An Online Framework for Adapting Security Policies in Dynamic IT Environments

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
September 8, 2022

***Abstract*—We present an online framework for learning and updating security policies in dynamic IT environments. It includes three components: a digital twin of the target system, which continuously collects data and evaluates learned policies; a system identification process, which periodically estimates system models based on the collected data; and a policy learning process that is based on reinforcement learning. To evaluate our framework, we apply it to an intrusion prevention use case that involves a dynamic IT infrastructure. Our results demonstrate that the framework automatically adapts security policies to changes in the IT infrastructure and that it outperforms a state-of-the-art method.**

***Index Terms*—Network security, security management, digital twin, reinforcement learning, Markov decision process, MDP, POMDP.**

I. Introduction

A promising direction of recent research is obtaining security policies through reinforcement learning methods (see survey [1]). In this line of investigation, the problem of finding a security policy is generally modeled as a Markov decision problem, and policies are learned through simulation. While encouraging results have been obtained following this approach [2], [3], [2], [4], [5], [6], key challenges remain [7]. One of them is to narrow the gap between the environment where a policy is learned and the real system, where the policy is applied. Another challenge is to adapt a policy to a changing IT environment. Research to date has focused on stationary environments, which limits the applicability of the results.

In operational IT environments, various changes occur on a continuous basis. Components may fail, load patterns can shift, bandwidth can fluctuate, and components may be updated, for example. When such changes occur, security policies need to be adapted. To address this problem, we present an *online* framework that periodically updates security policies through *reinforcement learning*. We show that our framework automatically adapts security policies to changes in IT environments and that it outperforms state-of-the-art methods, which either are limited to simulations [2], [3], [2], [4], [5], [6] or to stationary environments [8], [9], [10], [11], [12].

II. Example Use Case: Intrusion Prevention

We develop and evaluate our framework considering an *intrusion prevention* use case that involves an IT infrastructure of an organization (see Fig. 1). The operator of this infrastructure, which we call the defender, takes measures to protect it against an attacker while providing services to a client population. The infrastructure includes a set of servers that run the services and an Intrusion Prevention System (IPS) that logs events in real-time. Clients access the services through a public gateway, which also is open to the attacker.

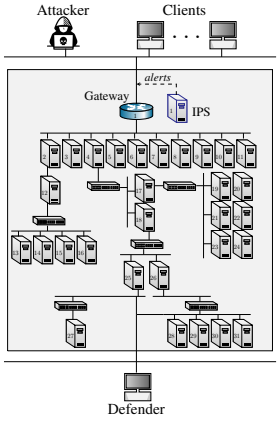

Fig. 1: The IT infrastructure and the actors in the use case.

The number of clients changes over time, which makes the infrastructure *dynamic*. At certain times, the infrastructure is under low load, and at other times, the number of clients spikes, which causes a high load on the infrastructure.

The attacker's goal is to intrude on the infrastructure and compromise its servers. To achieve this, the attacker explores the infrastructure through reconnaissance and exploits its vulnerabilities while avoiding detection.

The defender continuously monitors the infrastructure through analyzing IPS alerts and other statistics. It can take a limited number of defensive actions, each of which has a cost and a chance of preventing an intrusion. An example of a defensive action is to drop network traffic that triggers IPS alerts of a certain priority.

When deciding the time for taking a defensive action, the defender balances two objectives: (*i*) maintain services to its clients; and (*ii*), prevent an intrusion at lowest cost. The optimal policy for the defender is to monitor the infrastructure

