$ comparisons for the update at time step t.

5 EVALUATION

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

5.1 Evaluation setup

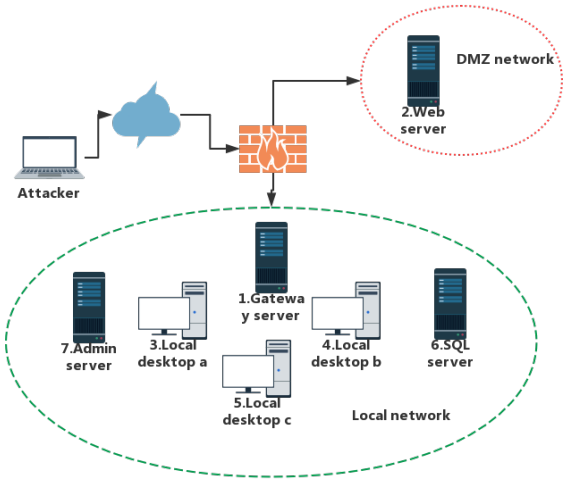

Figure 2: Test network.

Machine	Vulnerability	CVE#
Gateway server	Untrusted cookie in OpenSSH	2007-4752
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
SQL server	SQL injection	2008-5416
Admin server	MS SMV service Stack buffer overflow	2008-4050

Table 1: Vulnerabilities in the test network.

We setup a test network similar to the one in [21], which is shown in Figure 2. The network consists of 7 machines located in two subnets. The Web server is located in the DMZ network while local desktops, the Gateway server, SQL server and 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 [21] and listed in Table 1. These vulnerabilities can produce multiple attack scenarios. In the evaluation, we use BAG to

simulate one attack scenario where the attacker starts from either the Gateway server or the Web server and tries to compromise the whole network. From either the Gateway server or the Web server, local desktop a is accessible by exploiting MS Video ActiveX buffer overflow. And from the Web server, local desktop b is accessible by exploiting LICQ buffer overflow. With local user privilege, local desktop c can be compromised by exploiting remote login. And the SQL server can be compromised through any of the three local desktops by exploiting SQL injection. Finally, with information in local desktop c and the SQL server, the Admin server can be compromised by exploiting MS SMV service Stack buffer overflow.

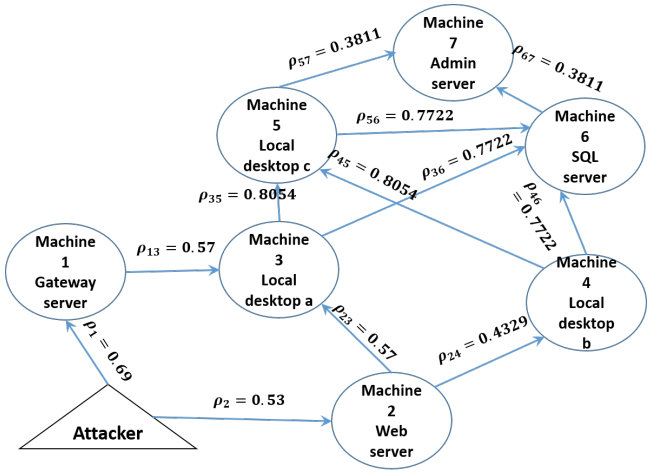

Figure 3: BAG of the test network.

BAG. We simulate the above attack scenario with the BAG shown in Figure 3. The Web server and the Gateway server are leaf nodes and accessible to the remote attacker. And the rest 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 2.1, the exploit probabilities are calculated based on the exploitability metric of CVSS scores. 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 network, adjacent network, local access or physical access. AC describes the conditions beyond the attacker's control that must exist in order 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. More details on CVSS metrics and their scoring computation can be found in the CVSS specification document [25]. Based on the features of CVSS exploitability metrics, 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 $|\mathcal{S}| = 2^7 = 128$ 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/she knows which compromised machines are recovered by the defender. And if the attacker has no available machine to exploit for next time step; e.g., no machine is compromised, he/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 7 machines and to reimage at most 3 out of 7 machines. There are total $|O| = 2^3 \times \binom{7}{3} = 280$ observations and $|\mathcal{A}| = \binom{7}{3} \times (\binom{7}{0} + \binom{7}{1} + \binom{7}{2} + \binom{7}{3}) = 2240$ actions.

