Version 0.5.0

Kim Hammar

March 2, 2024

Contents

1 The Cyber Security Learning Environment (CSLE)

This section contains an overview of the Cyber Security Learning Environment (CSLE), its purpose, its origin, and its organization.

1.1 What is CSLE?

CSLE is a framework for evaluating and developing reinforcement learning agents for control problems in cyber security. Everything from network emulation, to simulation, and learning in CSLE have been co-designed to provide an environment where it is possible to train and evaluate reinforcement learning agents for practical cyber security tasks.

CSLE is an opinionated framework, which is based on a specific method for learning and evaluating security strategies for a given IT infrastructure (see Fig. 1). This method includes two systems: an emulation system and a simulation system.

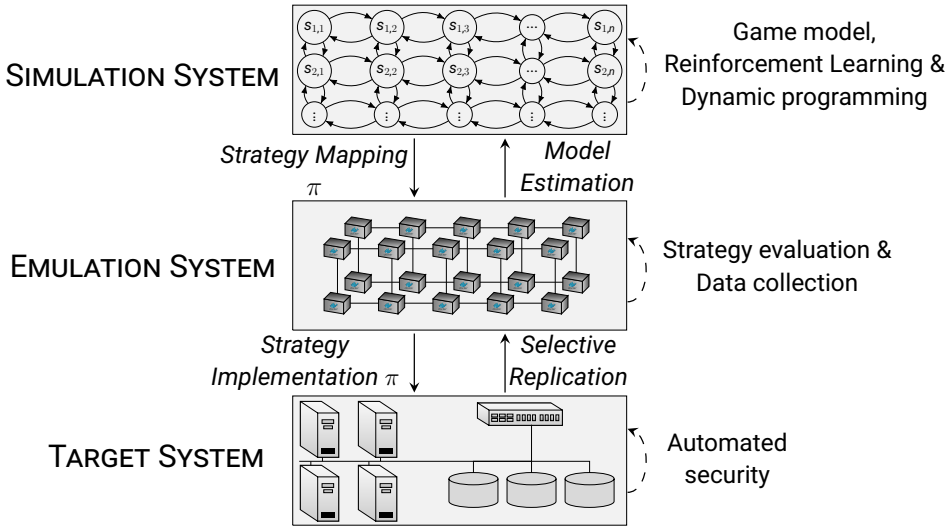

Figure 1: The method used to automatically find effective security strategies in CSLE.

The emulation system closely approximates the functionality of the target infrastructure and is used to run attack scenarios and defender responses. Such runs produce system measurements and logs, from which infrastructure statistics are estimated, which then are used to instantiate Markov decision processes (MDPs).

The simulation system is used to simulate the instantiated MDPs and to learn security strategies through reinforcement learning. Learned strategies

are extracted from the simulation system and evaluated in the emulation system.

Three benefits of this method are: (*i*) that the emulation system provides a realistic environment to evaluate strategies; (*ii*) that the emulation system allows evaluating strategies without affecting operational workflows on the target infrastructure; and (*iii*) that the simulation system enables efficient and rapid learning of strategies.

1.2 Why CSLE?

As the ubiquity and evolving nature of cyber attacks is of growing concern to society, *automation* of security processes and functions has been recognized as an important part of the response to this threat. A promising approach to achieve this automation is *reinforcement learning*, which has proven effective in finding near-optimal solutions to control problems in several domains (e.g., robotics [29] and industry automation [32]), and is actively investigated as an approach to automated security (see survey [13]). While encouraging results have been obtained in this line of research, key challenges remain. Chief among them is narrowing the gap between the environment where the reinforcement learning agents 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. Another limitation of prior research is the lack of common benchmarks and toolsets.

CSLE was developed to address precisely the above limitations. By using high-fidelity emulations, it narrows the gap between the evaluation environment and a real system, and by being open-source, it provides a foundation for further research to build on.

Recently, efforts to build similar frameworks as CSLE have started (see [13] for a review). Most notably, there is CyberBattleSim by Microsoft [3], CyBorg by the Australian department of defence [47], Yawning Titan by the UK Defence Science and Technology Laboratory (DSTL) [1], CyGIL by Defence Research and Development Canada (DRDC) [33], NASimEmu by the Czech Technical University in Prague [28], and FARLAND, which is developed at USA's National Security Agency (NSA) [36]. Some of these frameworks only include simulation systems and some of them include both simulation systems and emulation systems. In contrast to these frameworks, CSLE is fully open-source, includes both a simulation system and an emulation system, and has been thoroughly tested on an intrusion response use case (see the list of publications in Section 1.4.3).

1.3 Background

The development of CSLE started as a research project in 2020 at KTH Royal Institute of Technology, led by Prof. Rolf Stadler and Prof. Pontus Johnson. This research has been supported in part by the Swedish armed forces and is conducted at the division of Network and Systems Engineering (NSE) and the KTH Center for Cyber Defense and Information Security (CDIS). The project is currently maintained by Kim Hammar (kimham@kth.se).

1.4 Resources

Resources for getting started with CSLE are described below. Section 1.4.1 describes documentation resources, Section 1.4.2 describes where to find the source code and binaries of CSLE, and Section 1.4.3 describes scientific publications related to CSLE.

1.4.1 Documentation

Apart from the official release documentation (i.e., the document you are reading now), which is available in PDF format for each release, the latest documentation of CSLE can always be found at: https://limmen.dev/csle/. Video demonstrations of earlier version of CSLE are available on YouTube:

- demonstration of v0.0.1: https://www.youtube.com/watch?v=18P7MjPKNDg.
- demonstration of v0.2.0: https://www.youtube.com/watch?v=iE2KPmtIs2A&.

1.4.2 Source code, Binaries, and Docker Images

The source code of CSLE is available on GitHub: https://github.com/Limmen/csle and the Docker images are available on DockerHub: https://hub.docker.com/u/kimham. Further, binaries of the Python libraries in CSLE are available on PyPi:

- https://pypi.org/project/csle-base
- https://pypi.org/project/csle-collector
- https://pypi.org/project/csle-common
- https://pypi.org/project/csle-attacker
- https://pypi.org/project/csle-defender

- https://pypi.org/project/gym-csle-stopping-game
- https://pypi.org/project/gym-csle-apt-game
- https://pypi.org/project/gym-csle-cyborg
- https://pypi.org/project/csle-ryu
- https://pypi.org/project/csle-rest-api
- https://pypi.org/project/csle-agents
- https://pypi.org/project/csle-system-identification
- https://pypi.org/project/csle-cli
- https://pypi.org/project/csle-cluster
- https://pypi.org/project/gym-csle-intrusion-response-game
- https://pypi.org/project/csle-tolerance

Lines of code CSLE consists of around 225,000 lines of Python [51], 40,000 lines of JavaScript [7], 3000 lines of Dockerfiles [35], 2500 lines of Makefile, and 1800 lines of Bash. The lines of code can be counted by executing the following commands from the project root:

```
find . -name '*.py' | xargs wc -l
cd management-system; find . -name '*.js' | xargs wc -l
find . -name 'Dockerfile' | xargs wc -l
find . -name 'Makefile' | xargs wc -l
find . -name '*.sh' | xargs wc -l
```

Listing 1: Commands for counting the lines of code in CSLE.

1.4.3 Publications

CSLE is an open-source project mainly developed for research purposes. Below is a list of publications related to CSLE. Feel free to contact us to have your publication added to this list.

- *Scalable Learning of Intrusion Response through Recursive Decomposition [20]*

  - **Published in:** International Conference on Decision and Game Theory for Security 2023.
  - **Authors:** Kim Hammar & Rolf Stadler.

- *Learning Near-Optimal Intrusion Responses Against Dynamic Attackers [20]*
  - **Published in:** IEEE Transactions on Network and Service Management 2023.
  - **Authors:** Kim Hammar & Rolf Stadler.

- *Digital Twins for Security Automation [16]*
  - **Published in:** NOMS 2023 IEEE/IFIP Network Operations and Management Symposium.
  - **Authors:** Kim Hammar & Rolf Stadler.

- *An Online Framework for Adapting Security Policies in Dynamic IT Environments [15]*
  - **Published in:** International Conference on Network and Service Management (CNSM 2022).
  - **Authors:** Kim Hammar & Rolf Stadler.

- *Learning Security Strategies through Game Play and Optimal Stopping [21]*
  - **Published in:** Proceedings of the ML4Cyber workshop ICML 2022.
  - **Authors:** Kim Hammar & Rolf Stadler.

- *Intrusion Prevention Through Optimal Stopping [18]*
  - **Published in:** IEEE Transactions on Network and Service Management, special issue on recent advanced in security, 2022.
  - **Authors:** Kim Hammar & Rolf Stadler.

- *A System for Interactive Examination of Learned Security Policies [14]*
  - **Published in:** NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium.
  - **Authors:** Kim Hammar & Rolf Stadler.

- *Learning Intrusion Prevention Policies through Optimal Stopping [19]*

  - **Published in:** International Conference on Network and Service Management (CNSM 2021).
  - **Authors:** Kim Hammar & Rolf Stadler.

- *Finding Effective Security Strategies through Reinforcement Learning and Self-Play [17]*
  - **Published in:** International Conference on Network and Service Management (CNSM 2020).
  - **Authors:** Kim Hammar & Rolf Stadler.

1.5 Contributing

To anyone interested in making CSLE better, there are many improvements that needs to be done. A list of tasks is available at: `https://github.com/Limmen/csle/issues`. Contributors are also welcome to report bugs and to request new features. To report a bug or make a feature request, use GitHub issues[1]. To submit a patch or a new feature, use GitHub pull requests[2]. Before you open a pull request, make sure that your code changes makes sense. In particular, read the developer guide in Section 4 and make sure that your code changes follow the guidelines stated there.

1.5.1 Getting Help

To contact the maintainers of CSLE, use GitHub issues or email. The GitHub issues page is reviewed regularly by the maintainers.

1.5.2 List of Contributors

The following people have contributed to the development of CSLE (contact `kimham@kth.se` if your name should be on the list but is not):

- Kim Hammar, creator and main developer.
- Rolf Stadler, technical advisor.
- Pontus Johnson, technical advisor.
- Antonio Frederico Nesti Lopes, software development.

[1] `https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues`

[2] `https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests`

- Jakob Stymne, software development.
- Arvid Lagerqvist, software development.
- Nils Forsgren, software development.
- Forough Shahab Samani, software development and documentation.

1.6 License

CSLE is released under the Creative Commons Attribution-ShareAlike (CC BY-SA) 4.0 international license[3].

1.7 Document structure

The remainder of this document is structured as follows. The following section, Section 2, describes the architecture and design of CSLE. The subsequent section contains installation instructions (Section 3). Section 4 and Section 5 contain the developer guide and usage tutorials, respectively. Lastly, Section 6 contains answers to frequently asked questions.

[3] https://creativecommons.org/licenses/by-sa/4.0/

2 Architecture and Design

In this section, we describe the architecture and design of CSLE. We first describe the overall architecture (Section 2.1) and then we describe each of the main components: the management system, the metastore, the emulation system, the simulation system, the web interface, and the command-line interface (Sections 2.2-2.7).

2.1 Overall Architecture

The overall architecture of CSLE is shown in Fig. 2.

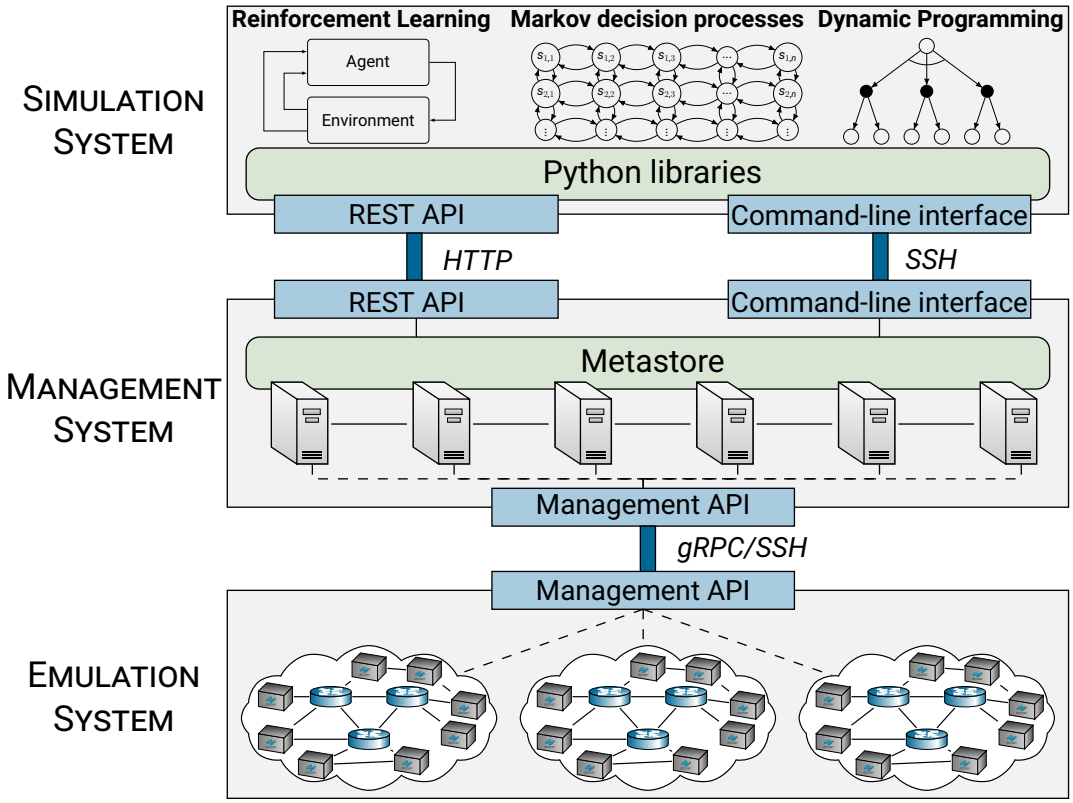

Figure 2: Architecture of CSLE; it consists of an emulation system (bottom part), a management system (middle part), and a simulation system (upper part); the three systems are connected through APIs and a shared metastore.

In the middle of Fig. 2 is the management system, which orchestrates the overall execution of the framework and stores metadata in a distributed

database called the metastore. It has three interfaces. One interface is a gRPC API to the emulation system, which emulates IT infrastructures using hardware and network virtualization [10]. The other two interfaces—a web interface and a CLI—are to the simulation system, which simulates Markov decision processes and runs reinforcement learning workloads.

Users access CSLE in three ways: (1) through software functions using Python libraries of the simulation system; (2) through a web browser using the web interface; and (3) through a terminal using the CLI.

Details about each component of CSLE are given in the sections below (Sections 2.2-2.7).

2.2 The Management System

The management system is the central component of CLSE and manages the overall execution of the framework. It is a distributed system that consist of $N \geq 1$ physical servers connected through an IP network. One of the servers is designated to be the "leader" and the other servers are "workers". Workers can perform 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 to the system state.

The leader is elected using the leader election protocol in [37], which uses the metastore for coordination (see Fig. 3). A new leader is elected by a quorum whenever the current leader fails or becomes unresponsive. This means that the management system tolerates up to $N/2 - 1$ failing servers.

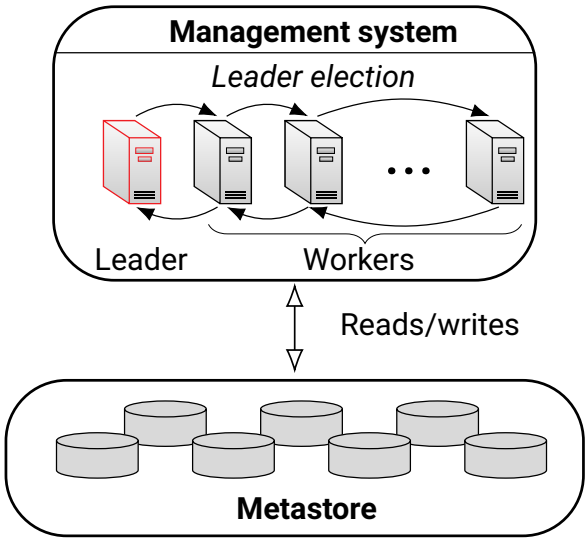

Figure 3: Architecture diagram of the management system; it is a distributed system with a designated leader, which is elected using a leader election algorithm that uses the metastore for coordination.

Each server of the management system executes the following services: (1) an instance of the emulation system; (2) a replica of the metastore; (3) an instance of the simulation system; (4) an HTTP server; and (5), a set of monitoring services, namely Prometheus [42], Grafana [12], cAdvisor [9], and Node exporter [41] (see Fig. 4).

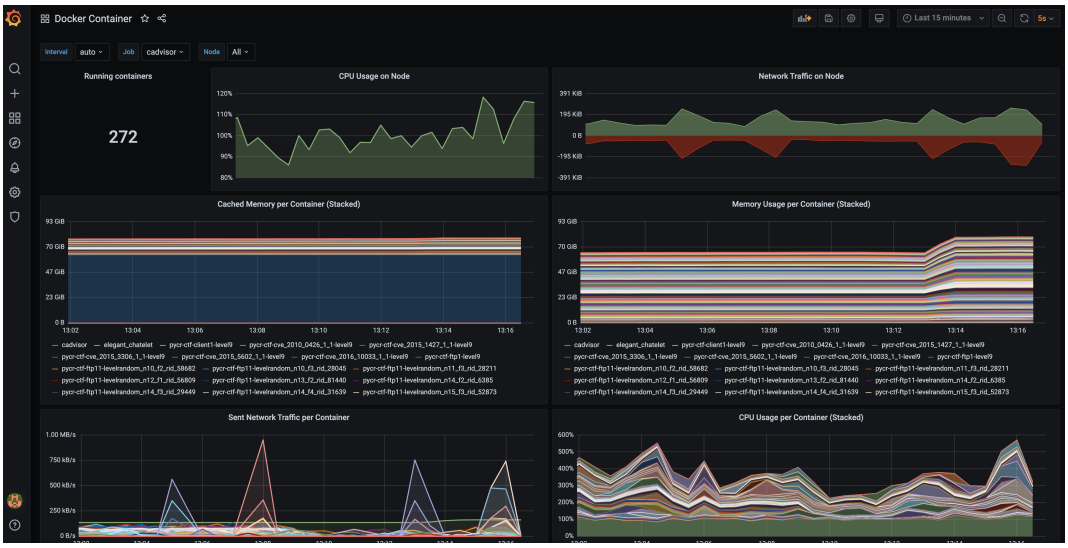

Figure 4: A Grafana dashboard for monitoring a physical server of the management system; cAdvisor and Node exporter are used to push system metrics to Prometheus, which stores the metrics in a time-series database, whose data is visualized with Grafana dashboards.

2.3 The Metastore

The metastore is a distributed relational database based on PostgreSQL [50] and Citus [5]. It stores metadata and management information about CSLE. This data includes: metadata about emulated IT infrastructures, metadata about reinforcement learning workloads, metadata about management users, metadata about simulation environments, and management information about virtual containers and networks (see Table 1 for a complete list).

The metastore is divided into N replicas. Each physical server of the management system stores one replica. Reads and writes to the metastore are distributed evenly across all replicas to maximize throughput. To coordinate writes and achieve consensus among replicas, a quorum-based two-phase commit (2PC) scheme is used. This means that the metastore tolerates $N/2 - 1$ failing replicas.

Table name	*Description*	*Columns*
emulations	Metadata about emulations.	id, name, config
emulations_traces	Attack and system traces.	id, emulation_name, trace
emulation_statistics	Count statistics of system metrics.	id, emulation_name, statistics
simulation_traces	Simulation traces-	id, gym_env, trace
emulation_simulation_traces	Mapping between emulation & simulation traces.	id, emulation_trace, simulation_trace
emulation_images	Pictures of emulation topologies.	id, emulation_name, image
simulations	Metadata about simulations.	id, name, config
experiment_executions	Metadata about experiments.	id, execution, simulation_name, emulation_name
simulation_images	Pictures of simulations.	id, simulation_name, image
multi_threshold_stopping_policies	Threshold-based policies.	id, policy, simulation_name
training_jobs	Training jobs.	id, config, simulation_name, emulation_name, pid
system_identification_jobs	System identification jobs.	id, config, emulation_name, pid
ppo_policies	PPO policies.	id, policy, simulation_name
system_models	System models.	id, emulation_name, system_model
data_collection_jobs	Data collection jobs.	id, config, emulation_name, pid
gaussian_mixture_system_models	Gaussian mixture system models.	id, model, emulation_name, emulation_statistic_id
tabular_policies	Tabular policies.	id, policy, simulation_name
alpha_vec_policies	Alpha vector policies.	id, policy, simulation_name
dqn_policies	DQN policies.	id, policy, simulation_name
fnn_w_softmax_policies	Feed-forward neural network policies.	id, policy, simulation_name
vector_policies	Vector policies.	id, policy, simulation_name
emulation_executions	Emulation executions.	ip_first_octet, emulation_name, info
empirical_system_models	Empirical system models.	id, model, emulation_name, emulation_statistic_name
mcmc_system_models	MCMC system models.	id, model, emulation_name, emulation_statistic_name
gp_system_models	Gaussian process system models.	id, model, emulation_name, emulation_statistic_id
management_users	Management users.	id, username, password, email, first_name, etc.
session_tokens	Session tokens.	token, timestamp, username
traces_datasets	Traces datasets.	id, name, description, etc.
statistics_datasets	Statistics datasets.	id, name, description, etc.
linear_threshold_stopping_policies	Linear threshold policies.	id, name, description, etc.

Table 1: Metastore tables in version 0.4.0 of CSLE.

2.4 The Emulation System

The emulation system uses a virtualization layer provided by Docker [35] containers and virtual networks to emulate IT infrastructures. An emulated IT infrastructure is defined by an *emulation configuration*, which includes the properties listed in Table 2. The set of configurations supported by the emulation system are stored in the metastore and can be seen as a *configuration space*, which defines the class of infrastructures that can be emulated (see Fig. 5).

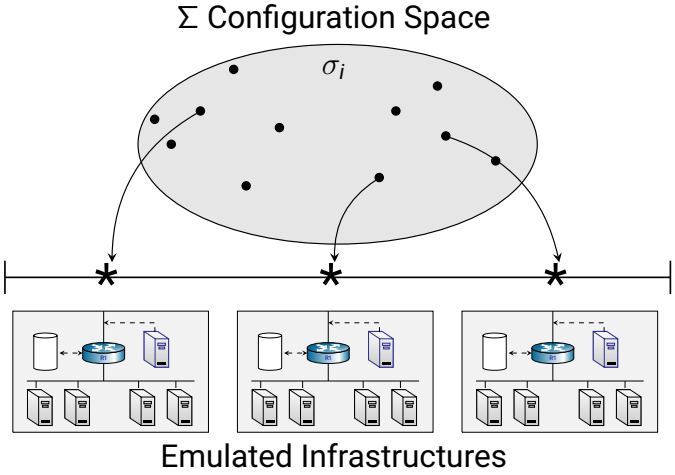

Figure 5: The configuration space of the emulation system, which defines the class of infrastructures that the emulation system can emulate.

Configuration property	*Description*
`name`	The name of the emulation.
`containers_config`	Configuration of each container (ip, hostname, resources, etc.).
`users_config`	Configuration of user accounts on each container.
`flags_config`	Configuration of flags on each container.
`vuln_config`	Configuration of vulnerabilities on each container.
`topology_config`	Configuration of the emulation's network topology.
`traffic_config`	Configuration of traffic generators and client population.
`resources_config`	Configuration of physical resources (CPU, memory, etc.).
`services_config`	Configuration of services running on each container.
`descr`	Description of the emulation.
`static_attacker_sequences`	Configuration of pre-defined attacker sequences.
`ovs_config`	Configuration of Open vSwitches (virtual OpenFlow switches).
`sdn_controller_config`	Configuration of SDN controllers.
`host_manager_config`	Configuration of host managers.
`snort_ids_manager_config`	Configuration of Snort IDS managers.
`ossec_ids_manager_config`	Configuration of OSSEC IDS managers.
`docker_stats_manager_config`	Configuration of Docker stats managers.
`beats_config`	Configuration of filebeats, packetbeats, etc.
`elk_config`	Configuration of the ELK stack in the management network.
`level`	Level of the emulation.
`version`	Version of the emulation.
`execution_id`	ID of the execution that the emulation belongs to (if any).
`csle_collector_version`	Version of the `csle-collector` library.
`csle_ryu_version`	Verion of the `csle-ryu` library.

Table 2: Properties of an emulation configuration.

Users of CSLE can create new emulation configurations and insert them into the metastore. CSLE also includes a set of pre-installed configurations

(see Table 3). One of these pre-installed configurations is `csle-level9-003` (as an example), whose topology is shown in Fig. 6 and whose configuration is listed in Table 4.

Emulation configuration	*Description*
`csle-level1-003`	Emulation with 7 components, 3 flags, password vulnerabilities, no IDS.
`csle-level2-003`	Emulation with 13 components, 6 flags, password vulnerabilities, no IDS.
`csle-level3-003`	Emulation with 34 components, 6 flags, password vulnerabilities, no IDS.
`csle-level4-003`	Emulation with 7 components, 3 flags, password vulnerabilities, IDS.
`csle-level5-003`	Emulation with 13 components, 6 flags, password vulnerabilities, IDS.
`csle-level6-003`	Emulation with 34 components, 6 flags, password vulnerabilities, IDS.
`csle-level7-003`	Emulation with 7 components, 3 flags, password & RCE vulnerabilities, IDS.
`csle-level8-003`	Emulation with 13 components, 6 flags, password & RCE vulnerabilities, IDS.
`csle-level9-003`	Emulation with 34 components, 6 flags, password & RCE vulnerabilities, IDS.
`csle-level10-003`	Emulation with 16 components, 12 flags, password & RCE vulnerabilities, IDS.
`csle-level11-003`	Emulation with 36 components, 6 flags, password & RCE vulnerabilities, IDS.
`csle-level12-003`	Emulation with 7 components, 3 flags, password & RCE vulnerabilities, IDS, SDN.

Table 3: Emulation configurations in version 0.4.0 of CSLE.

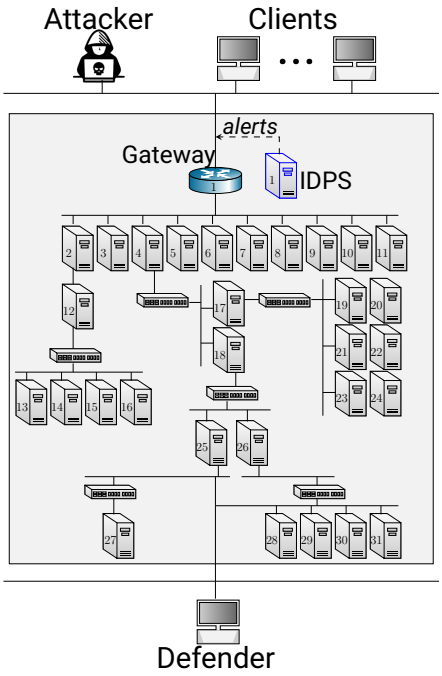

Figure 6: Topology of the emulation configuration `csle-level9-003`.

ID (s)	*OS:Services:Exploitable Vulnerabilities*
N_1	Ubuntu20:Snort(community ruleset v2.9.17.1),SSH:-
N_2	Ubuntu20:SSH,HTTP Erl-Pengine,DNS:SSH-pw
N_4	Ubuntu20:HTTP Flask,Telnet,SSH:Telnet-pw
N_{10}	Ubuntu20:FTP,MongoDB,SMTP,Tomcat,TS3,SSH:FTP-pw
N_{12}	Debian10.2:TS3,Tomcat,SSH:CVE-2010-0426,SSH-pw
N_{17}	Wheezy:Apache2,SNMP,SSH:CVE-2014-6271
N_{18}	Deb10.2:IRC,Apache2,SSH:SQL Injection
N_{22}	Deb10.2:PROFTPD,SSH,Apache2,SNMP:CVE-2015-3306
N_{23}	Deb10.2:Apache2,SMTP,SSH:CVE-2016-10033
N_{24}	Deb10.2:SSH:CVE-2015-5602,SSH-pw
N_{25}	Deb10.2: Elasticsearch,Apache2,SSH,SNMP:CVE-2015-1427
N_{27}	Deb10.2:Samba,NTP,SSH:CVE-2017-7494
N_3,N_{11},N_5-N_9	Ubuntu20:SSH,SNMP,PostgreSQL,NTP:-
N_{13-16},N_{19-21},N_{26},N_{28-31}	Ubuntu20:NTP, IRC, SNMP, SSH, PostgreSQL:-

Table 4: Configuration of the emulation configuration `csle-level9-003`, whose topology is shown in Fig. 6.

An *emulation execution* consists of a set of running containers and virtual networks, which are defined by an associated *emulation configuration*.