978-3-903176-51-5 © 2022 IFIP

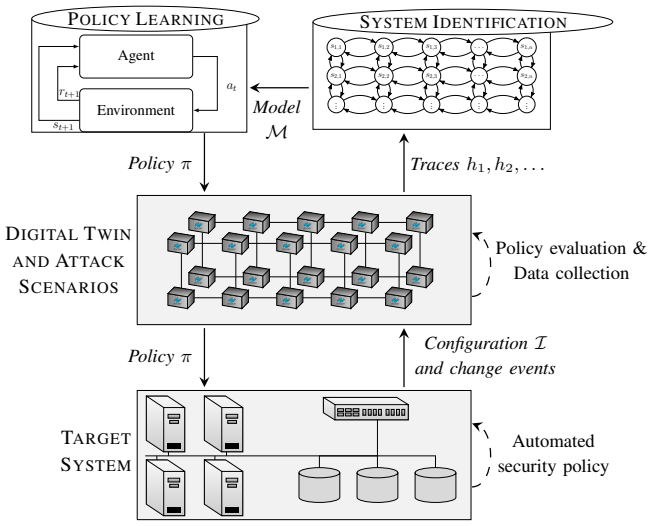

Fig. 2: Our online framework for learning and adapting security policies.

and maintain services until the attacker enters through the gateway, at which time the intrusion must be prevented at minimal cost through defensive actions. The challenge for the defender thus is to identify the precise time when an intrusion starts. In the following, we describe a general framework that addresses this challenge and learns optimal defender policies.

III. Our Framework for Learning Security Policies in Dynamic IT Environments

Our framework for learning security policies for a dynamic target system includes three main parts (see Fig. 2). First, a *digital twin* emulates the target system and its evolution in a virtualized environment. This environment is used to run attack scenarios and defender responses, from which traces with system statistics and logs are obtained. Second, a *system identification* algorithm estimates a model of the target system based on the collected traces, e.g. a dynamical system model, such as a Markov decision process. Third, the estimated model is simulated and the defender policy is obtained through *reinforcement learning*.

Digital twin. The digital twin collects traces and evaluates learned policies. It emulates the target system and is periodically updated when changes to the target system are detected. To evaluate a defender policy π, it runs attack scenarios and executes defender responses prescribed by π. If a policy π is deemed effective, the digital twin sends it to the target system for implementation. From each such evaluation, a trace of defender actions, system observations, and rewards $h = o_1, a_2, r_2, \ldots, a_{T-1}, r_T, o_T$ is obtained. The traces are then sent to the system identification process.

System identification process. The system identification process periodically receives traces h_t from the digital twin and runs a system identification algorithm φ, which learns a system model $\mathcal{M}$ based on h_t. After obtaining $\mathcal{M}$, the system identification process sends $\mathcal{M}$ to the policy learning process.

Algorithm 1: High-level execution of the framework

Input: $emulator$: method to create digital twin
φ: system identification algorithm
ϕ: policy learning algorithm

Algorithm $(emulator, \varphi, \phi)$
  do in parallel
    DigitalTwin($emulator$)
    SystemIdProcess(φ)
    LearningProcess(ϕ)
  end
Procedure DigitalTwin($emulator$)
  Loop
    $\pi \leftarrow$ ReceiveFromLearningProcess()
    $h_t \leftarrow$ CollectTrace(π)
    SendToSystemIdProcess(h_t)
    UpdateDigitalTwin($emulator$)
  EndLoop
Procedure SystemIdProcess(φ)
  Loop
    $h_1, h_2, \ldots \leftarrow$ ReceiveFromDigitalTwin()
    $\mathcal{M} \leftarrow \varphi(h_1, h_2, \ldots)$ // estimate model
    SendToLearningProcess($\mathcal{M}$)
  EndLoop
Procedure LearningProcess(ϕ)
  Loop
    $\mathcal{M} \leftarrow$ ReceiveFromSystemIdProcess()
    $\pi \leftarrow \phi(\mathcal{M})$ // learn policy π
    SendToDigitalTwin(π)
  EndLoop

Policy learning process. The policy learning process periodically receives a system model $\mathcal{M}$ from the system identification process, learns an effective policy π through simulating $\mathcal{M}$ and running the reinforcement learning algorithm ϕ, and then sends π to the digital twin for evaluation.

The pseudocode underlying our framework is given in Algorithm 1. A video demonstration of the framework is available at [13]. The framework can be customized by changing the method to create the digital twin of the target system, the system model $\mathcal{M}$, the system identification algorithm φ, and the reinforcement learning algorithm ϕ.

