Digital Twins for Security Automation

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

***Abstract*—We present a novel emulation system for creating high-fidelity digital twins of IT infrastructures. The digital twins replicate key functionality of the corresponding infrastructures and allow to play out security scenarios in a safe environment. We show that this capability can be used to automate the process of finding effective security policies for a target infrastructure. In our approach, a digital twin of the target infrastructure is used to run security scenarios and collect data. The collected data is then used to instantiate simulations of Markov decision processes and learn effective policies through reinforcement learning, whose performances are validated in the digital twin. This closed-loop learning process executes iteratively and provides continuously evolving and improving security policies. We apply our approach to an intrusion response scenario. Our results show that the digital twin provides the necessary evaluative feedback to learn near-optimal intrusion response policies.**

***Index Terms*—Digital twin, cybersecurity, network security, automation, reinforcement learning, BMDP, POMDP.**

I. Introduction

In the past few years, virtualization technologies have matured to the point that it is now feasible to deploy large virtual IT infrastructures on commodity hardware. Virtual infrastructures differ from physical ones in that they consist of lightweight virtual containers or virtual machines that enable a higher level of control by shifting functions from hardware to software. Building on this capability, *digital twin* has emerged as a key technology in system automation [1]. A digital twin is a virtual replica of a real-world system that provides a controlled environment for virtual operations, the outcomes of which can be used to optimize operations in the real-world system.

Digital twin has been adopted in several industry sectors, including the manufacturing industry (see example [1]), the automotive industry (see example [2]), and the healthcare industry (see example [3]). While digital twins have been adopted in all of these industries, it is only recently that it has gained traction in networking and security research, where example usages include: network planning, anomaly detection, and predictive analytics (see surveys [4]–[6]).

In this paper, we study a new use case of digital twin, namely: automating the process of finding effective security policies for IT infrastructures. Security policies have traditionally been defined by domain experts. Though 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. To address this challenge, significant efforts to automate the process of obtaining effective security policies is now under way within major IT vendors as well as throughout academia [7, §VII].

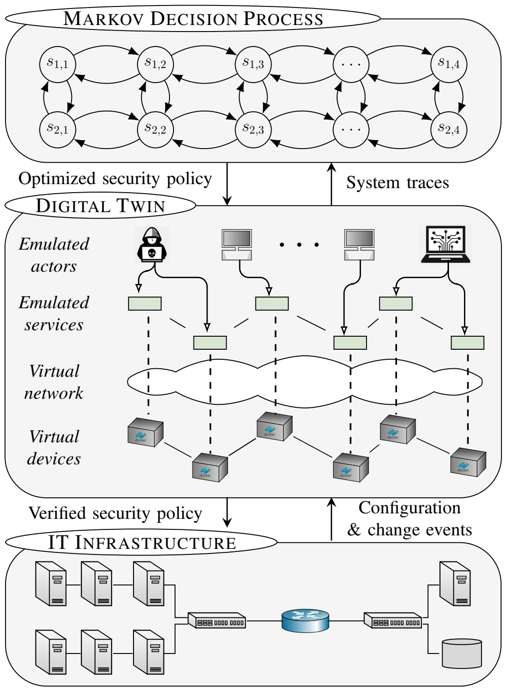


Fig. 1: Our approach to find and evaluate security policies for an IT infrastructure using a digital twin; the digital twin is a virtual replica of the IT infrastructure and is used to evaluate security policies and collect data; the collected data is used to instantiate simulations of Markov decision processes and to learn effective policies through reinforcement learning.

A promising direction of recent research is to learn security policies through reinforcement learning techniques [7]–[17]. While encouraging results have been obtained following this approach, key challenges remain. Chief among them is narrowing the gap between the environment where policies are evaluated and a scenario playing out in a real system. Most of the results obtained so far are limited to simulation environments, and it is not clear how they generalize to practical IT infrastructures.

In this work, we address the above challenge and present an approach for learning near-optimal security policies for an IT infrastructure that is centered around high-fidelity digital twins (see Fig. 1). Our approach includes the following steps. We first create a digital twin of the target infrastructure and use it to run attack scenarios and defender responses. Such runs produce system measurements and logs, from which we estimate infrastructure statistics. We then use the estimated statistics to instantiate simulations of Markov decision processes and learn near-optimal security policies, whose performance we assess in the digital twin. This closed-loop learning process executes iteratively and provides continuously evolving and improving security policies for the real-world IT infrastructure.

