Learning Intrusion Prevention Policies through Optimal Stopping

Kim Hammar †‡ and Rolf Stadler†‡
† Division of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden
‡ KTH Center for Cyber Defense and Information Security, Sweden
Email: {kimham, stadler}@kth.se

***Abstract*—We study automated intrusion prevention using reinforcement learning. In a novel approach, we formulate the problem of intrusion prevention as an optimal stopping problem. This formulation allows us insight into the structure of the optimal policies, which turn out to be threshold based. Since the computation of the optimal defender policy using dynamic programming is not feasible for practical cases, we approximate the optimal policy through reinforcement learning in a simulation environment. To define the dynamics of the simulation, we emulate the target infrastructure and collect measurements. Our evaluations show that the learned policies are close to optimal and that they indeed can be expressed using thresholds.**

***Index Terms*—Network Security, automation, optimal stopping, reinforcement learning, Markov Decision Processes**

I. Introduction

An organization's security strategy has traditionally been defined, implemented, and updated by domain experts [1]. Although this approach can provide basic security for an organization's communication and computing infrastructure, a growing concern is that infrastructure update cycles become shorter and attacks increase in sophistication. Consequently, the security requirements become increasingly difficult to meet. As a response, significant efforts are made to automate security processes and functions.

Over the last years, research directions emerged to automatically find and update security policies. One such direction aims at automating the creation of threat models for a given infrastructure [2]. A second direction focuses on evolutionary processes that produce novel exploits and corresponding defenses [3]. In a third direction, the interaction between an attacker and a defender is modeled as a game, which allows attack and defense policies to be analyzed and sometimes constructed using game theory [4], [5]. In a fourth direction, statistical tests are used to detect attacks [6]. Further, the evolution of an infrastructure and the actions of a defender is studied using the framework of dynamical systems. This framework allows optimal policies to be obtained using methods from control theory [7] or dynamic programming [8], [9]. In all of the above directions, machine learning techniques are often applied to estimate model parameters and policies [10], [11].

Many activities center around modeling the infrastructure as a discrete-time dynamical system in the form of a Markov Decision Process (MDP). Here, the possible actions of the defender are defined by the action space of the MDP, the defender policy is determined by the actions that the defender takes in different states, and the security objective is encoded in the reward function, which the defender tries to optimize.

To find the optimal policy in an MDP, two main methods are used: dynamic programming and reinforcement learning. The advantage of dynamic programming is that it has a strong theoretical grounding and oftentimes allows to derive properties of the optimal policy [12], [13]. The disadvantage is that it requires complete knowledge of the MDP, including the transition probabilities. In addition, the computational overhead is high, which makes it infeasible to compute the optimal policy for all but simple configurations [14], [13], [8]. Alternatively, reinforcement learning enables *learning* the dynamics of the model through exploration. With the reinforcement learning approach, it is often possible to compute close approximations of the optimal policy for non-trivial configurations [10], [15], [14], [16]. As a drawback, however, theoretical insights into the structure of the optimal policy generally remain elusive.

In this paper, we study an intrusion prevention use case that involves the IT infrastructure of an organization. The operator of this infrastructure, which we call the defender, takes measures to protect it against a possible attacker while, at the same time, providing a service to a client population. The infrastructure includes a public gateway through which the clients access the service and which also is open to a plausible attacker. The attacker decides when to start an intrusion and then executes a sequence of actions that includes reconnaissance and exploits. Conversely, the defender aims at preventing intrusions and maintaining the service to its clients. It monitors the infrastructure and can block outside access to the gateway, an action that disrupts the service but stops any ongoing intrusion. What makes the task of the defender difficult is the fact that it lacks direct knowledge of the attacker's actions and must infer that an intrusion occurs from monitoring data.

We study the use case within the framework of discrete-time dynamical systems. Specifically, we formulate the problem of finding an optimal defender policy as an *optimal stopping problem*, where stopping refers to blocking access to the gateway. Optimal stopping is frequently used to model problems in the fields of finance and communication systems [17], [18], [6], [19]. To the best of our knowledge, finding a intrusion

978-3-903176-36-2 ©2021 IFIP

prevention policy through solving an optimal stopping problem is a novel approach.

By formulating intrusion prevention as an optimal stopping problem, we know from the theory of dynamic programming that the optimal policy can be expressed through a threshold that is obtained from observations, i.e. from infrastructure measurements [13], [12]. This contrasts with prior works that formulate the problem using a general MDP, which does not allow insight into the structure of optimal policies [10], [11], [20], [21].

To account for the fact that the defender only has access to a limited number of measurements and cannot directly observe the attacker, we model the optimal stopping problem with a Partially Observed Markov Decision Process (POMDP). We obtain the defender policies by simulating a series of POMDP episodes in which an intrusion takes place and where the defender continuously updates its policy based on outcomes of previous episodes. To update the policy, we use a state-of-the-art reinforcement learning algorithm. This approach enables us to find effective defender policies despite the uncertainty about the attacker's behavior and despite the large state space of the model.

We validate our approach to intrusion prevention for a non-trivial infrastructure configuration and two attacker profiles. Through extensive simulation, we demonstrate that the learned defender policies indeed are threshold based, that they converge quickly, and that they are close to optimal.

We make two contributions with this paper. First, we formulate the problem of intrusion prevention as a problem of optimal stopping. This novel approach allows us a) to derive properties of the optimal defender policy using results from dynamic programming and b) to use reinforcement learning techniques to approximate the optimal policy for a non-trivial configuration. Second, we instantiate the simulation model with measurements collected from an emulation of the target infrastructure, which reduces the assumptions needed to construct the simulation model and narrows the gap between a simulation episode and a scenario playing out in a real system. This addresses a limitation of related work that rely on abstract assumptions to construct the simulation model [10], [11], [20], [21].

II. The Intrusion Prevention Use Case

We consider an intrusion prevention use case that involves the IT infrastructure of an organization. The operator of this infrastructure, which we call the defender, takes measures to protect it against an attacker while, at the same time, providing a service to a client population (Fig. 1). The infrastructure includes a set of servers that run the service and an intrusion detection system (IDS) that logs events in real-time. Clients access the service through a public gateway, which is also open to the attacker.

We assume that the attacker intrudes into the infrastructure through the gateway, performs reconnaissance, and exploits found vulnerabilities, while the defender continuously monitors the infrastructure through accessing and analyzing IDS statistics and login attempts at the servers. The defender has a single action to stop the attacker, which involves blocking all outside access to the gateway. As a consequence of this action, the service as well as any ongoing intrusion are disrupted.

