CSLE: A REINFORCEMENT LEARNING PLATFORM FOR AUTONOMOUS SECURITY MANAGEMENT

Kim Hammar [1]

ABSTRACT

Reinforcement learning is a promising approach to autonomous and adaptive security management in networked systems. However, current reinforcement learning solutions for security management are mostly limited to simulation environments and it is unclear how they generalize to operational systems. In this paper, we address this limitation by presenting CSLE: a reinforcement learning platform for autonomous security management that enables experimentation under realistic conditions. Conceptually, CSLE encompasses two systems. First, it includes an emulation system that replicates key components of the target system in a virtualized environment. We use this system to gather measurements and logs, based on which we identify a system model, such as a Markov decision process. Second, it includes a simulation system where security strategies are efficiently learned through simulations of the system model. The learned strategies are then evaluated and refined in the emulation system to close the gap between theoretical and operational performance. We demonstrate CSLE through four use cases: flow control, replication control, segmentation control, and recovery control. Through these use cases, we show that CSLE enables near-optimal security management in an environment that approximates an operational system.

1 INTRODUCTION

Managing the security of networked systems alongside their service requirements and physical infrastructures is a major technical challenge that has grown exponentially with the rise of cloud computing, distributed networks, and IoT services. Examples of security management tasks include incident response, risk analysis, strategy design, and threat hunting. Today, many of these tasks remain manual processes carried out by security experts. Although this approach can be effective, it is labor-intensive and requires significant skills. For example, a recent study reports a global shortage of more than 4 million security experts (ISC2, 2024).

A promising approach to address this challenge is to use reinforcement learning to automatically derive effective security strategies. For example, Li et al. (2024) use reinforcement learning to compute effective incident response strategies. Similarly, Kiely et al. (2025) use multi-agent reinforcement learning to derive effective defense strategies against advanced persistent threats. A comprehensive review of these developments is provided by Nguyen & Reddi (2023). While these works report encouraging results, key challenges remain. Chief among them is narrowing the gap between the environment where strategies are evaluated and a scenario playing out in an operational system. Most of the results obtained so far are limited to simulation environments, leaving their practical utility unproven.

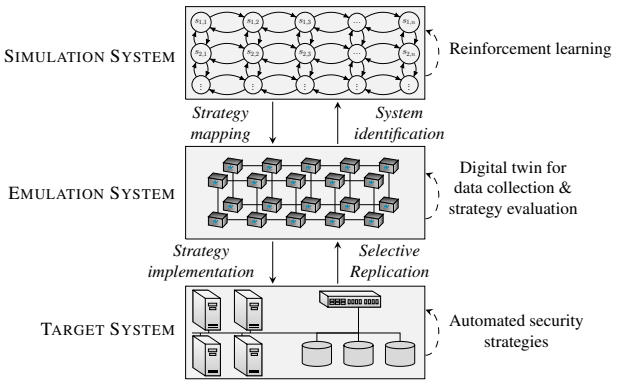


Figure 1. Architectural overview of CSLE: a reinforcement learning platform for autonomous security management.

In this paper, we address this limitation by presenting a platform that enables experimentation with reinforcement learning in realistic conditions. Conceptually, the platform consists of two systems, as illustrated in Fig. 1. First, we use an *emulation system* for creating a virtual replica (i.e., a *digital twin*) of the target system. This twin closely approximates the functionality and timing behavior of the

[1]Division of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden. Correspondence to: Kim Hammar <kimham@kth.se>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

target system, which allows us to run attack scenarios in a controlled environment. Such runs produce system measurements and logs, based on which we identify a system model, e.g., a Markov decision process. Second, we use a *simulation system* where near-optimal security strategies are incrementally learned through reinforcement learning. Learned strategies are extracted from the simulation system and evaluated in the digital twin. This process can be performed iteratively to provide progressively better security strategies that are adapted to changes in the target system, such as configuration changes and software updates.

We refer to the platform as CSLE, which stands for the "***C**yber **S**ecurity **L**earning **E**nvironment*." CSLE includes an initial set of 15 digital twin configurations, more than 50 simulated security scenarios, 34 implemented reinforcement learning algorithms, and 4 implemented system identification algorithms, all of which can be extended. To evaluate CSLE experimentally, we use it to learn effective security strategies for four security management tasks: flow control, replication control, segmentation control, and recovery control. Through these use cases, we show that CSLE enables autonomous security management in an environment that closely approximates an operational system. Moreover, we demonstrate the broad applicability of CSLE and its integration with various reinforcement learning techniques.

In summary, the main contributions of this paper are:

- We present CSLE, a reinforcement learning platform for autonomous security management that enables experimentation under realistic operating conditions.
- We evaluate CSLE on four different use cases: flow control, replication control, segmentation control, and recovery control. Our experimental results show that CSLE enables autonomous security management.

Open source The source code of CSLE is released under the CC-BY-SA 4.0 license and available in the repository at (Hammar, 2023). In addition to the source code, this repository also includes video demonstrations, Docker images, documentation, and datasets of system traces.

2 Autonomous Security Management through Reinforcement Learning

Before presenting our platform, we start by formulating security management as a reinforcement learning problem. To accomplish this formulation, we need a vocabulary in which to talk about the systems and actors involved. To this end, we refer to the operator of the target system as the *defender*, and we refer to an entity aiming to attack the system as the *attacker*. Both interact with the system by taking *actions* (e.g., attacks and responses), which affect the system's *state* (e.g., the system's security and service status). When selecting these actions, the defender and the attacker use measurements from the system (e.g., log files and security alerts), which we refer to as *observations*. A function that maps a sequence of observations to an action is called a *strategy*, and a strategy that is most advantageous according to some objective is *optimal*.

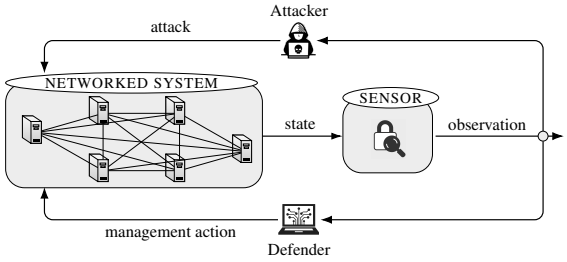


Figure 2. Autonomous and adaptive security management of a networked system as a reinforcement learning problem.

In the context of reinforcement learning, we can view security management as the problem of learning an effective defender strategy through repeated interaction with the system. In particular, by observing how the system responds to different actions, the defender can gradually improve its strategy to meet security objectives. However, many system-level challenges are encountered when applying this approach in practice. Chief among them are:

1. In an operational networked system, attacks and defender actions unfold over long time scales and can disrupt critical services. These factors make direct interaction with the system impractical for reinforcement learning, which typically requires executing thousands of actions to learn an effective strategy.
2. To enable safe and efficient learning, the learning process must, therefore, be executed in a simulation environment. However, the system's behavior is often too complex to model, which means that the simulation dynamics must be estimated from system measurements.
3. After learning a security strategy through simulation, it must be experimentally validated. In particular, the validity of the simulation must be verified by evaluating the learned strategy in an environment that closely approximates the target system.

In the next section, we review existing platforms that attempt to address these challenges and explain their limitations. We then introduce our platform (CSLE), which is designed to overcome these limitations and enable reinforcement learning experimentation in realistic operating conditions.