Some remarks on the implementation of the framework: To synchronize traces, system models, and policies between the digital twin, the policy learning process, and the system identification process, we have implemented a parameter-server architecture, through which the processes communicate in an asynchronous fashion. The parameter server maintains the latest policy, the latest system model, and a buffer with traces collected from the digital twin. When estimating a system model, a sliding window of the last K traces is used (in this paper $K = 50$). To speed up the collection of traces and the evaluation of policies, we run N digital twins in parallel

Actions
Continue, Revoke user certificates, Blacklist IPs, Block gateway
Drop traffic that generates IPS alerts of priority $k \in \{1, 2, 3, 4\}$

TABLE 1: Defender actions to emulate intrusion responses.

(in this paper $N = 5$).

IV. Instantiating our Framework for the Intrusion Prevention Use Case

In this section, we instantiate our framework for the intrusion prevention use case. We first describe our method for creating a digital twin of the target system shown in Fig. 1. We then formalize the use case and define our system model $\mathcal{M}$ together with the system identification algorithm φ, which instantiates $\mathcal{M}$ based on traces from the digital twin. Lastly, we describe our algorithm ϕ for learning the defender policy.

A. Creating the Digital Twin of the Target System

Emulating physical hosts. We emulate physical hosts of the target system using Docker containers. Resource constraints on the containers, e.g. CPU and memory constraints, are enforced using cgroups. The software functions running inside the containers are copies of key components of the target system, such as, web servers, databases, and the Snort IPS.

Emulating network connectivity. We emulate network connectivity between hosts using virtual links. Connections between servers in the target system are emulated as full-duplex lossless connections with capacity 1 Gbit/s. External connections between the gateway and the client population are emulated full-duplex connections of 100 Mbit/s capacity and 0.1% packet loss. (These numbers are drawn from empirical studies on enterprise and wide area networks [12].)

Emulating the client population. We emulate the client population by processes inside Docker containers that interact with the emulated hosts. Client arrivals follow a sine-modulated Poisson process having the service rate function $\lambda(t) = \bar{\lambda} + \alpha \sin(\frac{2\pi t}{\upsilon})$ and exponentially distributed service time with $\mu = \frac{1}{4}$ ($\bar{\lambda} = 20$, $\alpha = 20$, and $\upsilon = 800$).

Emulating defender and attacker actions. We emulate defender and attacker actions by executing the commands listed in Table 1 and Table 2, respectively. At every time-step, the defender takes an action from the list in Table 1, which is determined by the defender policy π. Once an intrusion has started, the attacker takes actions which are sampled uniformly at random from the list in Table 2.

B. System Model $\mathcal{M}$ of the Use Case

We model the intrusion prevention use case with a Partially Observed Markov Decision Process (POMDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}^{a_t}_{s_t,s_{t+1}}, \mathcal{R}^{a_t}_{s_t,s_{t+1}}, \gamma, \rho_1, T, \mathcal{O}, \mathcal{Z} \rangle$ [10], [2].

Time progression. Time evolves in discrete time-steps: $t = 1, \ldots, T$, which constitute an *episode*. The duration of a time-step in the digital twin is 30 seconds.

Type	*Actions*
Reconnaissance	TCP-SYN scan, UDP port scan, TCP Null scan, TCP Xmas scan, TCP FIN scan, ping-scan, "Vulscan", TCP connection scan,
Brute-force attack	Telnet, SSH, FTP, Cassandra, IRC, MongoDB, MySQL, SMTP, Postgres
Exploit	CVE-2017-7494, CVE-2015-3306, CVE-2010-0426, CVE-2015-5602, CVE-2014-6271 CVE-2016-10033, CVE-2015-1427, SQL Injection

TABLE 2: Attacker commands to emulate intrusions.

Attacker model. The attacker starts the intrusion at a random time during an episode and then follows a predefined strategy until the intrusion is prevented.