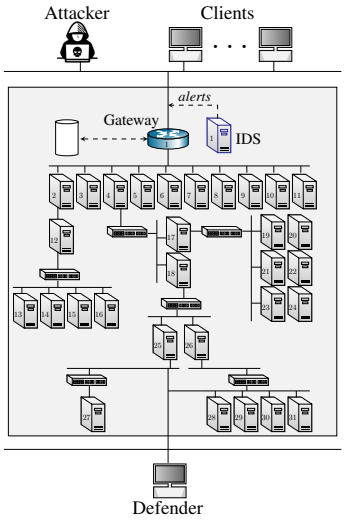

Fig. 1: The IT infrastructure and the actors in the use case.

When deciding whether to block the gateway, the defender must balance two objectives: to maintain the service to its clients and to keep a possible attacker out of the infrastructure. The optimal policy for the defender is to maintain service until the moment when the attacker enters through the gateway, at which time the gateway must be blocked. The challenge for the defender is to identify the precise time when this moment occurs.

In this work, we model the attacker as an agent that starts the intrusion at a random point in time and then takes a predefined sequence of actions, which includes reconnaissance to explore the infrastructure and exploits to compromise the servers.

We study the use case from the defender's perspective. The evolution of the system state and the actions by the defender are modeled with a discrete-time Partially Observed Markov Decision Process (POMDP). The reward function of this process encodes the benefit of maintaining service and the loss of being intruded. Finding an optimal defender policy thus means maximizing the expected reward. To find an optimal policy, we solve an *optimal stopping problem*, where the stopping action refers to blocking the gateway.

III. Theoretical Background

This section contains background information on Markov decision processes, reinforcement learning, and optimal stopping.

A. Markov Decision Processes

A Markov Decision Process (MDP) models the control of a discrete-time dynamical system [22], [23], [12]. An MDP is defined by a seven-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma, \rho_1, T \rangle$. $\mathcal{S}$ denotes the set of states and $\mathcal{A}$ denotes the set of actions.

$\mathcal{P}^a_{ss'}$ refers to the transition probability of moving from state s to state s' when taking action a (Eq. 1), which has the Markov property $\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \ldots, s_t]$. Similarly, $\mathcal{R}^a_{ss'} \in \mathbb{R}$ is the expected reward when taking action a in state s to transition to state s' (Eq. 2). Finally, $\gamma \in (0, 1]$ is the discount factor, $\rho_1 : \mathcal{S} \mapsto [0, 1]$ is the initial state distribution, and T is the time horizon.

$$\mathcal{P}^{a_t}_{s_t s_{t+1}} = \mathbb{P}[s_{t+1}|s_t, a_t] \tag{1}$$

$$\mathcal{R}^{a_t}_{s_t s_{t+1}} = \mathbb{E}[r_{t+1}|a_t, s_t, s_{t+1}] \tag{2}$$

A Partially Observed Markov Decision Process (POMDP) is an extension of an MDP [24], [13], [25]. The difference compared with an MDP is that, in a POMDP, the states $s \in \mathcal{S}$ are not directly observable. Instead, observations are defined by $o \in \mathcal{O}$, which depend on the state s as defined by an observation function $\mathcal{Z}$. Specifically, a POMDP is defined by a nine-tuple $\mathcal{M}_\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma, \rho_1, T, \mathcal{O}, \mathcal{Z} \rangle$. The first seven elements define an MDP. $\mathcal{O}$ denotes the set of observations and $\mathcal{Z}(o', s', a) = \mathbb{P}[o'|s', a]$ is the observation function, where $o' \in \mathcal{O}$, $s' \in \mathcal{S}$, and $a \in \mathcal{A}$.

By using so called belief states $b_t(s_t)$, where $b_t(s_t)$ is defined as the posterior $b_t(s_t) = \mathbb{P}[s_t|a_1, o_1, \ldots, a_{t-1}, o_t]$, a POMDP can be viewed as a special kind of MDP in which all of the essential results for MDPs hold [25], [13]. Formally, belief states are sufficient (Markovian) statistics of the posterior estimate of the state s_t based on the history h_t of actions and observations $h_t = (a_1, o_1, \ldots, a_{t-1}, o_t) \in \mathcal{H}$.

When observing o_{t+1} after taking action a_t, the belief state $b_t \in \mathcal{B}$ is updated using a recursive filter based on Bayes rule:

$$b_{t+1}(s_{t+1}) = C\mathcal{Z}(o_{t+1}, s_{t+1}, a_t) \sum_{s_t \in \mathcal{S}} \mathcal{P}^{a_t}_{s_t s_{t+1}} b_t(s_t) \tag{3}$$

where $\mathcal{B}$ denotes the space of belief states, $C = 1/\mathbb{P}[o_{t+1}|a_t, b_t]$ is a normalizing factor independent of s_{t+1} with the purpose of making b_{t+1} sum to one [25], and $\mathbb{P}[o_{t+1}|a_t, b_t] = \sum_{s' \in S} \mathcal{Z}(o_{t+1}, s', a_t) \sum_{s \in S} \mathcal{P}^{a_t}_{ss'} b_t(s)$.

B. The Reinforcement Learning Problem

The reinforcement learning problem can be formulated as that of finding a policy $\pi^* \in \Pi$ that maximizes the expected discounted cumulative reward of an MDP or POMDP over a horizon T [16], [14]:

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_\pi \left[\sum_{t=1}^{T} \gamma^{t-1} r_t \right] \tag{4}$$

where Π is the policy space, γ is the discount factor, and $\mathbb{E}_\pi$ denotes the expectation under π. In an MDP, a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ is a function that maps states to actions. In a POMDP, a policy is a function that maps either histories to actions $\pi : \mathcal{H} \mapsto \mathcal{A}$ or belief states to actions $\pi : \mathcal{B} \mapsto \mathcal{A}$.

The Bellman equations [26] (Eq. 5-6) relate the optimal policy π^* to the states (or histories/belief states) and actions of the MDP/POMDP: $\pi^* = \arg\max_{a_t \in \mathcal{A}} Q^*(s_t, a_t)$.

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} \mathbb{E}[r_{t+1} + \gamma V^*(s_{t+1})|s_t, a_t] \tag{5}$$

$$Q^*(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V^*(s_{t+1})|s_t, a_t] \tag{6}$$

A key theoretical result for POMDPs is that the value function $V^*(b_t)$ is piecewise linear and convex in the belief state [27].

To solve the Bellman equations and obtain the optimal policy π^*, two principal methods can be used: dynamic programming and reinforcement learning. Dynamic programming algorithms assume access to a perfect model of the MDP/POMDP and solve the Bellman equations iteratively to obtain the optimal policy π^* [12], [23], [13]. Conversely, reinforcement learning algorithms compute or approximate π^* *without* access to a perfect model [16], [14]. Three classes of such algorithms exist: value-based algorithms (e.g. Q-learning [28]), policy-based algorithms (e.g. Proximal Policy Optimization (PPO) [29]), and model-based algorithms (e.g. Dyna-Q [16]).