The digital twin provides three key functions for the approach described above: (*i*) it provides a safe and realistic test environment; (*ii*) it provides evaluative feedback that enables closed-loop learning of policies; and (*iii*), it allows collecting data and evaluating policies without affecting operational workflows on the real-world infrastructure.

We make two contributions with this paper. First, our emulation system for creating digital twins is novel and extends related works that have developed systems for creating digital twins for other networking use cases [6], [18]–[21]. Second, although a growing body of work investigates reinforcement learning as a way to learn security policies [7]–[17], our work is, to the best of our knowledge, the first study of high-fidelity digital twins in this context.

II. Emulation System for Creating Digital Twins

This section describes our emulation system for creating digital twins of IT infrastructures. We first give an overview of the system's architecture and implementation (§II-A). Then, we describe the process of creating a digital twin, which involves three main tasks of the emulation system. The first task is to replicate relevant parts of the physical infrastructure that is emulated, such as physical resources, network interfaces, and network conditions. This task is described in §II-B. The second task is to instrument the digital twin with monitoring and management capabilities. Since the digital twin will be used to evaluate security policies, it must be possible to monitor system metrics and perform control actions in real-time. We describe these capabilities in §II-C. The third task is to emulate security actors in the digital twin, a process which we describe in §II-D.

A. System Implementation and Architecture

The system takes infrastructure configurations as input and generates digital twins as output, which are deployed on a cluster of machines that runs a virtualization layer provided by Docker containers and virtual links. The set of configurations supported by the system can be seen as a *configuration space*, which defines the class of digital twins that can be created (see Fig. 2). An infrastructure configuration includes container specifications, service specifications, resource specifications, and network specifications. (A complete infrastructure configuration is available in [22].)

The system is implemented in Python, JavaScript, and Bash, and comprises around $160,000$ lines of code. It can be accessed in three ways: through Python libraries, through a web interface, and through a Command-Line Interface (CLI).

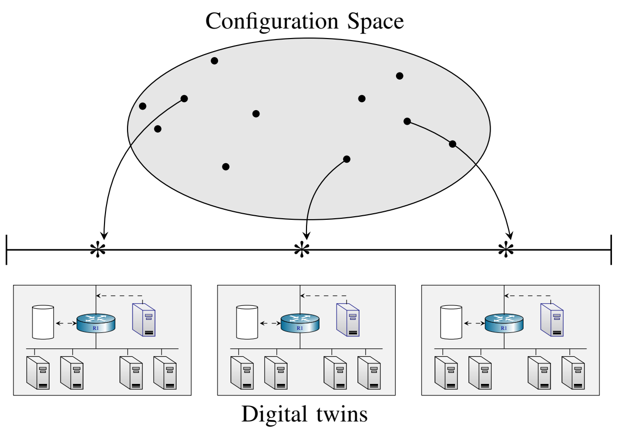


Fig. 2: The configuration space of the emulation system, which defines the set of digital twins that can be created.

The web interface is implemented in JavaScript and the CLI is implemented in Python.

It is a distributed system that consists of $N \geq 1$ physical servers connected through an IP network. It stores metadata in a distributed database referred to as the *metastore*, which is based on POSTGRES and CITUS [23] (see Fig. 3). The metastore consists of N replicas, one per physical server. To coordinate database updates and achieve consensus among replicas, a quorum-based two-phase commit scheme is used.

One of the servers is designated to be the "leader" and the others are "workers". Workers can execute local management actions but not actions that affect the overall system state. These actions are routed to the leader, which applies them sequentially to ensure consistent updates.

The leader is elected using the leader election protocol in [24], which uses the metastore for coordination. A new leader is elected by a quorum whenever the current leader fails or becomes unresponsive. This means that the system tolerates up to $N/2 - 1$ failing servers.

B. Emulating Physical Resources and Conditions

Given an infrastructure configuration, the emulation system creates a digital twin through the following steps.

Emulating physical hosts. Physical hosts are emulated with Docker containers [25], i.e. lightweight executable packages that include runtime systems, code, system tools, system libraries, and configurations. Resource allocation to containers, e.g. CPU and memory, is enforced using cgroups.

Emulating physical switches. Physical switches are emulated with Docker containers that run Open vSwitch (OVS) [26] and may connect to a controller through the OPENFLOW protocol [27]. (Since the switches are programmed through flow tables, they can act either as classical layer 2 switches or as routers, depending on the flow table configurations.)