Utility. As mentioned in Section 2.2, utilities are introduced to quantify security levels of the network. In the evaluation, we use time independent utility function $u(s, a) = r(s, a) - c(a)$ which consists of reward part and cost part. The reward part is defined based on the CIA triad. In particular, we use the impact metric of CVSS score to calculate $r(s, a)$ as follows:

$$r(s, a) = R - \sum_{\{s^i \in s | s^i = 1, i \notin a^r\}} [I_C(i) + I_I(i) + I_A(i)],$$

where $I_C(i), I_I(i), I_A(i)$ are the confidentiality, integrity, and availability impact score brought by the successful exploitation of the vulnerability on machine i. We choose CIA triad because they are commonly considered as the three most crucial components of network security. Each impact score is a real number scaling from 0 to 10 and a higher score means that the network is less secure. We use a constant R to represent the base security level of the clean network; i.e., the network with 0 compromised machine. In this network, $R = 70$. To represent the reward of keeping the network secure, we subtract the total impact score of compromised machines in the network after action a_t is taken from R. Therefore, a higher value of r represents that the network is more secure, and vice verse. And the cost induced by the defender's action is calculated as $c(a) = \sum_{\{i \in a^r | s^i = 0\}} I_A(i)$, which quantifies the cost induced by reimaging the clean machines. $c(a)$ is concerned with the availability cost. If a clean machine is reimaged, some resources on the machine becomes unavailable for trusted users.

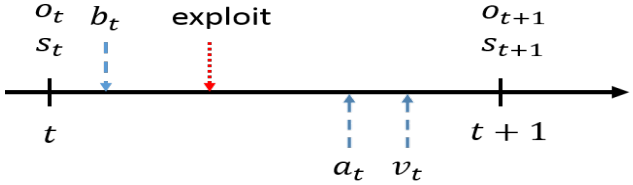

Figure 4: Events and updates in ACD problem.

Time line. We use a time line shown in Figure 4 to describe when the main events and updates happen in the ACD problem. We consider a time step starts when the defender receives an observation. At the beginning of time step t, the network is in state s_t and the defender updates its belief state b_t. The attacker exploits some available vulnerability while the defender chooses an action based on his/her belief state. After the action is chosen, the defender receives a utility value v_t. Note that in the ACD problem, the received utility value is not $u(s_t, a_t)$ because the defender cannot measure the value without knowing full state s_t. Instead, the defender uses the following measurement as the utility value $v_t = v(o_t, a_t) \triangleq R - \sum_{\{s^i \in o_t | s^i=1, i \notin a_t^r\}} [I_C(i)+I_I(i)+I_A(i)] - \sum_{\{i \in a_t^r | s^i=0 \text{ and } s^i \in o_t\}} I_A(i)$. That is, the defender can only measure the reward and cost based on the observation and action.

5.2 A customized algorithm

Based on the evaluation setup, we simulate the interactions among the attacker, network and defender in Python. All the simulations are conducted on an Intel(R) Core(TM) i5 machine with 8GB memory running OS X 10.11.6. As mentioned in Section 4.4, Algorithm 1 requires $O(|\mathcal{O}||\mathcal{S}|^2|\mathcal{B}_m|^2|\mathcal{A}_t|)$ products for the update at time step t. In our evaluation, the numbers of states, actions are observations are large. Even in the interval k_1, the update requires more than 10^{10} products. And Algorithm 1 takes more than two hours to finish one iteration while Algorithm 2 takes 0.69s to finish one iteration on average. This computation complexity makes Algorithm 1 infeasible in practice. Therefore, the defender in our evaluation applies Algorithm 2 instead.