C. Markovian Optimal Stopping Problems

Optimal stopping is a classical problem in statistics [30], [31], [32]. Different versions of the problem can be found in the literature. Including, discrete-time and continuous time, finite horizon and infinite horizon, fully observed and partially observed, and Markovian and non-Markovian. Consequently, there are also different solution methods, most prominent being the martingale approach [32] and the Markovian approach [31], [12], [23]. In this paper, we consider a partially observed Markovian optimal stopping problem in discrete-time with a finite horizon.

A Markovian optimal stopping problem can be seen as a specific kind of MDP or POMDP where the state of the environment evolves as a discrete-time Markov process $(s_t)_{t=1}^T$ which is fully or partially observed [23], [13]. At each time-step t of this decision process, two actions are available: "stop" and "continue". The *stop* action causes the interaction with the environment to stop and yields a stopping-reward. The time t of the stop action is defined by a *stopping time* τ, where τ is a random variable dependent on $s_1, \ldots, s_t$ [32]. Conversely, the *continue* action causes the environment to evolve to the next time-step and yields a continuation-reward. In this context, the objective is to find a stopping policy $\pi(s_t) \mapsto \{S, C\}$ that maximizes the expected reward, where $\pi(s_t) = S$ indicates a stopping action. This induces the following maximization at each time-step before stopping (Bellman equation [26]):

$$\max \left[\underbrace{\mathbb{E}\left[\mathcal{R}^S_{ss'}\right]}_{\text{stopping reward}}, \underbrace{\mathbb{E}\left[\mathcal{R}^C_{ss'} + \gamma V^*(s')\right]}_{\text{continuation reward}} \right] \tag{7}$$

To solve the maximization above, standard solution methods for MDPs and POMDPs can be applied, such as dynamic programming and reinforcement learning [12], [18]. Further, the solution can be characterized using dynamic programming theory as the least excessive (or superharmonic) majorant of the reward function, or using martingale theory as the Snell envelope of the reward function [32].

IV. Formalizing The Intrusion Prevention Use Case and Our Reinforcement Learning Approach

In this section, we first formalize the intrusion prevention use case described in Section II and then we introduce our

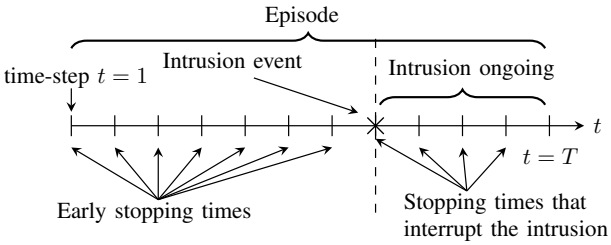

Fig. 2: Optimal stopping formulation of intrusion prevention; the horizontal axis represents time; T is the time horizon; the episode length is $T - 1$; the dashed line shows the intrusion start time; the optimal policy is to stop at the time of intrusion.

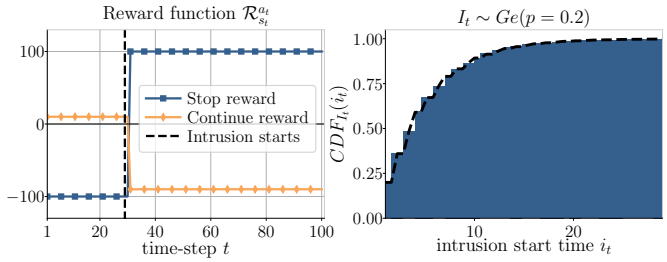

Fig. 3: Left: the reward function for the stop and continue actions; the intrusion starts at $t = 29$; right: the cumulative distribution function (CDF) of the intrusion start time.

solution method. Specifically, we first define a POMDP model of the intrusion prevention use case. Then, we describe our reinforcement learning approach to approximate the optimal defender policy. Lastly, we use the theory of dynamic programming to derive the threshold property of the optimal policy.

A. A POMDP Model of the Intrusion Prevention Use Case

We model the intrusion prevention use case as a partially observed optimal stopping problem where an intrusion starts at a geometrically distributed time and the stopping action refers to blocking the gateway (Fig. 2). This type of optimal stopping problem is often referred to as a *quickest change detection* problem [32], [31], [6].

To formalize this model, we use a POMDP. This model includes the state space and the observation space of the defender. It further includes the initial state distribution, the defender actions, the transition probabilities, the observation function, the reward function, and the optimization objective.

1) States $\mathcal{S}$, Initial State Distribution ρ_1, and Observations $\mathcal{O}$: The system state s_t is defined by the intrusion state $i_t \in \{0, 1\}$, where $i_t = 1$ if an intrusion is ongoing. Further, we introduce a terminal state $\emptyset$, which is reached either when the defender stops or when the attacker completes an intrusion.

At time $t = 1$ no intrusion is in progress. Hence, the initial state distribution is the degenerate distribution $\rho_1(s_1 = 0) = 1$.

The defender has a partial view of the system state and does not know whether an intrusion is in progress. Specifically, if the defender has not stopped, it observes three counters $o_t = (\Delta x_t, \Delta y_t, \Delta z_t)$. The counters are upper bounded, where $\Delta x_t \in [0, X_{max}]$, $\Delta y_t \in [0, Y_{max}]$, $\Delta z_t \in [0, Z_{max}]$ denote the number of severe IDS alerts, warning IDS alerts, and login attempts generated during time-step t, respectively. Otherwise, if the defender has stopped, it observes $o_t = s_t = \emptyset$.

2) Actions $\mathcal{A}$: The defender has two actions: "stop" (S) and "continue" (C). The action space is thus $\mathcal{A} = \{S, C\}$.

3) Transition Probabilities $\mathcal{P}^a_{ss'}$: We model the start of an intrusion by a Bernoulli process $(Q_t)_{t=1}^T$, where $Q_t \sim Ber(p = 0.2)$ is a Bernoulli random variable. The time t of the first occurrence of $Q_t = 1$ is the change point representing the start time of the intrusion I_t, which thus is geometrically distributed, i.e. $I_t \sim Ge(p = 0.2)$ (Fig. 3). As the geometric distribution has the memoryless property, the intrusion start time is Markovian.

An intrusion lasts for a finite number of time-steps, denoted by a constant i_T. This implies that an intrusion has ended if $t \geq I_t + i_T$.