Emulating physical network links. Network connectivity is emulated with virtual links implemented by Linux bridges. Network isolation between virtual containers on the same physical host is achieved through network namespaces, which create logical copies of the physical host's network stack.

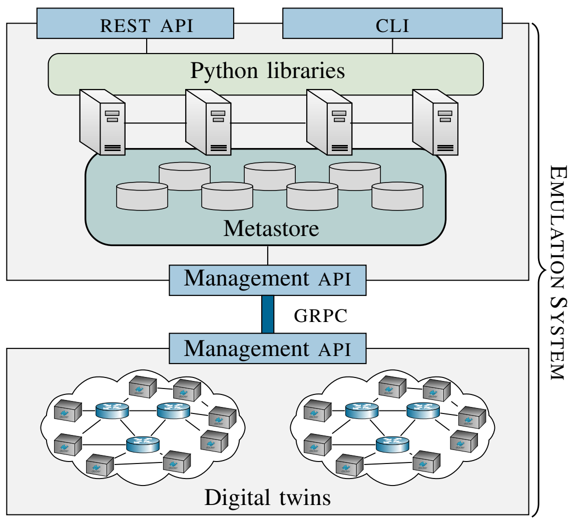


Fig. 3: Architecture of the emulation system for creating digital twins; it executes on a cluster of machines that runs a virtualization layer provided by Docker containers and virtual links; it uses a GRPC API to create and manage digital twins; it stores metadata in a distributed database (the metastore); it implements system functionality in Python libraries; and it has two interfaces: a web interface (a REST API) and a Command-Line Interface (a CLI).

In the case that an emulated network spans multiple physical servers, the emulated traffic is tunneled over the physical network using VXLAN tunnels. In other words, the physical network provides a substrate network, on top of which virtual networks are overlaid.

Emulating network conditions. Network conditions of virtual links are configured using the NetEm module in the Linux kernel [28]. This module allows fine-gained configuration of bit rates, packet delays, packet loss probabilities, jitter, and packet reordering probabilities.

C. *Management and Monitoring of Digital Twins*

To manage and monitor digital twins, the emulation system equips each digital twin with a management network, a set of management agents, and a set of monitoring agents. The system also includes a web interface for management purposes.

The management network. Emulated devices and management systems are connected through a *management network*. The reason for using a separate network to carry management traffic is to avoid interference and simplify control of the network [29].

Management agents. Each emulated device runs a *management agent*, which exposes a GRPC API. This API is invoked by security policies to perform control actions, e.g. restarting services and updating configurations. The API specification is available in [22].

Monitoring agents. Each emulated device runs a *monitoring agent*, which reads local metrics of the device and pushes those metrics to an event bus implemented with KAFKA [30] (see Fig. 4). The data in this bus is consumed by data pipelines implemented with SPARK [31], which process the data, write it to storage systems, and provide inputs to security policies for deciding on control actions. The number of metrics collected per time-step is in the order of thousands and scales linearly with the number of virtual hosts in the digital twin. The list of metrics is available in [22].

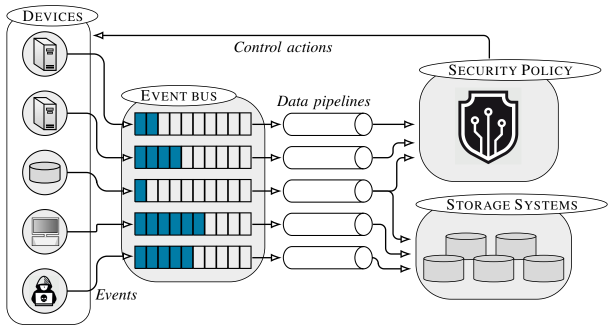


Fig. 4: Monitoring system of a digital twin; emulated devices run monitoring agents that periodically push metrics to an event bus; the data in this bus is consumed by data pipelines that process the data and write to storage systems; the processed data is used by an automated security policy to decide on control actions to execute in the digital twin.

Web interface. The emulation system has a web interface for viewing management information and requesting management operations (see Fig. 5). A video demonstration of the web interface is available in [22].

D. *Emulating Security Actors in a Digital Twin*

In this section, we describe how the emulation system implements security actors in a digital twin.