In general, $\mathcal{A}_t = \mathcal{A}$ for all t if there is no restriction on the action space; e.g., any machine can be reimaged any time in the ACD problem. In the evaluation of Algorithm 2, we let $v_t = u(o_t, a_t)$ and $\mathcal{A}_t = \mathcal{A}$ for all t. But note that the observation o_t does not include false positives (mentioned in Section 2.2). Based on this property, we propose a customized Algorithm 3 which restricts $\mathcal{A}_t$ at time step t based on o_t as follows: only the actions that reimage the compromised machines and avoid the clean machines in o_t will be considered at time step t (Lines 18 to 27).

Algorithm 3 Point-based Q-FPL for ACD

```
...
18: A_t = A
19: for a ∈ A do
20:   if any i ∈ a^r such that s^i ∈ o_t and s^i = 0 then
21:     A_t = A \ {a};
22:   else
23:     if any s^i ∈ o_t and s^i = 1 but i ∉ a^r then
24:       A_t = A \ {a};
25:     end if
26:   end if
27: end for
...
```

5.3 Results and discussion

To show the effectiveness of Algorithm 2 and Algorithm 3, we compare their performances with a baseline policy. The baseline policy uniformly chooses one action each time step and is referred to as uniform selection policy. The duration of each simulation (from the attack begins till the attack ends) is 100 time steps and we repeat 20 identical simulations. The comparison results are shown in Figure 5.

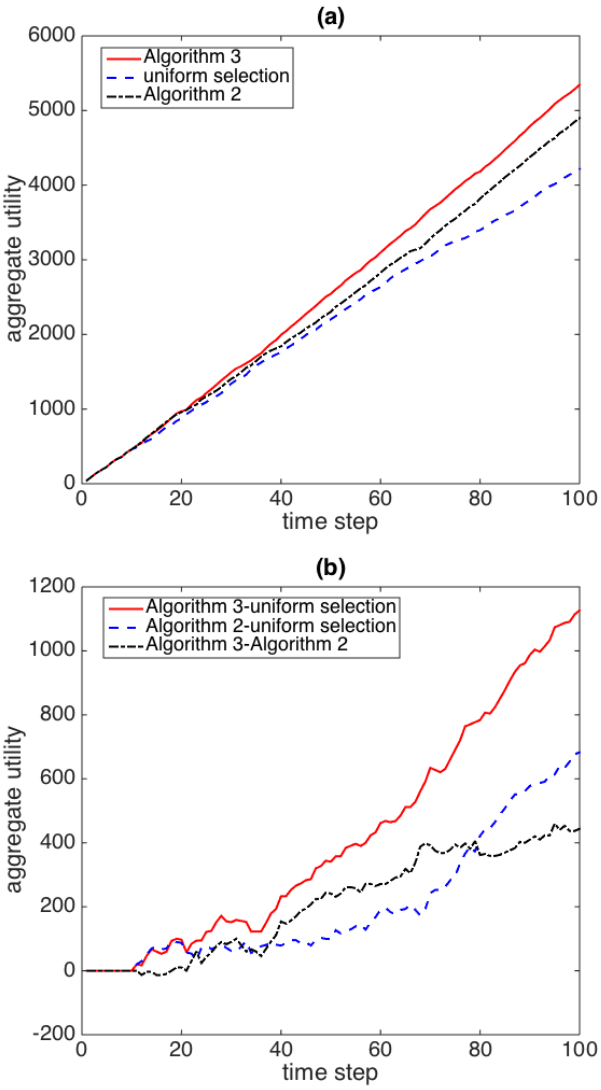

Figure 5: (a) shows the Aggregate utilities of three policies; (b) shows the gaps of aggregate utilities between Algorithm 3, Algorithm 2 and uniform selection policy.