We define the transition probabilities $\mathcal{P}^{a_t}_{s_t s_{t+1}} = \mathbb{P}\left[s_{t+1} | s_t, a_t\right]$ as follows:

$$\mathbb{P}\left[\emptyset | \cdot, S\right] = \mathbb{P}\left[\emptyset | \emptyset, \cdot\right] = 1 \tag{8}$$

$$\mathbb{P}\left[\emptyset | 1, \cdot\right] = 1 \qquad t \geq I_t + i_T \tag{9}$$

$$\mathbb{P}\left[0 | 0, C\right] = 1 - p \tag{10}$$

$$\mathbb{P}\left[1 | 0, C\right] = p \tag{11}$$

$$\mathbb{P}\left[1 | 1, C\right] = 1 \qquad t < I_t + i_T \tag{12}$$

All other transitions have probability 0.

Eq. 8-9 defines the transition probabilities to the terminal state $\emptyset$. The terminal state is reached either when taking the stop action S or when an intrusion completes ($t \geq I_t + i_T$). Eq. 10-12 define the transition probabilities when taking the continue action C. Eq. 10 captures the case where no intrusion occurs and where $i_{t+1} = i_t = 0$; Eq. 11 captures the start of an intrusion where $i_t = 0, i_{t+1} = 1$; and Eq. 12 describes the case where an intrusion is in progress and $i_{t+1} = i_t = 1$.

4) Observation Function $\mathcal{Z}(o', s', a)$: We assume that the number of IDS alerts and login attempts generated during a single time-step are random variables $X \sim f_X$, $Y \sim f_Y$, $Z \sim f_Z$, dependent on the intrusion state and defined on the sample spaces $\Omega_X = \{0, 1, \ldots, X_{max}\}$, $\Omega_Y = \{0, 1, \ldots, Y_{max}\}$, and $\Omega_Z = \{0, 1, \ldots, Z_{max}\}$. Consequently, the probability that Δx severe alerts, Δy warning alerts, and Δz login attempts are generated during time-step t is $f_{XYZ}(\Delta x, \Delta y, \Delta z | i_t, I_t, t)$.

We define the observation function $\mathcal{Z}(o', s', a) = \mathbb{P}[o' | s', a]$ as follows:

$$\mathcal{Z}\big((\Delta x, \Delta y, \Delta z), i_t, C\big) = f_{XYZ}(\Delta x, \Delta y, \Delta z | i_t, I_t, t) \tag{13}$$

$$\mathcal{Z}\big(\emptyset, \emptyset, \cdot\big) = 1 \tag{14}$$

5) Reward Function $\mathcal{R}^a_s$: The reward function is parameterized by the reward that the defender receives for stopping an intrusion ($R_{st} = 100$), the loss of stopping before an intrusion has started ($R_{es} = -100$), the reward for maintaining service ($R_{sla} = 10$), and the loss of being intruded ($R_{int} = -100$), respectively.

We define the deterministic reward function $\mathcal{R}_{s_t}^{a_t} = r(s_t, a_t)$ to be (Fig. 3):

$$r(\emptyset, \cdot) = 0 \tag{15}$$

$$r(i_t, S) = \mathbb{1}_{i_t=0} R_{es} + \mathbb{1}_{i_t=1} R_{st} \tag{16}$$

$$r(i_t, C) = R_{sla} + \mathbb{1}_{i_t=1} R_{int} \tag{17}$$

Eq. 15 states that the reward in the terminal state is zero. Eq. 16 indicates that stopping an intrusion incurs a reward but stopping before an intrusion starts yields a loss, where S is the stop action and $\mathbb{1}$ is the indicator function. Lastly, as can be seen from Eq. 17, the defender receives a positive reward for maintaining service and a loss for taking the continue action C while under intrusion. This means that the maximal reward is received if the defender stops when an intrusion starts.

6) Policy Space Π_θ, Time Horizon T, and Objective J: A policy π_θ is a function that maps either histories or belief states to actions: $\pi_\theta : \mathcal{H} \mapsto \mathcal{A}$ or $\pi_\theta : \mathcal{B} \mapsto \mathcal{A}$. Further, a policy π_θ is parameterized by a vector $\theta \in \mathbb{R}^d$, where $\pi_\theta \in \Pi_\theta$.

The time horizon is defined by the time-step when the terminal state $\emptyset$ is reached, which is a random variable $T_\emptyset$. Since we know the expectation of the intrusion time I_t and assume that the duration of an intrusion is finite, we conclude that the horizon is finite: $\mathbb{E}_{\pi_\theta}[T_\emptyset] < \infty$.

The optimal policy $\pi_\theta^* \in \Pi_\theta$ maximizes the expected cumulative reward over the random horizon $T_\emptyset$:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T_\emptyset} \gamma^{t-1} r(s_t, a_t)\right], \ \pi^* = \arg\max_{\pi_\theta \in \Pi_\theta} J(\theta) \tag{18}$$

where we set the discount factor $\gamma = 1$.

Eq. 18 defines the objective of the optimal stopping problem. In the following section, we describe our approach for solving this problem using reinforcement learning.

B. Our Reinforcement Learning Approach

Since the POMDP model is *unknown* to the defender, we use a model-free reinforcement learning approach to approximate the optimal policy. Specifically, we use the state-of-the-art reinforcement learning algorithm PPO [29] to learn a policy $\pi_\theta : \mathcal{H} \mapsto \mathcal{A}$ that maximizes the objective in Eq. 18.

Due to computational limitations (i.e. finite memory), we summarize the history $h_t = (a_1, o_1, a_2, o_2, \ldots, a_{t-1}, o_t)$ by a vector $\hat{h}_t = (x_t, y_t, z_t, t)$, where x_t, y_t, z_t are the accumulated counters of the observations $o_i = (\Delta x_i, \Delta y_i, \Delta z_i)$ for $i = 1, \ldots, t$: $x_t = \sum_{i=1}^t \Delta x_i$, $y_t = \sum_{i=1}^t \Delta y_i$, $z_t = \sum_{i=1}^t \Delta z_i$.

PPO implements the *policy gradient* method and uses stochastic gradient ascent with the following gradient [29]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\underbrace{\nabla_\theta \log \pi_\theta(a|\hat{h})}_{\text{actor}} \underbrace{A^{\pi_\theta}(\hat{h}, a)}_{\text{critic}}\right] \tag{19}$$

where $A^{\pi_\theta}(\hat{h}, a) = Q^{\pi_\theta}(\hat{h}, a) - V^{\pi_\theta}(\hat{h})$ is the so-called advantage function. We implement π_θ with a deep neural network that takes as input the summarized history $\hat{h}$ and produces as output a conditional probability distribution $\pi_\theta(a|\hat{h})$. The neural network follows an actor-critic architecture [33] and computes a second output that estimates the value function $V^{\pi_\theta}(\hat{h})$, which is used to estimate $A^{\pi_\theta}(\hat{h}, a)$ in Eq. 19.