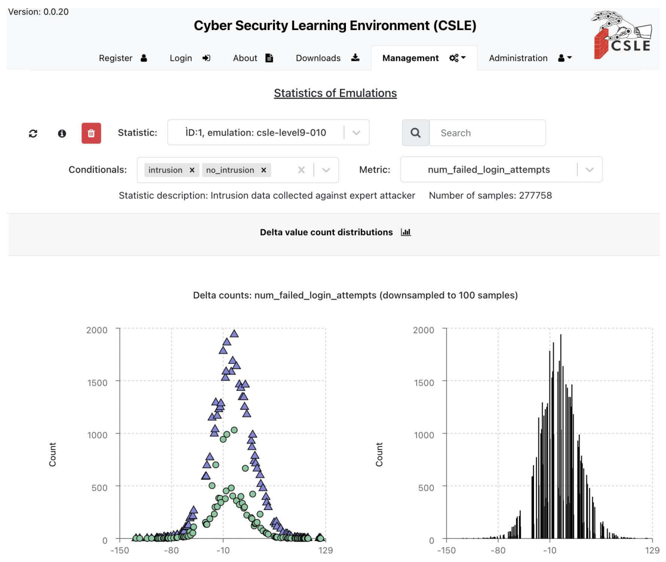
Emulating client populations. Client populations are emulated by processes that run in Docker containers and interact with emulated hosts through various network protocols, e.g. HTTP, SSH, and DNS. The clients select network functions from a pre-defined list, which is available in [22]. The functions are selected according to a Markov process. Client arrivals are emulated by a Poisson process with exponentially distributed service times.

Emulating attackers. Attackers are emulated by automated programs that select actions from a pre-defined set, which is available in [22]. The actions are selected according to an *attacker policy*, which may depend on system metrics collected by the monitoring agents.

Emulating defenders. Defender actions are emulated by executing system commands through the GRPC API described above. The actions are selected according to a *defender policy*, which may depend on system metrics collected by the monitoring agents. The defender actions are listed in [22].

III. Running Security Scenarios in a Digital Twin and Learning An Effective Security Policy

In this section, we describe how a digital twin created through the steps above can be used to play out security



Statistic: ÌD:1, emulation: csle-level9-010

Conditionals: intrusion ✕ no_intrusion ✕

Metric: num_failed_login_attempts

Statistic description: Intrusion data collected against expert attacker

Number of samples: 277758

Delta counts: num_failed_login_attempts (downsampled to 100 samples)

Fig. 5: The statistics page of the emulation system's web interface; this page allows a user to view empirical statistics of digital twins.

scenarios and automate the process of finding an effective security policy for an IT infrastructure. We first describe the security scenario (§III-A) and then we describe the learning process (§III-B).

A. *Example Security Scenario: Intrusion Response*

We consider an intrusion response scenario that involves the IT infrastructure of an organization (see Fig. 6 and Table 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 Detection and Prevention System (IDPS) that logs events in real-time. Clients access the services through a public gateway, which also is open to the attacker.

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 vulnerabilities while avoiding detection by the defender. The attacker follows a pre-defined attack policy, which is defined in [7].

The defender continuously monitors the infrastructure through accessing and analyzing IDPS alerts and other statistics. It can take a fixed number of defensive actions, each of which has a cost and a chance of stopping an ongoing attack. An example of a defensive action is to drop network traffic that triggers IDPS alerts of a certain priority. The defender takes defensive actions in a pre-determined order, starting with the action that has the lowest cost. The final action blocks all external access to the gateway, which disrupts any intrusion as well as the services to the clients.

When deciding the time for taking a defensive action, the defender balances two objectives: (*i*) maintain services to its clients; and (*ii*), stop a possible intrusion at the lowest

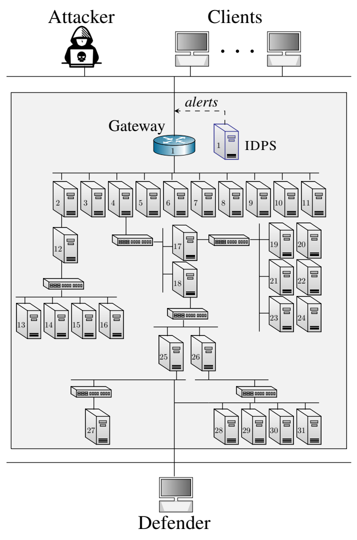


Fig. 6: The IT infrastructure and the actors in the intrusion response scenario.