Creating an execution involves two 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, operating systems, network interfaces, and network conditions. The second task is to instrument the emulated infrastructure with monitoring and management capabilities. Since the emulation execution will be used to evaluate reinforcement learning agents, it must be possible to monitor system metrics in real-time and to perform control actions prescribed by reinforcement learning agents.

In the following seven sections, Sections 2.4.1-2.4.7, we describe how the emulation system emulates physical hosts, network links, switches, network conditions, client populations, attackers, and defenders, respectively. In the subsequent three sections, we describe the monitoring and management capabilities of the emulation system (Sections 2.4.8-2.4.10). Lastly, in Section 2.4.11, we describe the process of creating an emulation execution from an emulation configuration.

2.4.1 Emulating Physical Hosts

Physical hosts are emulated with Docker containers. A container is a lightweight executable package that includes everything needed to emulate the host: a runtime system, code, system tools, system libraries, and configurations.

Multiple containers can run on the same physical host and share the same operating system kernel, in which case their resources and processes are isolated using cgroups[4] (see Fig. 7).

The CSLE Docker images are listed in Table 5 and include images for emulating application servers, Software-Defined Networking (SDN) controllers, clients, attackers, Intrusion Detection and Prevention Systems (IDPSs), and storage systems. Detailed configuration of each image is available at: https://hub.docker.com/r/kimham/.

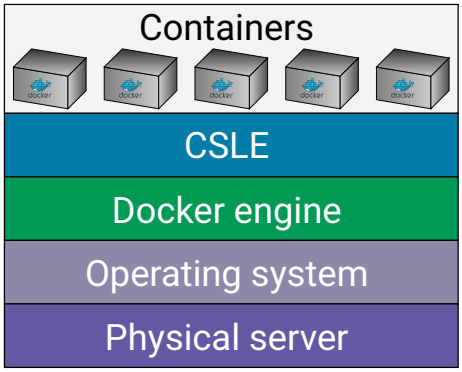

Figure 7: Software and hardware stack of a physical server that runs CSLE; closest to the hardware is the operating system; on top of the operating system is the Docker engine, which provides the base for running CSLE and virtual containers.

[4] https://man7.org/linux/man-pages/man7/cgroups.7.html

Name	*Description*	*OS*
cve_2010_0426_1	An image with the CVE-2010-0426 vulnerability.	Debian:Deb10.2
cve_2015_1427_1	An image with the CVE-2015-1427 vulnerability.	Debian:Deb10.2
cve_2015_3306_1	An image with the CVE-2015-3306 vulnerability.	Debian:Deb10.2
cve_2015_5602_1	An image with the CVE-2015-5602 vulnerability.	Debian:Deb10.2
cve_2016_10033_1	An image with the CVE-2016-10033 vulnerability.	Debian:Deb10.2
hacker_kali_1	An image with tools for penetration testing.	Kali
samba_1	An image with the CVE-2017-7494 vulnerability.	Debian:Deb10.2
samba_2	An image with the CVE-2017-7494 vulnerability.	Debian:Deb10.2
shellshock_1	An image with the CVE-2014-6271 vulnerability.	Debian:Wheezy
sql_injection_1	An image with SQL-injection vulnerability.	Debian:10.2
ftp_1	An image with an FTP server.	Ubuntu:14
ftp_2	An image with an FTP server.	Ubuntu:14
ssh_1	An image with a SSH server.	Ubuntu:20
ssh_2	An image with a SSH server.	Ubuntu:20
ssh_3	An image with a SSH server.	Ubuntu:20
telnet_1	An image with a Telnet server.	Ubuntu:20
telnet_2	An image with a Telnet server.	Ubuntu:20
telnet_3	An image with a Telnet server.	Ubuntu:20
honeypot_1	An image with various honeypot services.	Ubuntu:20
honeypot_2	An image with various honeypot services.	Ubuntu:20
router_1	A blank image.	Ubuntu:20
router_2	An image with the Snort IDPS.	Ubuntu:20
client_1	A blank image.	Ubuntu:20
blank_1	A blank image.	Ubuntu:20
pengine_exploit_1	An image with the pengine exploit.	Ubuntu:20
ovs_1	An image with the OVS virtual switch.	Ubuntu:20
ryu_1	An image with the Ryu SDN controller.	Ubuntu:20
elk_1	An image with the ELK stack.	Ubuntu:20
kafka_1	An image with Kafka.	Ubuntu:20
spark_1	An image with a Spark server.	Ubuntu:22

Table 5: Docker images in version 0.4.0 of CSLE.

2.4.2 Emulating Physical Network Links

Network connectivity is emulated by virtual links implemented by Linux bridges[5]. Network isolation between emulated hosts and emulated switches on the same physical host is achieved through network namespaces[6], which create logical copies of the physical host's network stack.

Each emulation execution has a dedicated virtual subnetwork with the

[5] https://www.kernel.org/doc/html/v5.17/networking/bridge.html
[6] https://man7.org/linux/man-pages/man7/network_namespaces.7.html

subnet mask:

```
executionID.emulationID.0.0/16
```

where `executionID` identifies the execution and `emulationID` identifies the emulation configuration. This addressing scheme allows several executions of the same emulation configuration to execute concurrently on the same physical servers without overlapping address spaces.

In the case that an emulated network spans multiple physical servers of the management system, VXLAN connections are used. These connections tunnel the emulated network traffic over the physical network (see Fig. 8). In other words, the physical network that connects the servers of the management system provides a substrate network, on top of which the emulation system deploys virtual networks.

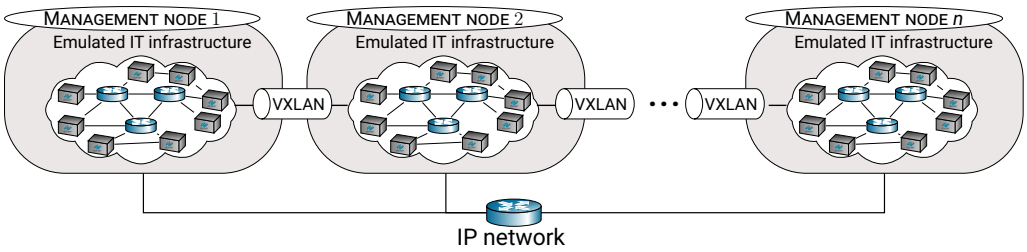

Figure 8: Architecture diagram of a distributed emulation execution; servers of the management system are connected through a physical IP network, over which virtual networks are overlayed using VXLAN tunnels.

2.4.3 Emulating Physical Switches

Physical switches are emulated with Docker containers that run Open vSwitch [40] (OVS) version 2.16 (see Fig. 9). The emulated switches connect to an SDN controller using the OpenFlow protocol version 1.3 [34] over a secure TLS tunnel. The SDN controller is emulated by a container that resides in the management network (see Section 2.4.8 for more details about the management network). (Since the switches are programmed through flow tables, they can act either as classical L2 switches or as routers, depending on the flow table configurations.)

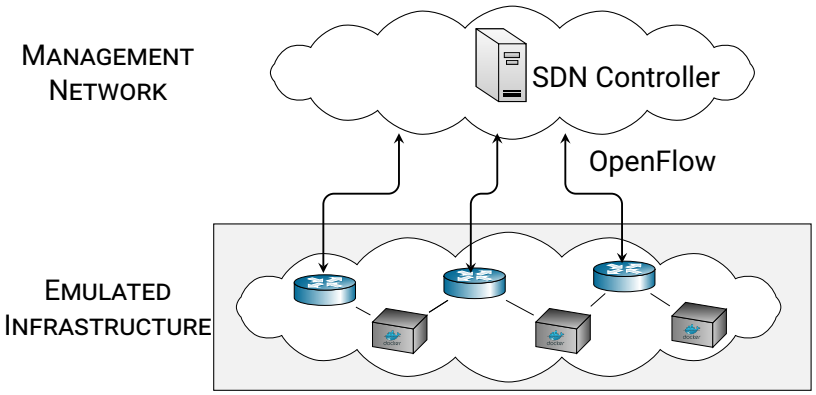

Figure 9: Physical switches in CSLE are emulated as Open vSwitch (OVS) switches [40]; each emulated switch runs the OpenFlow protocol and can be configured via a Software-Defined Networking (SDN) controller.

2.4.4 Emulating Network Conditions

Network conditions of virtual links are configured using the NetEm module in the Linux kernel [25], which allows fine-gained configuration of bit rates, packet delays, packet loss probabilities, jitter, and packet reordering probabilities (see Fig. 10).

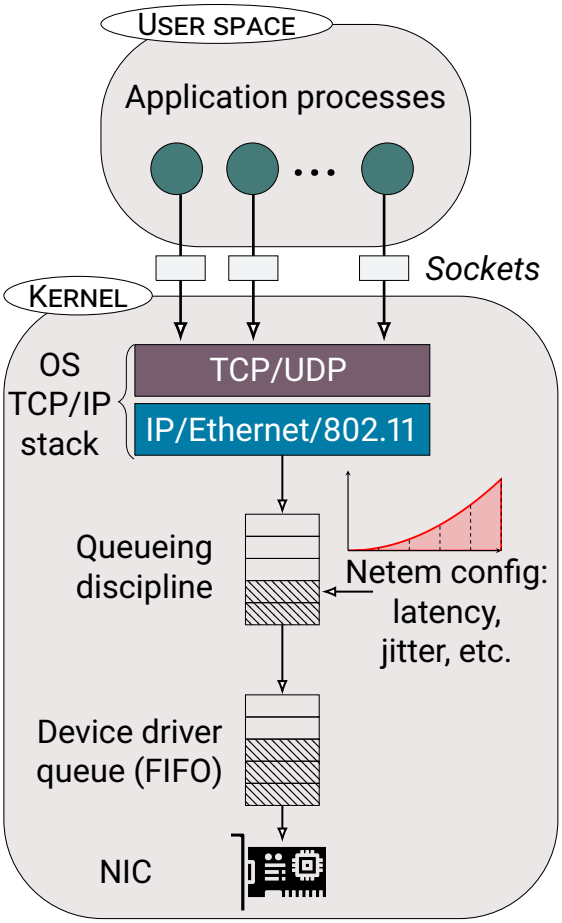

Figure 10: Method to emulate network conditions with the Netem module in the Linux kernel; Netem uses a dedicted queueing discipline in the Linux networking stack and emulates bit rates, latencies, packet loss probabilities, packet reordering probabilities, and jitter according to a predefined configuration.

2.4.5 Emulating Client Populations

Client populations in CSLE are emulated by processes that run inside Docker containers and interact with emulated hosts through various network protocols, e.g., HTTP, SSH, and DNS. The clients select functions from the list in Table 6 according to some probability distribution.

Function	*Description*
HTTP	Download web pages and use REST APIs.
SSH	Connect to emulated components using SSH.
SNMP	Perform network management operations through SNMP.
ICMP	Ping emulated components.
IRC	Communicate with IRC servers.
PostgreSQL	Communicate with PostgreSQL servers.
FTP	Download and upload files to FTP servers.
DNS	Make DNS queries.
Telnet	Connect to emulated components using Telnet.

Table 6: Network functions that emulated clients may invoke.

Client arrivals are emulated by a Poisson process with exponentially distributed service times (see Fig. 11). This process may be stationary or non-stationary, e.g., it may be sine-modulated to model periodic load patterns. The duration of a time-step in an emulation is defined by its configuration; a typical duration is 30 seconds.

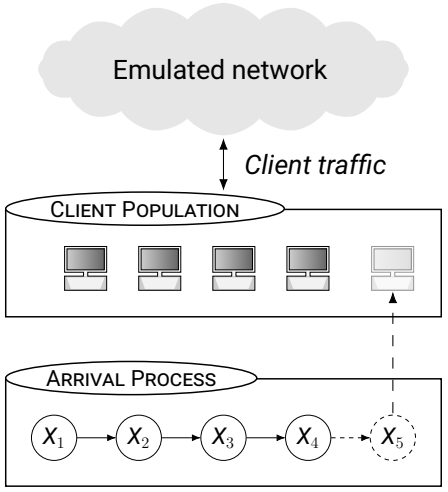

Figure 11: Clients are emulated by processes that invoke network functions of emulated hosts; client arrivals and service times are emulated by a Poisson process with exponentially distributed service times.

2.4.6 Emulating Attackers

Attackers are emulated by automated programs that select actions from a pre-defined set. The actions are selected according to an *attacker strategy*, which may depend on system metrics of the emulation. (These metrics are

collected in real-time by monitoring agents (see Section 2.4.9 for details).). The attacker commands supported in version 0.4.0 of CSLE are listed in Table 7.

Type	*Actions*
Reconnaissance	TCP-SYN scan, UDP port scan, TCP Null scan, TCP Xmas scan, TCP FIN scan, ping-scan, TCP connection scan, TCP-xmas-tree scan, OS-detection scan “Vulscan” vulnerability scanner “Vulners” vulnerability scanner Nikto web scan, masscan firewall walk, HTTP-enum scan finger scan
Brute-force attack	Telnet, SSH, FTP, Cassandra, IRC, MongoDB, MySQL, SMTP, Postgres
Exploit	CVE-2017-7494, CVE-2015-3306, CVE-2010-0426, CVE-2015-5602, CVE-2014-6271, CVE-2016-10033 CVE-2015-1427, SQL Injection

Table 7: Attacker commands supported in version 0.4.0 of CSLE.

2.4.7 Emulating Defenders

Defender actions are emulated by executing system commands on emulated hosts. These commands are invoked through a management API based on gRPC [10] and SSH (see Section 2.4.10). Similar to the attacker actions, the defender actions are prescribed by a *defender strategy*, which may depend on system metrics collected by monitoring agents (see Section 2.4.9). The defender actions supported in version version 0.4.0 of CSLE are listed in Table 8.

Index	*Action*
1	Revoke user certificates.
2	Blacklist IPs.
$3-37$	Drop traffic that generates IDPS alerts of priority $1-34$.
38	Block gateway.
39	Server migration between different zones of the infrastructure.
40	Traffic redirect from one server to another.
41	Server isolation.
42	Deploying new security functions (e.g., firewalls and IDPSs).
43	Server shutdown.

Table 8: Defender commands supported in version 0.4.0 of CSLE.

2.4.8 The Management Network

Each emulation execution has a dedicated virtual subnetwork, called the *management network*, which spans the following address space:

`executionID.emulationID.253.0/24`

This network interfaces directly with the management system and also has an interface to each emulated component. The purpose of this network is to connect management systems to the emulated devices, e.g., storage systems, monitoring systems, and SDN controllers (see Fig. 12). The reason for using a dedicated management network instead of carrying management traffic on the network that carries device-to-device traffic is to avoid interference and to simplify control of the execution [4].

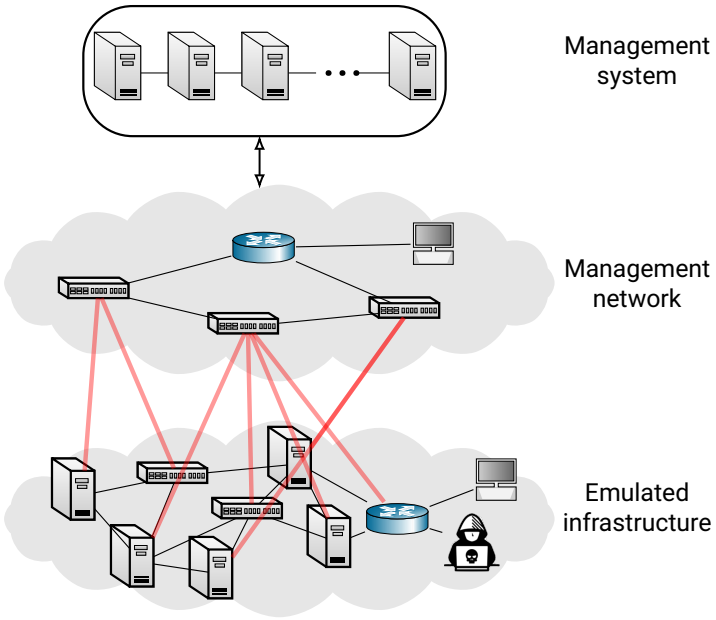

Figure 12: The management network; it is a dedicated network used for operation of the emulated infrastructure.

2.4.9 Monitoring Agents

Each emulated device runs a *monitoring agent* that reads local metrics of the device and pushes those metrics to a distributed Kafka queue [30], which is deployed inside the management network (see Fig. 13). The metrics collected at every time-step are listed in Tables 11-13.

The data in the Kafka queue is organized in a set of *topics* (see Table 9), each of which stores a sequence of records in a first-in first-out (FIFO) order for a pre-determined time window. These topics are distributed over a set of partitions, each of which is stored at a server in the Kafka cluster.

The topics are consumed by a set of data pipelines implemented with Spark [52], which process the data and write the results to Presto [46] and Elasticsearch [11] for persistent storage and search, respectively. These storage systems are then consumed by various downstream applications, e.g., dashboards applications and machine learning applications. The processed data is also input to reinforcement learning agents, which use it to decide on control actions.

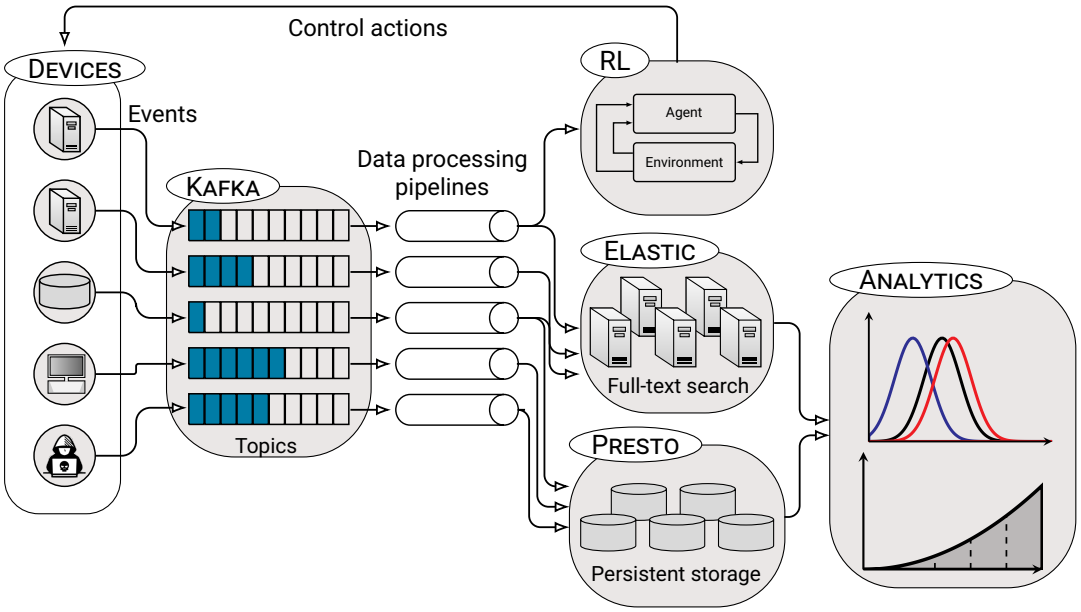

Figure 13: The monitoring system of CSLE; monitoring agents running on emulated devices push metric data to a distributed Kafka queue; the data in this queue is consumed by data pipelines that process the data and write to Presto and Elasticsearch for persistent storage and search, respectively; the processed data is used by Reinforcement Learning (RL) agents to decide on control actions, and by downstream applications to create dashboards and statistical models.

Topic	*Description*
`client_population`	Statistics of the client population (e.g., number of clients per time.step).
`snort_ids_log`	Information about Snort IDS alerts per time-step.
`ossec_ids_log`	Information about OSSEC IDS alerts per time-step.
`host_metrics`	Information about system metrics per host.
`docker_stats`	Resource statistics of docker containers.
`docker_host_stats`	Resource statistics of host servers.
`openflow_flow_stats`	Statistics of flows.
`openflow_port_stats`	Statistics of ports.
`openflow_flow_agg_stats`	Aggregate flow statistics.
`avg_openflow_flow_stats_per_switch`	Aggregate flow statstics per switch.
`avg_openflow_port_stats_per_switch`	Aggregate port statistics per switch.
`attacker_actions`	Attacker actions in the emulation.
`defender_actions`	Defender actions in the emulation.

Table 9: Kafka topics in version 0.4.0 of CSLE.

Metric	*Description*
num_clients	Number of active clients.
clients_arrival_rate	Arrival rate of clients.
pids	Number of pids per container.
cpu_percent	CPU utilization per container.
mem_current	The total memory usage per container.
mem_total	The memory limit per container.
mem_percent	The memory utilization per container.
blk_read	Number of blocks read per container.
blk_write	Number of blocks written per container.
net_rx	Number of received network bytes per container.
net_tx	Number of transmitted network bytes per container.
avg_pids	Average number of pids.
avg_cpu_percent	Average CPU utilization.
avg_mem_current	Average total memory usage.
avg_mem_total	Average memory limit.
avg_mem_percent	Average memory utilization.
avg_blk_read	Average number of blocks read.
avg_blk_write	Average number of blocks written.
avg_net_rx	Average number of received network bytes.
avg_net_tx	Average number of transmitted network bytes.
num_logged_in_users	Number of logged in users per container.
num_failed_login_attempts	Number of failed logins per container.
num_open_connections	Number of open connections per container.
num_login_events	Number of login events per container.
num_processes	Number of processes per container.
num_users	Number of users per container.
avg_num_logged_in_users	Average number of logged in users per container.
avg_num_failed_login_attempts	Average number of failed logins per container.
avg_num_open_connections	Average number of open connections per container.
avg_num_login_events	Average number of login events per container.
avg_num_processes	Average number of processes per container.
avg_num_users	Average number of users per container.

Table 10: Metrics collected per time-step by the monitoring system in version 0.4.0 of CSLE (1/4).

Metric	*Description*
defender_action_id	Defender action ID.
defender_action_name	Defender action name.
defender_action_cmds	List of defender action commands.
defender_action_type	Defender action type.
defender_action_descr	Defender action description.
defender_action_outcome	Defender action outcome.
defender_action_execution_time	Defender action execution time.
attacker_action_id	Attacker action ID.
attacker_action_name	Attacker action name.
attacker_action_cmds	List of attacker action commands.
attacker_action_type	Attacker action type.
attacker_action_descr	Attacker action description.
attacker_action_outcome	Attacker action outcome.
attacker_action_execution_time	Attacker action execution time.
attacker_action_vulnerability	Attacker action vulnerability.
snort_priority_alerts	List of number of Snort alerts of priorities 1-4.
snort_class_alerts	List of number of Snort alerts of classes 1-34.
snort_total_alerts	Total number of Snort alerts.
snort_alerts_weighted_by_priority	Number of Snort alerts weighted by priority.
ossec_level_alerts	Number of OSSEC IDS alerts of level 1-16 per container.
ossec_group_alerts	Number of OSSEC IDS alerts of group 1-12 per container.
ossec_total_alerts	Total number of OSSEC IDS alerts per container.
ossec_alerts_weighted_by_priority	Total number of OSSEC IDS alerts weighted by priority per container.
avg_ossec_level_alerts	Average number of OSSEC IDS alerts of level 1-16.
avg_ossec_group_alerts	Average number of OSSEC IDS alerts of group 1-12.
avg_ossec_total_alerts	Average number of OSSEC IDS alerts.
avg_ossec_alerts_weighted_by_priority	Average number of OSSEC IDS alerts weighted by priority.

Table 11: Metrics collected per time-step by the monitoring system in version 0.4.0 of CSLE (2/4).

Metric	*Description*
flow_num_packets	Number of packets per flow.
flow_num_bytes	Number of transceived bytes per flow.
flow_duration	Duration per flow.
flow_in_port	Input port per flow.
flow_out_port	Output port per flow.
flow_dst_mac_address	Destination MAC address per flow.
flow_datapath_id	Datapath id per flow.
avg_flow_num_bytes	Average number of transceived bytes per flow.
avg_flow_duration	Average duration per flow.
of_port_datapath_id	OpenFlow switch port datapath id.
of_port_num_received_packets	Number of received packets per OpenFlow port.
of_port_num_received_bytes	Number of received bytes per OpenFlow port.
of_port_num_received_errors	Number of received errors per OpenFlow port.
of_port_num_transmitted_packets	Number of transmitted packets per OpenFlow port.
of_port_num_transmitted_bytes	Number of transmitted bytes per OpenFlow port.
of_port_num_transmitted_errors	Number of transmitted errors per OpenFlow port.
of_port_num_received_dropped	Number of received dropped packets per OpenFlow port.
of_port_num_transmitted_dropped	Number of transmitted dropped packets per OpenFlow port.
of_port_num_received_frame_errors	Number of received frame errors per OpenFlow port.
of_port_num_received_overrun_errors	Number of received overrun errors per OpenFlow port.
of_port_num_received_crc_errors	Number of received CRC errors per OpenFlow port.
of_port_num_collisions	Number of received collisions per OpenFlow port.
of_port_duration_seconds	Duration uptime per OpenFlow port.
avg_of_port_num_received_packets	Average OpenFlow port number of received packets.
avg_of_port_num_received_bytes	Average OpenFlow port number of received bytes.
avg_of_port_num_received_errors	Average OpenFlow port number of received errors.
avg_of_port_num_transmitted_packets	Average OpenFlow port number of transmitted packets.
avg_of_port_num_transmitted_bytes	Average OpenFlow port number of transmitted bytes.
avg_of_port_num_transmitted_errors	Average OpenFlow port number of transmitted errors.
avg_of_port_num_received_dropped	Average OpenFlow port number of received dropped packets.
avg_of_port_num_transmitted_dropped	Average OpenFlow port number of transmitted dropped packets.
avg_of_port_num_received_frame_errors	Average OpenFlow port number of received frame errors.
avg_of_port_num_received_overrun_errors	Average OpenFlow port number of received overrun errors.
avg_of_port_num_received_crc_errors	Average OpenFlow port number of received CRC errors.
avg_of_port_num_collisions	Average OpenFlow port number of received collisions.
avg_of_port_duration_seconds	Average OpenFlow port duration uptime.

Table 12: Metrics collected per time-step by the monitoring system in version 0.4.0 of CSLE (3/4).

Metric	*Description*
switch_flow_num_packets	Number of packets per flow per switch.
switch_flow_num_bytes	Number of transceived bytes per flow per switch.
switch_flow_duration	Duration per flow per switch.
switch_flow_in_port	Input port per flow per switch.
switch_flow_out_port	Output port per flow per switch.
switch_flow_dst_mac_address	Destination MAC address per flow per switch.
switch_flow_datapath_id	Datapath id per flow per switch.
switch_avg_flow_num_bytes	Average number of transceived bytes per flow per switch.
switch_avg_flow_duration	Average duration per flow per switch.
switch_of_port_datapath_id	OpenFlow switch port datapath id.
switch_of_port_num_received_packets	Number of received packets per OpenFlow port per switch.
switch_of_port_num_received_bytes	Number of received bytes per OpenFlow port per switch.
switch_of_port_num_received_errors	Number of received errors per OpenFlow port per switch.
switch_of_port_num_transmitted_packets	Number of transmitted packets per OpenFlow port per switch.
switch_of_port_num_transmitted_bytes	Number of transmitted bytes per OpenFlow port per switch.
switch_of_port_num_transmitted_errors	Number of transmitted errors per OpenFlow port per switch.
switch_of_port_num_received_dropped	Number of received dropped packets per OpenFlow port per switch.
switch_of_port_num_transmitted_dropped	Number of transmitted dropped packets per OpenFlow port per switch.
switch_of_port_num_received_frame_errors	Number of received frame errors per OpenFlow port per switch.
switch_of_port_num_received_overrun_errors	Number of received overrun errors per OpenFlow port per switch.
switch_of_port_num_received_crc_errors	Number of received CRC errors per OpenFlow port per switch.
switch_of_port_num_collisions	Number of received collisions per OpenFlow port per switch.
switch_of_port_duration_seconds	Duration uptime per OpenFlow port per switch.
switch_avg_of_port_num_received_packets	Average OpenFlow port number of received packets.
switch_avg_of_port_num_received_bytes	Average OpenFlow port number of received bytes.
switch_avg_of_port_num_received_errors	Average OpenFlow port number of received errors.
switch_avg_of_port_num_transmitted_packets	Average OpenFlow port number of transmitted packets.
switch_avg_of_port_num_transmitted_bytes	Average OpenFlow port number of transmitted bytes.
switch_avg_of_port_num_transmitted_errors	Average OpenFlow port number of transmitted errors.
switch_avg_of_port_num_received_dropped	Average OpenFlow port number of received dropped packets.
switch_avg_of_port_num_transmitted_dropped	Average OpenFlow port number of transmitted dropped packets.
switch_avg_of_port_num_received_frame_errors	Average OpenFlow port number of received frame errors.
switch_avg_of_port_num_received_overrun_errors	Average OpenFlow port number of received overrun errors.
switch_avg_of_port_num_received_crc_errors	Average OpenFlow port number of received CRC errors.
switch_avg_of_port_num_collisions	Average OpenFlow port number of received collisions.
switch_avg_of_port_duration_seconds	Average OpenFlow port duration uptime.