The hyperparameters of our implementation are given in Appendix A and were decided based on smaller search in parameter space.

The defender policy is learned through simulation of the POMDP. First, we simulate a given number of episodes. We then use the episode outcomes and trajectories to estimate the expectation of the gradient in Eq. 19. Then, we use the estimated gradient and the PPO algorithm [29] with the Adam optimizer [34] to update the policy. This process of simulating episodes and updating the policy continues until the policy has sufficiently converged.

C. Threshold Property of the Optimal Policy

The policy that solves the optimal stopping problem is defined by the optimization objective in Eq. 18. From the theory of dynamic programming, we know that this policy satisfies the Bellman equation [22], [12], [13], [25]:

$$\pi^*\big(b(1)\big) = \arg\max_{a \in \mathcal{A}} \left[r\big(b(1), a\big) + \sum_{o \in \mathcal{O}} \mathbb{P}[o|b(1), a] V^*\big(b_o^a(1)\big)\right] \tag{20}$$

where $b(1) = \mathbb{P}[s_t = 1|h_t]$ is the belief that the system is in state 1 based on the observed history h_t. Consequently, $b(0) = 1 - b(1)$ (see Section III-A for an overview of belief states). Moreover, $b_o^a(1)$ is the belief state updated with the Bayes filter in Eq. 3 after taking action a and observing o. Further, $r\big(b(1), \cdot\big)$ is the expected reward of taking action a in belief state $b(1)$, and V^* is the value function.

We use Eq. 20 to derive properties of the optimal policy. Specifically, we establish the following structural result. The optimal policy π^* is a threshold policy of the form:

$$\pi^*\big(b(1)\big) = \begin{cases} S \text{ (stop)} & \text{if } b(1) \geq \alpha^* \\ C \text{ (continue)} & \text{otherwise} \end{cases} \tag{21}$$

where α^* is a unique threshold.

To see this, we consider both actions of the defender $\mathcal{A} = \{S, C\}$ and derive from Eq. 20:

$$\pi^*(b(1)) = \tag{22}$$

$$\text{argmax}\left[\underbrace{r\big(b(1), S\big)}_{\omega}, \underbrace{r\big(b(1), C\big) + \sum_{o \in \mathcal{O}} \mathbb{P}[o|b(1), C] V^*\big(b_o^C(1)\big)}_{\beta}\right]$$

In the above equation, ω is the expected reward for stopping and β is the expected cumulative reward for continuing. If $\beta = \omega$, both actions of the defender, continuing and stopping, maximize the expected cumulative reward. If $\omega \geq \beta$, it is optimal for the defender to stop.

By rearranging the terms in Eq. 22 and expanding the expressions we derive that it is optimal to stop iff:

$$b(1) \geq \tag{23}$$

$$\underbrace{\frac{110 + \sum_{o \in \mathcal{O}} V^*\big(b_o^C(1)\big)\Big(p\mathcal{Z}(o, 1, C) + (1-p)\mathcal{Z}(o, 0, C)\Big)}{300 + \sum_{o \in \mathcal{O}} V^*\big(b_o^C(1)\big)\Big(p\mathcal{Z}(o, 1, C) + (1-p)\mathcal{Z}(o, 0, C) - \mathcal{Z}(o, 1, C)\Big)}}_{\alpha_{b(1)}}$$

Client	*Functions*	*Application servers*
1	HTTP, SSH, SNMP, ICMP	N_2, N_3, N_{10}, N_{12}
2	IRC, PostgreSQL, SNMP	$N_{31}, N_{13}, N_{14}, N_{15}, N_{16}$
3	FTP, DNS, Telnet	N_{10}, N_{22}, N_4

TABLE 1: Emulated client population; each client interacts with application servers using a set of functions.

This shows that the optimal policy is determined by the scalar thresholds $\alpha_{b(1)}$. Moreover, it can be shown that the difference $\big(b(1) - \alpha_{b(1)}\big)$ is increasing in $b(1)$ and that there exists a minimum $\alpha^* = \alpha_{b(1)}$ such that $b(1) \geq \alpha_{b(1)}$. As a consequence, if the posterior that an intrusion has started exceeds α^* (i.e. $b(1) \geq \alpha^*$), the optimal policy prescribes the stopping action. For this reason, the optimal policy is called a *threshold based policy*. The full proof of Eq. 21 can be found in the extended arXiv version of this paper [35]. Due to space limitation we do not include the full proof here.

V. Emulating the Target Infrastructure to Instantiate the Simulation

To simulate episodes of the POMDP we must know the distributions of alerts and login attempts. We estimate these distributions using measurements from an emulation system. This procedure is detailed in this section.

A. Emulating the Target Infrastructure

The emulation system executes on a cluster of machines that runs a virtualization layer provided by Docker [36] containers and virtual connections. The emulation is configured following the topology in Fig. 1 and the configuration in Appendix B. It emulates the clients, the attacker, and the defender, as well as 31 physical components of the target infrastructure (e.g application servers and the gateway). Each physical entity is emulated using a Docker container. The containers replicate important functions of the target infrastructure, including web servers, databases, SSH servers, etc.

The emulation evolves in discrete time-steps of 30 seconds. During each time-step, the attacker and the defender can perform one action each.

1) Emulating the Client Population: The client population is emulated by three client processes that interact with the application servers through different functions at short intervals, see Table 1.

2) Emulating the Attacker: The start time of an intrusion is controlled by a Bernoulli process as explained in Section IV. We have implemented two types of attackers, NoisyAttacker and StealthyAttacker, both of which execute the sequence of actions listed in Table 2. The actions consist of reconnaissance commands and exploits. During each time-step, one action is executed.

The two types of attackers differ in the reconnaissance command. NoisyAttacker uses a TCP/UDP scan for reconnaissance while StealthyAttacker uses a ping-scan. Since the ping-scan generates fewer IDS alerts than the TCP/UDP scan, it makes the actions of StealthyAttacker harder to detect.

Time-steps t	*Actions*
1–$I_t \sim Ge(0.2)$	(Intrusion has not started)
$I_t + 1$–$I_t + 7$	Recon, brute-force attacks (SSH,Telnet,FTP) on N_2, N_4, N_{10}, login(N_2, N_4, N_{10}), backdoor(N_2, N_4, N_{10}), Recon
$I_t + 8$–$I_t + 11$	CVE-2014-6271 on N_{17}, SSH brute-force attack on N_{12}, login (N_{17}, N_{12}), backdoor(N_{17}, N_{12})
$I_t + 12$–$X + 16$	CVE-2010-0426 exploit on N_{12}, Recon SQL-Injection on N_{18}, login(N_{18}), backdoor(N_{18})
$I_t + 17$–$I_t + 22$	Recon, CVE-2015-1427 on N_{25}, login(N_{25}) Recon, CVE-2017-7494 exploit on N_{27}, login(N_{27})

TABLE 2: Attacker actions to emulate an intrusion.