ID (s)	*OS:Services:Exploitable Vulnerabilities*
N_1	UBUNTU 20:SNORT (community ruleset v2.9.17.1),SSH:-
N_2	UBUNTU 20:SSH,HTTP,DNS:SSH-pw
N_4	UBUNTU 20:HTTP,TELNET,SSH:TELNET-pw
N_{10}	UBUNTU 20:FTP,MONGODB,SMTP,TOMCAT,TS 3,SSH:FTP-pw
N_{12}	JESSIE:TS 3,TOMCAT,SSH:CVE-2010-0426,SSH-pw
N_{17}	WHEEZY:APACHE 2,SNMP,SSH:CVE-2014-6271
N_{18}	Deb9.2:IRC,APACHE 2,SSH:SQL injection
N_{22}	JESSIE:FTP,SSH,APACHE 2,SNMP:CVE-2015-3306
N_{23}	JESSIE:APACHE 2,SMTP,SSH:CVE-2016-10033
N_{24}	JESSIE:SSH:CVE-2015-5602,SSH pw
N_{25}	JESSIE: Elasticsearch,APACHE 2,SSH,SNMP:CVE-2015-1427
N_{27}	JESSIE:SAMBA, NTP,SSH:CVE-2017-7494
N_3,N_{11},N_5-N_9	UBUNTU 20:SSH,SNMP,POSTGRES,NTP:-
N_{13-16},N_{19-21},N_{26},N_{28-31}	UBUNTU 20:NTP, IRC, SNMP, SSH, POSTGRES:-

TABLE 1: Configuration of the target infrastructure (Fig. 6).

cost. The optimal policy for the defender is to monitor the infrastructure and maintain services until the moment when the attacker enters through the gateway, at which time the attack must be stopped at minimal cost through defensive actions. The challenge for the defender is to identify this precise moment.

B. *Learning An Effective Intrusion Response Policy*

We formulate the above intrusion response use case from the defender's perspective as a Partially Observed Markov Decision Process (POMDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}^{a_t}_{s_t,s_{t+1}}, \mathcal{R}^{a_t}_{s_t}, \gamma, \rho_1, T, \mathcal{O}, \mathcal{Z} \rangle$. Theoretical details of this model are available in our previous work [7, §IV].

To approximate an optimal defender policy π^*, we use the following reinforcement learning approach. We first create a digital twin of the target IT infrastructure using the emulation system described in §II (the infrastructure's topology is shown in Fig. 6 and its configurations is listed in Table 1). We then use the digital twin to play out security scenarios, which generate system traces. Next, we use the generated traces to instantiate the POMDP $\mathcal{M}$ and learn π^* through simulations of $\mathcal{M}$. Lastly, we evaluate the learned policy in the digital twin.

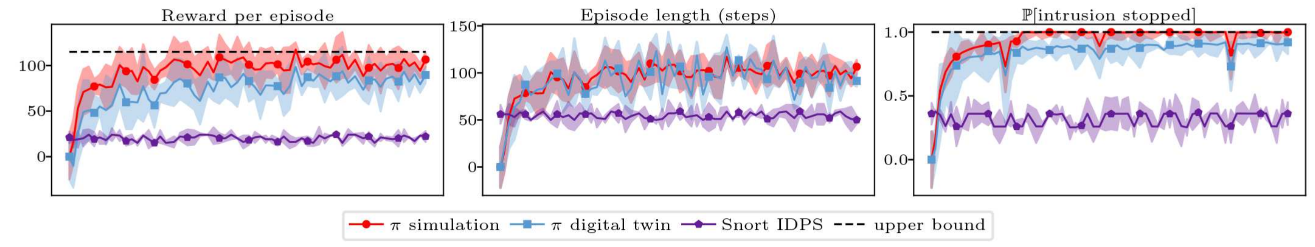


Fig. 7: Learning curves obtained during training of the defender policy π; red curves show simulation results and blue curves show results from the digital twin; the purple and black curves relate to baseline policies; the columns from left to right show performance metrics: episodic reward, episode length, and empirical stopping probability; the curves show the mean and 95% confidence interval for five training runs with different random seeds.