Platform	Simulation	Emulation	Open source	RL Library	Management	Validated	Maintained	Distributed
CSLE (our platform)	✓	✓	✓	✓	✓	✓	✓	✓
CyberBattleSim	✓	✗	✓	✓	✗	✗	✓	✗
CyBorg	✓	✗	✓	✗	✗	✗	✗	✗
Yawning Titan	✓	✗	✓	✗	✗	✗	✗	✗
NaSim	✓	✗	✓	✗	✗	✗	✗	✗
ATMoS	✗	✓	✓	✗	✗	✓	✗	✗
Gym-FlipIt	✓	✗	✓	✗	✗	✗	✗	✗
Gym-IDSgame	✓	✗	✓	✗	✗	✗	✗	✗
MAB-Malware	✗	✓	✓	✗	✗	✓	✓	✗
Malware-RL	✗	✓	✓	✗	✗	✓	✓	✗
PenGym	✓	✓	✓	✗	✗	✗	✓	✗
CyGil	✗	✓	✗	✗	✗	✗	?	?
NaSimEmu	✓	✓	✓	✗	✗	✗	✓	✗
Farland	✓	✓	✗	✗	✗	✓	?	?
CyberWheel	✓	✓	✓	✗	✗	✓	✓	✗
CyberShield	✓	✗	✗	✗	✗	✗	✓	✗
Cyborg++	✓	✗	✓	✗	✗	✗	✓	✗
CyGym	✓	✗	✓	✗	✗	✗	✓	✗
C-CyberBattleSim	✓	✗	✓	✗	✗	✗	✓	✗

Table 1. Comparison between reinforcement learning platforms for autonomous security management based on key features: support for simulation-based optimization; support for emulation-based evaluation; open source code; whether the platform provides a library with implemented reinforcement learning algorithms to facilitate strategy optimization and system identification; whether the platform provides a management system for automating experiments and debugging strategies; whether the platform has been experimentally validated on practical use cases; whether the platform is actively maintained; and whether the platform supports distributed deployment.

3 Reinforcement Learning Platforms for Autonomous Security Management

Over the past 5 years, several reinforcement learning environments for autonomous security management have been developed. They include CyberBattleSim by Microsoft (Blum, 2021), CyBorg by the Australian department of defense (Standen et al., 2021), NaSim by the University of Queensland (Schwartz et al., 2020), Yawning Titan by the UK defense science and technology laboratory (Andrew et al., 2022), CyGil by Canada's department of defense (Li et al., 2021), NaSimEmu by the Czech Technical University in Prague (Janisch et al., 2023), ATMoS by the University of Waterloo (Akbari et al., 2020), Gym-FlipIt by Northeastern University (Oakley & Oprea, 2019), Gym-IDSgame by KTH Royal Institute of Technology (Hammar & Stadler, 2020), MAB-Malware by the University of California (Riverside) (Song et al., 2022), Malware-RL by the University of Virginia (Anderson et al., 2018), PenGym by Japan's advanced institute of science and technology (Huynh Phuong Thanh et al., 2024), Farland by USA's national security agency (Molina-Markham et al., 2021), CyberWheel by the Oak Ridge national laboratory (Oesch et al., 2024), CyberShield by the University of Malaga (Carrasco et al., 2024), Cyborg++ by the Alan Turing Institute (Emerson et al., 2024), CyGym by Washington University (Lanier & Vorobeychik, 2025), and C-CyberBattleSim by the University of Lorraine (Terranova et al., 2025).

Like CSLE, all of the referenced platforms include capabilities for learning security strategies using reinforcement learning. However, they differ from CSLE in several important ways, as highlighted in Table 1. First, most existing platforms are confined to simulations. By contrast, CSLE is centered around an emulation system based on virtualization. The benefit of our approach is that it narrows the gap between the environment where security strategies are evaluated and a scenario playing out in an operational system. Second, many of the referenced platforms are not open source and most of them are no longer maintained. By contrast, CSLE is open source and has an active development community. Third, CSLE has been experimentally validated on a range of practical use cases, whereas most other platforms have only been evaluated on a single simulation use case. Fourth, unlike the other platforms, CSLE supports distributed deployment, which improves scalability. Moreover, CSLE incorporates a novel management system that provides infrastructure for automating reinforcement learning experiments and debugging the learned strategies.

Lastly, we note that a few platforms for autonomous system operations based on large language models (LLMs) have recently emerged, most notably ITBench (Jha et al., 2025) and AIOpsLab (Chen et al., 2025). These platforms focus on using LLMs to automate general system operation tasks. By contrast, CSLE is explicitly designed for automating security management tasks. Another difference is that the referenced platforms are designed for evaluating LLM-based agents, whereas CSLE is designed for develop-

ing reinforcement learning agents. This difference in scope leads to fundamental differences in platform architecture. In particular, LLMs do not require the same system-level support as reinforcement learning agents do. For example, CSLE supports identifying simulation models, optimizing strategies through reinforcement learning, and transferring strategies from simulation to emulation. None of these functions is provided by ITBench and AIOpsLab.

4 Architecture of CSLE

The architecture of CSLE is illustrated in Fig. 1 and is centered around an emulation system for creating a digital twin, i.e., a virtual replica of the target system.[1] We use this twin to run automated attack scenarios and defender responses. Such runs produce system measurements and logs, from which we estimate infrastructure statistics. These statistics allow us to instantiate a mathematical model of the target system through *system identification*. We then leverage this model to learn effective security strategies through simulation, whose performance is assessed using the digital twin. This closed-loop process can be executed iteratively to provide progressively better security strategies that are adapted to changes in the target system; see Fig. 3.

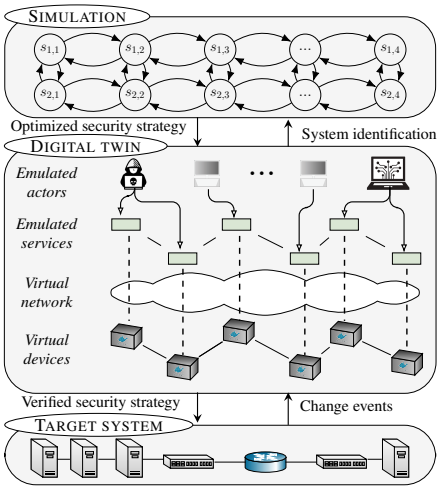


Figure 3. A digital twin in CSLE is a virtual replica of a target system that runs the same software and configuration, but on virtualized hardware. Moreover, the twin controls network delays and emulates actors to replicate operational workloads. The twin is used in CSLE for strategy evaluation and system identification.

4.1 Emulation System

As described above, the emulation system in CSLE is used to create a digital twin of the target system. The concept of a *digital twin* emerged in the 1960s when NASA used virtual environments to evaluate failure scenarios for lunar landers (Allen B. Danette, 2021). Since then, digital twin has emerged as a key technology in automation and has been adopted in several industries, including the manufacturing industry [see e.g., (Tao et al., 2019)], the automotive industry [see e.g., (Biesinger & Weyrich, 2019)], the healthcare industry [see e.g., (Liu et al., 2019)], and the technology industry [see e.g., (Wu et al., 2021)].

In CSLE, a digital twin is a virtual replica of a networked system that provides a controlled environment for virtual operations (e.g., cyberattacks and responses), the outcomes of which can be used to optimize operations in the target system. Such a twin enables us to systematically test security strategies under different conditions, including varying attacks, workloads, and network latencies.

4.2 Simulation System

The simulation system in CSLE is used to run simulations and execute reinforcement learning algorithms. Although these algorithms could in principle be executed in the digital twin, this approach is not practical due to the long execution times required for carrying out actions and collecting observations in the digital twin. For instance, executing a cyberattack or a defensive reconfiguration in a digital twin can take several minutes. In contrast, the simulation system abstracts these processes as actions in a Markov decision process, which reduces the execution time to milliseconds.

With a *simulation*, we mean an execution of a *discrete-time dynamical system* of the form

$$s_{t+1} \sim f(s_t, a_t^{(\mathrm{D})}, a_t^{(\mathrm{A})}), \tag{1}$$

where s_t is the system state at time t, $a_t^{(\mathrm{D})}$ is the defender action, $a_t^{(\mathrm{A})}$ is the attacker action, f is the system dynamics, and $s \sim f$ means that s is sampled from f. For example, the dynamics f may represent a Markov decision process (MDP) or a Markov game. Each simulation path $s_1, s_2, \ldots, s_t$ is associated with security consequences and costs. The goal of reinforcement learning is to identify the defender actions that control the simulation in an optimal manner according to a specified security objective.

4.3 Reinforcement Learning Methodology

The emulation and simulation systems in CSLE enable a reinforcement learning methodology with the following steps.

Step 1 *Defining the target system.*

- This is the system where the learned security strategies are intended to be deployed. In CSLE, the target system is defined through a configu-

[1] By *target system*, we mean the system where the learned security strategies are intended to be deployed.

ration file that specifies the system components, the network topology, the services, etc.

Step 2 *Creating a digital twin of the target system.*

- Given the target system specification, the creation of a digital twin in CSLE is automated through the emulation system.