Table 13: Metrics collected per time-step by the monitoring system in version 0.4.0 of CSLE (4/4).

2.4.10 Management Agents

In addition to the monitoring agent, each emulated device runs a *management agent*, which is connected to the management network (described in Section 2.4.8) and exposes a gRPC API [10] for management actions (see Fig. 14). This API is invoked by the management system to perform management operations on behalf of CSLE users and reinforcement learning agents, e.g., restarting services, rotating log files, or updating firewall configurations. The management functions available in the API are listed in Tables 14-16.

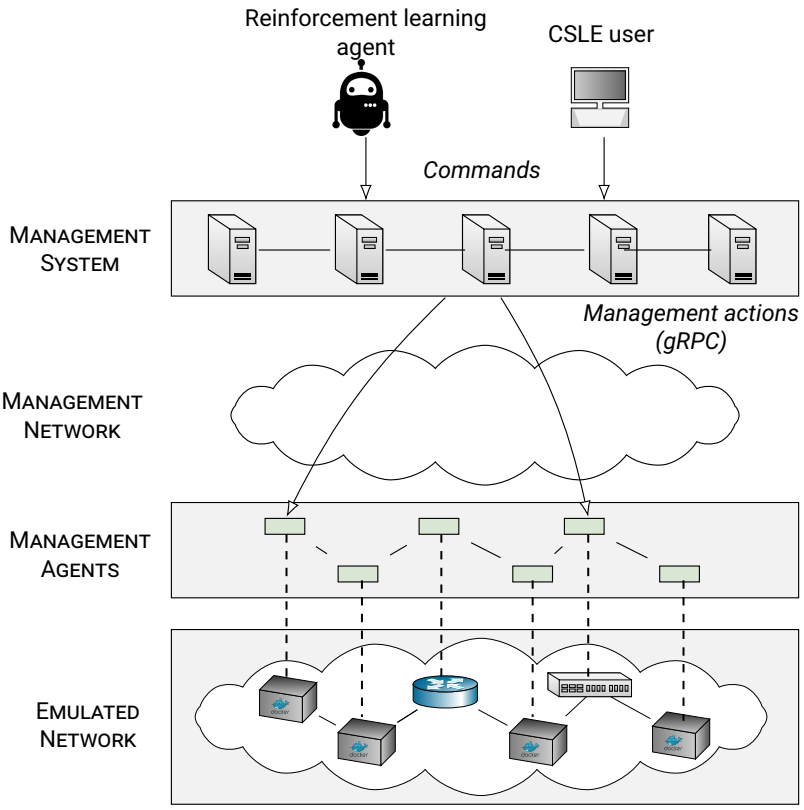

Figure 14: The management setup in CSLE; each emulated device runs a management agent which is connected to the management network and exposes a gRPC API for management actions, which is invoked by the management system on behalf of CSLE users and reinforcement learning agents.

Remote procedure call	*Description*
`getClients()`	Gets the number of clients.
`stopClients()`	Stops the client population.
`startClients()`	Starts the client population.
`startProducer()`	Starts the Kafka producer thread of the client population.
`stopProducer()`	Stops the Kafka producer thread of the client population.
`getDockerStatsMonitorStatus()`	Gets the status of the Docker statsmanager.
`stopDockerStatsMonitor()`	Stops the Docker statsmanager.
`startDockerStatsMonitor()`	Starts the Docker statsmanager.
`getElkStatus()`	Gets the status of the ELK stack.
`stopElk()`	Stops the ELK stack.
`startElk()`	Starts the ELK stack.
`stopElastic()`	Stops Elasticsearch.
`startElastic()`	Starts Elasticsearch.
`stopLogstash()`	Stops Logstash.
`startLogstash()`	Starts Logstash.
`stopKibana()`	Stops Kibana.
`startKibana()`	Starts Kibana.
`stopHostMonitor()`	Stops the host monitor of a particular host.
`startHostMonitor()`	Starts the host monitor of a particular host.
`getHostStatus()`	Gets the status of a host.
`getHostMetric()`	Gets a list of metrics of a given host.
`stopFilebeat()`	Stops filebeat on a given host.
`startFilebeat()`	Starts filebeat on a given host.
`configFilebeat()`	Configures filebeat on a given host.
`stopPacketbeat()`	Stops packetbeat on a given host.
`updateFirewall()`	Updates the firewall configuration on a given host.
`createUserAccount()`	Creates a user account on a given host.
`revokeUserCertificates()`	Revokes user certificates on a given host.

Table 14: Remote procedure calls (RPCs) in the gRPC API exposed by management agents in CSLE version 0.4.0 (1/3).

Remote procedure call	*Description*
`startPacketbeat()`	Starts packetbeat on a given host.
`configPacketbeat()`	Configures packetbeat on a given host.
`stopMetricbeat()`	Stops metricbeat on a given host.
`startMetricbeat()`	Starts metricbeat on a given host.
`configMetricbeat()`	Configures metricbeat on a given host.
`stopHeartbeat()`	Stops heartbeat on a given host.
`startHeartbeat()`	Starts heartbeat on a given host.
`configHeartbeat()`	Configures heartbeat on a given host.
`getKafkaStatus()`	Gets the status of the Kafka cluster.
`stopKafka()`	Stops the Kafka cluster.
`startKafka()`	Starts the Kafka cluster.
`createTopic()`	Creates a topic in Kafka.
`deleteTopic()`	Deletes a topic in Kafka.
`getOSSECIdsAlerts()`	Gets the list of OSSEC IDS alerts of a given host.
`stopOSSECIdsMonitor()`	Stops the OSSEC IDS monitor of a given host.
`startOSSECIdsMonito()`	Starts the OSSEC IDS monitor of a given host.
`stopOSSECIds()`	Stops the OSSEC IDS on a given host.
`startOSSECIds()`	Starts the OSSEC IDS on a given host.

Table 15: Remote procedure calls (RPCs) in the gRPC API exposed by management agents in CSLE version 0.4.0 (2/3).

Remote procedure call	*Description*
`getOSSECIdsMonitorStatus()`	Gets the status of the OSSEC IDS monitor of a given host.
`getRyuStatus()`	Gets the status of the Ryu SDN manager.
`stopRyu()`	Stops the Ryu SDN manager.
`startRyu()`	Starts the Ryu SDN manager.
`stopRyuMonitor()`	Stops the Ryu monitor.
`startRyuMonitor()`	Starts the Ryu monitor.
`getSnortIdsAlerts()`	Gets the list of Snort IDS alerts.
`stopSnortIdsMonitor()`	Stops the Snort IDS monitor.
`startSnortIdsMonitor()`	Starts the Snort IDS monitor.
`getSnortIdsMonitorStatus()`	Gets the status of the Snort IDS monitor.
`stopSnortIds()`	Stops the Snort IDS.
`startSnortIds()`	Starts the Snort IDS.
`getTrafficStatus()`	Gets the status of the traffic generator of a given host.
`stopTraffic()`	Stops the traffic generator on a given host.
`startTraffic()`	Starts the traffic generator on a given host.
`updateSnortConfig()`	Updates the Snort configuration on a given host.
`updateOssecConfig()`	Updates the OSSEC IDS configuration on a given host.
`updateRyuConfig()`	Updates the Ryu configuration on a given host.

Table 16: Remote procedure calls (RPCs) in the gRPC API exposed by management agents in CSLE version 0.4.0 (3/3).

2.4.11 Starting an Emulation Execution

An emulation execution corresponds to a set of running Docker containers and virtual networks. Starting an execution takes around 10-15 minutes. To start an execution, the emulation system first reads the emulation configuration from the metastore and starts the containers that are described in the configuration. Once the containers have started, the emulation system applies the emulation configuration. This step involves executing a sequence of Remote Procedure Calls (RPCs) through the gRPC API described in Section 2.4.10, e.g., RPCs to create user accounts, to start services, and to configure services (see Fig. 15).

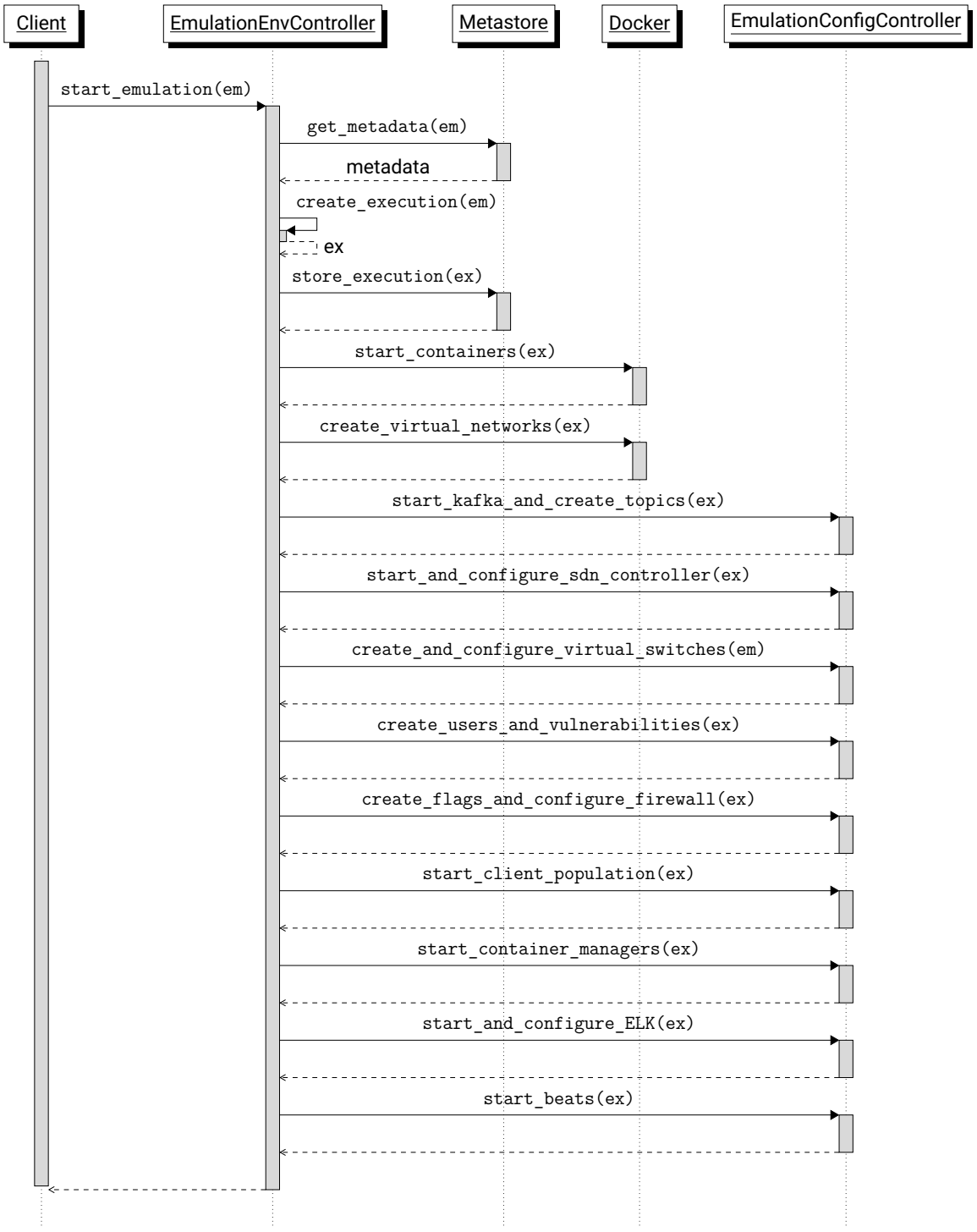

Figure 15: Sequence diagram of the emulation system's procedure to start an emulation execution.

2.5 The Simulation System

The simulation system uses the data collected from the emulation system to instantiate simulations, which are used to find effective defender strategies through reinforcement learning and other optimization techniques (see Fig. 16). It consists of simulation configurations and Python libraries that implement numerical algorithms for computing defender strategies, for simulating Markov decision processes, and for system identification.

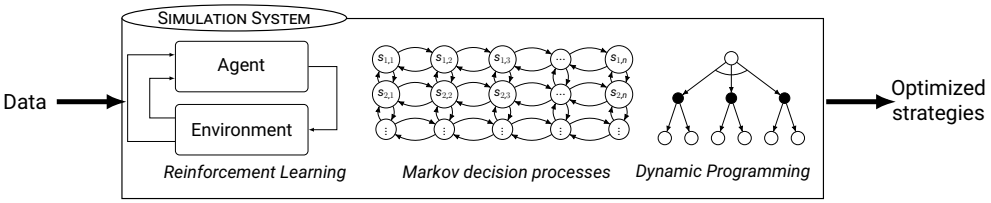

Figure 16: Overview of the simulation system; it uses data collected from emulated IT infrastructures to instantiate simulations of Markov decision processes (MDPs) and to optimize defender strategies through reinforcement learning and other optimization methods.

2.5.1 Simulation Environments

The simulation environments are listed in Table 17. Similar to an emulation execution, which is defined by an emulation configuration, a simulation is defined by a *simulation configuration*, which includes the set of properties listed in Table 18. Each simulation environment is implemented in Python and the configuration files are stored in the metastore. Simulations are typically based either on Markov decision processes or game-theoretic models.

Name	*Description*
Optimal stopping game	Zero-sum one-sided partially observed stochastic game in [21].
Optimal stopping MDP	MDP based on the optimal stopping formulation in [18].
Optimal stopping POMDP	POMDP based on the optimal stopping formulation in [18].
Local intrusion response game	Zero-sum partially observed stochastic game with public observations.
Local intrusion response POMDP attacker	POMDP for the attacker in the local intrusion response game.
Local intrusion response POMDP defender	POMDP for the defender in the local intrusion response game.
Workflow intrusion response game	Zero-sum partially observed stochastic game with public observations.
Workflow intrusion response POMDP attacker	POMDP for the attacker in the workflow intrusion response game.
Workflow intrusion response POMDP defender	POMDP for the defender in the workflow intrusion response game.
Intrusion recovery POMDP	POMDP intrusion recovery.
CMDP for replication factor control	CMDP for replication factor control
APT game	Zero-sum one-sided partially observed stochastic APT game.
APT MDP	MDP formulation of APT from attacker's perspective.
APT POMDP	POMDP formulation of APT from the defender's perspective.

Table 17: Simulation environments supported in version 0.4.0 of CSLE.

Configuration property	*Description*
`name`	Name of the simulation environment.
`gym_env_name`	Name of the OpenAI gym environment.
`descr`	Description of the simulation environment.
`simulation_env_input_config`	Input configuration to the simulation.
`players_config`	Players configuration of the simulation.
`state_space_config`	State space configuration of the simulation.
`joint_action_space_config`	Joint action space configuration of the simulation.
`joint_observation_space_config`	Joint observation space config of the simulation.
`time_step_type`	Time-step type of the simulation
`reward_function_config`	Reward function configuration of the of the simulation.
`transition_operator_config`	Transition operator configuration of the simulation.
`observation_function_config`	Observation function configuration of the simulation.
`emulation_statistic_id`	Id of the emulation statistic.
`initial_state_distribution_config`	Initial state distribution configuration of the simulation.
`version`	Version of the simulation environment.
`env_parameters_config`	Parameters that are not part of the state but that the poliy depends on.
`plot_transition_probabilities`	Boolean parameter whether to plot transition probabilities or not.
`plot_observation_function`	Boolean parameter whether to plot the observation function or not.
`plot_reward_function`	Boolean parameter whether to plot the reward function or not.

Table 18: Properties of a simulation environment configuration.

2.5.2 Numerical Algorithms

The numerical algorithms for strategy optimization and system identification are listed in Table 19. These algorithms are implemented in Python and are based on PyTorch [2], NumPy [24], PuLP [49], CvxPy [6], Openspiel [31], Stable-Baselines3 [43], Emukit [38], and CMA-Python [23]. A code example of running the Q-learning algorithm is shown below:

```
import csle_common.constants.constants as constants
from csle_common.dao.training.experiment_config import
↪  ExperimentConfig
from csle_common.metastore.metastore_facade import
↪  MetastoreFacade
from csle_common.dao.training.agent_type import AgentType
from csle_common.dao.training.hparam import HParam
from csle_common.dao.training.player_type import PlayerType
from csle_agents.agents.q_learning.q_learning_agent import
↪  QLearningAgent
import csle_agents.constants.constants as agents_constants
# Select simulation configuration
simulation_env_config = MetastoreFacade.get_simulation_by_name(
"csle-stopping-mdp-attacker-002")
# Setup experiment with hyperparameters
experiment_config = ExperimentConfig(output_dir="..",
↪  title="Q-learning test", random_seeds=[..],
↪  agent_type=AgentType.Q_LEARNING, hparams={..},
↪  player_type=PlayerType.ATTACKER, player_idx=1)
agent =
↪  QLearningAgent(simulation_env_config=simulation_env_config,
↪  experiment_config=experiment_config, save_to_metastore=True)
# Run the algorithm
experiment_execution = agent.train()
# Save the results and the learned policies
MetastoreFacade.save_experiment_execution(experiment_execution)
for policy in experiment_execution.result.policies.values():
    MetastoreFacade.save_tabular_policy(tabular_policy=policy)
```

Listing 2: Example of strategy optimization through the Q-learning algorithm in CSLE.

Name	*Algorithm type*
Bayesian optimization	Black-box optimization
Cross-entropy method	Black-box optimization
Simulated annealing	Black-box optimization
CMA-ES	Evolutionary computation
Nelder-mead	Black-box optimization
Particle swarm	Black-box optimization
Differential evolution	Evolutionary computation
Deep Q-network (DQN)	Reinforcement learning
Fictitious Self-Play	Computational game theory
Heuristic Search Value Iteration (HSVI)	Dynamic programming
Heuristic Search Value Iteration (HSVI) for one-sided game POSGs	Computational game theory
Kiefer Wolfowitz	Stochastic approximation
Linear programming	Computational game theory/Markov decision theory
Policy iteration	Dynamic programming
Value iteration	Dynamic programming
Proximal Policy Optimization (PPO)	Reinforcement learning
Q-learning	Reinforcement learning
Random search	Black-box optimization
REINFORCE	Reinforcement learning
SARSA	Reinforcement learning
Shapley iteration	Computational game theory
Sondik's value iteration	Dynamic programming
T-FP	Reinforcement learning
T-SPSA	Reinforcement learning
Expectation maximization	System identification
POMCP	POMDP planning

Table 19: Numerical algorithms supported in version 0.4.0 of CSLE.

2.6 The Web Interface

The web interface of CSLE can be used to view management information and to request management operations (see Fig. 17). It can also be used to monitor emulations in real-time (see Fig. 18 and Fig. 19), to start or stop services (see Fig. 20), to monitor reinforcement learning workloads (see Fig. 21), to access terminals of emulated components (see Fig. 22), and to examine security policies (see Fig. 2.6.1).

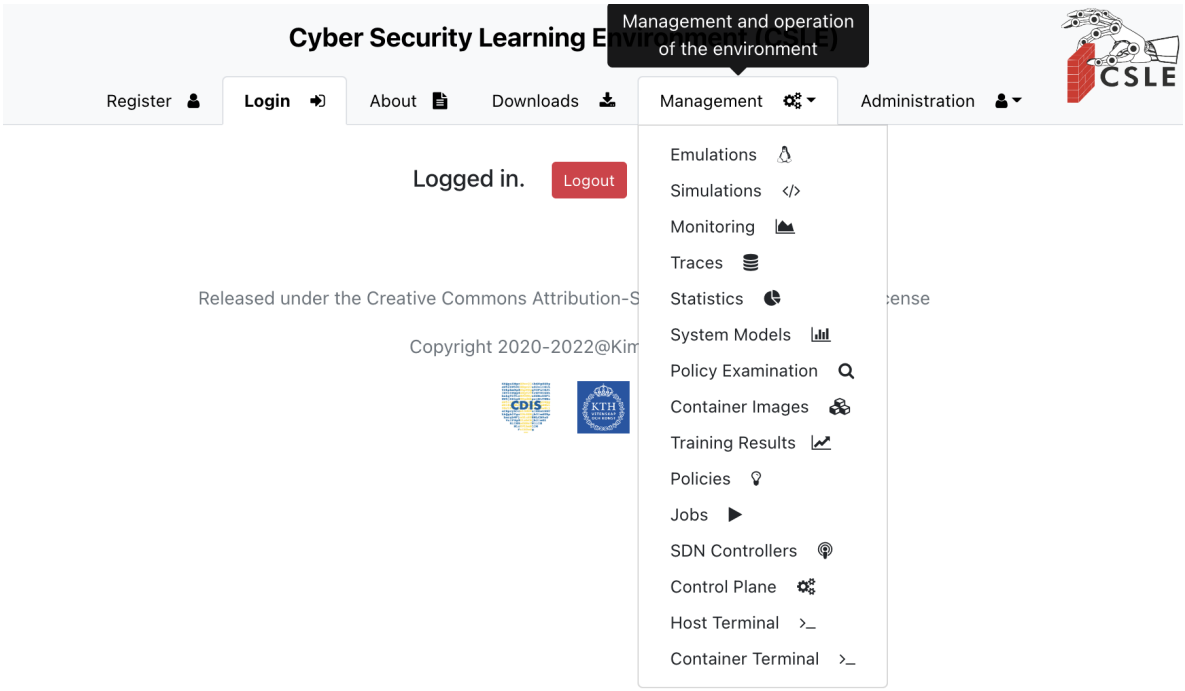

Figure 17: The web interface of CSLE; this interface shows management information and and can be used to request management operations.

Figure 18: The monitoring page of the web interface; this page allows a user to view system metrics of a given emulation execution in real-time.

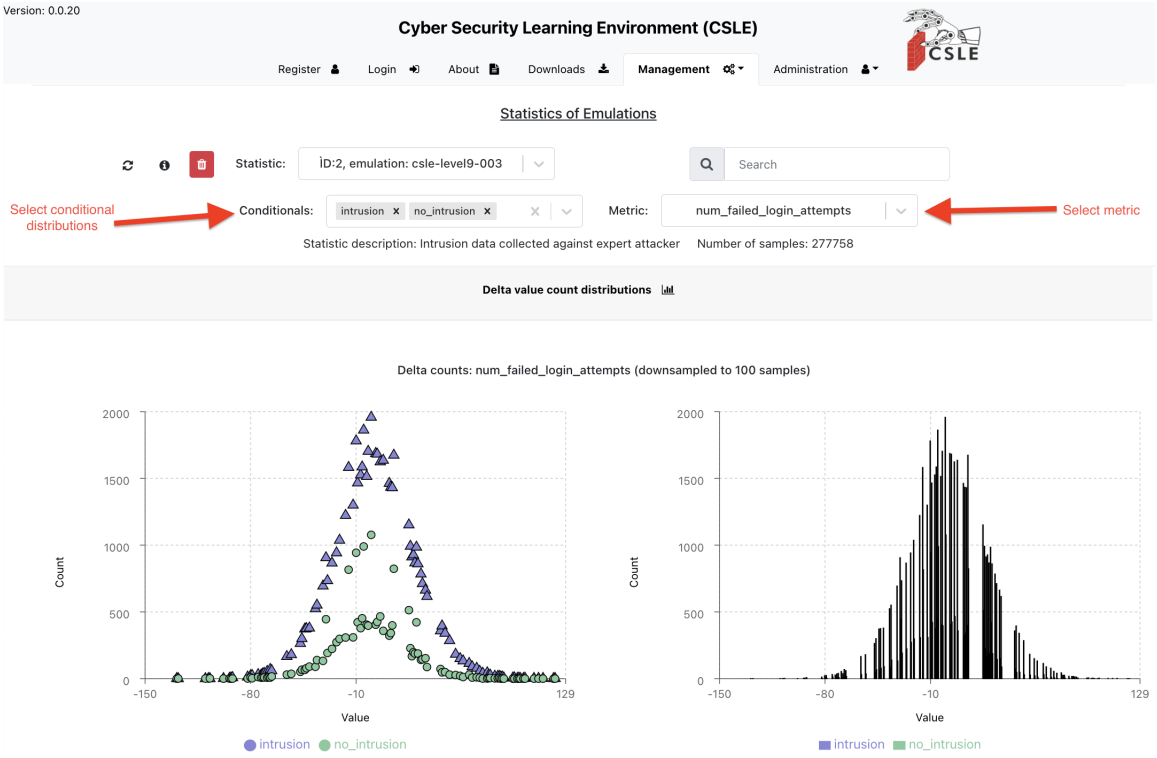

Figure 19: The statistics page of the web interface; this page allows a user to view empirical statistics of emulated infrastructures, e.g., empirical distributions of system metrics.

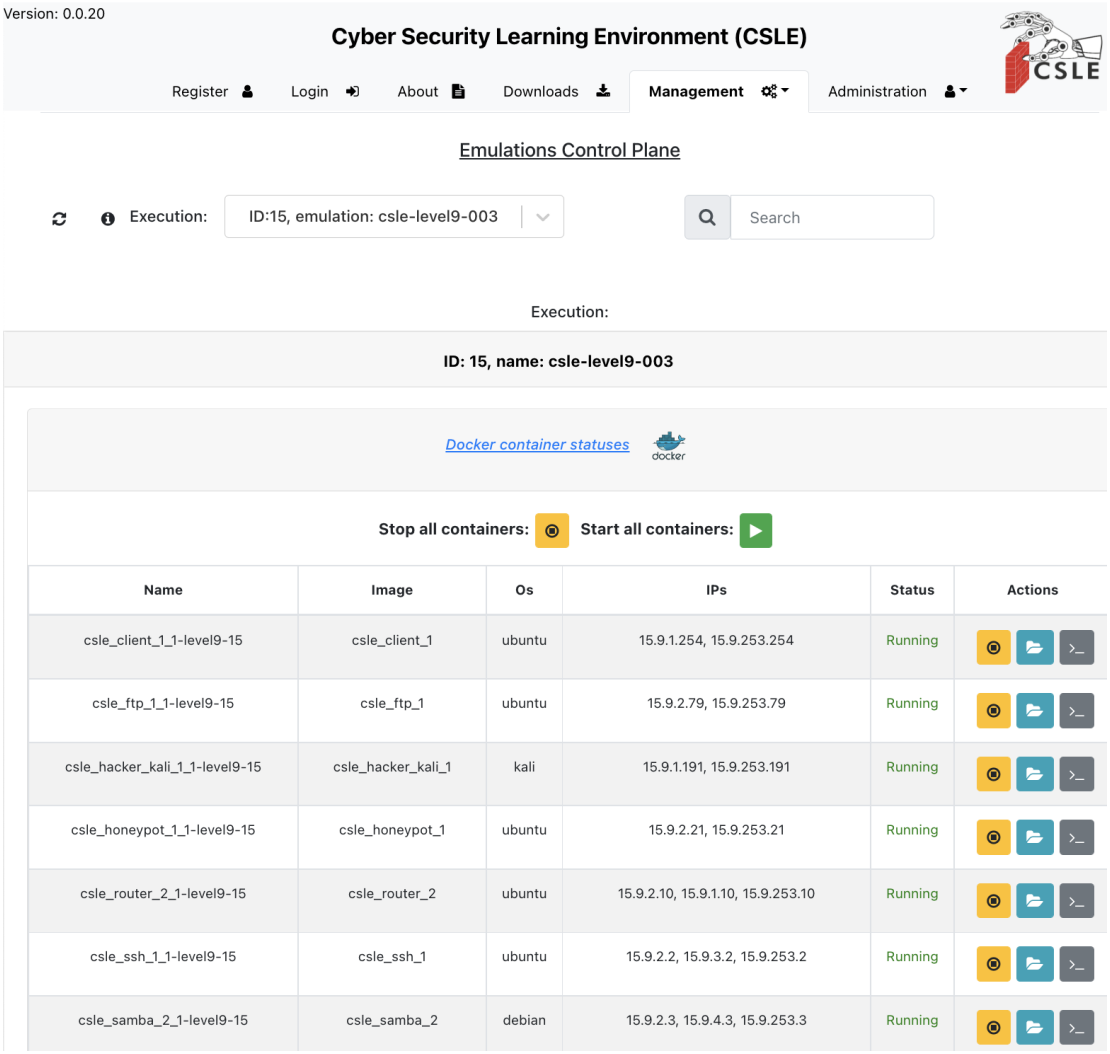

Figure 20: The control plane of the web interface; this page allows a user to request management operations, e.g., starting and stopping emulated components or services.