State space $\mathcal{S}$. At time-step t, the state s_t is 0 if no intrusion occurs, it is 1 if an intrusion is ongoing, and it is $\emptyset$ if the episode has ended. Hence, $\mathcal{S} = \{0, 1, \emptyset\}$.

Initial state distribution ρ_1. The initial state distribution is the degenerate distribution $\rho_1(0) = 1$.

Action space $\mathcal{A}$. At time-step t, the defender either decides to *continue* to monitor the infrastructure ($a_t = C$), or, it decides to take a *stop* action ($a_t = S$), which corresponds to a defensive action against a possible intrusion (see Table 1). The action space thus is $\mathcal{A} = \{C, S\}$. A stop action can be invoked $L \geq 1$ times. The number of stop actions remaining at time-step t is denoted by $l_t \in \{1, \ldots, L\}$.

Transition probabilities $\mathcal{P}^{a_t}_{s_t,s_{t+1},l_t}$. If $s_t = 0$, the attacker starts an intrusion with probability p. In case of an intrusion, the state transition $0 \to 1$ occurs. Similarly, if the defender takes a defensive action when $s_t = 1$, the intrusion is prevented with probability ψ_{l_t}. In case the intrusion is prevented, the state transition $1 \to \emptyset$ occurs. Hence, the transition probabilities $\mathcal{P}^{a_t}_{s_t,s_{t+1},l_t} = \mathbb{P}_{l_t}[s_{t+1}|s_t, a_t]$ are:

$$\mathbb{P}_1[\emptyset|\cdot, 1] = \mathbb{P}_{l_t}[\emptyset|\emptyset, \cdot] = 1 \tag{1}$$
$$\mathbb{P}_{l_t}[0|0, a_t] = 1 - p \quad \text{if } l_t - \alpha > 0 \tag{2}$$
$$\mathbb{P}_{l_t}[1|0, a_t] = p \quad \text{if } l_t - \alpha > 0 \tag{3}$$
$$\mathbb{P}_{l_t}[1|1, a_t] = 1 - \psi_{l_t}\alpha \quad \text{if } l_t - \alpha > 0 \tag{4}$$
$$\mathbb{P}_{l_t}[\emptyset|1, a_t] = \psi_{l_t}\alpha \quad \text{if } l_t - \alpha > 0 \tag{5}$$

where $\alpha = \mathbb{1}_{\{a_t = S\}}$ ($\mathbb{1}$ denotes the indicator function).

Observation space $\mathcal{O}$. The state s_t is hidden from the defender, who instead observes $o_t \in \mathcal{O} = \{0, \ldots, 10^4\}$, which refers to the sum of the number of IPS alerts during time-step t weighted by priority.

Observation function $\mathcal{Z}_{t,O}$. The observation o_t is drawn from a random variable O whose distribution $\mathcal{Z}_{t,O}$ depends on the current state s_t and is *dynamic*, i.e depends on t.

Belief space $\mathcal{B}$. Based on the observations, the defender forms a belief about s_t, which is expressed in the *belief state* $b_t(s_t) = \mathbb{P}[s_t|h^{(1)}_t] \in \mathcal{B}$. $\mathcal{B}$ is the unit $(|\mathcal{S}| - 1)$-simplex. Since $s_t \in \{0, 1\}$ and $b_t(0) = 1 - b_t(1)$, b_t is determined by $b_t(1)$. Hence, we can model $\mathcal{B} = [0, 1]$.

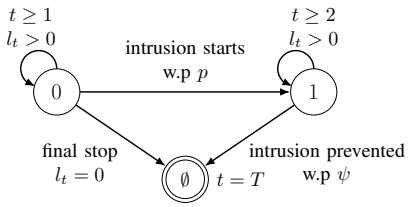

Fig. 3: State transition diagram of the POMDP $\mathcal{M}$: each circle represents a state; an arrow represents a state transition; a label indicates the event that triggers the state transition (w.p means "with probability"); an episode starts in state $s_1 = 0$ with $l_1 = L$ and it ends when $l_t = 0$ or an intrusion is prevented.