Metric	*Command in the Emulation*
Login attempts	`cat /var/log/auth.log`
IDS Alerts	`cat /var/snort/alert.csv`

TABLE 3: Commands used to measure the emulation.

3) Emulating Actions of the Defender: The defender takes an action every time-step. The continue action has no effect on the emulation. The stop action changes the firewall configuration of the gateway and drops all incoming traffic.

B. Estimating the Distributions of Alerts and Login Attempts

In this section, we describe how we collect data from the emulation and how we use the data to estimate the distributions of alerts and login attempts.

1) Measuring the Number of IDS alerts and Login Attempts in the Emulation: At the end of every time-step, the emulation system collects the metrics Δx, Δy, Δz, which contain the alerts and login attempts that occurred during the time-step. The metrics are collected by parsing the output of the commands in Table 3. For the evaluation reported in this paper, we collected measurements from 11000 time-steps.

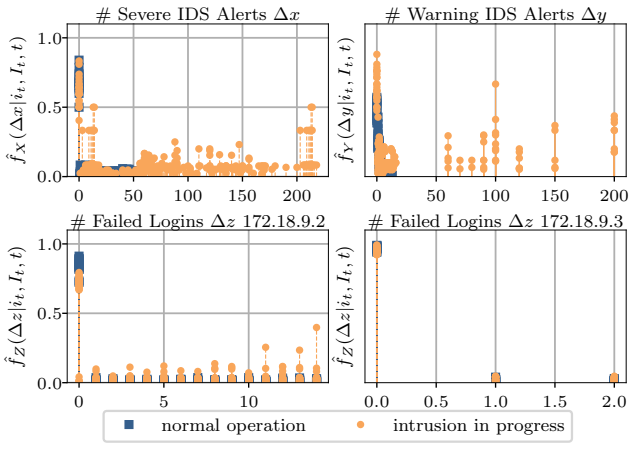

Fig. 4: Empirical distributions of IDS alerts (top row) and login attempts on two servers (bottom row); the graphs include several distributions that are superimposed.

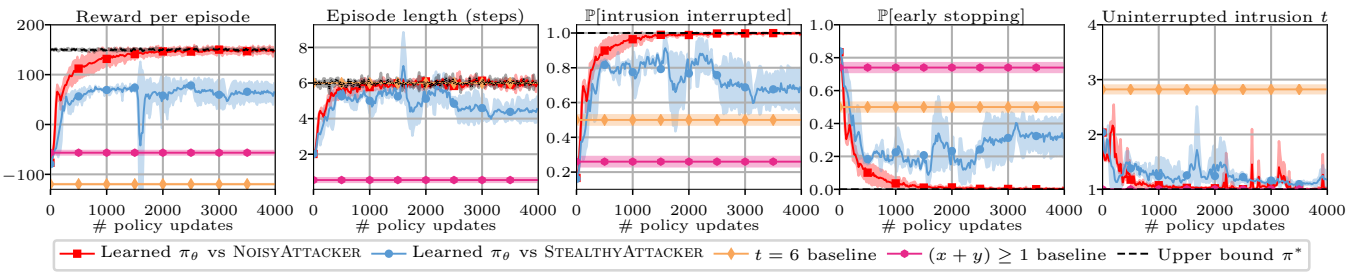

Fig. 5: Learning curves; the graphs show from left to right: episodic reward, length of an episode, empirical detection probability, empirical early stopping probability, and the number of steps between the start of an intrusion and the stop action; the curves show the averages and the standard deviations of three training runs with different random seeds.

2) Estimating the Distributions of Alerts and Login Attempts of the Target Infrastructure: Using the collected measurements, we compute the empirical distribution $\hat{f}_{XYZ}$, which is our estimate of the corresponding distribution f_{XYZ} in the target infrastructure. For each (I_t, t) pair, we obtain one empirical distribution.

Fig. 4 shows some of these distributions, which are superimposed. Although the distribution patterns generated during an intrusion and during normal operation overlap, there is a clear difference.

C. Simulating Episodes of the POMDP

During a simulation of the POMDP, the system state evolves according the dynamics described in Section IV and The observations evolve according to the estimated distribution $\hat{f}_{XYZ}$. In the initial state, no intrusion occurs. In every episode, either the defender stops before the intrusion starts or exactly one intrusion occurs, the start of which is determined by a Bernoulli process (see Section IV).

A simulated episode evolves as follows. During each time-step, if an intrusion is ongoing, the attacker executes an action in the predefined sequence listed in Table 2. Subsequently, the defender samples an action from the defender policy π_θ. If the action is stop, the episode ends. Otherwise, the simulation samples the number of alerts and login attempts occurring during this time-step from the empirical distribution $\hat{f}_{XYZ}$. It then computes the reward of the defender using the reward function defined in Section IV. The activities of the clients are not explicitly simulated but are implicitly represented in $\hat{f}_{XYZ}$. The sequence of time-steps continues until the defender stops or an intrusion completes, after which the episode ends.

VI. Learning Intrusion Prevention Policies using Simulation

To evaluate our reinforcement learning approach for finding defender policies, we simulate episodes of the POMDP where the defender policy is updated and evaluated. We evaluate the approach with respect to the convergence of policies and compare the learned policies to two baselines and to an ideal policy which presumes knowledge of the exact time of intrusion.

The evaluation is conducted using a Tesla P100 GPU and the hyperparameters for the learning algorithm are listed in Appendix A. Our implementation as well as the measurements for the results reported in this paper are publicly available [37].

A. Evaluation Setup

We train two defender policies against NoisyAttacker and StealthyAttacker until convergence, which occurs after some 400 iterations. In each iteration, we simulate 4000 time-steps and perform 10 updates to the policy. After each iteration, we evaluate the defender policy by simulating 200 evaluation episodes and compute various performance metrics.

We compare the learned policies with two baselines. The first baseline is a policy that always stops at $t = 6$, which corresponds to the expected time of intrusion $\mathbb{E}[I_t] = 6$ (see Section IV). The policy of the second baseline always stops after the first IDS alert occurs, i.e. $(x + y) \geq 1$.

To evaluate the stability of the learning curves' convergence, we run each training process three times with different random seeds. One training run requires approximately six hours of processing time on a P100 GPU.

B. Analyzing the Results

The red curves in Fig. 5 show the performance of the learned policy against NoisyAttacker, and the blue curves show the policy performance against StealthyAttacker. The purple and the orange curves give the performance of the two baseline policies. The dashed black curves give an upper bound on the performance of the optimal policy π^*, which is computed assuming that the defender knows the exact time of intrusion.