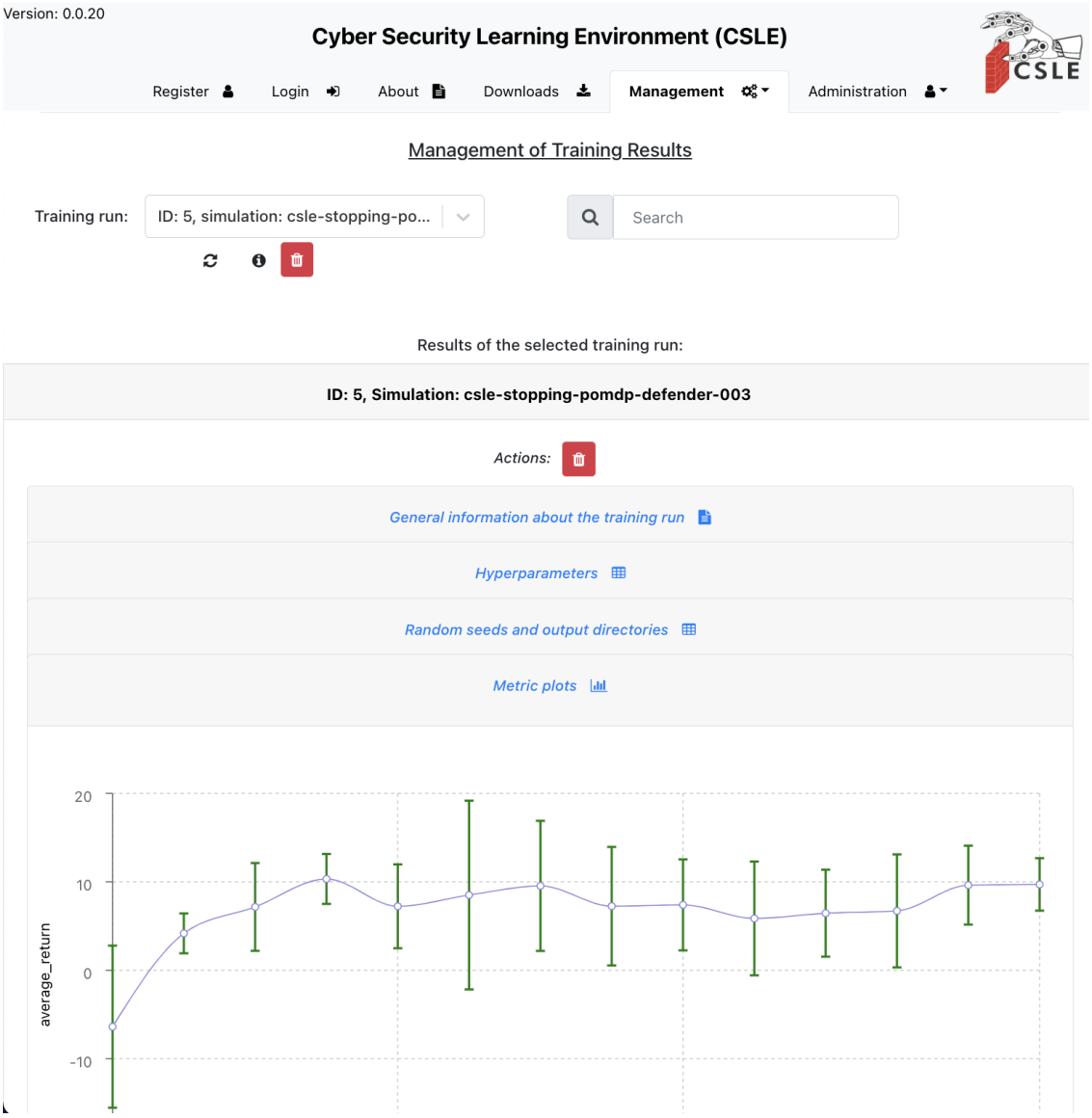

Figure 21: The training results page of the web interface; this page allows a user to view the results of reinforcement learning experiments.

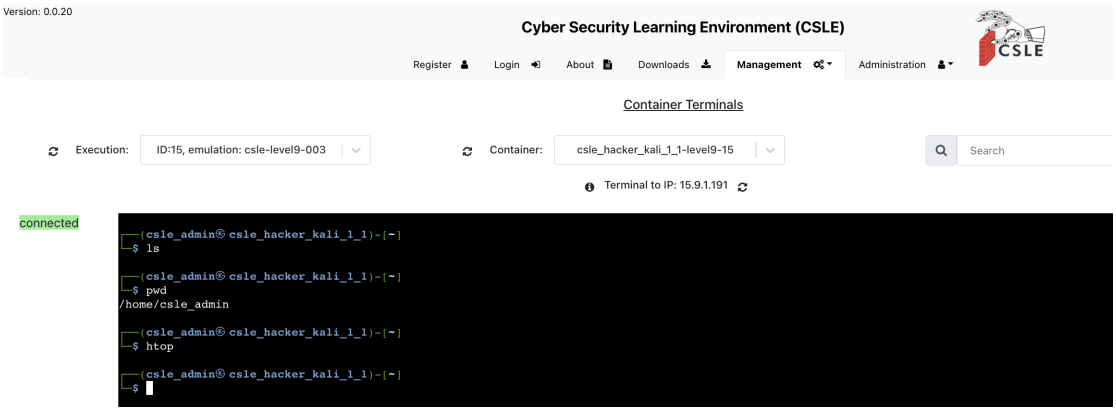

Figure 22: The container-terminal page of the web interface; this page allows a user to execute arbitrary bash commands inside a container of an emulated infrastructure.

2.6.1 The Policy Examination System

The policy examination system allows a user to traverse episodes of Markov decision processes in a controlled manner and to track the actions triggered by security policies. Similar to a software debugger, a user can continue or or halt an episode at any time-step and inspect parameters and probability distributions of interest. The system enables insight into the structure of a given policy and in the behavior of a policy in edge cases. It is integrated with the rest of the web interface and can be accessed as a regular web page (see Fig. 23).

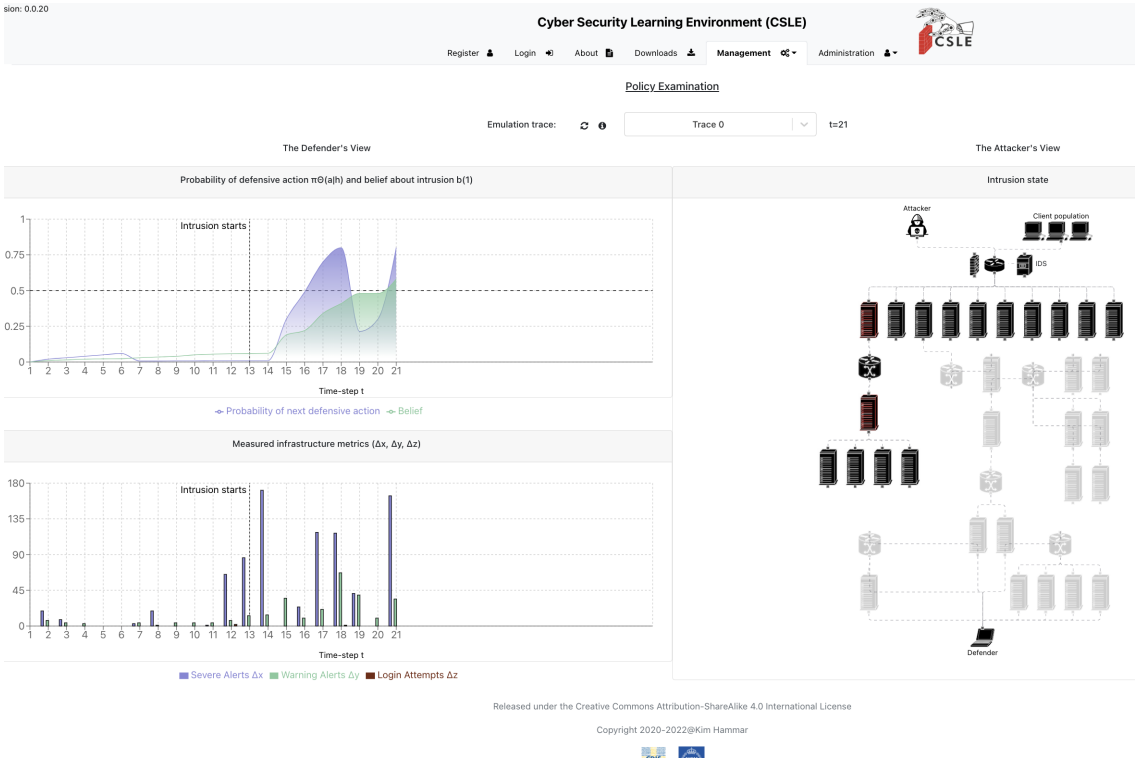

Figure 23: The demonstration view of the policy examination system; the system allows a user to interactively step through an episode of a Markov decision process.

Figure 23 pictures the main user interface of the policy examination system. The left part of the interface shows the defender's view and the right part shows the attacker's view. The plot on the upper left shows the defender's belief about the infrastructure's state and the probability that the defender takes an action at each time-step. The plot on the lower left shows the distribution of infrastructure metrics that the defender observes. The graphic on the right shows the attacker's view, depicted as an overlay on the IT-infrastructure's topology.

The policy examination system provides insight into the structure of the defender policy and its behavior in edge-cases. It can be used to observe correlations among the attacker's actions, the infrastructure metrics, and the actions that the defender policy prescribes. It can also be used to examine which actions of the attacker are difficult for the defender to detect or trigger actions by the defender.

2.6.2 Implementation

The web interface is implemented by a web application written in JavaScript [7] and the React framework [8], which executes on the management system. Each physical server of the management system runs a Flask web server [39] that exposes an HTTP REST API (the REST API is detailed in the subsequent section). In front of these web servers is a Nginx [44] reverse proxy that load balances client requests, decrypts incoming HTTPS traffic, and encrypts outgoing HTTP traffic using TLS (see Fig. 24).

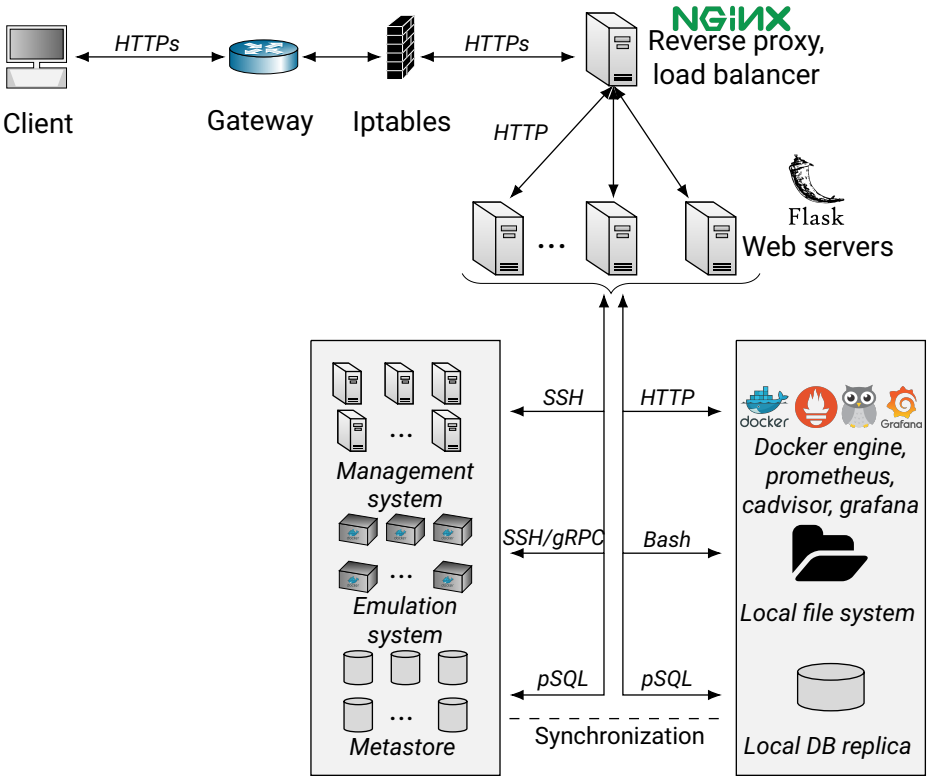

Figure 24: Architecture diagram of the CSLE web application; a Nginx reverse proxy load balances requests to a group of Flask web servers, which expose HTTP REST APIs; these APIs can be used to view management information and to request management operations.

The web-based terminal emulation shown in Fig. 22 is implemented as follows. To create a web-based terminal session, the web client's browser first opens a websocket to one of the HTTP servers. The server then opens an SSH tunnel to the emulated device and creates a terminal session. This tunnel is then used to pipe commands and terminal responses between the client and the terminal (see Fig. 25).

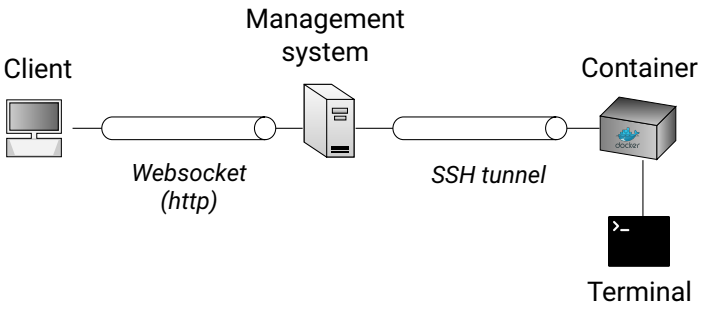

Figure 25: Architecture diagram of the web-based terminal emulation in CSLE; the web application in the client's browser opens a web socket to an HTTP server running on one of the physical servers of the management system; the web server then pipes the websocket to an SSH tunnel that connects to a terminal of the emulated component.

2.6.3 The REST API

The endpoints of the REST API are listed in Tables 20-23. An example HTTP request and response is given below.

```
1  curl https://csle.dev/emulations?ids=true\&token=<my_token>
2  [{"emulation":"csle-level1-003","id":1,"running":false},
3  {"emulation":"csle-level2-003","id":2,"running":false},
4  {"emulation":"csle-level3-003","id":3,"running":false},
5  {"emulation":"csle-level4-003","id":4,"running":false},
6  {"emulation":"csle-level5-003","id":5,"running":false},
7  {"emulation":"csle-level6-003","id":6,"running":false},
8  {"emulation":"csle-level7-003","id":7,"running":false},
9  {"emulation":"csle-level8-003","id":8,"running":false},
10 {"emulation":"csle-level9-003","id":9,"running":true},
11 {"emulation":"csle-level10-003","id":10,"running":false},
12 {"emulation":"csle-level11-003","id":11,"running":false},
13 {"emulation":"csle-level12-003","id":12,"running":false}]
```

Listing 3: An HTTP request for the `emulations` resource of the CSLE REST API (line 1) and the corresponding HTTP response (lines 2-13); the request includes the flag `ids=true`, which means that the response only includes the identifiers, names, and statuses of the emulations, rather than the full configurations; the request is performed with `curl` [48].

Resource	*Method*
/emulations	GET, DELETE
/emulations?ids=true&token=<token>	GET, DELETE
/emulations/<em_id>?token=<token>	GET, DELETE, POST
/emulations/<em_id>/executions?token=<token>	GET, DELETE
/emulations/<em_id>/executions/<exec_id>?token=<token>	GET, DELETE
/emulations/<em_id>/executions/<exec_id>/monitor/<min>?token=<token>	GET
/simulations?token=<token>	GET, DELETE
/simulations?ids=true&token=<token>	GET, DELETE
/simulations/<simulation_id>?token=<token>	GET, DELETE
/cadvisor?token=<token>	POST, GET
/nodeexporter?token=<token>	POST, GET
/grafana?token=<token>	POST, GET
/prometheus?token=<token>	POST, GET
/alpha-vec-policies?token=<token>	GET, DELETE
/alpha-vec-policies?ids=true&token=<token>	GET, DELETE
/alpha-vec-policies/<policy_id>?token=<token>	GET, DELETE
/dqn-policies?token=<token>	GET, DELETE
/dqn-policies?ids=true&token=<token>	GET, DELETE
/dqn-policies/<policy_id>?token=<token>	GET, DELETE
/ppo-policies?token=<token>	GET, DELETE
/ppo-policies?ids=true&token=<token>	GET, DELETE
/ppo-policies/<policy_id>?token=<token>	GET, DELETE
/vector-policies?token=<token>	GET, DELETE
/vector-policies?ids=true&token=<token>	GET, DELETE
/vector-policies/<policy_id>?token=<token>	GET, DELETE
/tabular-policies?token=<token>	GET, DELETE
/tabular-policies?ids=true&token=<token>	GET, DELETE
/tabular-policies/<policy_id>?token=<token>	GET, DELETE
/multi-threshold-policies?token=<token>	GET, DELETE
/multi-threshold-policies?ids=true&token=<token>	GET, DELETE
/multi-threshold-policies/<policy_id>?token=<token>	GET, DELETE
/linear-threshold-policies?token=<token>	GET, DELETE
/linear-threshold-policies?ids=true&token=<token>	GET, DELETE
/linear-threshold-policies/<policy_id>?token=<token>	GET, DELETE
/fnn-w-softmax-policies?token=<token>	GET, DELETE
/fnn-w-softmax-policies?ids=true&token=<token>	GET, DELETE
/fnn-w-softmax-policies/<policy_id>?token=<token>	GET, DELETE
/training-jobs?token=<token>	GET, DELETE
/training-jobs?ids=true&token=<token>	GET, DELETE
/training-jobs/<job_id>?token=<token>	GET, DELETE, POST
/data-collection-jobs?token=<token>	GET, DELETE
/data-collection-jobs?ids=true&token=<token>	GET, DELETE
/data-collection-jobs/<job_id>?token=<token>	GET, DELETE, POST
/system-identification-jobs?token=<token>	GET, DELETE
/system-identification-jobs?ids=true&token=<token>	GET, DELETE
/system-identification-jobs/<job_id>?token=<token>	GET, DELETE

Table 20: REST API resources in version 0.4.0 of CSLE (1/4).

Resource	*Method*
/emulation-traces?token=<token>	GET, DELETE
/emulation-traces?ids=true&token=<token>	GET, DELETE
/emulation-traces/<trace_id>?token=<token>	GET, DELETE
/simulation-traces?token=<token>	GET, DELETE
/simulation-traces?ids=true&token=<token>	GET, DELETE
/simulation-traces/<trace_id>?token=<token>	GET, DELETE
/emulation-simulation-traces?token=<token>	GET, DELETE
/emulation-simulation-traces?ids=true&token=<token>	GET, DELETE
/emulation-simulation-traces/<trace_id>?token=<token>	GET, DELETE
/images?token=<token>	GET
/file?token=<token>	POST
/experiments?token=<token>	GET, DELETE
/experiments?ids=true&token=<token>	GET, DELETE
/experiments/<trace_id>?token=<token>	GET, DELETE
/sdn-controllers?token=<token>	GET, DELETE
/sdn-controllers?ids=true&token=<token>	GET, DELETE
/emulation-statistics?token=<token>	GET, DELETE
/emulation-statistics?ids=true&token=<token>	GET, DELETE
/emulation-statistics/<trace_id>?token=<token>	GET, DELETE
/gaussian-mixture-system-models?token=<token>	GET, DELETE
/gaussian-mixture-system-models?ids=true&token=<token>	GET, DELETE
/gaussian-mixture-system-models/<trace_id>?token=<token>	GET, DELETE
/empirical-system-models?token=<token>	GET, DELETE
/empirical-system-models?ids=true&token=<token>	GET, DELETE
/empirical-system-models/<trace_id>?token=<token>	GET, DELETE
/mcmc-system-models?token=<token>	GET, DELETE
/mcmc-system-models?ids=true&token=<token>	GET, DELETE
/mcmc-system-models/<trace_id>?token=<token>	GET, DELETE
/gp-system-models?token=<token>	GET, DELETE
/gp-system-models?ids=true&token=<token>	GET, DELETE
/gp-system-models/<model_id>?token=<token>	GET, DELETE
/system-models?ids=true&token=<token>	GET
/login	POST
/traces-datasets?token=<token>	GET, DELETE
/traces-datasets?ids=true&token=<token>	GET, DELETE
/traces-datasets/<traces_datasets_id>?token=<token>	GET, DELETE
/traces-datasets/<traces_datasets_id>?token=<token>&download=true	GET
/server-cluster?token=<token>	GET
/pgadmin?token=<token>	GET,POST
/postgresql?token=<token>	GET,POST
/docker?token=<token>	GET,POST
/nginx?token=<token>	GET,POST
/flask?token=<token>	GET,POST
/clusterstatuses?token=<token>	GET
/logs/nginx?token=<token>	POST
/logs/postgresql?token=<token>	POST
/logs/docker?token=<token>	POST
/logs/flask?token=<token>	POST
/logs/clustermanager?token=<token>	POST

Table 21: REST API resources in version 0.4.0 of CSLE (2/4).

Resource	*Method*
/statistics-datasets?token=<token>	GET, DELETE
/statistics-datasets?ids=true&token=<token>	GET, DELETE
/statistics-datasets/<statistics_datasets_id>?token=<token>	GET, DELETE
/statistics-datasets/<statistics_datasets_id>?token=<token>&download=true	GET
/emulation-executions?token=<token>	GET, DELETE
/emulation-executions?ids=true&token=<token>	GET, DELETE
/emulation-executions/<exec_id>?token=<token>	GET, DELETE
/emulation-executions/<exec_id>/info?token=<token>&emulation=	GET
/emulation-executions/<exec_id>/docker-stats-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/docker-stats-monitor?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/client-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/client-population?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/client-producer?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/kafka-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/kafka?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/snort-ids-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/snort-ids-monitor?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/snort-ids?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/ossec-ids-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/ossec-ids?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/ossec-ids-monitor?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/host-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/host-monitor?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/elk-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/elastic?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/logstash?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/kibana?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/container?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/elk-stack?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/traffic-generator?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/filebeat?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/packetbeat?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/metricbeat?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/ryu-manager?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/ryu-monitor?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/ryu-controller?token=<token>&emulation=	POST
/emulation-executions/<exec_id>/switches?token=<token>&emulation=	POST
/users?token=<token>	GET, DELETE, PUT
/users?ids=true&token=<token>	GET, DELETE
/users/<user_id>?token=<token>	GET, DELETE, PUT
/users/create	POST
/logs?token=<token>	GET
/logs/docker-stats-manager?token=<token>	POST
/logs/prometheus?token=<token>	POST

Table 22: REST API resources in version 0.4.0 of CSLE (3/4).

Resource	*Method*
/logs/grafana?token=<token>	POST
/logs/cadvisor?token=<token>	POST
/logs/pgadmin?token=<token>	POST
/logs/container?token=<token>	GET
/logs/node-exporter?token=<token>	POST
/logs/client-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/kafka-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/elk-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/ryu-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/traffic-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/snort-ids-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/ossec-ids-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/host-manager?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/ossec-ids?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/snort-ids?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/kafka?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/elk-stack?token=<token>&emulation=&executionid=<exec_id>	POST
/logs/ryu-controller?token=<token>&emulation=&executionid=<exec_id>	POST
/config?token=<token>	GET, PUT
/config/registration-allowed	GET
/version	GET
/about-page	GET
/login-page	GET
/register-page	GET
/emulation-statistics-page	GET
/emulations-page	GET
/images-page	GET
/downloads-page	GET
/jobs-page	GET
/monitoring-page	GET
/policies-page	GET
/policy-examination-page	GET
/sdn-controllers-page	GET
/control-plane-page	GET
/user-admin-page	GET
/system-admin-page	GET
/logs-admin-page	GET
/simulations-page	GET
/system-models-page	GET
/traces-page	GET
/training-page	GET
/container-terminal-page	GET
/container-terminal?token=<token>	Websockets

Table 23: REST API resources in version 0.4.0 of CSLE (4/4).

2.7 The Command-Line Interface (CLI)

The Command-Line Interface (CLI) allows a user to inspect the state of emulations and to execute management operations from the command-line. The commands available in the CLI are listed in Tables 24-25. An example command and the resulting output is shown below.

```
1  kim@gpu2 ~> csle ls emulations --all
2  CSLE emulations:
3  csle-level9-003 [running]
4  csle-level1-003 [stopped]
5  csle-level2-003 [stopped]
6  csle-level3-003 [stopped]
7  csle-level4-003 [stopped]
8  csle-level5-003 [stopped]
9  csle-level6-003 [stopped]
10 csle-level7-003 [stopped]
11 csle-level8-003 [stopped]
12 csle-level10-003 [stopped]
13 csle-level11-003 [stopped]
14 csle-level12-003 [stopped]
```

Listing 4: The command `csle ls emulations -all` (line 1) and the output generated by the CSLE CLI (lines 2-14).

Command	*Description*
`csle attacker <em> <execid>`	Opens an attacker shell in a given execution.
`csle clean all`	Removes all containers, networks, emulations, etc.
`csle clean containers` <execid>	Removes all Docker containers in a given execution.
`csle clean emulations` <execid>	Removes all emulations in a given execution.
`csle clean emulation_traces` <execid>	Removes all emulation traces in a given execution.
`csle clean simulation_traces` <execid>	Removes all simulation traces in a given execution.
`csle clean emulation_statistics` <execid>	Removes all emulation statistics in a given execution.
`csle clean emulation_executions` <execid>	Removes all emulation executions in a given execution.
`csle em <emname>`	Get status of a given emulation.
`csle em <emname> -host`	Get status of host managers in a given emulation.
`csle em <emname> -stats`	Get status of stats managers in a given emulation.
`csle em <emname> -kafka`	Get status of kafka managers in a given emulation.
`csle em <emname> -snortids`	Get status of snortids in a given emulation.
`csle em <emname> -clients`	Get status of clients in a given emulation.
`csle em <emname> -executions`	Get status of executions in a given emulation.
`csle em <emname> -executions`	Get status of executions in a given emulation.
`csle init`	Initializes CSLE and sets up management accounts.
`csle install emulations`	Install emulation environments.
`csle install simulations`	Install simulation environments.
`csle install <emname>`	Install a given emulation.
`csle install <simname>`	Install a given simulation.
`csle install derived images`	Install derived Docker images.
`csle install base images`	Install base Docker images.
`csle install metastore`	Install the metastore.
`csle install all`	Install everything.
`csle ls all`	List all information about entities.
`csle ls containers`	List all containers.
`csle ls networks`	List all networks.
`csle ls images`	List all Docker images.
`csle ls emulations`	List all emulations.
`csle ls environments`	List all environments.
`csle ls prometheus`	List status of Prometheus.
`csle ls node_exporter`	List status of node exporter.
`csle ls cadvisor`	List status of cadvisor.
`csle ls statsmanager`	List status of statsmanager.
`csle ls flask`	List status of the REST API server.
`csle ls simulations`	List status of simulations.
`csle ls emulation_executions`	List status of emulation executions.
`csle ls <entity> -all`	List extended information of the given entity.
`csle ls <entity> -running`	Only list information about running entities.
`csle ls <entity> -stopped`	Only list information about stopped entities.
`csle rm <network>`	Removes the network with the given name.
`csle rm <container>`	Removes the container with the given name.
`csle rm <image>`	Removes the Docker image with the given name.
`csle rm networks`	Removes all networks.
`csle rm images`	Removes all images.
`csle rm containers`	Removes all containers.
`csle rm help`	Lists all available commands.

Table 24: Commands available in the CSLE command-line interface (version 0.4.0) (1/2).

Command	*Description*
`csle shell <containername>`	Opens shell in a given container.
`csle start prometheus -ip <ip>`	Starts prometheus on a given IP.
`csle start nginx -ip <ip>`	Starts nginx on a given IP.
`csle start postgresql -ip <ip>`	Starts postgresql on a given IP.
`csle start docker -ip <ip>`	Starts the Docker engine on a given IP.
`csle start clustermanager`	Starts the cluster manager locally.
`csle start node_exporter -ip <ip>`	Starts node exporter on a given IP.
`csle start grafana -ip <ip>`	Starts grafana on a given IP.
`csle start cadvisor -ip <ip>`	Starts cadvisor on a given IP.
`csle start flask -ip <ip>`	Starts the REST API server on a given IP.
`csle start <containername>`	Starts the given container.
`csle start <emulationname>`	Starts the given emulation.
`csle start <emulationname> -no_network`	Starts the given emulation without virtual networks.
`csle start <emulationname> -no_traffic`	Starts the given emulation without traffic generators.
`csle start <emulationname> -no_clients`	Starts the given emulation without clients.
`csle start <emulationname> -id`	Starts the given emulation with execution id.
`csle start all`	Starts everything.
`csle start all -id`	Starts everything in an execution.
`csle start statsmanager -ip <ip>`	Starts the statsmanager on a given IP.
`csle start <trainingjobid>`	Starts trainingjob with a given id.
`csle start <systemidjobid>`	Starts system identification job with a given id.
`csle start <image> <containername>`	Starts a container with a given image and name.
`csle start_traffic <emulationname> <executionid>`	Starts the traffic and clients in execution.
`csle statsmanager <port> <logdir> <logfile> <maxworkers>`	Starts the statsmanager locally.
`csle stop <emulationname> <execid>`	Stops the emulation execution.
`csle stop <prometheus> -ip <ip>`	Stops prometheus on a given IP.
`csle stop <cadvisor> -ip <ip>`	Stops cadvisor on a given IP.
`csle stop <grafana> -ip <ip>`	Stops grafana on a given IP.
`csle stop <flask> -ip <ip>`	Stops the REST API server on a given IP.
`csle stop <containername> <execid>`	Stops the container.
`csle stop <statsmanager> -ip <ip>`	Stops the statsmanager on a given ip.
`csle stop emulation_executions`	Stops all emulation executions.
`csle stop nginx -ip <ip>`	Stops Nginx on the given IP.
`csle stop docker -ip <ip>`	Stops the Docker engine on the given IP.
`csle stop postgresql -ip <ip>`	Stops Postgresql on the given IP.
`csle stop all`	Stops everything that runs.
`csle stop_traffic <emulationname> <execid>`	Stops client population and traffic in execution.
`csle systemidentificationjob <jobid>`	Starts job.
`csle trainingjob <jobid>`	Starts job.
`csle datacollectionjob <jobid>`	Starts job.
`csle uninstall emulations`	Uninstalls emulation environments.
`csle uninstall simulations`	Uninstalls simulation environments.
`csle uninstall <emname>`	Uninstalls emulation.
`csle uninstall <simname>`	Uninstalls simulation.
`csle uninstall derived_images`	Uninstalls derived images.
`csle uninstall base_images`	Uninstalls base images.
`csle uninstall metastore`	Uninstalls metastore.
`csle uninstall all`	Uninstalls everything.