Step 3 *Collecting data from the digital twin.*

- After creating the digital twin, we use it to run attack scenarios. Such runs produce system traces (i.e., sequences of system metrics), which we collect through CSLE's monitoring system.

Step 4 *Identifying a system model.*

- Having collected system measurements from the digital twin, we use the collected data to identify a model (e.g., through statistical learning) that can be used for running simulations, such as a Markov decision process (MDP).

Step 5 *Learning an effective security strategy.*

- Given the identified system model (e.g., an MDP), we apply reinforcement learning techniques to learn an effective security strategy.

Step 6 *Evaluating the learned strategy in the digital twin.*

- After the learning process has converged, we evaluate the learned security strategy in the digital twin. Such evaluation involves measuring system metrics from the digital twin in real time (e.g., security alerts), using them as input to the security strategy, and executing the action prescribed by the strategy in the digital twin.

Step 7 *Deploying the learned strategy in the target system.*

- If the evaluation is satisfactory, we deploy the learned strategy in the target system. Otherwise, we collect more data to update the simulation and then learn a new strategy. This procedure of updating the simulation and re-learning the strategy is repeated until a strategy with satisfactory performance is obtained.

5 Implementation of CSLE

We have implemented CSLE in Python [$\approx 275,000$ lines of code], JavaScript [$\approx 40,000$ lines of code], and Bash [$\approx 5,000$ lines of code]. From an architectural point of view, the implementation can be divided into three systems: the emulation system, the simulation system, and the management system; see Fig. 4. Broadly speaking, the emulation system creates digital twins, the simulation system runs reinforcement learning algorithms, and the management system orchestrates the platform. The rest of this section delves into the technical details of these three systems.

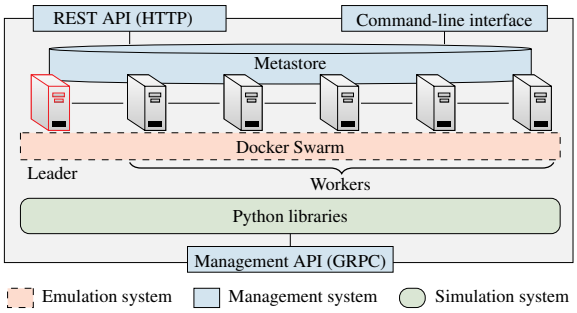


Figure 4. The architecture of CSLE. It is a distributed platform with N servers ($N = 6$ in this example), which are connected through a database (the metastore) and a virtualization layer provided by Docker Swarm. CSLE has four interfaces: a Python API, a GRPC API, a REST API, and a command-line interface.

5.1 Infrastructure

CSLE runs on a distributed system with $N \geq 1$ servers connected through an IP network. Each server runs a virtualization layer provided by Docker Swarm (Merkel, 2014) and can be accessed through Python libraries, a web interface, a command-line interface, and a GRPC interface (Google, 2022). Platform metadata is stored in a distributed database referred to as the *metastore*, which is based on Citus (Cubukcu et al., 2021). This database consists of N replicas, one per server. One replica is a designated *leader* and is responsible for coordination. The others are *workers*. A new leader is elected by a quorum whenever the current leader fails or becomes unresponsive. CSLE thus tolerates up to $\lfloor \frac{N-1}{2} \rfloor$ failing servers. This design enables horizontal scaling as the number of servers increases.

Deployment of CSLE in both on-premise and cloud infrastructures is automated using Ansible (Red Hat, 2024). This automation enables on-demand deployment, allowing CSLE to be launched dynamically for specific experiments or to run continuously as part of an operational environment.

5.2 The Emulation System

The purpose of the emulation system in CSLE is to create a digital twin that replicates relevant components of the target system. Creating such a twin involves three tasks: (*i*) emulating the target system's physical infrastructure, such as processors, network interfaces, and network conditions; (*ii*) emulating actors, i.e., attackers, defenders, and clients; and (*iii*) instrumenting the twin with monitoring and management capabilities. Each of these tasks is detailed below.

Emulating hosts and switches

We emulate hosts and switches with Docker containers (Merkel, 2014), i.e., lightweight executable packages that include runtime systems, code, libraries, and configurations. This virtualization lets us quickly instantiate large digital twins; see Fig. 5. Resource allocation to containers, e.g., CPU and memory, is enforced using Cgroups. Containers that emulate switches run OVS (Pfaff et al., 2015) and connect to controllers through OpenFlow (McKeown et al., 2008). Since the switches are programmed through flow tables, they can act as layer-two switches or as routers.

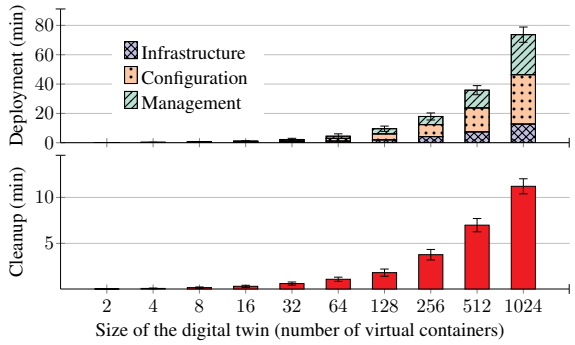


Figure 5. Time to deploy and cleanup a digital twin in CSLE. Deploying the twin involves creating containers, attaching them to networks, configuring them, and starting management services. Cleanup involves stopping and deleting containers and networks. The time measurements were performed for a digital twin with a single network running on a server with a 24-core Intel Xeon Gold 2.10 GHz CPU and 768 GB RAM. Numbers and error bars indicate the mean and the standard deviation from 5 measurements.

The hosts and switches of the digital twin are specified through a configuration file written in Python, which CSLE parses before deploying the twin. We provide a code snippet of the configuration file in Listing 1.

```
from csle_common.dao.emulation_config.
    node_container_config import NodeContainerConfig
from csle_common.dao.emulation_config.
    node_firewall_config import NodeFirewallConfig
node_cfg = NodeContainerConfig(name="my-image", os="
    Ubuntu22", ips=[..], subnets=[..], interfaces=[..],
      cpus=1, memory_gb=4)
node_fw_config = NodeFirewallConfig(host="..",
    default_gw="",default_input="ACCEPT",default_output
    ="ACCEPT",default_forward="ACCEPT",fw_rules=[..])
```

Listing 1. Python code for configuring a container.

Emulating network links

We emulate network connectivity in digital twins through virtual links implemented by Linux bridges and network namespaces. If an emulated network spans multiple physical servers, we tunnel the traffic over the physical network using VXLAN (Mahalingam et al., 2014). In other words, the physical network of the servers provides a substrate, on top of which the emulated networks are overlaid.

Network conditions of virtual links are created using the NetEm module in the Linux kernel (Hemminger, 2005). This module allows setting bit rates, packet delays, packet loss probabilities, and jitter. For example, the standard configuration in CSLE emulates connections between servers in an IT system with full-duplex, lossless connections of 1 Gbit/s capacity in both directions. Similarly, the default configuration for external communications is full-duplex connections of 100 Mbit/s capacity and 0.1% packet loss with random bursts of 1% packet loss. These numbers are based on measurements on enterprise and wide-area networks; see e.g., (Kushida & Shibata, 2002; Paxson, 1997).

The network conditions are configured in CSLE through Python objects. We provide an example in Listing 2.

```
from csle_common.dao.emulation_config.
    node_network_config import NodeNetworkConfig
NodeNetworkConfig(interface="eth0", packet_delay_ms=2,
    jitter_ms=0.5, delay_distribution="pareto", corrupt
    =0.02, duplicate=0.00001, correlation=25, reorder
    =2, rate_limit_mbit=100)
```

Listing 2. Python code for configuring a network interface.

Emulating actors

All actors in CSLE are programmatically controlled through a management API based on GRPC, which allows changing configuration parameters, starting new actors, and stopping running ones. This automation enables attackers, clients, and defenders to operate in a fully autonomous environment.