Defender policy π_l. A defender policy $\pi_l : \mathcal{B} \rightarrow \Delta(\mathcal{A})$ maps a belief state to a probability distribution over $\mathcal{A}$. ($\Delta(\mathcal{A})$ denotes the set of probability distributions over $\mathcal{A}$.)

Time horizon T. Fig. 3 depicts the state transition diagram of $\mathcal{M}$. The diagram describes an *episode* of $\mathcal{M}$, which starts at $t = 1$ and ends at $t = T$. The time horizon T is a random variable that indicates the time t when the terminal state $\emptyset$ is reached. It follows from Eqs. 1-5 that $\mathbb{E}_{\pi_l}[T] < \infty$ for any policy π_l that is guaranteed to use L stop actions as $t \rightarrow \infty$. ($\mathbb{E}_{\pi}$ denotes the expectation under π.)

Reward function $\mathcal{R}^{a_t}_{s_t,l_t}$. The reward function is parameterized by the reward that the defender receives for stopping an intrusion $R_{st} > 0$, the loss of being intruded $R_{int} < 0$, and the cost $R_{cost} < 0$ of taking a defensive action:

$$\mathcal{R}_{\emptyset,0} = 0 \tag{6}$$

$$\mathcal{R}^{S}_{s_t,l_t} = R_{cost}/l_t + s_t R_{st}/l_t \qquad s_t \in \{0,1\} \tag{7}$$

$$\mathcal{R}^{C}_{s_t,l_t} = s_t R_{int} \qquad s_t \in \{0,1\} \tag{8}$$

Objective function J. The optimal defender policy π^*_l maximizes the expected reward over the time-horizon T:

$$\pi^*_l \in \arg\max_{\pi_l \in \Pi_l} J(\pi_l), \quad J(\pi_l) = \mathbb{E}_{\pi_l}\left[\sum_{t=1}^{T} \gamma^{t-1} \mathcal{R}^{a_t}_{s_t,l_t}\right] \tag{9}$$

(For a theoretical analysis of our model and its connection with optimal stopping theory, see [10], [2].)

C. System Identification Algorithm φ to Instantiate $\mathcal{M}$

The system identification algorithm φ instantiates $\mathcal{M}$ by estimating the observation function $\mathcal{Z}_{t,O}$ based on traces collected from the digital twin. We implement φ by fitting a Gaussian mixture model $\hat{\mathcal{Z}}_{t,O}$ over the discrete observation space $\mathcal{O}$ of $\mathcal{M}$. For both values of s, we obtain the conditional distribution $\hat{\mathcal{Z}}_{t,O|s}$ through expectation-maximization.

D. Policy Learning Algorithm ϕ to Learn Defender Policies

The policy learning process in Algorithm 1 learns defender policies for $\mathcal{M}$ through the reinforcement learning algorithm ϕ. In our previous work, we proved that, at each time-step t, there exists an optimal defender policy π^*_t that uses L thresholds $\alpha^*_{1,t} \geq \alpha^*_{2,t}, \ldots, \alpha^*_{L,t} \in \mathcal{B}$ [10, Theorem 1]. For this reason, we implement ϕ as follows.

We parameterize the defender policy $\pi_{t,l}$ with a parameter vector $\theta \in \mathbb{R}^L$ and define $\pi_{\theta,t,l}$ to be a smooth stochastic policy that approximates a threshold policy:

$$\pi_{\theta,t,l}\big(S|b_t(1)\big) = \left(1 + \left(\frac{b_t(1)(1-\sigma(\theta_l))}{\sigma(\theta_l)(1-b_t(1))}\right)^{-20}\right)^{-1} \tag{10}$$

where $\sigma(\cdot)$ is the sigmoid function and $\sigma(\theta_1)$, $\sigma(\theta_2)$, …, $\sigma(\theta_L) \in \mathcal{B}$ are the L thresholds.