Figure 5-(a) shows the average aggregate utilities over 20 simulations at the particular time steps. We can see the aggregate utilities of Algorithm 3 are always the highest among all three policies. Besides, the aggregate utilities of Algorithm 2 are higher than those of uniform selection policy. The uniform selection policy performs worst because it does not learn anything and only randomly chooses actions, which could induce only cost at some time steps. In addition, we also compare the gaps among all three policies over time. Figure 5-(b) shows that both Algorithm 3 and Algorithm 2

increase their leads in general on aggregate utilities compared with uniform selection policy. We can conclude that both Algorithm 2 and Algorithm 3 enable the defender to identify effective defense policies when utility functions are unknown.

We further discuss the differences between Algorithm 2 and Algorithm 3 via evaluation results. First, Algorithm 3 outperforms Algorithm 2 in terms of aggregate utilities. Second, the gaps between Algorithm 3 and Algorithm 2 enlarge but the gaps between their slops reduce as time goes by. To explain the differences, we have some conjectures based on the intuitions of the algorithms. First, the customized Algorithm 3 eliminates the actions that conflict with the current observation. Notice that the observation has no-false-positive feature. That is, the defender would not reimage clean machines. So it performs better than Algorithm 2. Second, Algorithm 3 might suffer from the false negatives induced by the observations; i.e., clean machines in the current observation are already compromised when the new action is taken. Then the available action set $\mathcal{A}_t$ for Algorithm 3 might not contain optimal action at time step t while Algorithm 2 does not rule out any action at any time. Therefore, as time goes by, the advantage of Algorithm 3 might decrease.

6 CONCLUSION

This paper proposes online algorithms to solve POMDP problems with unknown utility functions. Further, the algorithm is customized to guide the defender to effectively defend a computer network against a sequence of intrusions. The effectiveness of the algorithm is verified in numerical simulations based on real-world attacks on a small-scale computer network.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. Z. Hu and M. Zhu are partially supported by ARO under Grant No.: W911NF-13-1-0421 (MURI) and NSF under Grant No.: CNS-1505664. P. Liu is partially supported by ARO under Grant No.: W911NF-13-1-0421 (MURI), AFOSR under Grant No.: W911NF1210055 and NSF under Grant No.: CNS-1422594.

REFERENCES

[1] Karl J. Åström. 1965. Optimal control of Markov processes with incomplete state information. *J. Math. Anal. Appl.* 10, 1 (1965), 174 – 205. https://doi.org/10.1016/0022-247X(65)90154-X

[2] Richard E. Bellman. 2003. *Dynamic Programming*. Dover Publications.

[3] Richard E. Bellman and Stuart E. Dreyfus. 1962. *Applied dynamic programming*. Princeton University Press.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. 1996. *Neuro-Dynamic Programming* (1st ed.). Athena Scientific.

[5] David Bigelow, Thomas Hobson, Robert Rudd, William Streilein, and Hamed Okhravi. 2015. Timely Rerandomization for Mitigating Memory Disclosures. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. Denver, Colorado, USA, 268–279.

[6] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. 1994. Acting Optimally in Partially Observable Stochastic Domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2) (AAAI'94)*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1023–1028.

[7] Ping Chen, Jun Xu, Zhiqiang Lin, Dongyan Xu, Bing Mao, and Peng Liu. 2015. A Practical Approach for Adaptive Data Structure Layout Randomization. In *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15)*. Vienna, Austria, 69–89.

[8] CVE-2014-0160. 2014. Heartbleed Bug. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160. (2014).

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

[10] Zakir Durumeric, James Kasten, David Adrian, Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. Vancouver, BC, Canada, 475–488.

[11] Zhisheng Hu, Ping Chen, Yang Lu, Minghui Zhu, and Peng Liu. 2016. Towards a science for adaptive defense: Revisit server protection. In *IEEE International Conference on Collaboration and Internet Computing*. Pittsburgh, 112–121.

[12] Zhisheng Hu, Minghui Zhu, Ping Chen, and Peng Liu. 2016. On convergence rates of robust adaptive game theoretic learning algorithms. *ArXiv e-prints* (Dec. 2016). arXiv:math.OC/1612.04724 https://arxiv.org/abs/1612.04724.