We emulate clients through processes in the digital twin that access services on emulated hosts. The client population is defined by (*i*) an arrival process (e.g., a Poisson process) that controls the rate at which new client processes are started; (*ii*) a service time distribution (e.g., an exponential distribution) that controls how long a client will consume services before terminating; (*iii*) a service configuration that specifies the services of the digital twin that clients will consume; and (*iv*) a Markov process that controls the sequence of service invocations that a client makes. All of these parameters are configured in CSLE through a Python file. We provide a code snippet of this file in Listing 3.

```
from csle_collector.client_manager.dao.client import
    Client
from csle_collector.client_manager.dao.
    constant_arrival_config import
    ConstantArrivalConfig
clients=[Client(service_distribution=[0.5,0.2,0.3],
    arrival_config=ConstantArrivalConfig(lamb=20), mu
    =4, exponential_service_time=True)]
```

Listing 3. Python code for configuring the client population.

Figure 6 shows the resource usage of two digital twins as a function of the client arrival rate. We observe, as expected, that the resource usage increases with the load imposed on the twins. In particular, higher client arrival rates lead to increased CPU utilization since the twins must process a larger number of service requests. In contrast, the memory usage remains stable when increasing the load.

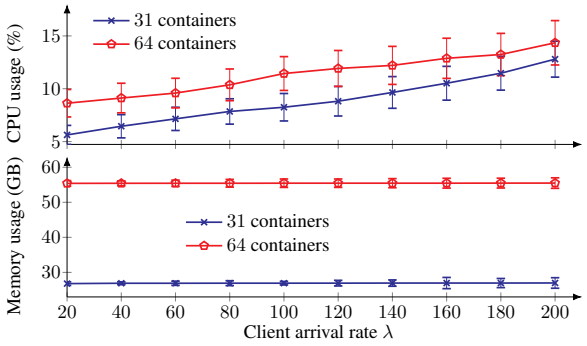


Figure 6. Resource usage of two digital twins in function of the client (Poisson) arrival rate λ. Numbers and error bars indicate the mean and the standard deviation from 5 evaluations. The CPU and memory usages are averaged over a monitoring period of 30 minutes. The blue curves relate to digital twins of an IT infrastructure with 31 and 64 hosts, respectively. The network topologies are shown in Figs. 9.a–b and the configurations are available in the supplementary material (Tables 5 and 6). Each client consumes a randomly selected service of the infrastructure for a time that is sampled from an exponential distribution with mean value $\mu = 60$ seconds. We run the digital twins on a server with a 24-core Intel Xeon Gold 2.10 GHz CPU and 768 GB RAM.

Similar to how clients are emulated, attackers in CSLE are implemented as autonomous processes that execute actions from a pre-defined list, including reconnaissance commands, privilege escalation actions, and exploits. Table 2 lists some of the attacker actions that are automated in CSLE. The defender is emulated in a similar way, with actions implemented as system commands that can reconfigure network components, isolate hosts, or perform other mitigation steps. We provide several examples of defender actions in Table 3.

5.3 The Management System

The role of the management system in CSLE is to support the operation of digital twins and facilitate end-to-end reinforcement learning experiments. In particular, the management system provides APIs for real-time monitoring and control of digital twins, as well as a web interface for managing reinforcement learning experiments and deployments.

Each emulated device in a digital twin runs a *management agent*, which exposes a GRPC API (Google, 2022). This API is invoked to perform control actions, e.g., restarting services and updating configurations. The communication

Type	*Actions*	MITRE ATT&CK technique
Reconnaissance	TCP SYN scan, UDP scan	T1046 service scanning.
	TCP XMAS scan	T1046 service scanning.
	Vulscan	T1595 active scanning.
	ping-scan	T1018 system discovery.
Brute-force	Telnet, SSH	T1110 brute force.
	FTP, Cassandra	T1110 brute force.
	IRC, MongoDB, MySQL	T1110 brute force.
	SMTP, Postgres	T1110 brute force.
Exploit	CVE-2017-7494	T1210 service exploitation.
	CVE-2015-3306	T1210 service exploitation.
	CVE-2010-0426	T1068 privilege escalation.
	CVE-2015-5602	T1068 privilege escalation.
	CVE-2015-1427	T1210 service exploitation.
	CVE-2014-6271	T1210 service exploitation.
	CVE-2016-10033	T1210 service exploitation.
	SQL injection	T1210 service exploitation.

Table 2. Examples of attacker actions in CSLE; actions are identified by identifiers in the common vulnerabilities and exposures (CVE) database (The MITRE Corporation, 2022); the actions are also linked to the corresponding attack techniques in the MITRE ATT&CK taxonomy (Strom et al., 2018).

channels to the agents are provided by a *management network*. The reason for using a separate network to carry management traffic is to avoid interference and simplify control of the digital twin (Clemm & Cisco Systems, 2007).

We provide an example of using the management system to execute control actions inside a digital twin in Listing 4. To complement the Python APIs, the management system also includes a web interface and a command-line interface, both of which provide the same functions as the Python API. A video demonstration of the management system is available at (Hammar, 2023) and screenshots of the web interface are provided in Appendix A in the supplementary material.

```
from csle_common.metastore.metastore_facade import
    MetastoreFacade
from csle_common.util.emulation_util import
    EmulationUtil
twin=MetastoreFacade.get_twin(name="")
EmulationUtil.execute_ssh(cmds=[cmd], ip="", twin=twin)
```

Listing 4. Python code for executing a control action.

To monitor processes and services running inside the digital twin, we use a monitoring system based on a publish-subscribe architecture; see Fig. 7. Following this architecture, each emulated device in a digital twin runs a *monitoring agent*, which reads local metrics of the host and pushes those metrics to an event bus implemented with Kafka (Kreps, 2011). The data in this bus is consumed by data pipelines, which process the data and write it to storage systems. In particular, the data is exported to an Elasticsearch database that can be queried and visualized through Kibana dashboards for real-time monitoring.

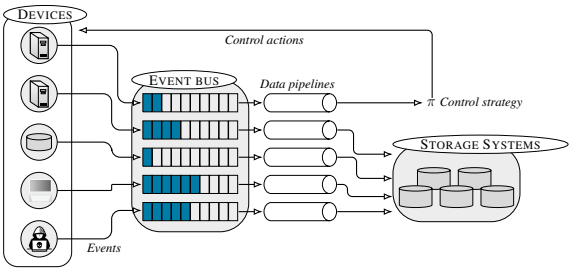


Figure 7. Monitoring system of a digital twin in CSLE. Emulated devices run monitoring agents that periodically push metrics to an event bus, which is consumed by pipelines that process the data and write to storage systems; the processed data is also used as input to automated control strategies to decide on control actions.

Action	MITRE D3FEND technique
Revoke user certificates	D3-CBAN certificate revocation.
Blacklist IPs	D3-NTF network traffic filtering.
Drop traffic	D3-NTF network traffic filtering.
Block gateway	D3-NI network isolation.
Migrate servers between zones	D3-NI network isolation.
Redirect traffic	D3-NTF network traffic filtering.
Isolate a server	D3-NI network isolation.
Deploy new security functions	D3-NTPM network policy mapping.
Shutdown a server	D3-HS host shutdown.
Replicate a service	D3-SVCDM service mapping.
Start decoy services	D3-D3 decoy environment.

Table 3. Examples of defender actions in CSLE; the actions are linked to the corresponding defense techniques in the MITRE D3FEND taxonomy (Kaloroumakis & Smith, 2021).

Figure 8 shows performance statistics related to the monitoring system. In particular, Fig. 8.a shows that the CPU overhead introduced by the monitoring agents is around 6%, while the memory overhead is approximately 1%. Both values can be considered relatively low. Furthermore, Fig. 8.b shows that the number of monitoring events produced by the monitoring agents per monitoring interval increases with the size of the digital twin and also depends on the specific system configuration. Larger twins typically contain more monitored components, which naturally results in a higher number of generated events. In addition, the event rate is influenced by the types and number of monitoring mechanisms deployed. For example, configurations that include a larger number of intrusion detection systems (as is the case for the system in Fig. 9.b) generate more monitoring events.

5.4 The Simulation System

The simulation system in CSLE is implemented in Python and consists of reinforcement learning environments and algorithms for learning security strategies. All environments follow the OpenAI Gym interface (Towers et al., 2024), which allows integration with standard reinforcement learning frameworks. Each environment defines a Markov decision process or a game and can be configured through Python configuration files. CSLE includes an initial set of 34 reinforcement learning algorithms, over 50 simulation environments, and 4 identification algorithms. We provide an example of using the simulation system to run a reinforcement learning algorithm in Listing 5.