Table 25: Commands available in the CSLE command-line interface (version 0.4.0) (2/2).

3 Getting Started

This section contains instructions for getting started with CSLE. It includes instructions for installation and operation of CSLE, as well as instructions on how to uninstall CSLE.

3.1 Installing CSLE

The installation of CSLE can be divided in four main steps (see Fig. 26). The first step is "Installation setup", which comprises installation configuration and installation of build tools. In the second step, the metastore and the simulation system are installed. Lastly, in the third and the fourth steps, the emulation system and the management system are installed, respectively.

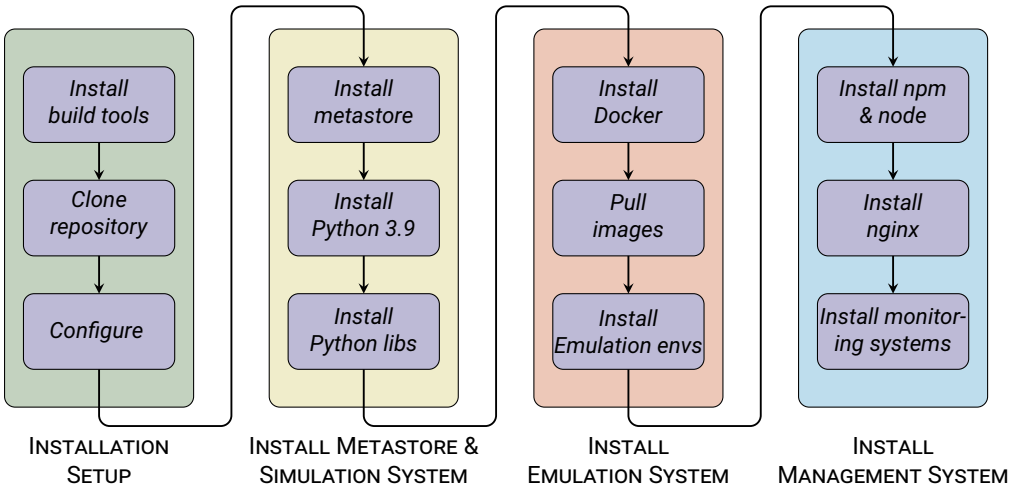

Figure 26: Steps to install CSLE.

3.1.1 Prerequisites

To install and run CSLE you need at least one server or virtual machine that meets the following criteria:

- Ubuntu 18.04+;
- at least 16GB RAM (the exact amount of RAM necessary depends on the emulations to deploy; 16GB is sufficient for the smallest emulations, e.g., in the size of 5-10 containers);
- at least 2 CPUs;
- 50 GB of free hard-disk space;

- outside Internet access;
- and a UNIX user account with `sudo` privileges.

Below are a list of dependencies that will be installed with CSLE (unless they are already installed):

- Docker 20.10.14+;
- Python 3.9+;
- Python libraries: `torch`, `numpy`, `gym`, `docker`, `paramiko`, `stable-baselines3`, `psycopg`, `pyglet`, `flask`, `click`, `waitress`, `scp`, `psutil`, `grpcio`, `grpcio-tools`, `scipy`, `confluent-kafka`, `requests`, `pyopenssl`, `sphinx`, `mypy`, `mypy-extensions`, `mypy-protobuf`, `types-PyYAML`, `types-protobuf`, `types-paramiko`, `types-requests`, `types-urllib3`, `flake8`, `flake8-rst-docstrings`, `pytest`, `gevent`, `eventlet`, `dnspython`, `csle-ryu-fork`, `gpytorch`, `pulp`, `Bayesian optimization`, `emukit`, `cma`, `pycryptodome`.
- PostgreSQL 15+;
- Citus 11.2+;
- `build-essential`, `make`, `git`, `bzip2`;
- Prometheus 2.23+;
- Node exporter 1.0.1+;
- cAdvisor (any version);
- Grafana (any version);
- Node v16.13.1+;
- and `npm` v6.14.8+.

3.1.2 Physical Resources

CSLE can be installed on a cluster of servers. All servers need to be connected through an IP network. One of the servers should be designated as the "leader" (see Fig. 3). The other servers are workers. (If you install CSLE on a single server then that server is the master and there are no workers.) If not stated otherwise, all commands in the installation instructions below should be executed on every server in the cluster. Certain commands should only be executed on the leader, which will be clearly indicated.

3.1.3 Installation setup

In this step of the installation, the source code of CSLE is downloaded, configuration parameters are set, and tools required in later steps of the installation are installed.

Start with installing the necessary build tools and version management tools by running the following commands:

```
sudo apt install build-essential
sudo apt install make
sudo apt install git
sudo apt install bzip2
sudo apt install nginx
wget
 ↪  https://repo.anaconda.com/archive/Anaconda3-2022.10-Linux-x86_64.sh
chmod u+rwx Anaconda3-2022.10-Linux-x86_64.sh
./Anaconda3-2022.10-Linux-x86_64.sh
```

Listing 5: Commands to install build tools and version management tools.

Next, clone the CSLE repository and setup environment variables by running the commands:

```
git clone https://github.com/Limmen/csle
export CSLE_HOME=/path/to/csle/ # for bash
set -gx CSLE_HOME "/path/to/csle"  # for fish
```

Listing 6: Commands to clone the CSLE repository and setup environment variables.

Further, add the following line to `.bashrc` to set the environment variable `CSLE_HOME` permanently in the bash shell:

```
export CSLE_HOME=/path/to/csle/
```

Listing 7: Line to add to `.bashrc` to set the `CSLE_HOME` environment variable.

and add the following line to the fish configuration file to set the environment variable `CSLE_HOME` permanently in the fish shell:

```
set -gx CSLE_HOME "/path/to/csle"
```

Listing 8: Line to add to the fish configuration file to set the `CSLE_HOME` environment variable.

After performing the steps above, you should have the directory layout shown in Fig. 27.

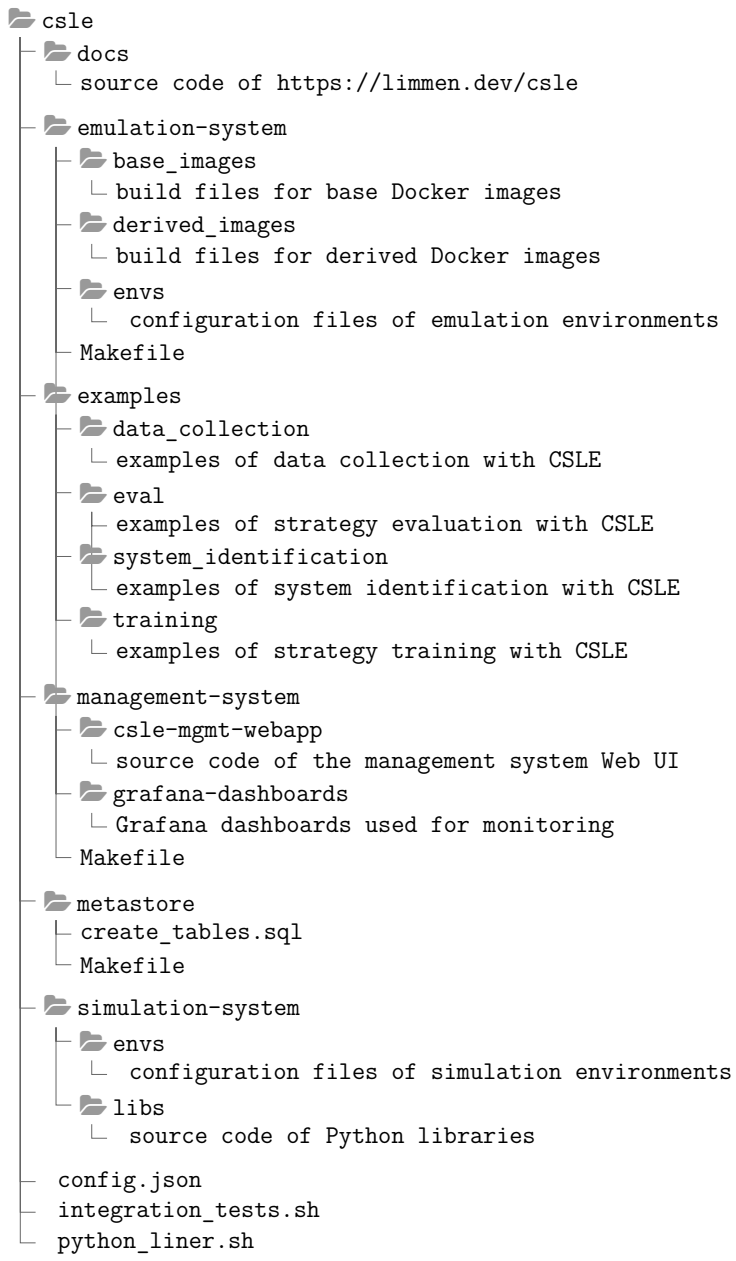

Figure 27: The directory layout of the CSLE code repository.

Next, create a directory to store PID files by running the following commands (change <my_user> to your username):

```
mkdir /var/log/csle
sudo chmod -R u+rw /var/log/csle
sudo chown -R <my_user> /var/log/csle
```

Listing 9: Commands to setup the pid file directory of CSLE.

Similarly, create a directory to store log files by running the following commands (change `<my_user>` to your username):

```
mkdir /tmp/csle
sudo chmod -R u+rw /tmp/csle
sudo chown -R <my_user> /tmp/csle
```

Listing 10: Commands to setup the log file directory of CSLE.

Next, add the following line to the `sudoers` file using `visudo` (change `<my_user>` to your username):

☞ Warning: take care when editing the sudoers file. If the sudoers file becomes corrupted it can make the system unusable.

```
<my_user> ALL = NOPASSWD: /usr/sbin/service docker stop,
↪  /usr/sbin/service docker start, /usr/sbin/service docker
↪  restart, /usr/sbin/service nginx stop, /usr/sbin/service
↪  nginx start, /usr/sbin/service nginx restart,
↪  /usr/sbin/service postgresql start, /usr/sbin/service
↪  postgresql stop, /usr/sbin/service postgresql restart,
↪  /bin/kill, /usr/bin/journalctl -u docker.service -n 100
↪  --no-pager -e
```

Listing 11: Line to add to the sudoers file.

By adding the above line to the `sudoers` file, CSLE will be able to view logs and start and stop management services without requiring a password to be entered. (Note that the exact paths used above may differ on your system, very the paths by running the command `whereis service`, `whereis journalctl`, etc.)

Next, setup SSH keys so that all servers (leader and workers) have SSH access to each other without requiring a password. To do this, start by generating an SSH key pair on each server using the command `ssh-keygen`. After

generating the keys, copy the public key of each server (e.g., `id_rsa.pub`) to the file `.ssh/authorized_keys` on the other servers.

Lastly, define default username and password to the management system by editing the file: `csle/config.json`.

3.1.4 Installing the Metastore

The metastore is based on PostgreSQL and Citus. Installing the metastore thus corresponds to installing and configuring PostgreSQL and Citus.

To install PostgreSQL v15 and the Citus extension v11.2, run the following commands:

```
curl https://install.citusdata.com/community/deb.sh | sudo bash
sudo apt-get -y install postgresql-15-citus-11.2
sudo pg_conftool 15 main set shared_preload_libraries citus
sudo pg_conftool 15 main set listen_addresses '*'
```

Listing 12: Commands to install PostgreSQL and the Citus extension.

Verify the installed version of PostgreSQL by running the command:

```
psql --version
```

Listing 13: Commands to verify the installed version of PostgreSQL.

Next, setup a password for the `postgres` user by running the commands:

```
sudo -u postgres psql  # start psql session
psql> \password postgres # set postgres password
```

Listing 14: Commands to setup a password for the `postgres` user.

Next, setup password authentication for the `postgres` user and allow remote connections by performing the following steps:

1. Open the file "`/etc/postgresql/<YOUR_VERSION>/main/pg_hba.conf`" and replace the existing content to match your IP addresses and desired security level:

```
local all  postgres md5
host all all 127.0.0.1/32 trust
host all all ::1/128 trust
host all all 172.31.212.0/24 trust
```

Listing 15: Lines to add to `pg_hba.conf`; note: to allow external connections `127.0.0.1/32` can be changed to `0.0.0.0/0`.

2. Restart PostgreSQL with the command:

```
sudo service postgresql restart
```

Listing 16: Command to restart PostgreSQL.

3. Run the following command to have PostgreSQL restarted automatically when the server is restarted:

```
sudo update-rc.d postgresql enable
```

Listing 17: Command to make PostgreSQL start automatically when the server starts.

After completing the steps above, create the CSLE database and setup the Citus extension by running the following command:

```
cd metastore; make db
```

Listing 18: Commands to create the CSLE database and setup the Citus extension.

Next, edit the file `csle/metastore/create_cluster.sql` and configure IP addresses of the worker servers and of the leader. Then, **on the leader**, run the following commands to setup the Citus cluster and create the tables:

```
cd metastore; make cluster
cd metastore; make tables
```

Listing 19: Commands to setup the Citus cluster and create tables.

Next, update the variable called `HOST` under the class `METADATA_STORE` in the file

`csle/simulation-system/libs/csle-common/src/csle\_common/constants/constants.py`

Next, define ips of the cluster nodes and thet metastore leader by editing the file: `csle/config.json`.

Lastly, make the PostgreSQL log files readable by your user by running the commands:

```
sudo chmod -R u+rw /var/log/postgresql
sudo chown -R <my_user> /var/log/postgresql
```

Listing 20: Commands to make the PostgreSQL log files readable for a given user.

PostgreSQL Debugging If PostgreSQL is not starting for some reason, check the reason by running the command:

```
sudo service postgresql@15-main status
```

Listing 21: Command to check the status of PostgreSQL.

3.1.5 Installing the Simulation System

The simulation system consists of a set of Python libraries and a set of configuration files. To install the simulation system, the Python libraries need to be installed and the configuration files need to be inserted into the metastore.

If you do not have Python >3.9 in your base environment, start with installing Python 3.9 by running the commands:

```
conda create -n py39 python=3.9
conda activate py39
```

Listing 22: Command to install Python 3.9 using Anaconda [27].

The simulation system includes 12 Python libraries: `csle-collector`, `csle-ryu`, `csle-common`, `csle-attacker`, `csle-defender`, `csle-system-identification`,

gym-csle-stopping-game, csle-agents, csle-rest-api, csle-cli, csle-cluster, gym-csle-intrusion-response-game, csle-base, csle-tolerance, gym-csle-cyborg,and gym-csle-apt-game. These libraries can either be installed from PyPi[7] or directly from source.

To install all libraries at once from PyPi, run one of the following commands:

```
pip install csle-base csle-collector csle-ryu csle-common
  ↪  csle-attacker csle-defender csle-system-identification
  ↪  gym-csle-stopping-game csle-agents csle-rest-api csle-cli
  ↪  csle-cluster gym-csle-intrusion-response-game csle-tolerance
  ↪  gym-csle-apt-game gym-csle-cyborg
cd simulation-system/libs; ./local_install.sh
```

Listing 23: Commands to install all CSLE python libraries from PyPi.

To install the libraries one by one rather than all at once, follow the instructions below.

Install csle-base from PyPi by running the command:

```
pip install csle-base
```

Listing 24: Command to install csle-base from PyPi.

Alternatively, install csle-base from source by running the commands:

```
cd simulation-system/libs/csle-base/
pip install -e .
cd ../../../
```

Listing 25: Commands to install csle-base from source.

[7] https://pypi.org/

Next, install `csle-collector` from PyPi by running the command:

```
pip install csle-collector
```

Listing 26: Command to install `csle-collector` from PyPi.

Alternatively, install `csle-collector` from source by running the commands:

```
cd simulation-system/libs/csle-collector/
pip install -e .
cd ../../../
```

Listing 27: Commands to install `csle-collector` from source.

Next, install `csle-ryu` from PyPi by running the command:

```
pip install csle-ryu
```

Listing 28: Command to install `csle-ryu` from PyPi.

Alternatively, install `csle-ryu` from source by running the commands:

```
cd simulation-system/libs/csle-ryu/
pip install -e .
cd ../../../
```

Listing 29: Commands to install `csle-ryu` from source.

Next, install `csle-common` from PyPi by running the command:

```
pip install csle-common
```

Listing 30: Command to install `csle-common` from PyPi.

Alternatively, install `csle-common` from source by running the commands:

```
cd simulation-system/libs/csle-common/
pip install -e .
cd ../../../
```

Listing 31: Commands to install `csle-common` from source.

Next, install `csle-attacker` from PyPi by running the command:

```
pip install csle-attacker
```

Listing 32: Command to install `csle-attacker` from PyPi.

Alternatively, install `csle-attacker` from source by running the commands:

```
cd simulation-system/libs/csle-attacker/
pip install -e .
cd ../../../
```

Listing 33: Commands to install `csle-attacker` from source.

Next, install `csle-defender` from PyPi by running the command:

```
pip install csle-defender
```

Listing 34: Command to install `csle-defender` from PyPi.

Alternatively, install `csle-defender` from source by running the commands:

```
cd simulation-system/libs/csle-defender/
pip install -e .
cd ../../../
```

Listing 35: Commands to install `csle-defender` from source.

Next, install `csle-system-identification` from PyPi by running the command:

```
pip install csle-system-identification
```

Listing 36: Command to install `csle-system-identification` from PyPi.

Alternatively, install `csle-system-identification` from source by running the commands:

```
cd simulation-system/libs/csle-system-identification/
pip install -e .
cd ../../../
```

Listing 37: Commands to install `csle-system-identification` from source.

Next, install `gym-csle-stopping-game` from PyPi by running the command:

```
pip install gym-csle-stopping-game
```

Listing 38: Command to install `gym-csle-stopping-game` from PyPi.

Alternatively, install `gym-csle-stopping-game` from source by running the commands:

```
cd simulation-system/libs/gym-csle-stopping-game/
pip install -e .
cd ../../../
```

Listing 39: Commands to install `gym-csle-stopping-game` from source.

Next, install `gym-csle-apt-game` from PyPi by running the command:

```
pip install gym-csle-apt-game
```

Listing 40: Command to install `gym-csle-apt-game` from PyPi.

Alternatively, install `gym-csle-apt-game` from source by running the commands:

```
cd simulation-system/libs/gym-csle-apt-game/
pip install -e .
cd ../../../
```

Listing 41: Commands to install `gym-csle-apt-game` from source.

Next, install `gym-csle-cyborg` from PyPi by running the command:

```
pip install gym-csle-cyborg
```

Listing 42: Command to install `gym-csle-cyborg` from PyPi.

Alternatively, install `gym-csle-cyborg` from source by running the commands:

```
cd simulation-system/libs/gym-csle-cyborg/
pip install -e .
cd ../../../
```

Listing 43: Commands to install `gym-csle-cyborg` from source.

Next, install `csle-agents` from PyPi by running the command:

```
pip install csle-agents
```

Listing 44: Command to install `csle-agents` from PyPi.

Alternatively, install `csle-agents` from source by running the commands:

```
cd simulation-system/libs/csle-agents/
pip install -e .
cd ../../../
```

Listing 45: Commands to install `csle-agents` from source.

Next, install `csle-rest-api` from PyPi by running the command:

```
pip install csle-rest-api
```

Listing 46: Command to install `csle-rest-api` from PyPi.

Alternatively, install `csle-rest-api` from source by running the commands:

```
cd simulation-system/libs/csle-rest-api/
pip install -e .
cd ../../../
```

Listing 47: Commands to install `csle-rest-api` from source.

Next, install `csle-cli` from PyPi by running the command:

```
pip install csle-cli
```

Listing 48: Command to install `csle-cli` from PyPi.

Alternatively, install `csle-cli` from source by running the commands:

```
cd simulation-system/libs/csle-cli/
pip install -e .
cd ../../../
```

Listing 49: Commands to install `csle-cli` from source.

Next, install `csle-cluster` from PyPi by running the command:

```
pip install csle-cluster
```

Listing 50: Command to install `csle-cluster` from PyPi.

Alternatively, install `csle-cluster` from source by running the commands:

```
cd simulation-system/libs/csle-cluster/
pip install -e .
cd ../../../
```

Listing 51: Commands to install `csle-cluster` from source.

Next, install `gym-csle-intrusion-response-game` from PyPi by running the command:

```
pip install gym-csle-intrusion-response-game
```

Listing 52: Command to install `gym-csle-intrusion-response-game` from PyPi.

Alternatively, install `gym-csle-intrusion-response-game` from source by running the commands:

```
cd simulation-system/libs/gym-csle-intrusion-response-game/
pip install -e .
cd ../../../
```

Listing 53: Commands to install `gym-csle-intrusion-response-game` from source.

Next, install `csle-tolerance` from PyPi by running the command:

```
pip install csle-tolerance
```

Listing 54: Command to install `csle-tolerance` from PyPi.

Alternatively, install `csle-tolerance` from source by running the commands:

```
cd simulation-system/libs/csle-tolerance/
pip install -e .
cd ../../../
```

Listing 55: Commands to install `csle-tolerance` from source.

Finally, **on the leader node only**, insert the simulation configurations into the metastore by running the commands:

```
cd simulation-system/envs
make install
cd ../../
```

Listing 56: Commands to insert simulation configurations into the metastore.

3.1.6 Installing the Emulation System

The emulation system consists of a set of configuration files and a set of Docker images, which are divided into a set of "base images" and a set of "derived images". The base images contain common functionality required by all images in CSLE whereas the derived images add specific configurations to the base images, e.g., specific vulnerabilities. To install the emulation system, the configuration files must be inserted into the metastore and the Docker images must be built or downloaded.

Start with adding Docker's official GPG key to Ubuntu's package manager by running the commands (you can also follow the instructions for installing Docker at: https://docs.docker.com/engine/install/ubuntu/):

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
↪  gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture)
↪  signed-by=/etc/apt/keyrings/docker.gpg]
↪  https://download.docker.com/linux/ubuntu $(lsb_release -cs)
↪  stable" | sudo tee /etc/apt/sources.list.d/docker.list >
↪  /dev/null
```

Listing 57: Commands to add Docker's official GPG key to Ubuntu's package manager.

Next, install Docker by running the commands:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
sudo groupadd docker
sudo usermod -aG docker $USER
```

Listing 58: Commands to install Docker.

After running the commands above, start a new shell for the changes to take effect.

Next, setup a docker swarm by running the following command on the leader:

```
docker swarm init --advertise-addr <ip address of the leader>
```

Listing 59: Command to initialize a Docker swarm.

After running the above command, a secret token will be returned. Use this token to run the following command on each worker to add it to the swarm:

```
docker swarm join --token <my_roken> leader_ip:2377
```

Listing 60: Commands to add a worker node to the Docker swarm.

☞ If you forget the swarm token, you can display it by running the following command on the leader: `docker swarm join-token worker`.

On the leader you can verify that the swarm is setup correctly by running:

```
docker node ls
```

Listing 61: Commands to check the nodes in the Docker swarm.

After completing the Docker installation, pull the base images of CSLE from DockerHub[8] by running the commands:

```
cd emulation-system/base_images
make pull
cd ../../
```

Listing 62: Commands to pull CSLE base images from DockerHub.

Alternatively, you can build the base images locally (this takes several hours) by running the commands:

```
cd emulation-system/base_images
make build
cd ../../
```

Listing 63: Commands to build CSLE base images.

Next, pull the derived images of CSLE from DockerHub by running the commands:

```
cd emulation-system/derived_images
make pull
cd ../../
```

Listing 64: Commands to pull CSLE derived images from DockerHub.

Alternatively, you can build the derived images locally by running the commands:

[8] https://hub.docker.com/u/kimham

```
cd emulation-system/derived_images
make build
cd ../../
```

Listing 65: Commands to build CSLE derived images.

Next, insert the emulation configurations into the metastore by running the commands **on the leader node only**:

```
cd emulation-system/envs
make install
cd ../../
```

Listing 66: Commands to insert emulation configurations into the metastore.

Alternatively, you can install the base images, the derived images, and the emulation configurations all at once by running the commands:

```
cd emulation-system
make build
cd ../
```

Listing 67: Commands to install base images, derived images, and emulation environments.

A few configuration parameters of the kernel need to be updated to be able to execute emulations. In particular, the configuration variables `max_map_count`, `max_user_watches`, and the `ulimit` need to be updated.

Update `max_map_count` by editing the file `/etc/sysctl.conf` and add the following line:

```
vm.max_map_count=262144
```

Listing 68: Line to add to `/etc/sysctl.conf`.

Alternatively, for a non-persistent configuration, run the command:

```
sysctl -w vm.max_map_count=262144
```

Listing 69: Command to update the configuration variable `max_map_count`.

You can check the configuration by running the command:

```
sysctl vm.max_map_count
```

Listing 70: Command to check the configuration of "max_map_count".

Next, update the ulimit on the number of open filedescriptors by editing the file /etc/security/limits.conf and add the following lines:

```
<myuser>              soft       nofile          102400
<myuser>              hard       nofile          1024000
```

Listing 71: Lines to add to /etc/security/limits.conf.

You can check the configuration by running the command:

```
ulimit -aH
```

Listing 72: Command to check the ulimit configuration.

Finally, update max_user_watches by running the command:

```
echo fs.inotify.max_user_watches=524288 | sudo tee -a
↪  /etc/sysctl.conf && sudo sysctl -p
```

Listing 73: Command to set the configuration variable max_user_watches.

3.1.7 Installing the Management System

The management system consists of monitoring systems (i.e., Grafana, Prometheus, Node exporter, cAdvisor, and pgadmin) and the web application that implements the web interface, which is based on node.js[9]. Hence, installing the management system corresponds to installing these services and applications.

Start by installing node.js, its version manager nvm, and its package manager npm on all servers (the leader and the workers) by running the commands:

[9] https://nodejs.org/en/

```
curl
↪  https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.2/install.sh
↪ --output nvm.sh
chmod u+rwx nvm.sh
```

```
./nvm.sh
```

Listing 74: Commands to install `nvm`.

Then setup the `nvm` environment variables by adding the following lines to `.bashrc`:

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

Listing 75: Commands to setup NVM environment variables.

If you are using the fish shell, you can setup `nvm` using the following commands:

```
curl
↪  https://raw.githubusercontent.com/oh-my-fish/oh-my-fish/master/bin/install
↪  | fish # install OMF
omf install https://github.com/fabioantunes/fish-nvm # Install
↪  fish-nvm
omf install bass # Install bass
```

Listing 76: Commands to setup `nvm` with the fish shell.

Then you can install `node.js` and `npm` using the commands:

```
nvm -v # Verify nvm installation
nvm install node # Install node
npm install -g npm # Update npm
node -v # Verify version of node
npm -v # Verify version of npm
```

Listing 77: Commands to install `node.js` and `npm`.

Next install and build the web application of the management system by running the following commands:

```
cd csle/management-system/csle-mgmt-webapp
npm install
npm run build
```

Listing 78: Commands to install the web application of the CSLE management system.

☞ Note: when you run the command `npm install` you may need to add the flag `-legacy-peer-deps`. Further, if you have an old operating system, you may need to run `export NODE_OPTIONS=-openssl-legacy-provider` before running `npm run build`.

Next, install and start `pgadmin` **on the leader only** by running the following commands:

```
docker pull dpage/pgadmin4
docker run -p 7778:80 -e "PGADMIN_DEFAULT_EMAIL=user@domain.com"
↪ -e "PGADMIN_DEFAULT_PASSWORD=SuperSecret" -d dpage/pgadmin4
```

Listing 79: Commands to start `pgadmin`.