We then simulate $\mathcal{M}$ for a given number of episodes where the defender takes actions according to $\pi_{\theta,t,l}$. Then, we use the episode outcomes and trajectories to estimate the gradient $\nabla_\theta J(\theta)$ (see Eq. 9). Next, we use the estimated gradient and stochastic gradient ascent to update θ. This process of simulating episodes and updating θ continues until $\pi_{\theta,t,l}$ has sufficiently converged. The pseudocode is available at [10].

V. Learning Security Policies for a Dynamic IT Infrastructure

We evaluate our framework by using it to obtain defender policies for the intrusion prevention use case described in Section II. The results are shown in Fig. 4.

Evaluation setup. The topology of the target system is shown in Fig. 1 and its configuration is listed in [10][Table 6]. We run the framework (Algorithm 1) for 50 hours. Our framework updates the system model and the defender policy every 10 minutes. The execution environment is a server with two 24-core Intel Xeon Gold 2.10 GHz CPUs with hyperthreading, 768 GB RAM, 4 NVIDIA Quadro RTX8000 GPUs, and Ubuntu 20.04. The hyperparameters can be found in [10].

Baseline algorithms. We compare the performance of the policies obtained with our framework to two baselines. The first baseline is obtained with an ideal policy which presumes knowledge of the exact intrusion time. The second baseline uses the method proposed in [10], which recently achieved state-of-the-art results for the same intrusion prevention use case but for a stationary environment. Similar to the proposed framework, [10] uses emulation, system identification, and reinforcement learning. The main difference between the proposed framework and [10] is that the latter assumes a stationary environment and does not update the defender policy periodically.

Discussion of the evaluation results. The upper plot in Fig. 4 shows how the number of clients evolves, the middle plot shows how the mean values of the estimated IPS alert distributions change over time (see Section IV-C), and the lower plot shows how the performance of the learned policies evolves (see Section IV-D).

Fig. 4 contains the evaluation results. First, we observe that the load on the system is dynamic and that the number of clients follows a sinusoidal pattern (upper plot in Fig. 4). Second, we note that the distributions of IPS alerts estimated by our framework change over time and are correlated with the number of clients (red and blue curves in the middle plot of Fig. 4). Third, we see in the lower plot of Fig. 4 that the pink