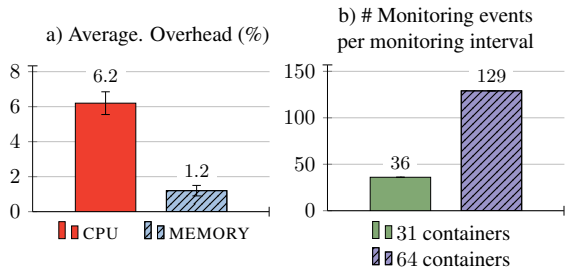


Figure 8. Statistics of the monitoring system in CSLE. Plot a) shows the average overhead of a monitoring agent and plot b) shows the number of monitoring events per monitoring interval for two digital twins deployed with CSLE. The network topologies of the digital twins with 31 and 64 containers are shown in Fig. 9.a and Fig. 9.b, respectively. Numbers and error bars indicate the mean and the standard deviation from 5 evaluations.

```
from csle_agents.agents.sarsa.sarsa_agent import
    SARSAAgent
from csle_common.metastore.metastore_facade import
    MetastoreFacade
from csle_common.dao.training.experiment_config import
    ExperimentConfig
simulation = MetastoreFacade.get_simulation(..)
experiment = ExperimentConfig(..)
agent = SARSAAgent(simulation, experiment)
execution = agent.train()
MetastoreFacade.save_experiment_execution(execution)
for strategy in execution.result.strategies.values():
    MetastoreFacade.save_strategy(strategy)
```

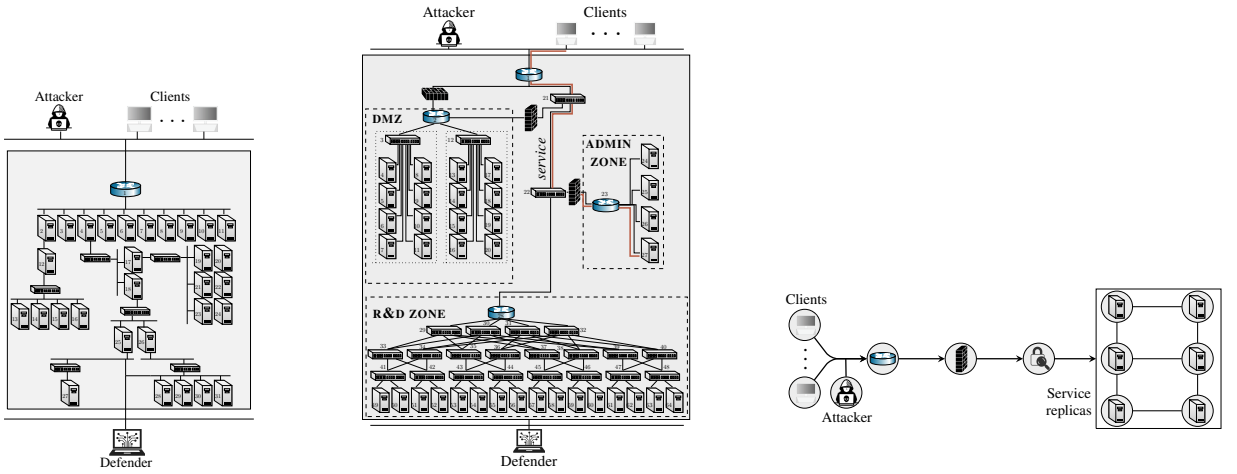
Listing 5. Python code for running the SARSA reinforcement learning algorithm in the simulation system.

6 Example Use Cases

We demonstrate CSLE by applying it to four different security use cases. Each use case involves a target system and a system operator, which we refer to as the *defender*; see Fig. 9. (The detailed system configurations are available in Appendix B.) The use cases are described below.

6.1 Flow Control

This use case involves an IT system that provides services to clients through a public gateway; see Fig. 9.a. While the gateway enables legitimate access for clients, it also exposes an entry point for potential attackers attempting to intrude on the system and compromise components. To protect the system against such intrusions, the defender continuously monitors network traffic and analyzes security



a) Target system for the flow control use case. **b)** Target system for the segmentation use case. **c)** Target system for the replication and recovery use cases.

Figure 9. Target systems for the use cases in the experimental evaluation. The system configurations are available in Appendix B.

alerts to identify suspicious or malicious activity. Based on these observations, the defender can control network flows to mitigate potential network intrusions. For example, the defender can block suspicious flows or redirect them to a honeypot. When making these decisions, the defender aims to balance the dual objectives of preserving service availability for clients and mitigating potential attacks.

6.2 Network Segmentation Control

This use case involves a cloud infrastructure that is segmented into zones with virtual nodes that run network services; see Fig. 9.b. Services are realized by *workflows* that clients access through a cloud gateway, which is also open to an attacker. The attacker aims to intrude on the infrastructure, compromise nodes, and disrupt workflows. To counter these threats, the defender continuously monitors the infrastructure by accessing and analyzing intrusion detection alerts and other statistics. Based on this information, the defender can respond to possible intrusions by changing the network segmentation. For example, the defender can migrate nodes between zones, change access controls, or shut down nodes. When deciding between these actions, the defender balances two conflicting objectives: maximizing workflow utility towards clients and minimizing the operational cost of possible intrusions and defensive actions.

6.3 Recovery Control

This use case involves a replicated system that provides a web service to a client population; see Fig. 9.c. Because multiple replicas deliver the same service, the system can continue operating even when some replicas are compromised. To track the evolving security status of the system, the defender analyzes security alerts that indicate potential compromises of service replicas. Based on these observations, the defender decides when and which replicas to recover in order to maintain service availability. When making these recovery decisions, the goal is to ensure that compromised service replicas are recovered faster than new compromises occur while minimizing the recovery costs.

6.4 Replication Control

In this use case, we study the problem of learning adaptive replication strategies for the system illustrated in Fig. 9.c. The goal is to enable the system to autonomously adjust the number of replicas in response to changing security and performance conditions. In particular, the defender observes indicators of replica failures or degradations (e.g., security alerts) and uses these signals to dynamically decide when to launch new replicas or retire existing ones. By adapting the replication factor, the system can maintain service availability even under fluctuating attack intensities or workload demands. The key challenge is to minimize the number of replicas (to reduce operational costs) while still satisfying service availability and reliability constraints.

7 System Models

For each of the use cases described above, we consider two different reinforcement learning problems. First, we consider the problem of learning an optimal control strategy against an attacker that follows a fixed strategy. We model this problem as a Markov decision process (MDP) or a partially observed MDP (POMDP), depending on whether the system state is observable. Second, we consider the problem of learning a control strategy that is effective against

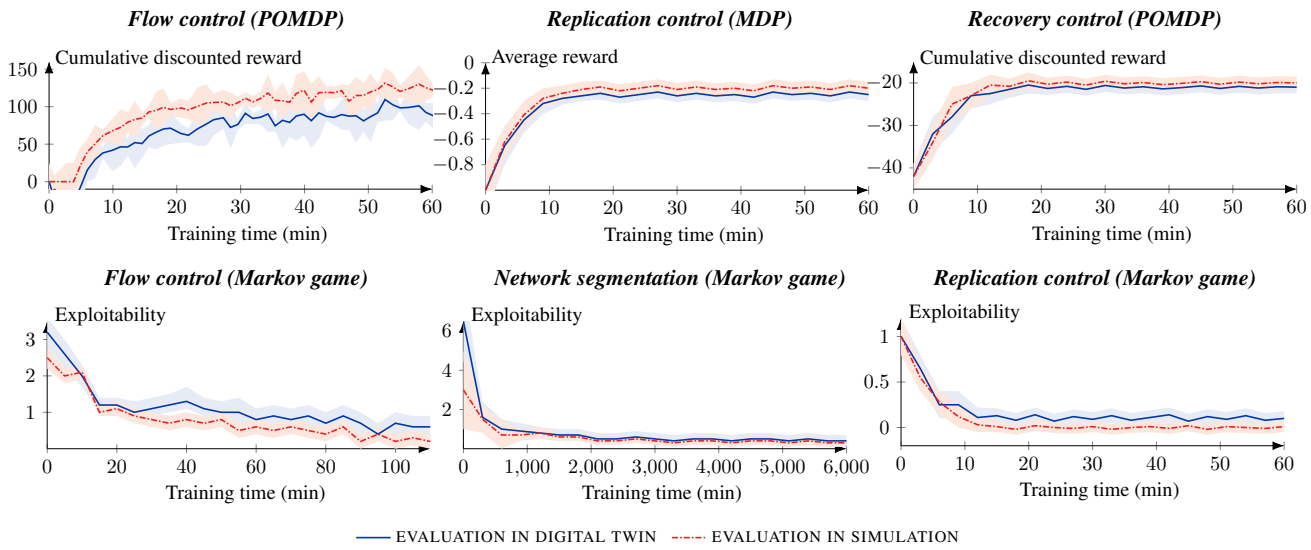


Figure 10. Convergence curves for the different use cases. The red curves relate to the performance in the simulations and the blue curves relate to the performance when evaluating the learned strategies in the digital twins. Curves show the mean values from evaluations with 5 random seeds; shaded areas indicate standard deviations. The x-axes indicate the training times in the simulations. The top row relates to simulations of decision-theoretic models. The bottom row relates to simulations of game-theoretic models.