Next, configure Nginx on the leader by editing the file:

```
/etc/nginx/sites-available/default
```

Listing 80: Nginx configuration file.

Replace the current configuration with the following:

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html;

        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location /pgadmin {
                proxy_set_header X-Script-Name /pgadmin;
                proxy_set_header Host $host;
                proxy_pass http://localhost:7778/;
                proxy_redirect off;
        }

        location / {
                proxy_pass http://localhost:7777/;
                proxy_buffering off;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-Host $host;
                proxy_set_header X-Forwarded-Port $server_port;
        }
}
```

Listing 81: Content of"`/etc/nginx/sites-available/default`" on the leader.

Restart Nginx on the leader by running the command:

```
sudo service nginx restart
```

Listing 82: Command to restart Nginx.

If you have HTTPS enabled on the REST API and have certificates you can configure them in Nginx on the leader by editing the file

```
/etc/nginx/sites-available/default
```

Listing 83: Nginx configuration file.

as follows:

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;
        server_name _;
        return 301 https://$host$request_uri;

        location /pgadmin {
                proxy_set_header X-Script-Name /pgadmin;
                proxy_set_header Host $host;
                proxy_pass http://localhost:7778/;
                proxy_redirect off;
        }
}


 server {
     listen 443 ssl default_server;
     listen [::]:443 ssl default_server;
     ssl_certificate /var/log/csle/certs/csle.dev.crt;
     ssl_certificate_key /var/log/csle/certs/csle_private.key;
     root /var/www/html;
     index index.html index.htm index.nginx-debian.html;
     server_name csle.dev;
     location / {
     proxy_pass http://localhost:7777/;
     proxy_buffering off;
     proxy_set_header X-Real-IP $remote_addr;
     proxy_set_header X-Forwarded-Host $host;
     proxy_set_header X-Forwarded-Port $server_port;
     }
 }
```

Listing 84: Contents of the file "`/etc/nginx/sites-available/default`" on the leader to allow HTTPS traffic to the web interface.

Next, configure Nginx on the workers by editing the following file

```
/etc/nginx/sites-available/default
```

Listing 85: Nginx configuration file.

Open the file on each worker and replace the current configuration with the following (replace “leader-ip” with the actual ip):

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;
        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;
        server_name _;
        location /pgadmin {
                proxy_set_header X-Script-Name /pgadmin;
                proxy_set_header Host $host;
                proxy_pass http://leader-ip:7778/;
                proxy_redirect off;
        }

        location / {
                proxy_pass http://localhost:7777/;
                proxy_buffering off;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-Host $host;
                proxy_set_header X-Forwarded-Port $server_port;
        }
}
```

Listing 86: Content of “`/etc/nginx/sites-available/default`” on a worker.

Next make the Nginx log files readable by your user by running the commands:

```
sudo chmod -R u+rw /var/log/nginx
sudo chown -R <my_user> /var/log/nginx
```

Listing 87: Commands to make the Nginx log files readable for a given user.

Lastly, restart Nginx on each worker and on the leader by running the command:

```
sudo service nginx restart
```

Listing 88: Command to restart Nginx.

After completing the steps above, install Prometheus and node exporter by running the following commands (these commands should be ran on all nodes, both the leader and the workers):

```
cd management-system
chmod u+x install.sh
./install.sh
```

Listing 89: Commands to install the management system and associated tools.

Next, configure the IP by editing the following file on each server (leader and workers):

```
csle/management-system/csle-mgmt-webapp/src
    /components/Common/serverIp.js
```

Listing 90: File to configure the IPs of servers in the management system.

Next, configure the port of the web interface by editing the following file on each server:

```
  csle/management-system/csle-mgmt-webapp/src
      /components/Common/serverPort.js
```

Listing 91: File to configure the port of the web interface.

To start and stop the monitoring systems using the CSLE CLI, their binaries need to be added to the system path.

Add Prometheus binary to the system path by adding the following line to `.bashrc` on all nodes:

```
export PATH=/path/to/csle/management-system/prometheus/:$PATH
```

Listing 92: Line to add to `.bashrc` to add Prometheus to the path.

If you have fish shell instead of bash, add the following line to the configuration file of fish:

```
fish_add_path /path/to/csle/management-system/prometheus/
```

Listing 93: Line to add to the configuration file of fish to add Prometheus to the path.

Similarly, to add the Node exporter binary to the path, add the following line to `.bashrc` on all nodes:

```
export PATH=/path/to/csle/management-system/node_exporter/:$PATH
```

Listing 94: Line to add to `.bashrc` to add Node exporter to the path.

If you have fish shell instead of bash, add the following line to the configuration file of fish:

```
fish_add_path /path/to/csle/management-system/node_exporter/
```

Listing 95: Line to add to the configuration file of fish shell to add Node exporter to the path.

Finally, start the csle daemons and setup the management user account with administrator privileges by running the following command on all nodes:

```
csle init
```

Listing 96: Command to start the csle daemon and setup the management user account with administrator privileges.

3.2 Quick Start

To verify that CSLE was installed correctly, run the command:

```
csle ls --all
```

Listing 97: Command to list all information about CSLE.

The above command lists information about the CSLE installation, including the emulation configurations that are installed in the metastore. Select one emulation configuration to start an execution, e.g., `csle-level9-003`, and then run the command:

```
csle start csle-level9-003
```

Listing 98: Command to start an execution of the emulation configuration csle-level9-003.

The above command will start all containers of the emulation and apply the configuration. After this process has completed, start the monitoring systems and the management system by executing the commands:

```
csle start grafana
csle start cadvisor
csle start prometheus
csle start nodeexporter
csle start flask
csle start nginx
```

Listing 99: Command to start management services of CSLE.

After the above commands have completed, the web interface of CSLE can be accessed at the URL: `http://localhost:7777/`, from which you can find links to Grafana's interface, Prometheus interface, cAdvisor's interface, and interfaces to storage systems (i.e., Kafka, Presto, and Elasticsearch).

To get started with reinforcement learning in CSLE, run the examples in the folder: `csle/examples`, e.g.,

```
python csle/examples/training/ppo/
       stopping_pomdp_defender/run_vs_random_attacker_v_001.py
```

Listing 100: Command to run the Proximal Policy Optimization (PPO) algorithm for learning defender strategies in CSLE.

3.3 Uninstalling CSLE

To uninstall CSLE, run the commands:

```
csle rm all
csle clean emulations
csle rm emulations
pip uninstall gym-csle-stopping-game
pip uninstall gym-csle-apt-game
pip uninstall gym-csle-cyborg
pip uninstall gym-csle-intrusion-response-game
pip uninstall csle-tolerance
pip uninstall csle-common
pip uninstall csle-collector
pip uninstall csle-agents
pip uninstall csle-ryu
pip uninstall csle-rest
pip uninstall csle-system-identification
pip uninstall csle-attacker
pip uninstall csle-base
pip uninstall csle-cluster
pip uninstall csle-cli
cd emulation-system && make rm
cd metastore; make clean
```

Listing 101: Commands to uninstall CSLE.

Also remove the `CSLE_HOME` environment variable from `.bashrc`.

3.4 Operating CSLE

This section describes commands and procedures that are useful when operating CSLE. The framework can be operated in two ways, either trough the web interface or through the CLI (see Fig. 28). This section focuses on the

CLI, but more or less the same commands can be invoked through the web interface, which should be self-explanatory.

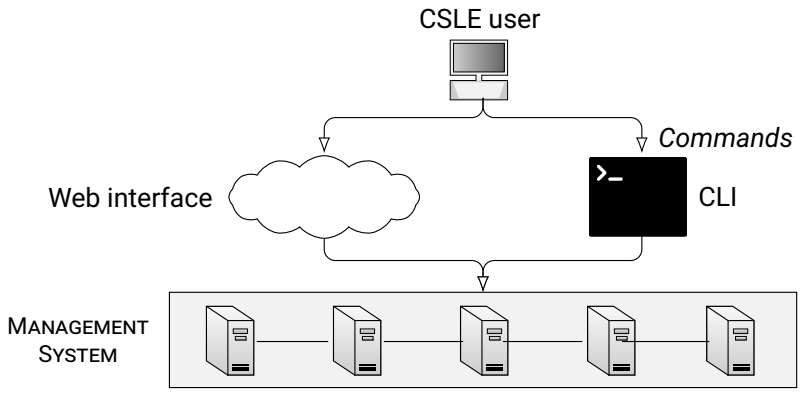

Figure 28: User interfaces of CSLE; a user can execute commands to the management system through two interfaces: a web interface and a Command-Line Interface (CLI).

3.4.1 Listing the Available Commands in the CLI

To list the available commands in the CSLE CLI, run the command:

```
csle help
```

Listing 102: Command to list the available commands in the CSLE CLI.

3.4.2 Listing the State of CSLE

Information about the CSLE installation and its current state can be listed by executing the following command:

```
csle ls --all
```

Listing 103: Command to list the state of a CSLE installation.

3.4.3 Starting, Stopping, and Resetting the Metastore

The metastore can be started by executing the following command:

```
sudo service postgresql start
```

Listing 104: Command to start the metastore.

The metastore can be stopped by executing the command:

```
sudo service postgresql stop
```

Listing 105: Command to stop the metastore.

To reset the metastore, execute the commands:

```
cd metastore; make clean
cd metastore; make build
```

Listing 106: Commands to reset the metastore.

3.4.4 Starting and Stopping the REST API Server

The REST API server can be started by executing the command:

```
csle start flask
```

Listing 107: Command to start the REST API server.

The REST API server can be stopped by executing the command:

```
csle stop flask
```

Listing 108: Command to stop the REST API server.

3.4.5 Starting and Stopping Monitoring Systems

The monitoring systems Grafana, cAdvisor, Node exporter, and Prometheus can be started by executing the commands:

```
csle start grafana
csle start cadvisor
csle start nodeexporter
csle start prometheus
```

Listing 109: Commands to start monitoring systems.

Similarly, Grafana, cAdvisor, Node exporter, and Prometheus can be stopped by executing the commands:

```
csle stop grafana
csle stop cadvisor
csle stop nodeexporter
csle stop prometheus
```

Listing 110: Commands to stop monitoring systems.

3.4.6 Starting and Stopping Emulation Executions

To start an execution of an emulation configuration with the name `csle-level9-003`, execute the following command:

```
csle start csle-level9-003
```

Listing 111: Command to start an execution of the emulation with configuration `csle-level9-003`.

Similarly, to stop an execution of an emulation configuration with the name `csle-level9-003`, execute the following command:

```
csle stop csle-level9-003
```

Listing 112: Command to stop an execution of the emulation with configuration `csle-level9-003`.

The above command will stop all containers but will not remove them, which means that the emulation can be started again with the same configuration by running the command in Listing 111.

To stop an emulation execution and remove all of its containers and virtual networks, run the command:

```
csle clean csle-level9-003
```

Listing 113: Command to stop and clean an execution of the emulation with configuration `csle-level9-003`.

3.4.7 Access a Terminal in an Emulated Container

To see which containers are running, execute the command:

```
csle ls --all
```

Listing 114: Command to list running CSLE containers.

To open a terminal in a running container, e.g., a container with the name `mycontainer`, run the command:

```
csle shell mycontainer
```

Listing 115: Command to open a terminal in an emulated container.

3.5 Examples

Examples of CSLE usages can be found at: https://github.com/Limmen/csle/examples. This folder includes examples of the four main usages of CSLE: (1) data collection; (2) system identification; (3) strategy training; and (4), strategy evaluation.

3.5.1 Data collection

A common usage of CSLE is to run emulated cyber attacks against an emulated IT infrastructure and record the resulting traces of system metrics and logs. To do this with CSLE, you have to a) choose a running emulation execution; b) define the sequence of attacker actions; c) define the sequence of defender actions; and d), start the attacker and defender sequences. A code example of performing these steps using the CSLE APIs is given below.

```
import csle_common.constants.constants as constants
from ..emulation_attacker_action import EmulationAttackerAction
from ..emulation_defender_action import EmulationDefenderAction
from ..emulation_attacker_stopping_actions import
↪  EmulationAttackerStoppingActions
from ..emulation_defender_stopping_actions import
↪  EmulationDefenderStoppingActions
from ..emulation_env_config import EmulationEnvConfig
from ..metastore_facade import MetastoreFacade
from ..container_controller import ContainerController
from csle_system_identification.emulator import Emulator

# Select emulation execution
execution = get_emulation_execution(ip_first_octet=..,
↪  emulation_name=..)
emulation_env_config = executions.emulation_env_config
attacker_sequence = [..] # Define attacker sequence
defender_sequence = [..] # Define defender sequence
# Starting the attacker and defender
Emulator.run_action_sequences(
     emulation_env_config=emulation_env_config,
     attacker_sequence=attacker_sequence,
     defender_sequence=defender_sequence)

# Extract recorded traces and statistics
statistics = MetastoreFacade.list_emulation_statistics()
traces = MetastoreFacade.list_emulation_traces()
```

Listing 116: Example of collecting attack and system traces with CSLE.

3.5.2 System Identification

A key step in CSLE's reinforcement learning method is *system identification* (see Fig. 1). In this step, data collected from emulated IT infrastructures are used to identify system parameters and models. A code example of system identification with CSLE is given below.

```
import csle_common.constants.constants as constants
from csle_common.dao.system_identification.
     system_identification_config
     import SystemIdentificationConfig
from csle_common.metastore.metastore_facade import
↪  MetastoreFacade
from csle_common.dao.system_identification.system_model_type
↪  import SystemModelType
from csle_common.dao.training.hparam import HParam
from csle_system_identification.expectation_maximization.
     expectation_maximization_algorithm \
     import ExpectationMaximizationAlgorithm
     import csle_system_identification.constants.constants as
      ↪  sid_consts
# Select emulation config
emulation_env_config =
↪  MetastoreFacade.get_emulation_by_name("csle-level9-002")
# Extract statistics from the metastore
emulation_statistic =
↪  MetastoreFacade.get_emulation_statistic(id=1)
system_identification_config = SystemIdentificationConfig(
output_dir=f"{constants.LOGGING.DEFAULT_LOG_DIR}em_level9_test",
title="Expectation-Maximization level 9 test",
model_type=SystemModelType.GAUSSIAN_MIXTURE,
log_every=1,
hparams={..})
algorithm = ExpectationMaximizationAlgorithm(
emulation_env_config=emulation_env_config,
emulation_statistics=emulation_statistic,
system_identification_config=system_identifcation_config)
system_model = algorithm.fit() # Run the algorithm
MetastoreFacade.save_gaussian_mixture_system_model(
gaussian_mixture_system_model=system_model) # Save system model
↪  in metastore
```

Listing 117: Example of system identification through expectation maximization with CSLE.

3.5.3 Strategy Training

CSLE includes several numerical algorithms for optimizing defender strategies, e.g., reinforcement learning algorithms, dynamic programming algorithms, game-theoretic algorithms, and general optimization algorithms. A code example of learning security strategies through the Proximal Policy Optimization (PPO) reinforcement learning algorithm is given below.

```
import csle_common.constants.constants as constants
from csle_common.dao.training.experiment_config import
↪  ExperimentConfig
from csle_common.metastore.metastore_facade import
↪  MetastoreFacade
from csle_common.dao.training.agent_type import AgentType
from csle_common.dao.training.hparam import HParam
from csle_common.dao.training.player_type import PlayerType
from csle_agents.agents.ppo.ppo_agent import PPOAgent
import csle_agents.constants.constants as agents_constants

emulation_env_config =
↪  MetastoreFacade.get_emulation_by_name("..")
# Select simulation environment
simulation_env_config =
↪  MetastoreFacade.get_simulation_by_name("..")
# Setup experiment with hyperparameters
experiment_config = ExperimentConfig(output_dir="..", title="PPO
↪  test", random_seeds=[..], agent_type=AgentType.PPO,
↪  log_every=1, hparams={..}, player_type=PlayerType.DEFENDER,
↪  player_idx=0)
agent = PPOAgent(emulation_env_config=emulation_env_config,
↪  simulation_env_config=simulation_env_config,
↪  experiment_config=experiment_config)
# Run the algorithm
experiment_execution = agent.train()
# Save the results and the learned policies
MetastoreFacade.save_experiment_execution(experiment_execution)
for policy in experiment_execution.result.policies.values():
    MetastoreFacade.save_ppo_policy(ppo_policy=policy)
```

Listing 118: Example of strategy optimization through the Proximal Policy Optimization (PPO) reinforcement learning algorithm in CSLE.

3.5.4 Strategy Evaluation

CSLE can be used to evaluate learned security strategies in emulated infrastructures running on the emulation system (see Fig. 1). Below is a code example of strategy evaluation with CSLE.

```
import gym
import csle_common.constants.constants as constants
from csle_common.metastore.metastore_facade import
↪  MetastoreFacade
from csle_common.dao.training.multi_threshold_stopping_policy
↪  import MultiThresholdStoppingPolicy
from
↪  gym_csle_stopping_game.envs.stopping_game_pomdp_defender_env
↪  import StoppingGamePomdpDefenderEnv
from csle_common.dao.training.player_type import PlayerType
from csle_common.dao.training.agent_type import AgentType
# Select emulation to be used as evaluation environment
emulation_env_config =
↪  MetastoreFacade.get_emulation_by_name("..")
# Select simulation environment
simulation_env_config =
↪  MetastoreFacade.get_simulation_by_name("..")
config = simulation_env_config.simulation_env_input_config
env = gym.make(simulation_env_config.gym_env_name,
↪  config=config)
# Define security policy to evaluate
tspsa_policy = MultiThresholdStoppingPolicy(..)
# Perform the evaluation
StoppingGamePomdpDefenderEnv.emulation_evaluation(env=env,
↪  n_episodes=10, intrusion_seq=[..],
↪  defender_policy=tspsa_policy,
↪  emulation_env_config=emulation_env_config,
↪  simulation_env_config=simulation_env_config)
```

Listing 119: Example of evaluating learned security strategies in the emulation system of CSLE.

4 Developer Guide

This section contains guidelines for developers. It includes development conventions (Section 4.1), instructions on how to install development tools (Section 4.2), guidelines for writing documentation (Section 4.3), guidelines for debugging (Section 4.4), instructions on how to run tests (Section 4.5), instructions on how to perform static code analysis (Section 4.6), guidelines for dependency management (Section 4.7), and guidelines for release management (Section 4.8).

4.1 Development Conventions

Being a project of certain size and complexity, it is important that the general development conventions of CSLE are followed. By adopting good practices it will be easier to deal with the growth of this project.

Please note that if you decide to contribute to CSLE you automatically accept the licensing conditions of this project (CC BY-SA 4.0 license).

4.1.1 General Principles

CSLE tries to follow a few high-level principles in making both technical and community decisions, which are listed below. They are goals to shoot for, and may not be followed perfectly all the time.

- **Code over configuration.** We aim to make CSLE fully programmable, everything from starting/stopping emulations to configuring a service running on a container should be possible through a program function. To achieve this level of programmability, as much as possible of CSLE should be defined in code rather than configuration files. If a configuration file is necessary, it should be defined in a serialization format that easily can be converted back to code, e.g., a JSON file that maps to a Python class. Another benefit of defining configuration in code is that we can run the configuration through the code quality toolchain to identify bugs and style errors.
- **Separation of concerns.** CSLE is divided into components (systems), e.g., the management system, the emulation system, and the simulation system. These components interact via APIs and via a shared database (the metastore). As much as possible, each individual component should be independent of the other components. This principle allows users to install individual components without having to install the other components.

- **Release early and often.** We should emphasize smaller, more iterative releases over large and complex ones. This keeps our documentation in-line with the latest releases and also minimizes the disruption (and subsequent maintenance burden) associated with big changes. The process for creating a release is relatively simple and quick, so don't hesitate to release patch versions (or minor versions) as appropriate.

4.1.2 Structure of CSLE

CSLE is made up of numerous libraries and artifacts in different programming languages (mainly Python, JavaScript, Bash, and Dockerfiles). When you initially consider contributing to CSLE, you might be unsure about which of those libraries implement the functionality you want to change or report a bug for. The list below, which explains the main components of CSLE and their location in the code repository, should help you with that.

- `examples/`. Examples of using CSLE.
- `emulation-system/base_images` and `emulation-system/derived_images`. Dockerfiles that define the container images of the emulation system.
- `emulation-system/envs`. Emulation configurations.
- `metastore`. SQL files that define the data model of the metastore.
- `management-system/csle-mgmt-webapp`. Web application that implements the web interface of CSLE (see Section 2.6).
- `simulation-system/envs`. Simulation configurations.
- `simulation-system/libs/csle-base`. A Python library with base classes and definitions used by the other python libraries in CSLE.
- `simulation-system/libs/csle-agents`. A Python library with implementations of control, learning, and game-theoretic algorithms for finding defender strategies.
- `simulation-system/libs/csle-attacker`. A Python library that contains code for emulating attacker actions (see Section 2.4.6).
- `simulation-system/libs/csle-cli`. A Python library with the CSLE CLI (see Section 2.7).
- `simulation-system/libs/csle-collector`. A Python library that implements the monitoring and management agents (see Section 2.4.9 and Section 2.4.10).

- `simulation-system/libs/csle-common`. A Python library that contains common functionality to all CSLE Python libraries.
- `simulation-system/libs/csle-defender`. A Python library that contains code for emulating defender actions (see Section 2.4.7).
- `simulation-system/libs/csle-rest-api`. A Python library with the CSLE REST API (see Section 2.6.3).
- `simulation-system/libs/csle-ryu`. A Python library with RYU SDN controllers.
- `simulation-system/libs/csle-cluster`. A Python library with a gRPC server for cluster management in CSLE.
- `simulation-system/libs/csle-system-identification`. A Python library with implementations of system identification algorithms to learn system models based on measured data and traces.
- `simulation-system/libs/gym-csle-stopping-game`. A gym environment for the optimal stopping game in [21].
- `simulation-system/libs/gym-csle-apt-game`. A gym environment for an APT stopping game.
- `simulation-system/libs/gym-csle-cyborg`. A gym environment wrapper for CybORG.
- `simulation-system/libs/gym-csle-intrusion-response-game`. A gym environment for the intrusion response game in [22].
- `simulation-system/libs/csle-tolerance`. An intrusion-tolerant system: Tolerance: (T)w(o)-(l)ev(e)l (r)ecovery (a)nd respo(n)se (c)ontrol with f(e)edback. (to be published).

4.1.3 Code Readability

The code style in CSLE is based on the PEP 8 standard [45] and defined using the `flake8` Python linter and the `eslint` JavaScript linter. You can configure your IDE or editor to automatically enforce these guidelines.

The configuration file of the Python linter is located at:

```
1 csle/.flake8
```

Listing 120: Configuration file for the `flake8` Python linter.

The configuration file of the JavaScript linter is located at:

```
csle/management-system/csle-mgmt-webapp/.eslintrc.json
```

Listing 121: Configuration file for the `eslint` JavaScript linter.

Names of variable, functions, methods etc. should be clear and descriptive, not cryptic. All Python functions and variables should be written in `snake_case`, e.g., `stop_all_executions()`. All JavaScript functions and variables should be in `CamelCase`, e.g., `getAgentType()`.

It is common practice to name simple loop variables i, j, and k, so there's no need to give them silly names like `the_index` unless it's necessary for some reason or other.

Avoid long lines (>120 characters) if possible. This principle makes pull requests smaller, makes the code more readable, and benefits co-developers editing code in Emacs/Vim over SSH and/or in narrower windows.

It is highly recommended that you configure your editor or Integrated Development Environment (IDE) to automatically enforce the style guidelines.

4.1.4 Comments in Code

All functions and classes should have comments that document the input/output arguments, the purpose of the function/class, and its return value. These comments are used to automatically generate API documentation (this process is described in Section 4.3).

Example of a comment to a Python function:

```
@staticmethod
def stop_all_executions() -> None:
    """
    Stops all emulation executions

    :return: None
    """
    executions = MetastoreFacade.list_emulation_executions()
    for exec in executions:
        EmulationEnvController.stop_containers(execution=exec)
        ContainerController.stop_docker_stats_thread(
                            execution=exec)
```

Listing 122: Example of a Python function with a comment.

Example of a comment to a JavaScript function:

```
const convertListToCommaSeparatedString = (listToConvert) => {
    /**
     * Converts a list of strings into a single comma-separated
     * string
     *
     * @param listToConvert the list to convert
     * @returns {string} the comma separated string
     */
    var str = ""
    for (let i = 0; i < listToConvert.length; i++) {
        if (i !== listToConvert.length-1) {
            str = str + listToConvert[i] + ", "
        } else {
            str = str + listToConvert[i]
        }
    }
    return str
}
```

Listing 123: Example of a JavaScript function with a comment.

4.1.5 Unit and Integration Testing

Every new change to the code must pass the tests (tests are performed automatically on each pull request). Whenever a new feature is added to CSLE, a corresponding test should be added (see Section 4.5 for instructions on how to add tests and see Section 4.1.7 for the pipeline that automates the tests).

4.1.6 How We Use Git

We use the Git-Flow branching model, in which branches are categorized into:

- A master branch.
- A main development branch.
- One or several feature branches.
- One or several hotfix branches.

The code on the main branch (often called `main` or `release`) is stable, properly tested and is the version of the code that a typical user should pick. No changes are made directly on the master branch.

The code on the development branch (often called `develop`) should be working, but without guarantees. For small features, development might well happen directly on the development branch and the code here may therefore sometimes be broken (this should ideally never happen though). When the development branch is deemed "done" and has undergone testing and review, it is merged into the master branch. The release is then tagged with an appropriate release version.

A feature branch (often called `feature/some_name` where `some_name` is a very short descriptive name of the feature) is branched off from the main development branch when a new "feature" is being implemented. A new feature is any logically connected set of changes to the code base regardless of how many files are being changed.

A hotfix branch (often called `hotfix/some_name`) is a branch that implements a bugfix to a release. In terms of branching, it is thus very similar to a feature branch but for the master branch rather than for the development branch. A hotfix should fix critical errors that were not caught in testing before the release was made. Hotfixes should not implement new behavior, unless this is needed to fix a critical bug. Hotfixes need to undergo review before they are merged back into the master and development branches.

The master and development branches are never deleted, while the others are transient (temporary, for the duration of the development and code review).

The benefits of this type of branching model in development are:

- Co-developers work on separate branches, and do not "step on each other's toes" during the development process, even if they push their work back to GitHub.
- Co-developers and users have a stable master branch to use.
- Features in "feature" branches are independent of each other. Any conflicts are resolved when merging.

Commit often, possibly several times a day. It's easier to roll back a small commit than to roll back large commits. This practice also makes the code easier to review. Remember to push the commits to GitHub every once in a while too. Write a helpful commit message with each commit that describes what the changes are and possibly even why they were necessary.

4.1.7 Continuous Integration

We use continuous integration (CI) with GitHub Actions[10] to build the project and run tests on every pull request submitted to CSLE. The CI pipeline used in CSLE is illustrated in Fig. 29. Developers make commits on a branch that is separated from the master/main branch. Once a developer has completed a bugfix or a new feature, he/she submits a pull request to GitHub. The pull request then triggers a set of automated tests and automated builds using GitHub actions. If all automated tests pass, the pull request undergoes code review, otherwise the developer has to fix the failing tests or builds. Finally, when the code review and associated fixes are completed, the pull request is merged into the main/master branch and may optionally trigger a release pipeline where build artifacts are pushed to code servers (i.e., DockerHub and PyPi).

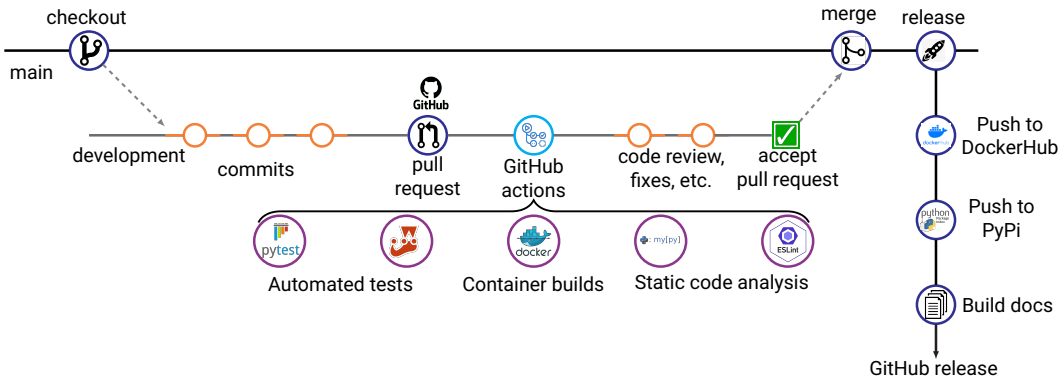

Figure 29: The continuous integration (CI) pipeline of CSLE.

4.1.8 Bug Reports

Please be aware of the following things when filing bug reports:

1. Avoid raising duplicate issues. Use the GitHub issue search feature to check whether your bug report or feature request has been mentioned in the past. Duplicate bug reports and feature requests are a huge maintenance burden on the limited resources of CSLE. If it is clear from your report that you would have struggled to find the original issue, that's ok, but if searching for a selection of words in your issue title would have found the original issue, then the duplicate issue will likely be closed abruptly.

[10] https://github.com/features/actions

2. When filing bug reports about exceptions or tracebacks, please include the complete traceback. Partial tracebacks, or just the exception text, are not helpful. Issues that do not contain complete tracebacks may be closed without warning.

3. Make sure you provide a suitable amount of information to work with. This means that you should provide:

    - Guidance on how to reproduce the issue. Ideally, this should be a small code sample that can be run immediately by the maintainers. Failing that, let us know what you're doing, how often it happens, what environment you're using, etc. Be thorough: it prevents us needing to ask further questions.
    - Tell us what you expected to happen. When we run your example code, what are we expecting to happen? What does "success" look like for your code?
    - Tell us what actually happens. It's not helpful for you to say "it doesn't work" or "it fails". Tell us how it fails: do you get an exception? A hang? A non-200 status code? How was the actual result different from your expected result?

    If you do not provide all of these things, it will take us much longer to fix your problem. If we ask you to clarify these things and you never respond, we will close your issue without fixing it.

4.1.9 Contribution flow

Our contribution flow is very straightforward and follows an issue-pull request workflow. Contributors need to fork the repository for having their contributions landed on the project. If you are looking to contribute, we have laid down a series of steps that we would like you to follow.

1. **Find an issue to work on.** The first step is to go through the issues to find one that you would like to contribute to (or possibly open a new issue). Exploring the issues and pull requests will give you an idea of how the contribution flow works. Upon finding something to work on, you can either request for the issue to be assigned to you (if someone else has created the Issue) or you can make your own. To ensure that the issue is received positively by the maintainers, make sure of the following:

    - If you are filing a bug report, make sure that the report follows the guidelines stated in Section 4.1.8.

- If you are filing a feature request, make sure that you pitch in your idea well and in a constructive manner.

2. **Getting your pull request merged.** To make sure you land a great contribution, we would request you to follow the standard Git & GitHub workflow for code collaboration. We would recommend:
   - Every pull request should have the corresponding issue linked to it.
   - Every pull request should pass the automated CI checks.
   - Every pull request should be as atomic as possible.
   - Every pull request should include a test verifying the new/fixed behavior.
3. **Reviewing pull requests.** After landing your pull request, the maintainers will review it and provide actionable feedback. Maintainers expect the comments to be resolved once a review has been completed. You can provide updates if you are still working on it to help us understand the areas where we can help.

4.2 Installing development tools

This section contains instructions on how to install development tools that CSLE uses, such as test libraries and static code analyzers.

To install all Python build tools at once, run the following commands:

```
cd csle/simulation-system/libs/csle-base; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-collector; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-ryu; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-common; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-attacker; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-defender; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-system-identification; pip
↪  install -r requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/gym-csle-stopping-game; pip
↪  install -r requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/gym-csle-apt-game; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/gym-csle-cyborg; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-agents; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-rest-api; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-cli; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-cluster; pip install -r
↪  requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/gym-csle-intrusion-response-game;
↪  pip install -r requirements_dev.txt; cd ../../../../
cd csle/simulation-system/libs/csle-tolerance; pip install -r
↪  requirements_dev.txt; cd ../../../../
```

Listing 124: Commands to install the development dependencies.

It is also possible to install all development tools at once by running the script `simulation-system/libs/local_install_dev.sh`.

Alternatively, install each development library separately by following the commands below.

The Python build tool, which is used to package Python libraries, can be installed by running the command:

```
pip install -q build
```

Listing 125: Command to install the Python build tool.

The `twine` tool (a tool for publishing Python packages to PyPi) can be installed by running the command:

```
python3 -m pip install --upgrade twine
```

Listing 126: Command to install `twine`.

The `flake8` Python linter can be installed by running the command:

```
python -m pip install flake8
```

Listing 127: Command to install `flake8`.

The `flake8-rst-docstrings` extension to the `flake8` Python linter can be installed by running the command:

```
python -m pip install flake8-rst-docstrings
```

Listing 128: Command to install `flake8-rst-docstrings`.

The `mypy` static type checker for Python can be installed by running the command:

```
python3 -m pip install -U mypy mypy-extensions mypy-protobuf
↪  types-PyYAML types-protobuf types-paramiko types-requests
↪  types-urllib3
```

Listing 129: Command to install `mypy` and the associated type libraries.

The `pytest` and associated test libraries for Python can be installed by running the command:

```
pip install -U pytest mock pytest-mock pytest-cov pytest-grpc
```

Listing 130: Command to install `pytest` and `mock`.

Ruby and its bundler, which are used to generate the CSLE documentation page (`https://limmen.dev/csle/`), can be installed by running the commands:

```
sudo apt-get install ruby ruby-dev
sudo gem install bundler
```

Listing 131: Commands to install Ruby and its bundler.

The `sphinx` Python library for automatic generation of API documentation can be installed by running the commands:

```
pip install sphinx sphinxcontrib-napoleon sphinx-rtd-theme
```

Listing 132: Commands to install `sphinx`.

Lastly, the `tox` Python library for automatic testing can be installed by running the command:

```
pip install tox
```

Listing 133: Command to install `tox`.

4.3 Writing Documentation

There are three kinds of documentation in CSLE:

- Release documentation (the document you are reading now). This document is generated for each major release only. It is a PDF generated through LaTeX.
- Live web documentation at `https://limmen.dev/csle/`. This documentation is generated with Ruby. The source files are located at `csle/docs`. This documentation is updated automatically on every code commit to the project. To generate the web documentation manually, run the command:

```
bundle exec Jekyll serve
```

Listing 134: Command to generate the web documentation at `https://limmen.dev/csle/`.

- Python API documentation. This documentation is generated automatically from code comments using `sphinx`. To generate the API documentation, run the command:

```
simulation-system/libs/generate_docs.sh
```

Listing 135: Command to generate the Python API documentation.

4.4 Log files and Debugging

Log files of physical servers in the management system are located at `/tmp/csle` and PID files are located at `/var/log/csle`. Log files of emulated devices are located at `/`.

To debug management services, check the log files and locate the error. Then try to fix the error and restart the services using the CLI. To debug emulated components, use SSH or `docker exec`[11] to open a terminal inside the component and check the log files.

If you cannot resolve the issue, we recommend restarting all services and see if you can reproduce the issue. If you can reproduce the issue and you believe it is a bug, please file a bug report following the instructions listed in Section 4.1.8.

[11] `https://docs.docker.com/engine/reference/commandline/exec/`

4.5 Tests

CSLE uses `pytest`[12] for Python tests and integration tests, and `jest` for JavaScript tests[13].

The Python unit tests are available at:

- `csle/simulation-system/libs/csle-base/tests`
- `csle/simulation-system/libs/csle-agents/tests`
- `csle/simulation-system/libs/csle-attacker/tests`
- `csle/simulation-system/libs/csle-collector/tests`
- `csle/simulation-system/libs/csle-common/tests`
- `csle/simulation-system/libs/csle-defender/tests`
- `csle/simulation-system/libs/csle-rest-api/tests`
- `csle/simulation-system/libs/csle-ryu/tests`
- `csle/simulation-system/libs/csle-system-identification/tests`
- `csle/simulation-system/libs/gym-csle-stopping-game/tests`
- `csle/simulation-system/libs/gym-csle-apt-game/tests`
- `csle/simulation-system/libs/gym-csle-cyborg/tests`
- `csle/simulation-system/libs/csle-cluster/tests`
- `csle/simulation-system/libs/gym-csle-intrusion-response-game/tests`
- `csle/simulation-system/libs/csle-tolerance/tests`

[12] https://docs.pytest.org/en/7.2.x/
[13] https://jestjs.io/

To run the Python unit tests, execute the command:

```
simulation-system/libs/unit_tests.sh
```

Listing 136: Command to run the Python unit tests.

When adding new Python unit tests note that:

- All unit tests must be written in a `tests/` directory inside the Python project.
- File names should strictly start with "`tests_`".
- Function names should strictly start with "`test`".

The JavaScript unit tests are available at:

```
csle/management-system/csle-mgmt-webapp/src/
```

Listing 137: Directory with the JavaScript unit tests.

To run the JavaScript unit tests, execute the command:

```
cd management-system/csle-mgmt-webapp; npm test
```

Listing 138: Command to run the JavaScript unit tests.

To run the CSLE integration tests, execute the command:

```
simulation-system/libs/integration_tests.sh
```

Listing 139: Command to run the CSLE integration tests.

4.6 Static Code Analysis

Static code analyzers are used in CSLE to enforce style guidelines and discover bugs. The `flake8` linter (with the `flake8-rst-docstrings` extension) and the `mypy` type checker are used to analyze the Python code and the `eslint` linter is used to analyze the JavaScript code.

To run the Python linter on the CSLE code base, execute the command:

```
./python_linter.sh
```

Listing 140: Command to run the Python linter.

Alternatively, the following commands can be executed for the same effect as the command above:

```
flake8 simulation-system/
flake8 emulation-system/envs
flake8 examples/
```

Listing 141: Commands to run the Python linter.

To run the Python static type checker, execute the command:

```
simulation-system/libs/type_checker.sh
```

Listing 142: Command to run the Python static type checker.

To run the JavaScript linter, run the commands:

```
cd management-system/csle-mgmt-webapp/; npm run lint
cd management-system/csle-mgmt-webapp/; npm run lint:fix
```

Listing 143: Commands to run the JavaScript linter.

4.7 Dependency Management

Python dependencies in CSLE are managed with PyPi [14] and JavaScript dependencies are managed with npm [15].

The Python dependencies are defined in the following files:

- `csle/simulation-system/libs/csle-base/requirements.txt`
- `csle/simulation-system/libs/csle-base/requirements_dev.txt`
- `csle/simulation-system/libs/csle-base/setup.cfg`

[14] https://pypi.org/
[15] https://www.npmjs.com/

- csle/simulation-system/libs/csle-agents/requirements.txt
- csle/simulation-system/libs/csle-agents/requirements_dev.txt
- csle/simulation-system/libs/csle-agents/setup.cfg
- csle/simulation-system/libs/csle-attacker/requirements.txt
- csle/simulation-system/libs/csle-attacker/requirements_dev.txt
- csle/simulation-system/libs/csle-attacker/setup.cfg
- csle/simulation-system/libs/csle-cli/requirements.txt
- csle/simulation-system/libs/csle-cli/requirements_dev.txt
- csle/simulation-system/libs/csle-cli/setup.cfg
- csle/simulation-system/libs/csle-cluster/requirements.txt
- csle/simulation-system/libs/csle-cluster/requirements_dev.txt
- csle/simulation-system/libs/csle-cluster/setup.cfg
- csle/simulation-system/libs/gym-csle-intrusion-response-game/requirements.tx
- csle/simulation-system/libs/gym-csle-intrusion-response-game/requirements_de
- csle/simulation-system/libs/gym-csle-intrusion-response-game/setup.cfg
- csle/simulation-system/libs/csle-collector/requirements.txt
- csle/simulation-system/libs/csle-collector/requirements_dev.txt
- csle/simulation-system/libs/csle-collector/setup.cfg
- csle/simulation-system/libs/csle-common/requirements.txt
- csle/simulation-system/libs/csle-common/requirements_dev.txt
- csle/simulation-system/libs/csle-common/setup.cfg
- csle/simulation-system/libs/csle-defender/requirements.txt
- csle/simulation-system/libs/csle-defender/requirements_dev.txt
- csle/simulation-system/libs/csle-defender/setup.cfg
- csle/simulation-system/libs/csle-rest-api/requirements.txt

- csle/simulation-system/libs/csle-rest-api/requirements_dev.txt
- csle/simulation-system/libs/csle-rest-api/setup.cfg
- csle/simulation-system/libs/csle-ryu/requirements.txt
- csle/simulation-system/libs/csle-ryu/requirements_dev.txt
- csle/simulation-system/libs/csle-ryu/setup.cfg
- csle/simulation-system/libs/csle-system-identification/requirements.txt
- csle/simulation-system/libs/csle-system-identification/requirements_dev.txt
- csle/simulation-system/libs/csle-system-identification/setup.cfg
- csle/simulation-system/libs/gym-csle-stopping-game/requirements.txt
- csle/simulation-system/libs/gym-csle-stopping-game/requirements_dev.txt
- csle/simulation-system/libs/gym-csle-stopping-game/setup.cfg
- csle/simulation-system/libs/csle-tolerance/requirements.txt
- csle/simulation-system/libs/csle-tolerance/requirements_dev.txt
- csle/simulation-system/libs/csle-tolerance/setup.cfg
- csle/simulation-system/libs/gym-csle-apt-game/requirements.txt
- csle/simulation-system/libs/gym-csle-apt-game/requirements_dev.txt
- csle/simulation-system/libs/gym-csle-apt-game/setup.cfg
- csle/simulation-system/libs/gym-csle-cyborg/requirements.txt
- csle/simulation-system/libs/gym-csle-cyborg/requirements_dev.txt
- csle/simulation-system/libs/gym-csle-cyborg/setup.cfg

These files need to be updated whenever a Python dependency is added or removed. The dependency structure among the CSLE Python libraries is shown in Fig. 30.

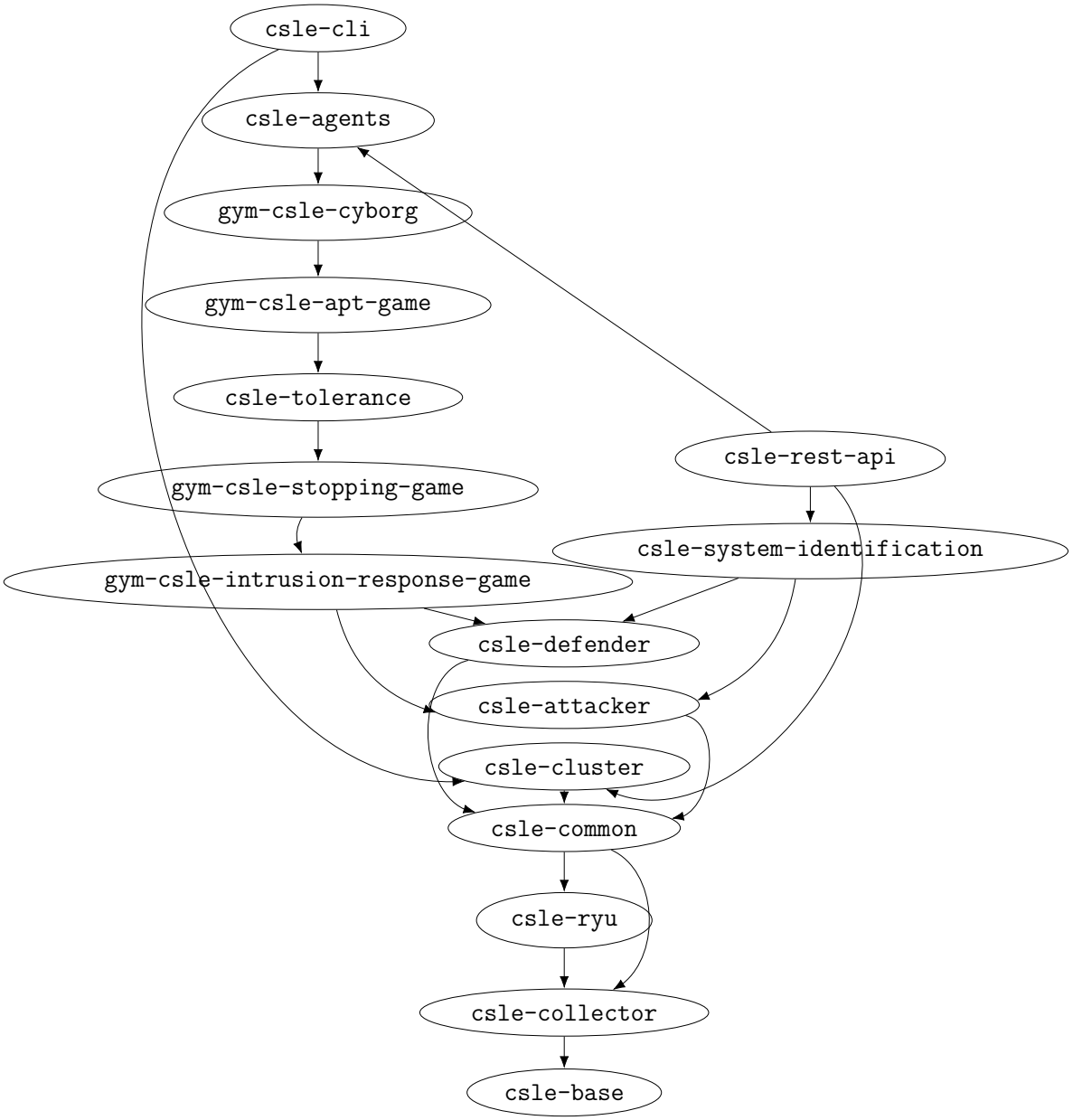

Figure 30: Dependency graph showing the dependencies among the CSLE Python libraries; an arrow from X to Y indicates that X depends on Y; dependency arrows are transitive.

JavaScript dependencies are defined in the file:

`csle/management-system/csle-mgmt-webapp/package.json`

This file should be updated whenever a JavaScript dependency is added or removed.

4.8 Release Management

CSLE has semantic versioning, i.e., versions of CSLE are of the form `MAJOR.MINOR.PATCH`.

- The `MAJOR` version is incremented when incompatible API changes are made.
- The `MINOR` version is incremented new functionality is added in a backwards compatible manner.
- The `PATCH` version is incremented when backwards compatible bug fixes are made.

To generate a new release of CSLE, the following steps must be performed:

1. Publish new versions of the Python libraries to: `https://pypi.org/`.
2. Publish new versions of the Docker containers to: `https://hub.docker.com/`.
3. Publish a new version of the documentation to: `http://limmen.dev/csle`.
4. Make a GitHub release with all artifacts (`https://github.com/Limmen/csle`).

Python releases. To make a new Python release, do the following:

1. Copy the `csle/simulation_system/libs/.pypirc_template` file to your home directory and rename it to `.pypirc`, e.g., `/home/kim/.pypirc`.
2. Create a PyPi token following the intructions here: `https://pypi.org/manage/account/token/`.
3. Add the PyPi token to the `.pypirc` file.
4. Edit the `RELEASE_CONFIG` variable to match the versions of the release in the file:

```
./simulation-system/libs/make_release.py
```

Listing 144: File to generate Python releases.

5. Run the command below to build all the Python libraries and push the built artifacts to PyPi:

```
python ./simulation-system/libs/make_release.py
```

Listing 145: Command to generate a new Python release.

Documentation releases. After making a new Python release, the new API documentation can be generated by running the command:

```
simulation-system/libs/generate_docs.sh
```

Listing 146: Command to generate release documentation.

Before running the above command, update the release version in the following files:

```
csle/simulation-system/libs/csle-collector/docs/source/conf.py
csle/simulation-system/libs/csle-ryu/docs/source/conf.py
csle/simulation-system/libs/csle-common/docs/source/conf.py
csle/simulation-system/libs/csle-attacker/docs/source/conf.py
csle/simulation-system/libs/csle-defender/docs/source/conf.py
csle/simulation-system/libs/csle-system-identification/docs/source/conf.py
csle/simulation-system/libs/gym-csle-stopping-game/docs/source/conf.py
csle/simulation-system/libs/gym-csle-apt-game/docs/source/conf.py
csle/simulation-system/libs/gym-csle-cyborg/docs/source/conf.py
csle/simulation-system/libs/csle-agents/docs/source/conf.py
csle/simulation-system/libs/csle-rest-api/docs/source/conf.py
csle/simulation-system/libs/csle-cluster/docs/source/conf.py
csle/simulation-system/libs/gym-csle-intrusion-response-game/docs/source/conf.py
csle/simulation-system/libs/csle-tolerance/docs/source/conf.py
```

Listing 147: Configuration files for automatic generation of API documentation.

JavaScript releases. Update the version parameter in the following file:

```
csle/management-system/csle-mgmt-webapp/package.json
```

Listing 148: Package file for the management system webapp.

Docker releases. To make a new Docker release, do the following:

1. Edit the `VERSION` variable in the file:

```
./emulation-system/base_images/Makefile
```

Listing 149: Makefile for base Docker images in CSLE.

2. Edit the `VERSION` variable in the file:

```
./emulation-system/derived_images/Makefile
```

Listing 150: Makefile for derived Docker images in CSLE.

3. Edit the versions of the derived images in the Dockerfiles inside the following directory:

```
./emulation-system/derived_images/
```

Listing 151: Directory with derived Docker images in CSLE.

4. Edit the versions of the base images in the Dockerfiles inside the following directory:

```
./emulation-system/base_images/
```

Listing 152: Directory with base Docker images in CSLE.

5. Edit the version parameters in all emulation configurations in the following directory:

```
./emulation-system/envs
```

Listing 153: Directory with emulation configurations.

6. Make a new release of `https://github.com/Limmen/exploit-CVE-2017-7494`:

- First clone the repository:

```
git clone https://github.com/Limmen/exploit-CVE-2017-7494
cd exploit-CVE-2017-7494
```

Listing 154: Command to clone `https://github.com/Limmen/exploit-CVE-2017-7494`.

- Then update the version in the file `exploit-CVE-2017-7494/Makefile`.
- Then build and push the image by running the commands:

```
make build
make push
```

Listing 155: Commands to build and push the exploit-CVE-2017-7494 image.

7. Build the images by running the command:

```
cd emulation-system/; make build
```

Listing 156: Command to build the Docker images in CSLE.

8. Push the images to DockerHub by running the command:

```
cd emulation-system/; make push
```

Listing 157: Command to push Docker images to DockerHub.

9. Insert the new emulation configuration into the metastore by running the command:

```
cd emulation-system/; make emulations
```

Listing 158: Command to insert emulation configurations into the metastore.

GitHub releases. To make a release in GitHub, follow the official guide[16].

[16] `https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository`

5 How-to Guides and Tutorials

This section contains guides and tutorials on how to use CSLE. Users of CSLE can request additional tutorials to be added to this section by creating GitHub Issues.

5.1 How to: Add Base Containers

To add a base container with the name "`my_container`", do the following steps:

1. Create a sub-directory with the name `my_container` in the following directory:

```
csle/emulation-system/base_images/docker_files
```

Listing 159: Directory with base Docker images.

2. Add a Dockerfile to the following directory:

```
csle/emulation-system/base_images/docker_files/my_container
```

Listing 160: Directory with the Dockerfile of a container with the name `my_container`.

3. Add build and push instructions to the following Makefile:

```
csle/emulation-system/base_images/Makefile
```

Listing 161: Makefile for base Docker images.

4. Update the following README file:

```
csle/emulation-system/base_images/README.md
```

Listing 162: README file for base Docker images.

5. Build and push the image to DockerHub by running the commands:

```
cd csle/emulation-system/base_images; make my_container
cd csle/emulation-system/base_images; make push_my_container
```

Listing 163: Commands to build and push the base Docker image `my_container` to DockerHub.

5.2 How to: Add Derived Containers

To add a derived container with the name "`my_container`", perform the following steps:

1. Create a sub-directory with the name `my_container` in the directory:

```
csle/emulation-system/derived_images/
```

Listing 164: Directory with derived Docker images.

2. Add a Dockerfile inside the directory:

```
csle/emulation-system/derived_images/my_container/
```

Listing 165: Directory with the Dockerfile for a derived Docker image with the name `my_container`.

3. Add build and push instructions to the following Makefile:

```
csle/emulation-system/derived_images/Makefile
```

Listing 166: Makefile for derived Docker images.

4. Update the following README file:

```
csle/emulation-system/derived_images/README.md
```

Listing 167: README file for derived Docker images.

5. Build and push the image to DockerHub by running the commands:

```
cd csle/emulation-system/derived_images; make my_container
cd csle/emulation-system/derived_images; make push_my_container
```

Listing 168: Commands to build and push the derived Docker image `my_container` to DockerHub.

5.3 How to: Add Emulation Configurations

To add a new emulation configuration with the name "`level_13`" and version 0.0.3, perform the following steps:

1. Create a sub-directory called `level_13` in the folder:

```
csle/emulation-system/envs/003/
```

Listing 169: Directory with emulation configurations with version 0.0.3.

2. Add the emulation configuration file `config.py` to the directory:

```
csle/emulation-system/envs/003/level_13/
```

Listing 170: Directory with emulation configuration file for the emulation with the name `level_13` and version 0.0.3.

3. Update the following README file:

```
csle/emulation-system/envs/003/README.md
```

Listing 171: README file for emulation configurations with version 0.0.3.

4. Update the following Makefile with installation instructions:

```
csle/emulation-system/envs/003/README.md
```

Listing 172: Makefile for emulation configurations with version 0.0.3.

5. Insert the emulation configuration into the metastore by running the command:

```
csle/emulation-system/envs; make install
```

Listing 173: Command to insert emulation configurations into the metastore.

5.4 How to: Add Simulation Configurations

To add a new simulation environment with the name "`test_env`" and version 0.0.1, perform the following steps:

1. Create a sub-directory called `test_env` in the folder:

```
csle/simulation-system/envs/
```

Listing 174: Directory with simulation configurations.

2. Add the simulation configuration file `config_v_001.py` to the directory:

```
csle/simulation-system/envs/test_env
```

Listing 175: Directory with configuration file for the simulation environment with the name `test_env`.

3. Update the following README file:

```
csle/simulation-system/envs/README.md
```

Listing 176: README file for simulation configurations.

4. Update the following Makefile with installation instructions:

```
csle/simulation-system/envs/Makefile
```

Listing 177: Makefile for simulation configurations.

5. Insert the simulation configuration into the metastore by running the command:

```
csle/simulation-system/envs; make install
```

Listing 178: Command to insert simulation configurations into the metastore.

5.5 How to: Update Monitoring Agents

To update the monitoring agents that run on the emulated devices, perform the following steps:

1. Do the necessary code changes to the monitoring agent by editing the files in the directory:

```
csle/simulation-system/libs/csle-collector/src/
```

Listing 179: Directory with source code of the monitoring agents in CSLE.

2. Update the gRPC API by editing the protocol-buffer[17] files in the directory:

```
csle/simulation-system/libs/csle-collector/protos/
```

Listing 180: Directory with protocol buffers for the monitoring agents in CSLE.

3. Regenerate the gRPC Python files by running the commands:

[17] https://developers.google.com/protocol-buffers

```
python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/client_manager/.
 ↪   ./protos/client_manager.proto
python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/kafka_manager/.
 ↪   ./protos/kafka_manager.proto

python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/elk_manager/.
 ↪   ./protos/elk_manager.proto
python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/docker_stats_manager/.
 ↪   ./protos/docker_stats_manager.proto

python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/snort_ids_manager/.
 ↪   ./protos/snort_ids_manager.proto
python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/host_manager/.
 ↪   ./protos/host_manager.proto

python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/ossec_ids_manager/.
 ↪   ./protos/ossec_ids_manager.proto
python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/traffic_manager/.
 ↪   ./protos/traffic_manager.proto

python -m grpc_tools.protoc -I./protos/
 ↪  --python_out=./src/csle_collector/.
 ↪  --grpc_python_out=./src/csle_collector/ryu_manager/.
 ↪   ./protos/ryu_manager.proto
```

Listing 181: Commands to generate gRPC Python files from protobuf files.

5.6 How to: Add Numerical Algorithms

To add a new numerical algorithm called "`my_algorithm`" to the simulation system, perform the following steps:

1. Create a new sub-directory called `my_algorithm` inside the directory:

```
csle/simulation-system/libs/csle-agents/src/
     csle_agents/agents/
```

Listing 182: Directory with numerical algorithms in CSLE.

2. Implement the algorithm in a file called `my_algorithm.py` in the directory:

```
csle/simulation-system/libs/csle-agents/src/
    csle_agents/agents/my_agent
```

Listing 183: Directory with algorithm implementation for the algorithm with the name `my_agent` in CSLE.

   The implementation should be a Python class called "`MyAlgorithm`" that inherits from the class "`BaseAgent`", which is defined in the file:

```
csle/simulation-system/libs/csle-agents/src/csle_agents/
    agents/base/base_agent.py
```

Listing 184: Base agent file in CSLE.

3. Add an example of how to use the algorithm to the following folder:

```
csle/examples
```

Listing 185: Directory with example usages of CSLE.

5.7 How to: Add Configuration Parameters

To add a new configuration parameter, perform the following steps:

1. Add the configuration parameter to the file: `csle/config.json`.
2. Add the new configuration parameter to the file:

```
csle/simulation-system/libs/csle-common/src/
    csle_common/dao/emulation_config/config.py
```

Listing 186: Python file that manages the CSLE configuration.

3. Add the new parameter to an appropriate class in the following file (add a new class if needed):

```
csle/simulation-system/libs/csle-common/src
    /csle_common/constants/constants.py
```

Listing 187: Python file with constants in `csle-common`.

6 Frequently Asked Questions

1. Q — **How to add a new management user?**

   A — New management users can be added by administrators through the web interface.

2. Q — **Why does my machine crash when I try to run the emulation environment `csle-level9-003`?**

   A — Most likely because the machine is running out of memory. The emulation environment `csle-level9-003` includes 34 containers, each of which requires at least 1GB of RAM, meaning that the machine needs at least 34GB of RAM to run this emulation environment.

3. Q — **Why does my reinforcement learning algorithm not converge?**

   A — There are many possible reasons for this; it could be a hyperparameter problem, a bug in your training environment, or simply slow convergence. These issues are not related to CSLE's implementation.

4. Q — **How can I use CSLE with my own custom code?**

   A — You have two options. One option is to integrate your code with CSLE as a new simulation or emulation environment (see Section 5 for instructions on how to do this). Another option is that you write custom code to integrate CSLE with your existing code base. The latter option is not something that CSLE maintainers will provide support for.

5. Q — **How do I generate plots?**

   A — CSLE does not have its own plotting library. Plots can be created through standard libraries, e.g., matplotlib [26].

6. Q — **How do I install CSLE?**

   A — See the instructions in Section 3.

7. Q — **Can I install CSLE on Windows?**

A — No.

8. Q — **Can I install CSLE on Mac?**

   A — Yes, you should be able to take the installation instructions for Linux and adapt them to a Mac environment. Note that Mac is not a platform that is actively supported in CSLE. Hence, you can not expect help from maintainers if you run into Mac-specific issues.

9. Q — **How do I use the Docker images in CSLE without the rest of the framework?**

   A — The CSLE Docker images can be pulled as standalone images from DockerHub [18].

10. Q — **How do I cite CSLE?**

    A — A list of publications is available in Section 1.4.3.

11. Q — **Can I use CSLE for commercial purposes?**

    A — Yes if it is compatible with the license (Creative Commons Attribution-ShareAlike 4.0 International License).

12. Q — **Can I use CSLE for research purposes?**

    A — Yes, CSLE was built for research purposes. If you use CSLE in your research please cite us.

13. Q — **Can I use CSLE in an air-gapped environment?**

    A — An air-gapped installation is possible but is not something that is officially supported in CSLE.

14. Q — **Can I run CSLE in the cloud?**

    A — Yes you can deploy CSLE on virtual machines provided by any cloud vendor.

15. Q — **Can I take the CSLE code as a base to develop a new framework?**

[18] https://hub.docker.com/r/kimham/

A — Yes as long as you respect the license (Creative Commons Attribution-ShareAlike 4.0 International License).

16. Q — **Can I deploy CSLE on a Kubernetes cluster?**

    A — It is possible but it is not something that we have documentation for.

17. Q — **How can I contribute to CSLE?**

    A — See the developer guide in Section 4.

18. Q — **How can I contact the maintainers of CSLE?**

    A — See Section 1.5.1.

A Setup PyCharm for Remote Development

The best way to debug and implement new features to the emulation system involves three steps: (1) deploy the latest version of the emulation system on your target cluster; (2) clone the latest source code both on the server and on your local machine (e.g., your laptop); (3) connect your Integrated Development Environment (IDE) to the cluster so that you can edit code directly on the cluster. This appendix walks through the steps of setting up the PyCharm IDE for remote development.

Step 1: Deploy CSLE on the servers. Follow the instructions in Section 3.

Step 2: Clone the source code. Run the following command on your local machine and on one of the servers where CSLE is installed:

```
1 git clone https://github.com/Limmen/csle
```

Listing 188: Command for cloning the source code of CSLE.

Step 3: Setup PyCharm for remote development. Start PyCharm on your local machine and open the CSLE project. Next, go to "Preferences" in PyCharm (see Fig. 31).

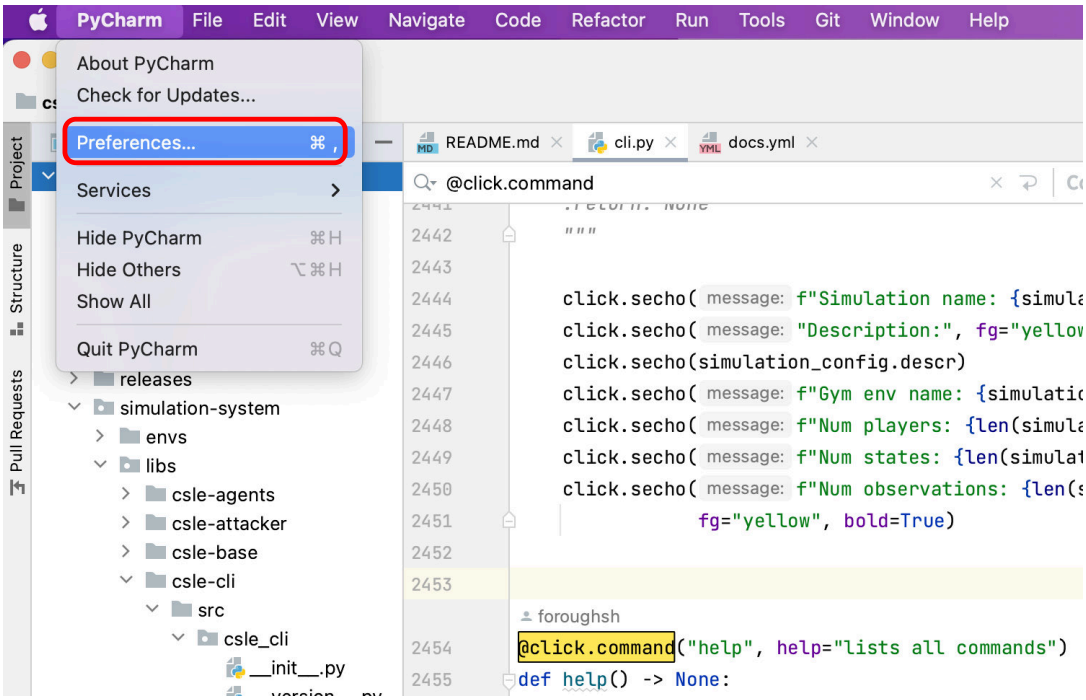

Figure 31: The preferences tab in PyCharm.

Then configure an SSH interpreter for the project by selecting the Python interpreter of the server where CSLE is installed (see Figs. 32-35).

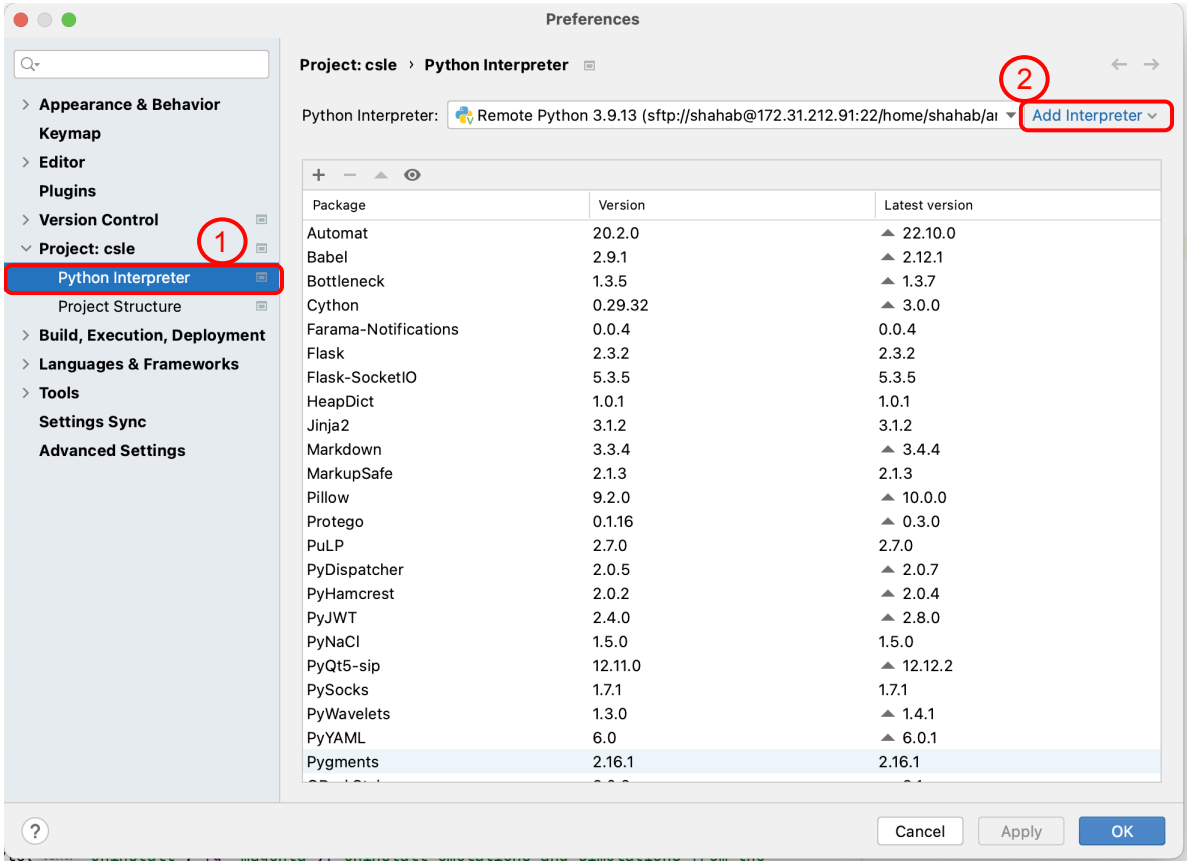

Figure 32: Configuration of a remote SSH Python interpreter in PyCharm (1/4).

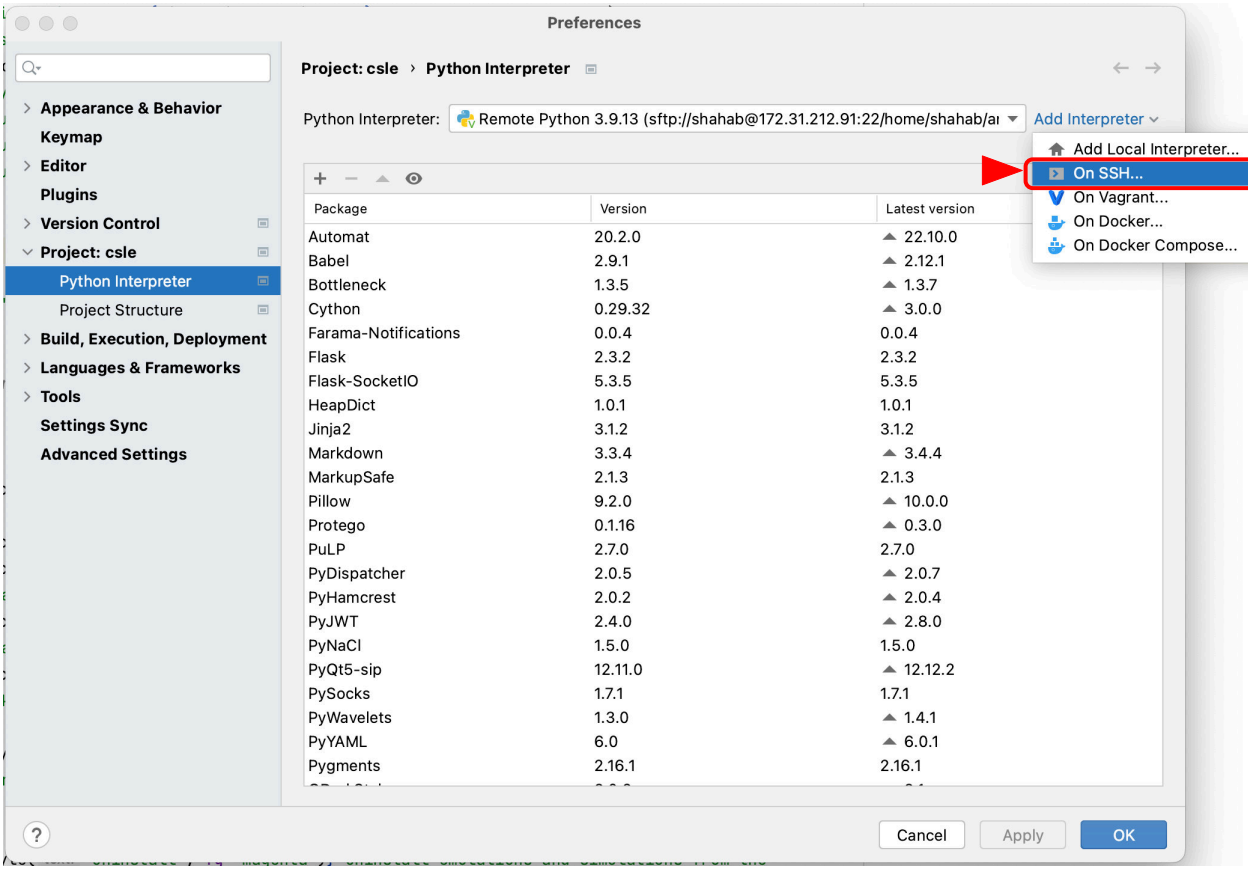

Figure 33: Configuration of a remote SSH Python interpreter in PyCharm (2/4).

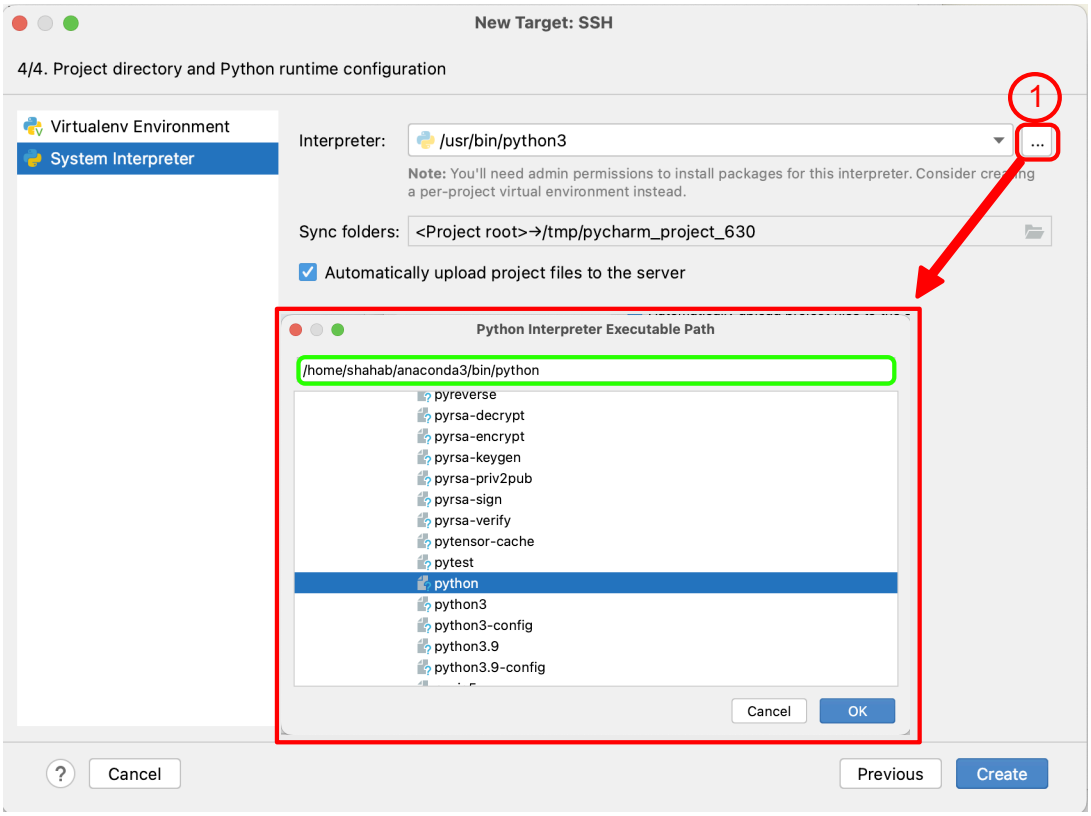

Figure 34: Configuration of a remote SSH Python interpreter in PyCharm (3/4).

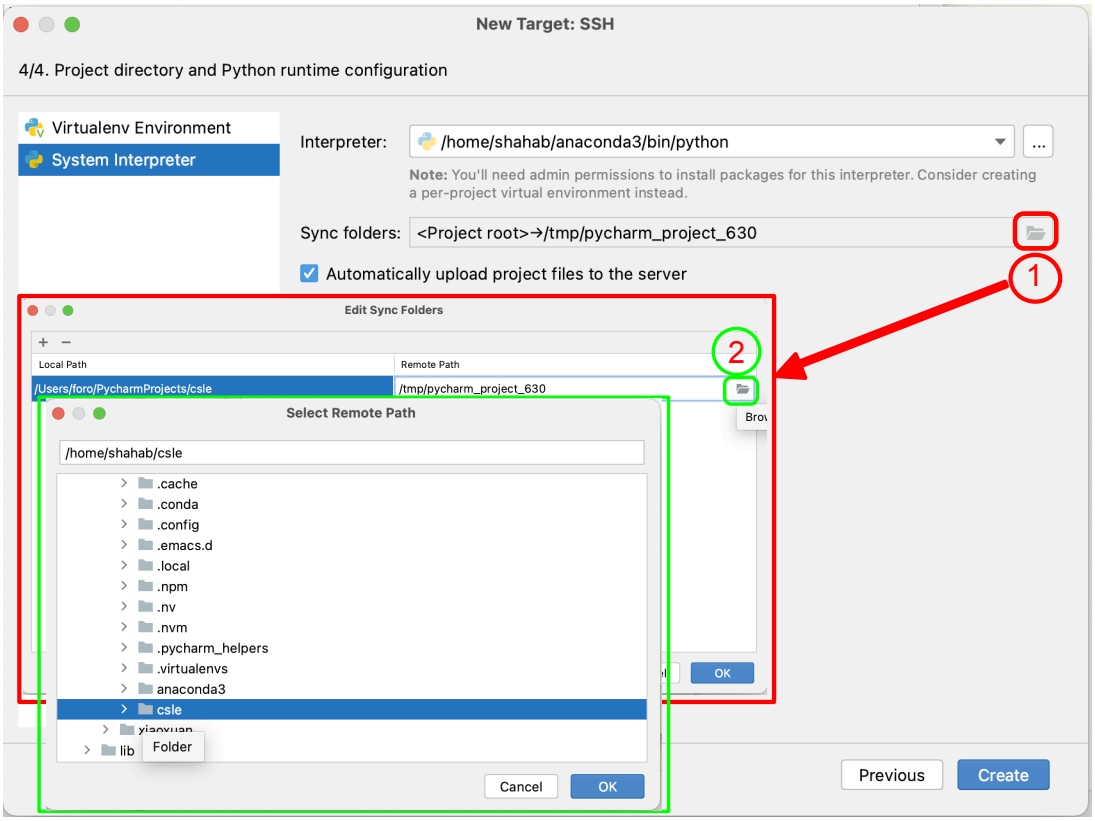

Figure 35: Configuration of a remote SSH Python interpreter in PyCharm (4/4).

To test that the remote SSH interpreter is working correctly and that the files are synchronized between the remote server and your local machine you can try to add a line of code in the IDE and verify that it also shows up on the server. You can also test the setup by running an example script in the `examples/` folder.

B Useful Git Commands

This appendix describes the minimal set of git commands necessary to follow the CSLE contribution workflow (see Section 4).

When you start working on a new feature, first make sure that you have pulled the latest code changes by running the following commands in the root of the CSLE repository:

```
git checkout master
git pull origin master
```

Listing 189: Commands for pulling the latest code changes from the master branch.

If the above command did not succeed due to code conflicts you can either resolve the conflicts or choose to reset the code to the latest commit (this will override any local changes) by running the commands:

```
git reset --hard origin/master
git clean -fd
```

Listing 190: Commands for resetting the local version of the code to the upstream version. (Warning: this will override any local changes to the code.)

Next, create a new branch for your code changes by running the command:

```
git checkout -b mybranchname
```

Listing 191: Command for creating a new branch with name "`mybranchname`".

Then you can make your code changes. When you have made a set of changes you can commit them by running the commands:

```
git add --all
git commit -m "comment for the commit"
```

Listing 192: Commands committing new code changes.

You can make several commits. When you feel that your new feature is ready to be merged into master you can push the code to GitHub by running the command:

```
git push origin mybranchname
```

Listing 193: Command for pushing the branch "`mybranchname`" to GitHub.

After pushing the changes you can go with your web browser to:

`https://github.com/Limmen/csle`

and open a new pull request.

When the pull request is merged into the master branch you can delete the local branch and pull the new changes from the master branch by running the commands:

```
git checkout master
git pull origin master
git branch -d mybranchname
```

Listing 194: Commands for (1) switching to the master branch; (2) pulling the latest changes; and (3) delete the old feature branch with name "`mybranchname`".

After running the above commands you can verify that your commits are available in the master branch by running the command:

```
git log
```

Listing 195: Command for viewing git commits.

References

[1] Alex Andrew et al. "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.

[2] A.Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[3] William Blum. *Gamifying machine learning for stronger security and AI models*. Microsoft Research. 2021.

[4] A. Clemm and Inc Cisco Systems. *Network Management Fundamentals*. Cisco Press fundamentals series. Cisco Press, 2007. ISBN: 9781587201370. URL: https://books.google.se/books?id=NvEVQmjScjYC.

[5] Umur Cubukcu et al. “Citus: Distributed PostgreSQL for Data-Intensive Applications”. In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD ’21. Virtual Event, China: Association for Computing Machinery, 2021, 2490–2502. ISBN: 9781450383431. DOI: 10.1145/3448016.3457551. URL: https://doi.org/10.1145/3448016.3457551.

[6] Steven Diamond and Stephen Boyd. “CVXPY: A Python-embedded modeling language for convex optimization”. In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[7] Brendan Eich. “JavaScript at ten years”. In: *Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming, ICFP 2005, Tallinn, Estonia, September 26-28, 2005*. Ed. by Olivier Danvy and Benjamin C. Pierce. ACM, 2005, p. 129. DOI: 10.1145/1086365.1086382. URL: https://doi.org/10.1145/1086365.1086382.

[8] Facebook. *React*. https://github.com/facebook/react. 2021. URL: https://github.com/facebook/react.

[9] Google. *cAdvisor*. https://github.com/google/cadvisor. 2022. URL: https://github.com/google/cadvisor.

[10] Google. *Google Remote Procedure Call*. 2022.

[11] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. 1st. O’Reilly Media, Inc., 2015. ISBN: 1449358543.

[12] Grafana Labs. *Grafana Documentation*. 2018. URL: https://grafana.com/docs/ (visited on 07/25/2019).

[13] Kim Hammar. *Awesome Reinforcement Learning for Cyber Security*. https://github.com/Limmen/awesome-rl-for-cybersecurity. 2021. URL: https://github.com/Limmen/awesome-rl-for-cybersecurity.

[14] Kim Hammar and Rolf 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. DOI: 10.1109/NOMS54207.2022.9789707.

[15] Kim Hammar and Rolf Stadler. “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. DOI: 10.23919/CNSM55787.2022.9964838.

[16] Kim Hammar and Rolf Stadler. “Digital Twins for Security Automation”. In: *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. 2023, pp. 1–6. DOI: 10.1109/NOMS56928.2023.10154288.

[17] Kim Hammar and Rolf Stadler. “Finding Effective Security Strategies through Reinforcement Learning and Self-Play”. In: *International Conference on Network and Service Management (CNSM 2020)*. Izmir, Turkey, 2020.

[18] Kim Hammar and Rolf Stadler. “Intrusion Prevention Through Optimal Stopping”. In: *IEEE Transactions on Network and Service Management* 19.3 (2022), pp. 2333–2348. DOI: 10.1109/TNSM.2022.3176781.

[19] Kim Hammar and Rolf Stadler. “Learning Intrusion Prevention Policies through Optimal Stopping”. In: *International Conference on Network and Service Management (CNSM 2021)*. http://dl.ifip.org/db/conf/cnsm/cnsm2021/1570732932.pdf. Izmir, Turkey, 2021.

[20] Kim Hammar and Rolf Stadler. “Learning Near-Optimal Intrusion Responses Against Dynamic Attackers”. In: *IEEE Transactions on Network and Service Management* (2023), pp. 1–1. DOI: 10.1109/TNSM.2023.3293413.

[21] Kim Hammar and Rolf Stadler. “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.

[22] Kim Hammar and Rolf Stadler. *Scalable Learning of Intrusion Responses through Recursive Decomposition*. 2023. arXiv: 2309.03292 [eess.SY]. URL: https://arxiv.org/abs/2309.03292.

[23] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. *CMA-ES/pycma on Github*. Zenodo, DOI:10.5281/zenodo.2559634. Feb. 2019. DOI: 10.5281/zenodo.2559634. URL: https://doi.org/10.5281/zenodo.2559634.

[24] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[25] Stephen Hemminger. “Network emulation with NetEm”. In: *Linux Conf* (2005).

[26] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[27] Anaconda Inc. *Anaconda*. https://docs.conda.io/. 2022. URL: https://docs.conda.io/.

[28] Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. *NASimEmu: Network Attack Simulator & Emulator for Training Agents Generalizing to Novel Scenarios*. 2023. arXiv: 2305.17246 [cs.CR].

[29] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement Learning in Robotics: A Survey”. In: *International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.

[30] Jay Kreps. “Kafka : a Distributed Messaging System for Log Processing”. In: 2011.

[31] Marc Lanctot et al. *OpenSpiel: A Framework for Reinforcement Learning in Games*. 2019. DOI: 10.48550/ARXIV.1908.09453. URL: https://arxiv.org/abs/1908.09453.

[32] Nevena Lazic et al. “Data center cooling using model-predictive control”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/059fdcd96baeb75112f09fa1dcc740cc-Paper.pdf.

[33] Li Li, Raed Fayad, and Adrian Taylor. *CyGIL: A Cyber Gym for Training Autonomous Agents over Emulated Network Systems*. 2021. arXiv: 2109.03331 [cs.CR].

[34] Nick McKeown et al. “OpenFlow: Enabling Innovation in Campus Networks”. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (2008), 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[35] Dirk Merkel. “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux journal* 2014 (2014), p. 2.

[36] Andres Molina-Markham et al. “Network Environment Design for Autonomous Cyberdefense”. In: (2021). https://arxiv.org/abs/2103.07583.

[37] Salman Niazi et al. “Leader Election Using NewSQL Database Systems”. In: *Distributed Applications and Interoperable Systems*. Ed. by Alysson Bessani and Sara Bouchenak. Cham: Springer International Publishing, 2015, pp. 158–172. ISBN: 978-3-319-19129-4.

[38] Andrei Paleyes et al. *Emulation of physical processes with Emukit*. 2021. arXiv: 2110.13293 [cs.LG].

[39] Pallets. *Flask*. https://github.com/pallets/flask. 2021. URL: https://github.com/pallets/flask.

[40] Ben Pfaff et al. “The Design and Implementation of Open vSwitch”. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130. ISBN: 978-1-931971-218. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff.

[41] Prometheus. *Node exporter*. https://github.com/prometheus/node_exporter. 2022. URL: https://github.com/prometheus/node_exporter.

[42] Bjorn Rabenstein and Julius Volz. "Prometheus: A Next-Generation Monitoring System (Talk)". In: Dublin: USENIX Association, May 2015.

[43] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[44] Will Reese. "Nginx: The High-Performance Web Server and Reverse Proxy". In: *Linux J.* 2008.173 (2008). ISSN: 1075-3583.

[45] Guido van Rossum, Barry Warsaw, and Nick Coghlan. *PEP 8 - Style Guide for Python Code*. https://peps.python.org/pep-0008/. 2001. URL: https://peps.python.org/pep-0008/.

[46] Raghav Sethi et al. "Presto: SQL on Everything". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2019, pp. 1802–1813. DOI: 10.1109/ICDE.2019.00196.

[47] Maxwell Standen et al. "CybORG: A Gym for the Development of Autonomous Cyber Agents". In: *CoRR* (). https://arxiv.org/abs/2108.09118.

[48] Daniel Stenberg. *Curl*. https://curl.se/. 2022. URL: https://curl.se/.

[49] Mitchell Stuart et al. *Optimization with PuLP*. https://coin-or.github.io/pulp/. 2022. URL: https://coin-or.github.io/pulp/.

[50] The PostgreSQL Global Development Group. *PostgreSQL*. 2021.

[51] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[52] Matei Zaharia et al. "Spark: Cluster Computing with Working Sets". In: *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA: USENIX Association, 2010, p. 10.