Listing 1 Command to start a digital twin with the configuration `twin-1` using the Command-Line Interface (CLI).

```
csle start twin-1
```

Listing 2 Code that uses Python libraries of the emulation system to a) execute attacker and defender sequences in a digital twin; b) estimate statistics and instantiate a Markov decision process; and c) run a reinforcement learning algorithm to approximate an optimal defender policy (in this example the T-SPSA algorithm is used [7, §IV.C]).

```python
import Metastore, Emulator,
↪   SystemIdentification,
↪   ExpectationMaximization, Experiment,
↪   TSPSAAgent from csle
# Get digital twin details from metastore
dt = Metastore.get_dt(..)
# Define attacker/defender sequences
attacker_sequence = [..]
defender_sequence = [..]
# Run sequences in the digital twin
Emulator.run_action_sequences(dt,
↪   attacker_sequence, defender_sequence)
# Extract recorded traces and statistics
stats = Metastore.statistics()
traces = Metastore.traces()
# Setup experiment and hyperparameters
sid_config = SystemIdentification(..)
# Define identification algorithm
id_algorithm = ExpectationMaximization(..)
# Run the algorithm
pomdp = id_algorithm.fit(stats)
# Define reinforcement learning agent
agent = TSPSAAgent(dt, pomdp, ..)
# Run the algorithm
results = agent.train()
# Save results and the learned policy
Metastore.save(results)
```

The command for creating the digital twin using the CLI of the emulation system is given in Listing 1 and the Python code for running security scenarios, collecting data, and learning the policy is given in Listing 2. (The hyperparameters and further details about the execution are available in [7].)

Evaluation results. The results are shown in Fig. 7. We compare the learned policy with the SNORT IDPS, which is a de-facto industry standard and can be considered state-of-the-art for our use case. The red curves in Fig. 7 represent the results from the POMDP simulations; the blue curves show the results from the digital twin; the purple curves give the performance of the SNORT IDPS baseline; and the dashed black curves give an upper bound to any optimal policy.

We observe that the learning curves converge quickly to constant mean values across all investigated performance metrics. From this observation, we conclude that the learned policy has converged as well. Second, we observe that the converged values of the learning curves are close to the dashed black curves, which suggests that the learned policy is near-optimal. We also observe that the learned policy does significantly better than the SNORT IDPS baseline. Third, although the learned policy, as expected, performs better in the simulator than in the digital twin, we are encouraged by the fact that the curves of the digital twin are close to those of the simulator, which gives us high confidence that the learned policy would perform as expected also in the real-world infrastructure.

Inspection of the learned policy. We use the web interface of the emulation system to examine the learned security policy (see Fig. 5 and [32]). From this examination, we conclude that the policy takes defensive actions as soon as the attacker starts its reconnaissance phase or when it launches its first exploit. Specifically, we find that if the attacker performs reconnaissance by executing a port scan, there is a spike in infrastructure metrics, which causes the learned policy to react by revoking user certificates. If the attacker uses other means of reconnaissance, e.g. a ping scan, we observe that the policy does not detect it since no change in infrastructure metrics is generated.

IV. Related Work

Digital twin has emerged as an enabling technology in many industry sectors, including the manufacturing industry (see example [1]), the automotive industry (see example [2]), and the healthcare industry (see example [3]). While digital twins have been widely adopted in all of these industries, it is only recently that it has been adopted in networking and cyber security research (see surveys [4]–[6]). Use cases of digital twin in networking and security include network planning, anomaly detection, and predictive analytics [4]–[6], [33].

Few papers have studied digital twins in the context of reinforcement learning [20], [34]–[36]. This paper differs from these works in two main ways. First, we apply reinforcement learning and digital twins in the context of cyber security, whereas the referenced works study use cases in manufacturing, autonomous driving, and network planning. Second, our system for creating digital twins is based on advanced emulation technologies, whereas the referenced works use simulators.

This paper is also related to ongoing efforts in building reinforcement learning frameworks for cyber defense [14]–[17]. Some of these frameworks use simulators and some of them use emulation systems that resemble ours. In contrast to these frameworks, our approach includes both an emulation system and a simulator. Further, our approach has been thoroughly tested on an intrusion response use case [7], [10]–[13], whereas only limited evaluations of the frameworks in [14]–[17] have been conducted.

V. Conclusion and Future Work

We present a novel emulation system for creating high-fidelity digital twins of IT infrastructures. We show that the digital twins allow to play out security scenarios in a safe environment and that they can be used to automate the process of finding effective security policies. We apply our approach to an intrusion response scenario that involves an IT infrastructure. Our evaluation results show that the digital twin provides the necessary evaluative feedback to learn near-optimal intrusion response policies.

In future work, we plan to extend our system to create digital twins of a diverse set of IT infrastructures and use them to find security policies for different use cases.