a *dynamic* attacker that adapts its strategy to circumvent the defenses. We model this problem as a Markov game. In total, we consider six reinforcement learning problems, which are listed in Table 4. We address each problem using the general reinforcement learning methodology described in §4. Mathematical formulations are given in Appendix D.

Model	*Algorithm*
Flow control POMDP	SPSA (Spall, 1992)
Replication control MDP	PPO (Schulman et al., 2017)
Recovery control POMDP	Rollout (Bertsekas, 2021)
Flow control game	Fictitious play with SPSA (Brown, 1951)
Segmentation game	Fictitious play with PPO (Brown, 1951)
Replication control game	PPO (Schulman et al., 2017)

Table 4. The reinforcement learning problems that we consider in the experimental evaluation and the algorithms that we use to address them. In the decision-theoretic problems (i.e., the MDP and POMDPs), the goal is to learn an optimal security strategy against a fixed attacker strategy, whereas in the game-theoretic problems, the goal is to learn an equilibrium strategy against an attacker that dynamically adapts its strategy to the defender's strategy.

8 Experimental Evaluation of CSLE

In this section, we present our experimental results. Following the methodology described in §4, we identify the parameters of each system model described in the preceding section (e.g., the MDP, POMDP, or game parameters) based on data collected from the digital twin. Then, given the identified model, we learn the security strategy through simulation, after which we evaluate the learned strategy in the digital twin. The reinforcement learning algorithms that we use are listed in Table 4. For details about the system identification and the collected data, see Appendix D.

8.1 Experimental Setup

We run the reinforcement learning algorithms until convergence and evaluate the learned strategies periodically during training, both in the simulation and in the digital twin. The hyperparameters that we use to instantiate the algorithms are listed in Appendix C. The environment for training strategies and running simulations is a Tesla P100 GPU. The digital twins are deployed on a server with a 24-core Intel Xeon Gold 2.10 GHz CPU and 768 GB RAM.

We consider two evaluation metrics: the reward and the exploitability. The reward measures the overall performance of the learned strategy in terms of its ability to achieve the defender's objective in each use case. For the definitions of the reward functions, see Appendix D. The exploitability, on the other hand, measures the distance of the learned strategies from a Nash equilibrium, where an exploitability of 0 means that the strategies are in equilibrium.

Baselines

We compare the reinforcement learning algorithms listed in Table 4 against two baseline methods, namely phasic policy gradient (PPG) (Cobbe et al., 2021) and neural fictitious

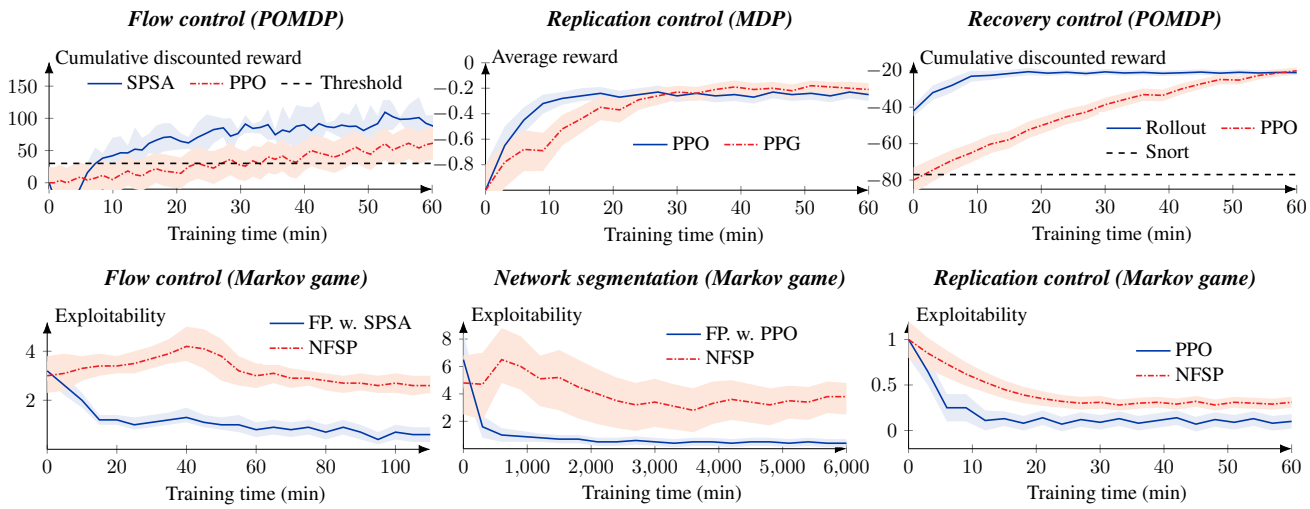


Figure 11. Performance comparison between reinforcement learning methods and baseline strategies in the digital twin. Curves show the mean values from evaluations with 5 random seeds; shaded areas indicate standard deviations. The top row relates to the decision-theoretic models. The bottom row relates to the game-theoretic models. The acronym FP stands for fictitious play.

self-play (NFSP) (Heinrich & Silver, 2016). Additionally, we compare the performance of the reinforcement learning methods with that of two static security strategies: a threshold flow control strategy and a recovery strategy based on the Snort intrusion detection and prevention system with community ruleset v2.9.17.1 (Roesch, 1999).

The threshold strategy blocks network flows when the defender's belief (probability) that the system is compromised exceeds a predefined threshold $\alpha = 0.75$. The belief is computed according to the POMDP model, which maintains a probabilistic estimate of the underlying system state based on the sequence of observed system events. When the belief that the system is in a compromised state surpasses the threshold, the strategy proactively blocks network flows in order to limit potential attacker movement and prevent further propagation of the compromise.

The Snort baseline follows a rule-based recovery strategy that represents a signature-based response mechanism. This strategy recovers a component of the target system (e.g., by redeploying it in a new virtual machine) when a Snort alert with priority medium or higher is generated.

8.2 Evaluation Results

The evaluation results are summarized in Figs. 10–11. The red and blue curves in Fig. 10 represent the results from the simulator and the digital twin, respectively. An analysis of these curves leads us to the following conclusions. The learning curves converge to nearly constant mean values for all use cases and evaluation metrics. From this observation, we conclude that the learned strategies have also converged.

Although the learned strategies, as expected, perform slightly better on the simulator than on the digital twin, we are encouraged by the fact that the curves of the digital twin are close to those of the simulator (cf. the blue and red curves). This small performance gap reflects inevitable discrepancies between the simulation model and the digital twin, such as differences in network latency, background processes, or unmodeled system dynamics.

Figure 11 shows a comparison between different reinforcement learning methods and baseline security strategies. We observe that the performance varies substantially. Overall, the results show that the reinforcement learning strategies significantly outperform the static strategies.

8.3 Discussion

The experimental results demonstrate that CSLE effectively enables the transfer of reinforcement learning-based security strategies from simulation to a digital twin that closely approximates an operational environment. The small performance gap observed between the simulator and the digital twin indicates that the identified simulation models capture the main dynamics of the target system. This transferability is a key step toward operational deployment, as it validates that security strategies learned in simulation remain effective when tested under realistic operating conditions.