The five graphs in Fig. 5 show that the learned policies converge, and that they are close to optimal in terms of achieving maximum reward, detecting intrusion, avoid stopping when there is no intrusion, and stopping right after an intrusion starts. Further, the learned policies outperform both baselines by a large margin (leftmost graph in Fig. 5).

The performance of the learned policy against NoisyAttacker is better than that against StealthyAttacker (leftmost graph in Fig. 5). This indicates that NoisyAttacker is easier to detect for the defender. For instance, the learned policy against StealthyAttacker has a higher probability of stopping early (second rightmost graph of Fig. 5). This

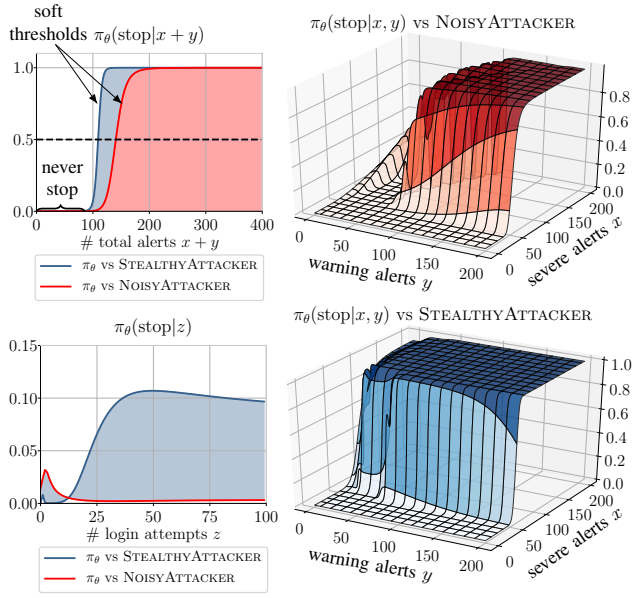

Fig. 6: Probability of the stop action by the learned policies in function of the number of alerts x, y and login attempts z.

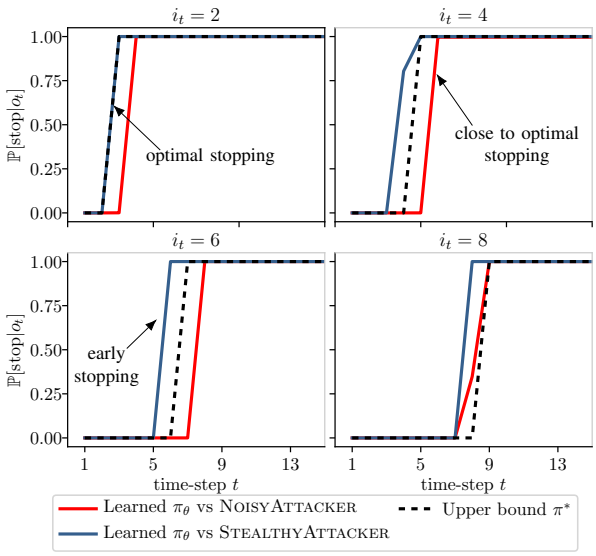

Fig. 7: Comparison of the learned policies π_θ and an upper bound on the optimal policy π^* for 4 sample episodes where the intrusions start at $i_t = 2, 4, 6, 8$.

can also be seen in the second leftmost graph of Fig. 5, which shows that, on average, the learned policy against StealthyAttacker stops after 5 time-steps and the learned policy against NoisyAttacker stops after 6 time-steps.

Looking at the baseline policies, we can see that the baseline $t = 6$ stops too early in 50 percent of the episodes, and the $(x + y) \geq 1$ baseline stops too early in 80 percent of the episodes (second rightmost graph in Fig. 5).

VII. Threshold Properties of the Learned Policies and Comparison with the Optimal Policy

When analyzing the learned policies, we find that they can be expressed through thresholds, just like the optimal policy (Section IV-C). However, in contrast to the optimal policy, the learned thresholds are based on the observed counters of alerts and login attempts rather than the belief state (which is unknown to the defender). Specifically, the top left graph in Fig. 6 shows that the learned policies against both attackers implement a soft threshold by stopping with high probability if the number of alerts $(x_t + y_t)$ exceeds 130. This indicates that if the total number of alerts is above 130, an intrusion has started, i.e. $x_t + y_t$ is used to approximate the posterior $b_t(1)$.

Moreover, the graphs in Fig. 6 also show the relative importance of severe alerts x_t, warning alerts y_t, and login attempts z_t for policy decisions. Specifically, it can be seen that x_t has the highest importance, y_t has a lower importance, and z_t has the least importance for policy decisions.

We also see that the learned policy against NoisyAttacker is associated with a higher alert threshold than that of StealthyAttacker (top left graph in Fig. 6). This is consistent with our comment in Section V-A2 that StealthyAttacker is harder to detect.

Lastly, Fig. 7 suggest that the thresholds of the learned policies are indeed close to the threshold of the optimal policy. For instance, the policy learned against NoisyAttacker stops immediately after the optimal stopping time.

VIII. Related Work

The problem of automatically finding security policies has been studied using concepts and methods from different fields, most notably reinforcement learning [15], game theory [4], dynamic programming [8], control theory [7], attack graphs [2], [9], statistical tests [6], and evolutionary computation [3].

Most research on reinforcement learning applied to network security is recent. Prior work that most resembles the approach taken in this paper includes our previous research [10] and the work in [11], [21], [20], and [19]. All of which study problems related to network intrusions using reinforcement learning.

This paper differs from prior work in the following ways: (1) we formulate intrusion prevention as an optimal stopping problem ([19] uses a similar formulation); (2) we use an emulated infrastructure to estimate the parameters of our simulation model, rather than relying on abstract assumptions like [10], [11], [20], [19]; (3) we derive a structural property of the optimal policy; (4) we analyze the learned policies and relate them to the optimal policy, an analysis which prior work lacks [10], [11], [20]; and (5) we apply state-of-the-art reinforcement learning algorithms, i.e. PPO [29], rather than traditional ones as used in [11], [20], [19], [21].

IX. Conclusion and Future Work

In this paper, we proposed a novel formulation of the intrusion prevention problem as one of optimal stopping. This allowed us to state that the optimal defender policy can be

Parameters	*Values*
γ, lr α, batch, # layers, # neurons, clip ϵ	1, 0.0005, 4000, 3, 64, 0.2
$X_{max}, Y_{max}, Z_{max}$, GAE λ, ent-coef	1000, 1000, 1000, 0.95, 0.0005

TABLE 4: Hyperparameters of the learning algorithm.