References

[1] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, 2019.

[2] F. Biesinger and M. Weyrich, "The facets of digital twins in production and the automotive industry," in *2019 23rd International Conference on Mechatronics Technology (ICMT)*, 2019, pp. 1–6.

[3] Y. Liu *et al.*, "A novel cloud-based framework for the elderly healthcare services using digital twin," *IEEE Access*, 2019.

[4] P. Almasan *et al.*, "Network digital twin: Context, enabling technologies, and opportunities," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 22–27, 2022.

[5] H. X. Nguyen, R. Trestian, D. To, and M. Tatipamula, "Digital twin for 5g and beyond," *IEEE Communications Magazine*, 2021.

[6] A. Pokhrel *et al.*, "Digital twin for cybersecurity incident prediction: A multivocal literature review," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020.

[7] K. Hammar and R. Stadler, "Intrusion prevention through optimal stopping," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2333–2348, 2022.

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

[9] ——, "Learning intrusion prevention policies through optimal stopping," in *International Conference on Network and Service Management (CNSM 2021)*, Izmir, Turkey, 2021, https://arxiv.org/pdf/2106.07160.pdf.

[10] ——, "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.

[11] ——, "An online framework for adapting security policies in dynamic it environments," in *2022 18th International Conference on Network and Service Management (CNSM)*, 2022, pp. 359–363.

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

[13] ——, "Learning near-optimal intrusion responses against dynamic attackers," 2023, https://arxiv.org/abs/2301.06085.

[14] M. Standen *et al.*, "Cyborg: A gym for the development of autonomous cyber agents," https://arxiv.org/abs/2108.09118.

[15] L. Li, R. Fayad, and A. Taylor, "Cygil: A cyber gym for training autonomous agents over emulated network systems," 2021.

[16] A. Molina-Markham *et al.*, "Network environment design for autonomous cyberdefense," 2021, https://arxiv.org/abs/2103.07583.

[17] A. Andrew, S. Spillard, J. Collyer, and N. Dhir, "Developing optimal causal cyber-defence agents via cyber security simulation," in *Proceedings of the ML4Cyber workshop, ICML 2022, Baltimore, USA*, 2022.

[18] H. Wang, Y. Wu, G. Min, and W. Miao, "A graph neural network-based digital twin for network slicing management," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1367–1376, 2022.

[19] L. Hui *et al.*, "Digital twin for networking: A data-driven performance modeling perspective," *IEEE Network*, pp. 1–8, 2022.

[20] C. Güemes-Palau *et al.*, "Accelerating deep reinforcement learning for digital twin network optimization with evolutionary strategies," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022, pp. 1–5.

[21] J. Deng *et al.*, "A digital twin approach for self-optimization of mobile networks," in *2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2021, pp. 1–6.

[22] K. Hammar, "Cyber security learning environment," 2023, https://limmen.dev/csle/.

[23] U. Cubukcu *et al.*, "Citus: Distributed postgresql for data-intensive applications," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD '21, 2021.

[24] S. Niazi, M. Ismail, G. Berthou, and J. Dowling, "Leader election using newsql database systems," in *Distributed Applications and Interoperable Systems*, 2015.

[25] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, p. 2, 2014.

[26] B. Pfaff *et al.*, "The design and implementation of open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015.

[27] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, p. 69–74, mar 2008.

[28] S. Hemminger, "Network emulation with netem," *Linux Conf*, 2005.

[29] A. Clemm and I. Cisco Systems, *Network Management Fundamentals*, ser. Cisco Press fundamentals series. Cisco Press, 2007.

[30] J. Kreps, "Kafka : a distributed messaging system for log processing," 2011.

[31] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USA: USENIX Association, 2010, p. 10.

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

[33] C. Gehrmann and M. Gunnarsson, "A digital twin based industrial automation and control system security architecture," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 669–680, 2020.

[34] K. Xia *et al.*, "A digital twin to train deep reinforcement learning agent for smart manufacturing plants: Environment, interfaces and intelligence," *Journal of Manufacturing Systems*, 2021.

[35] W. Yang *et al.*, "Optimizing federated learning with deep reinforcement learning for digital twin empowered industrial iot," *IEEE Transactions on Industrial Informatics*, 2023.

[36] J. Wu *et al.*, "Digital twin-enabled reinforcement learning for end-to-end autonomous driving," in *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2021, pp. 62–65.