For safety and operational reasons, we have not evaluated the learned strategies in a production environment. While the digital twin replicates the software, configuration, and timing behavior of a production environment, further study is needed to determine whether the learned strategies main-

tain comparable performance in a production environment.

8.4 Sensitivity to Model Misspecification

The effective transfer of the learned strategies from simulation to the digital twin indicates that the identified simulation dynamics capture the main characteristics of the target system. However, in practice, model misspecification may still arise due to factors such as measurement noise or changes in system behavior over time (e.g., data drift). To better understand the impact of such modeling inaccuracies, in this section, we analyze the sensitivity of the learned strategies to the misspecification of the system model.

From a theoretical perspective, the performance loss of learned strategies due to deviations between the model and the system dynamics is upper-bounded as follows.

Proposition 8.1 (Model misspecification error bound)**.** *Let $\tilde{f}$ denote the dynamics of the system model [cf. (1)] and let f denote the dynamics of the target system. Denote by $J_{\boldsymbol{\pi}}$ and $\tilde{J}_{\boldsymbol{\pi}}$ the value functions (i.e., the expected rewards) under a strategy pair $\boldsymbol{\pi} = (\pi_{\mathrm{D}}, \pi_{\mathrm{A}})$ in the target system and in the simulation, respectively. If the dynamics satisfy*

$$\sum_{s' \in \mathcal{S}} \left| f(s' \mid s, a^{(\mathrm{D})}, a^{(\mathrm{A})}) - \tilde{f}(s' \mid s, a^{(\mathrm{D})}, a^{(\mathrm{A})}) \right| \leq \alpha,$$

for all states and actions, and some constant α. Then

$$\|J_{\boldsymbol{\pi}} - \tilde{J}_{\boldsymbol{\pi}}\|_{\infty} \leq \frac{\alpha\gamma\beta}{(1-\gamma)^2},$$

where $\gamma < 1$ is the discount factor and β is defined by

$$\beta = \max_{s \in \mathcal{S}, a^{(\mathrm{D})} \in \mathcal{A}_{\mathrm{D}}, a^{(\mathrm{A})} \in \mathcal{A}_{\mathrm{A}}} |r(s, a^{(\mathrm{D})}, a^{(\mathrm{A})})|,$$

where r is the reward function, $\mathcal{S}$ is the state space, and $(\mathcal{A}_{\mathrm{D}}, \mathcal{A}_{\mathrm{A}})$ are the action spaces.

We present the proof of Prop. 8.1 in the supplementary material (Appendix E). This proposition states that the misspecification error grows proportionally with the error of the discrepancy between the state transitions in the system model and the digital twin, as quantified by the parameter α. In practice, this parameter can be estimated by comparing the simulated state trajectories of the model and the trajectories observed in the digital twin.

While Prop. 8.1 provides a worst-case bound on the impact of model misspecification, it is conservative and does not necessarily reflect the sensitivity of a specific system instance. To complement the theoretical analysis, we therefore conduct an empirical sensitivity analysis on the flow control model used in our experiments. In this model, the probability that the attacker successfully compromises the system is governed by a parameter $p \in [0, 1]$; see Appendix D.1 in the supplementary material for details. In the digital twin, we configure this parameter as $p = 0.01$. To introduce model misspecification, we vary p in the simulation model and evaluate how the performance of the learned defender strategy changes as the discrepancy between the simulation model and the digital twin increases. The results of this sensitivity analysis are summarized in Fig. 12.

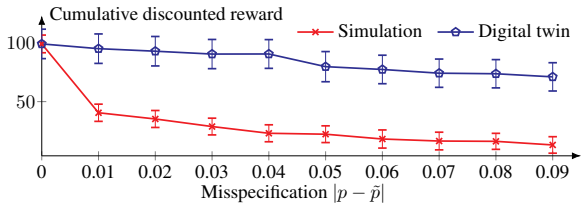


Figure 12. Analysis of the sensitivity to model misspecification in the flow control use case. Numbers and error bars indicate the mean and the standard deviation from 5 evaluations.

Figure 12 shows that small modeling errors lead to noticeable differences between the performance predicted by the simulation and the performance observed in the digital twin. However, these discrepancies in the simulated performance translate only to small variations in the performance of the learned defender strategy when it is evaluated in the digital twin. This indicates that, although the model is sensitive to misspecification when estimating performance, the learned strategy itself is relatively robust to such errors.

9 Conclusion

In this paper, we present CSLE, a comprehensive research platform for autonomous security management through reinforcement learning. This platform addresses the system-level challenges that arise in the operation and experimentation with reinforcement learning in networked systems. In particular, it is based on a novel methodology for learning security strategies that combines a digital twin with system identification and simulation-based reinforcement learning. Our evaluation across four security use cases demonstrates that this methodology narrows the gap between simulated and practical performance of learned security strategies.

Future Work

Future work will focus on further developing the CSLE platform and expanding its open-source ecosystem. We plan to continue improving the platform's usability by adding more learning resources, documentation, and example configurations to facilitate adoption by both researchers and practitioners. So far, our experimental validation of CSLE has focused primarily on IT systems. In future work, we aim to extend CSLE to cyberphysical systems.

REFERENCES

Akbari, I., Tahoun, E., Salahuddin, M. A., Limam, N., and Boutaba, R. ATMoS: Autonomous threat mitigation in SDN using reinforcement learning. In *NOMS IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2020. doi: 10.1109/NOMS47738.2020.9110426.

Allen B. Danette. Digital twins and living models at NASA, 2021. Digital Twin Summit.

Anderson, H. S., Kharkar, A., Filar, B., Evans, D., and Roth, P. Learning to evade static PE machine learning malware models via reinforcement learning, 2018. URL https://arxiv.org/abs/1801.08917.

Andrew, A., Spillard, S., Collyer, J., and Dhir, N. Developing optimal causal cyber-defence agents via cyber security simulation. In *Proceedings of the ML4Cyber workshop, ICML 2022, Baltimore, USA, July 17-23, 2022*. PMLR, 2022.

Bertsekas, D. P. *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Athena scientific optimization and computation series. Athena Scientific., 2021. ISBN 9781886529076.

Biesinger, F. and Weyrich, M. The facets of digital twins in production and the automotive industry. In *2019 23rd International Conference on Mechatronics Technology (ICMT)*, pp. 1–6, 2019. doi: 10.1109/ICMECT.2019.8932101.

Blum, W. Gamifying machine learning for stronger security and AI models, 2021.

Brown, G. W. Iterative solution of games by fictitious play, 1951. Activity analysis of production and allocation.

Carrasco, J. A. F., Pagola, I. A., Urrutia, R. O., and Roman, R. CYBERSHIELD: A competitive simulation environment for training AI in cybersecurity. In *2024 11th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pp. 11–18, 2024. doi: 10.1109/IOTSMS62296.2024.10710208.

Chen, Y., Shetty, M., Somashekar, G., Ma, M., Simmhan, Y., Mace, J., Bansal, C., Wang, R., and Rajmohan, S. AIOpsLab: a holistic framework to evaluate AI agents for enabling autonomous clouds. https://mlsys.org/virtual/2025/poster/3285, 2025. MLSys 2025 Poster.

Clemm, A. and Cisco Systems, I. *Network Management Fundamentals*. Cisco Press fundamentals series. Cisco Press, 2007. ISBN 9781587201370.

Cobbe, K. W., Hilton, J., Klimov, O., and Schulman, J. Phasic policy gradient. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2020–2027. PMLR, 18–24 Jul 2021.

Cubukcu, U. et al. Citus: Distributed PostgreSQL for data-intensive applications. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, 2021.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39: 1–38, 1977. URL http://web.mit.edu/6.435/www/Dempster77.pdf.

Emerson, H., Bates, L., Hicks, C., and Mavroudis, V. CybORG++: An enhanced gym for the development of autonomous cyber agents, 2024. URL https://arxiv.org/abs/2410.16324.

Glivenko, V. I. and Cantelli, F. P. Sulla determinazione empirica delle leggi di probabilità, 1933. Giorn. Ist. Ital. Attuari (in Italian). 4: 92–99.

Google. Google remote procedure call, 2022.

Hammar, K. Cyber security learning environment (CSLE), 2023. URL https://kim-hammar.github.io/csle/. Documentation: https://kim-hammar.github.io/csle/, traces: https://github.com/Kim-Hammar/csle/releases/tag/v0.4.0, source code: https://github.com/Kim-Hammar/csle, video demonstration: https://www.youtube.com/watch?v=iE2KPmtIs2A&.