ID (s)	*OS:Services:Exploitable Vulnerabilities*
1	Ubuntu20:Snort(community ruleset v2.9.17.1),SSH:-
2	Ubuntu20:SSH,HTTP Erl-Pengine,DNS:SSH-pw
4	Ubuntu20:HTTP Flask,Telnet,SSH:Telnet-pw
10	Ubuntu20:FTP,MongoDB,SMTP,Tomcat,Teamspeak3,SSH:FTP-pw
12	Jessie:Teamspeak3,Tomcat,SSH:CVE-2010-0426,SSH-pw
17	Wheezy:Apache2,SNMP,SSH:CVE-2014-6271
18	Deb9.2:IRC,Apache2,SSH:SQL Injection
22	Jessie:PROFTPD,SSH,Apache2,SNMP:CVE-2015-3306
23	Jessie:Apache2,SMTP,SSH:CVE-2016-10033
24	Jessie:SSH:CVE-2015-5602,SSH-pw
25	Jessie: Elasticsearch,Apache2,SSH,SNMP:CVE-2015-1427
27	Jessie:Samba,NTP,SSH:CVE-2017-7494
3,11,5-9	Ubuntu20:SSH,SNMP,PostgreSQL,NTP:-
13-16,19-21,26,28-31	Ubuntu20:NTP, IRC, SNMP, SSH, PostgreSQL:-

TABLE 5: Configuration of the target infrastructure (Fig. 1).

expressed using a threshold obtained from infrastructure measurements. Further, we used reinforcement learning to estimate the optimal defender policies in a simulation environment. In addition to validating the predictions from the theory, we learned from the simulations a) the relative importance of measurement metrics with respect to the threshold level and b) that different attacker profiles can lead to different thresholds of the defender policies.

We plan to extend this work in three directions. First, the model of the defender in this paper is simplistic as it allows only for a single stop action. We plan to increase the set of actions that the defender can take to better reflect today's defense capabilities, while still keeping the structure of the stopping formulation. Second, we plan to extend the observation capabilities of the defender to obtain more realistic policies. Third, in the current paper, the attacker policy is static. We plan to extend the model to include a dynamic attacker that can learn just like the defender. This requires a game-theoretic formulation of the problem.

APPENDIX

A. Hyperparameters: Table 4

B. Configuration of the Infrastructure in Fig. 1: Table 5

REFERENCES

[1] A. Fuchsberger, "Intrusion detection systems and intrusion prevention systems," *Inf. Secur. Tech. Rep.*, vol. 10, no. 3, p. 134–139, Jan. 2005.

[2] P. Johnson, R. Lagerström, and M. Ekstedt, "A meta language for threat modeling and attack simulations," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018, New York, NY, USA, 2018.

[3] R. Bronfman-Nadas, N. Zincir-Heywood, and J. T. Jacobs, "An artificial arms race: Could it improve mobile malware detectors?" in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, 2018.

[4] T. Alpcan and T. Basar, *Network Security: A Decision and Game-Theoretic Approach*, 1st ed. USA: Cambridge University Press, 2010.

[5] S. Sarıtaş, E. Shereen, H. Sandberg, and G. Dán, "Adversarial attacks on continuous authentication security: A dynamic game approach," in *Decision and Game Theory for Security*, Cham, 2019, pp. 439–458.

[6] A. G. Tartakovsky, B. L. Rozovskii, R. B. Blažek, and H. Kim, "Detection of intrusions in information systems by sequential change-point methods," *Statistical Methodology*, vol. 3, no. 3, 2006.

[7] W. Liu and S. Zhong, "Web malware spread modelling and optimal control strategies," *Scientific Reports*, vol. 7, p. 42308, 02 2017.

[8] M. Rasouli, E. Miehling, and D. Teneketzis, "A supervisory control approach to dynamic cyber-security," in *Decision and Game Theory for Security*. Cham: Springer International Publishing, 2014, pp. 99–117.

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

[10] K. Hammar and R. Stadler, "Finding effective security strategies through reinforcement learning and Self-Play," in *International Conference on Network and Service Management (CNSM 2020)*, Izmir, Turkey, 2020.

[11] R. Elderman, L. J. J. Pater, A. S. Thie, M. M. Drugan, and M. Wiering, "Adversarial reinforcement learning in a cyber security simulation," in *ICAART*, 2017.

[12] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Belmont, MA, USA: Athena Scientific, 2005, vol. I.

[13] V. Krishnamurthy, *Partially Observed Markov Decision Processes: From Filtering to Controlled Sensing*. Cambridge University Press, 2016.

[14] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming.* Belmont, MA: Athena Scientific, 1996.

[15] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *CoRR*, vol. abs/1906.05799, 2019.

[16] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[17] J. du Toit and G. Peskir, "Selling a stock at the ultimate maximum," *The Annals of Applied Probability*, vol. 19, no. 3, Jun 2009.

[18] A. Roy, V. S. Borkar, A. Karandikar, and P. Chaporkar, "Online reinforcement learning of optimal threshold policies for markov decision processes," *CoRR*, vol. abs/1912.10325, 2019.

[19] M. N. Kurt, O. Ogundijo, C. Li, and X. Wang, "Online cyber-attack detection in smart grid: A reinforcement learning approach," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5174–5185, 2019.

[20] F. M. Zennaro and L. Erdodi, "Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular q-learning," *CoRR*, vol. abs/2005.12632, 2020.

[21] J. Schwartz, H. Kurniawati, and E. El-Mahassni, "Pomdp + information-decay: Incorporating defender's behaviour in autonomous penetration testing," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, no. 1, pp. 235–243, Jun. 2020.

[22] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[23] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley and Sons, Inc., 1994.

[24] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[25] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," USA, 1996.

[26] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[27] E. J. Sondik, "The optimal control of partially observable markov processes over the infinite horizon: Discounted costs," *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978.

[28] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, 1989.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, 2017.

[30] A. Wald, *Sequential Analysis*. Wiley and Sons, New York, 1947.

[31] A. N. Shirayev, *Optimal Stopping Rules*. Springer-Verlag Berlin, 2007, reprint of russian edition from 1969.

[32] G. Peskir and A. Shiryaev, *Optimal stopping and free-boundary problems*, ser. Lectures in mathematics (ETH Zürich). Springer, 2006.

[33] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, international Conference for Learning Representations, San Diego.

[35] K. Hammar and R. Stadler, "Learning intrusion prevention policies through optimal stopping," https://arxiv.org/pdf/2106.07160.pdf,2021.

[36] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[37] K. Hammar and R. Stadler, "gym-optimal-intrusion-response," 2021, https://github.com/Limmen/gym-optimal-intrusion-response.