[13] Jeff Hughes, Lawrence Carin, and George Cybenko. 2008. Cybersecurity Strategies: The QuERIES Methodology. *Computer* 41 (2008), 20–26. https://doi.org/doi.ieeecomputersociety.org/10.1109/MC.2008.295

[14] Marcus Hutter and Jan Poland. 2005. Adaptive Online Prediction by Following the Perturbed Leader. *Journal of Machine Learning Research* 6 (2005), 639–660.

[15] Adam Kalai and Santosh Vempala. 2005. Efficient Algorithms for Online Decision Problems. *J. Comput. System Sci.* 71, 3 (Oct. 2005), 291–307. https://doi.org/10.1016/j.jcss.2004.10.016

[16] 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)*. San Jose, CA, USA.

[17] Yu Liu and Hong Man. 2005. Network Vulnerability Assessment Using Bayesian Networks. In *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005*. 61–71.

[18] Erik Miehling, Mohammad Rasouli, and Demosthenis Teneketzis. 2015. Optimal Defense Policies for Partially Observable Spreading Processes on Bayesian Attack Graphs. In *Proceedings of the Second ACM Workshop on Moving Target Defense (MTD '15)*. ACM, New York, NY, USA, 67–76. https://doi.org/10.1145/2808475.2808482

[19] George E. Monahan. 1982. A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science* 28, 1 (1982), 1–16. http://www.jstor.org/stable/2631070

[20] Hamed Okhravi, James Riordan, and Kevin Carter. 2014. Quantitative evaluation of dynamic platform techniques as a defensive mechanism. In *Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014*. Gothenburg, Sweden, 405–425.

[21] 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 (Jan 2012), 61–74. https://doi.org/10.1109/TDSC.2011.34

[22] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[23] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. 2012. POMDPs Make Better Hackers: Accounting for Uncertainty in Penetration Testing. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI'12)*. AAAI Press, 1816–1824. http://dl.acm.org/citation.cfm?id=2900929.2900985

[24] Mike Schiffman. 2017. Common Vulnerability Scoring System (CVSS). (2017). http://www.first.org/cvss.

[25] Mike Schiffman. 2017. Common Vulnerability Scoring System v3.0: Specification Document. (2017). https://www.first.org/cvss/cvss-v30-specification-v1.7.pdf.

[26] Guy Shani, Joelle Pineau, and Robert Kaplow. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27, 1 (01 Jul 2013), 1–51. https://doi.org/10.1007/s10458-012-9200-2

[27] 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.

[28] Symantec. 2015. Internet Security Threat Report. (2015). https://know.elq.symantec.com/LP=1542.

[29] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. In *Machine Learning*. 279–292.

[30] Herbert Weisberg. 1992. *Central tendency and variability*. Number 83. Sage University Paper Series on Quantitative Applications in the Social Sciences.

[31] Jun Xu, Pinyao Guo, Mingyi Zhao, Robert F. Erbacher, Minghui Zhu, and Peng Liu. 2014. Comparing different moving target defense techniques. In *First ACM Workshop on Moving Target Defense, in Association with 2014 ACM Conference on Computer and Communications Security*. Scottsdale, Arizona, 97–107.

[32] Lu Yu and Richard R. Brooks. 2013. Applying POMDP to Moving Target Optimization. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW '13)*. ACM, New York, NY, USA, Article 49, 4 pages. https://doi.org/10.1145/2459976.2460032

[33] Emmanuele Zambon and Damiano Bolzoni. 2006. Network Intrusion Detection Systems: False Positive reduction through Anomaly Detection. (2006). http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zambon.pdf.

[34] Minghui Zhu, Zhisheng Hu, and Peng Liu. 2014. Reinforcement learning algorithms for adaptive cyber defense against Heartbleed. In *First ACM Workshop on Moving Target Defense (MTD '14)*. Scottsdale, Arizona, USA, 51–58.

[35] 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.

[36] Quanyan Zhu and Tamer Başar. 2009. Dynamic policy-based IDS configuration. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. 8600–8605. https://doi.org/10.1109/CDC.2009.5399894

[37] 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* (2013), 305–329.