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

Hammar, K. and Stadler, R. Intrusion prevention through optimal stopping. *IEEE Transactions on Network and Service Management*, 19(3):2333–2348, 2022. doi: 10.1109/TNSM.2022.3176781.

Hammar, K. and Stadler, R. Scalable learning of intrusion response through recursive decomposition. In Fu, J., Kroupa, T., and Hayel, Y. (eds.), *Decision and Game Theory for Security*, pp. 172–192, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-50670-3.

Hammar, K. and Stadler, R. Intrusion tolerance for networked systems through two-level feedback control. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 338–352, 2024a. doi: 10.1109/DSN58291.2024.00042.

Hammar, K. and Stadler, R. Learning near-optimal intrusion responses against dynamic attackers. *IEEE Transactions on Network and Service Management*, 21(1):1158–1177, 2024b. doi: 10.1109/TNSM.2023.3293413.

Hammar, K. and Stadler, R. Intrusion tolerance as a two-level game. In Sinha, A., Fu, J., Zhu, Q., and Zhang, T. (eds.), *Decision and Game Theory for Security*, pp. 3–23, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-74835-6.

Hammar, K., Li, Y., Alpcan, T., Lupu, E. C., and Bertsekas, D. Adaptive network security policies via belief aggregation and rollout, 2025. URL https://arxiv.org/abs/2507.15163.

Heinrich, J. and Silver, D. Deep reinforcement learning from self-play in imperfect-information games, 2016. URL https://arxiv.org/abs/1603.01121.

Hemminger, S. Network emulation with NetEm. *Linux Conf*, 2005.

Huynh Phuong Thanh, N., Chen, Z., Hasegawa, K., Fukushima, K., and Beuran, R. PenGym: Pentesting training framework for reinforcement learning agents. pp. 498–509, 01 2024. doi: 10.5220/0012367300003648.

ISC2. 2024 Cybersecurity workforce study. Technical report, ISC2, October 2024. URL https://www.isc2.org/Insights/2024/10/ISC2-2024-Cybersecurity-Workforce-Study. Based on an online survey of 15,852 cybersecurity professionals conducted in April–May 2024 in collaboration with Forrester Research Inc.; accessed 2025-09-07.

Janisch, J., Pevný, T., and Lisý, V. NASimEmu: Network attack simulator & emulator for training agents generalizing to novel scenarios, 2023.

Jha, S., Arora, R. R., Watanabe, Y., Yanagawa, T., Chen, Y., Clark, J., Bhavya, B., Verma, M., Kumar, H., Kitahara, H., Zheutlin, N., Takano, S., Pathak, D., George, F., Wu, X., Turkkan, B. O., Vanloo, G., Nidd, M., Dai, T., Chatterjee, O., Gupta, P., Samanta, S., Aggarwal, P., Lee, R., Ahn, J.-W., Kar, D., Paradkar, A., Deng, Y., Moogi, P., Mohapatra, P., Abe, N., Narayanaswami, C., Xu, T., Varshney, L. R., Mahindru, R., Sailer, A., Shwartz, L., Sow, D., Fuller, N. C. M., and Puri, R. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., and Zhu, J. (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 27134–27197. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/jha25a.html.

Kaloroumakis, P. E. and Smith, M. J. Toward a knowledge graph of cybersecurity countermeasures. *The MITRE Corporation*, 11:2021, 2021.

Kearns, M. and Singh, S. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49 (2):209–232, Nov 2002. ISSN 1573-0565. doi: 10.1023/A:1017984413808. URL https://doi.org/10.1023/A:1017984413808.

Kiely, M., Ahiskali, M., Borde, E., Bowman, B., et al. CAGE Challenge 4: a scalable multi-agent reinforcement learning gym for autonomous cyber defence. *AI Magazine*, 46(e70021), 2025. doi: 10.1002/aaai.70021.

Kreps, J. Kafka : a distributed messaging system for log processing. 2011.

Kushida, T. and Shibata, Y. Empirical study of inter-arrival packet times and packet losses. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pp. 233–240, 2002. ISBN 0769515886.

Lanier, M. and Vorobeychik, Y. CyGym: a simulation-based game-theoretic analysis framework for cybersecurity, 2025. URL https://arxiv.org/abs/2506.21688.

Li, L., Fayad, R., and Taylor, A. CyGIL: A cyber gym for training autonomous agents over emulated network systems, 2021.

Li, T., Hammar, K., Stadler, R., and Zhu, Q. Conjectural online learning with first-order beliefs in asymmetric information stochastic games. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 6780–6785, 2024. doi: 10.1109/CDC56724.2024.10886479.

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

Mahalingam, M. et al. Virtual eXtensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks, 2014. URL https://www.rfc-editor.org/rfc/rfc7348.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, pp. 69–74, mar 2008. ISSN 0146-4833. doi: 10.1145/1355734.1355746. URL https://doi.org/10.1145/1355734.1355746.

Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014 (239):2, 2014.

Molina-Markham, A., Miniter, C., Powell, B., and Ridley, A. Network environment design for autonomous cyberdefense. 2021. https://arxiv.org/abs/2103.07583.

Nguyen, T. T. and Reddi, V. J. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3779–3795, 2023. doi: 10.1109/TNNLS.2021.3121870.

Oakley, L. and Oprea, A. QFlip: An adaptive reinforcement learning strategy for the FlipIt security game. In Alpcan, T., Vorobeychik, Y., Baras, J. S., and Dán, G. (eds.), *Decision and Game Theory for Security*, pp. 364–384, Cham, 2019. Springer International Publishing. ISBN 978-3-030-32430-8.

Oesch, S., Chaulagain, A., Weber, B., Dixson, M., Sadovnik, A., Roberson, B., Watson, C., and Austria, P. Towards a high fidelity training environment for autonomous cyber defense agents. In *Proceedings of the 17th Cyber Security Experimentation and Test Workshop*, CSET '24, pp. 91–99, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400709579. doi: 10.1145/3675741.3675752. URL https://doi.org/10.1145/3675741.3675752.

Paxson, V. End-to-end internet packet dynamics. In *IEEE/ACM Transactions on Networking*, pp. 277–292, 1997.

Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. The design and implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 117–130, Oakland, CA, May 2015. USENIX Association. ISBN 978-1-931971-218. URL https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff.

Red Hat. Ansible, 2024.

Roesch, M. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pp. 229–238, USA, 1999. USENIX Association.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. 2017. URL http://arxiv.org/abs/1707.06347.

Schwartz, J., Kurniawati, H., and El-Mahassni, E. POMDP + information-decay: Incorporating defender's behaviour in autonomous penetration testing. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1):235–243, Jun. 2020. URL https://ojs.aaai.org/index.php/ICAPS/article/view/6666.

Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D., and Yin, H. MAB-Malware: A reinforcement learning framework for blackbox generation of adversarial malware. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, pp. 990–1003, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391405. doi: 10.1145/3488932.3497768. URL https://doi.org/10.1145/3488932.3497768.

Spall, J. C. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 37(3): 332–341, 1992.

Standen, M., Lucas, M., Bowman, D., Richer, T. J., Kim, J., and Marriott, D. CybORG: A gym for the development of autonomous cyber agents. *CoRR*, 2021. https://arxiv.org/abs/2108.09118.

Strom, B. E., Applebaum, A., Miller, D. P., Nickels, K. C., Pennington, A. G., and Thomas, C. B. MITRE ATT&CK: Design and philosophy. In *Technical report*. The MITRE Corporation, 2018.

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

Terranova, F., Lahmadi, A., and Chrisment, I. Scalable and Generalizable RL Agents for Attack Path Discovery via Continuous Invariant Spaces. In *2025 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pp. 18, Gold Coast, Australia, October 2025. URL https://hal.science/hal-05182437.

The MITRE Corporation. CVE database, 2022. URL https://cve.mitre.org/.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., Cola, G. D., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., and Younis, O. G. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.

Wu, Y., Zhang, K., and Zhang, Y. Digital twin networks: A survey. *IEEE Internet of Things Journal*, 8(18):13789–13804, 2021. doi: 10.1109/JIOT.2021.3079510.