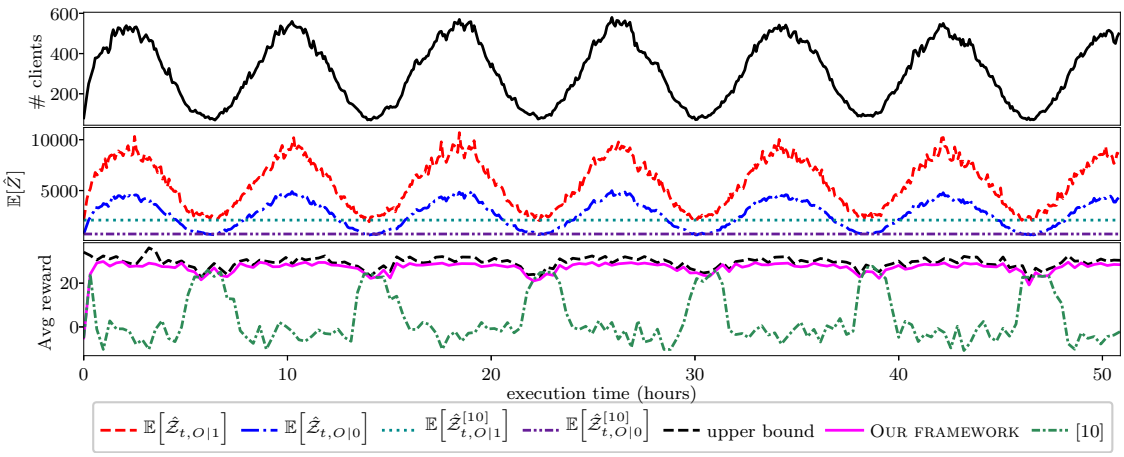

Fig. 4: Results from running our framework for 50 hours; the top plot shows the evolution of the number of clients; the middle plot shows the evolution of the mean values of the estimated distributions of IPS alerts, where $\mathbb{E}[\hat{\mathcal{Z}}_{t,O|s}]$ relate to our framework, $\mathbb{E}[\hat{\mathcal{Z}}^{[10]}_{t,O|s}]$ relate to the baseline [10], $s = 0$ indicate no intrusion, and $s = 1$ indicate intrusion; the lower plot shows the average episode reward of policies evaluated in the digital twin; the pink, green, and black curves relate to our framework, the baseline from [10], and the upper bound, respectively.

curve, which relates to our framework, is close to the dashed black curve, which gives an upper bound to any optimal policy. This suggests to us that our framework learns near-optimal policies that are adapted to the changing alert distribution. Lastly, we observe that the performance of the policy learned by the baseline method in [10] degrades as the load on the system changes. This suggests to us that the method proposed in [10] is not suited for a dynamic IT environment.

VI. Related Work

Most of the prior work on finding security policies through reinforcement learning are simulation-based studies [2], [3], [2], [4], [5], [6]. Our framework, in contrast, is based on emulation of the target system. Prior work that use similar emulation-based approaches as us include [8], [9], [10], [11], [12]. Compared to these papers, the main difference is that our framework is explicitly designed for online learning of policies in dynamic environments. The referenced works either assume a stationary environment or do not explicitly evaluate their approach in dynamic environments.

VII. Conclusion and Future Work

We present an online framework aimed at learning security policies for dynamic IT environments. The framework involves a digital twin and several parallel processes that a) collect system measurements; b) estimate system models through system identification; and c) learn policies through reinforcement learning. We apply the framework to an intrusion prevention use case and show that it dynamically adapts policies to changes in an IT environment and that it outperforms a state-of-the-art method for static environments.

In future work, we plan to extend our framework to obtain policies that generalize to a variety of infrastructure configurations and load patterns. This can be achieved by estimating system models based on data from several digital twins with different configurations and by expanding the observation space of the defender. We also plan to evaluate our framework for different use cases.

References

[1] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *CoRR*, 2019, http://arxiv.org/abs/1906.05799.

[2] K. Hammar and R. Stadler, "Learning intrusion prevention policies through optimal stopping," in *International Conference on Network and Service Management (CNSM 2021)*, Izmir, Turkey, 2021.

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

[4] M. Kurt *et al.*, "Online cyber-attack detection in smart grid: A reinforcement learning approach," *IEEE Transactions on Smart Grid*, 2019.

[5] A. Ridley, "Machine learning for autonomous cyber defense," 2018, the Next Wave, Vol 22, No.1 2018.

[6] M. Zhu, Z. Hu, and P. Liu, "Reinforcement learning algorithms for adaptive cyber defense against heartbleed," in *Proceedings of the First ACM Workshop on Moving Target Defense*, ser. MTD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 51–58.

[7] G. Dulac-Arnold, D. J. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," 2019.

[8] I. Akbari *et al.*, "Atmos: Autonomous threat mitigation in sdn using reinforcement learning," in *NOMS IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[9] Y. Liu *et al.*, "Deep reinforcement learning based smart mitigation of ddos flooding in software-defined networks," in *IEEE CAMAD*, 2018.

[10] K. Hammar and R. Stadler, "Intrusion prevention through optimal stopping," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022, https://ieeexplore.ieee.org/document/9779345.

[11] ——, "A system for interactive examination of learned security policies," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, pp. 1–3.

[12] ——, "Learning security strategies through game play and optimal stopping," in *Proceedings of the ML4Cyber workshop, ICML 2022, Baltimore, USA, July 17-23, 2022*. PMLR, 2022.

[13] ——, "A software framework for building self-learning security systems," in *NOMS 2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, https://www.youtube.com/watch?v=18P7MjPKNDg.