An Online Framework for Adapting Security Policies in Dynamic IT Environments

International Conference on Network and Service Management
Thessaloniki, Greece, Oct 31 - Nov 4 2022

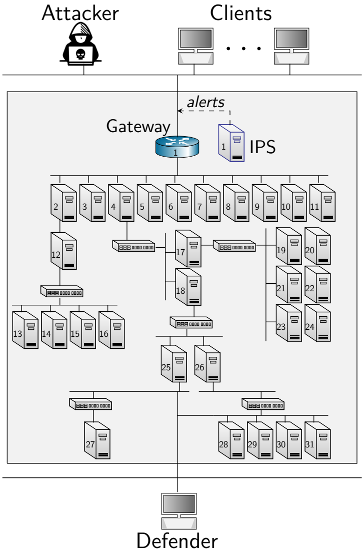
Kim Hammar & Rolf Stadler

kimham@kth.se stadler@kth.se

Division of Network and Systems Engineering
KTH Royal Institute of Technology

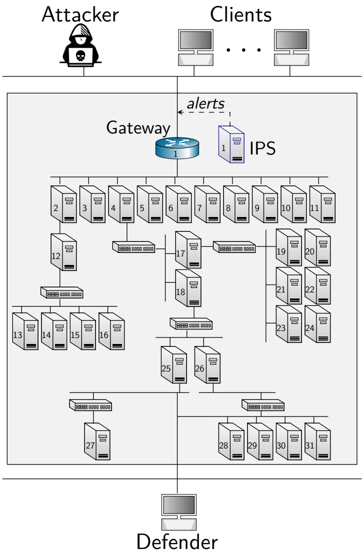
Challenges: Evolving and Automated Attacks

- **Challenges**
  - Evolving & automated attacks
  - Complex infrastructures



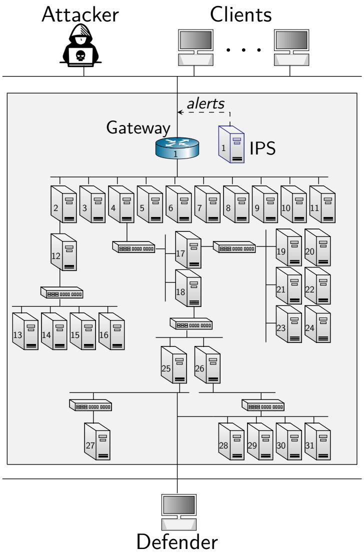
Goal: Automation and Learning

- **Challenges**
  - Evolving & automated attacks
  - Complex infrastructures
- **Our Goal**:
  - Automate security tasks
  - Adapt to changing attack methods

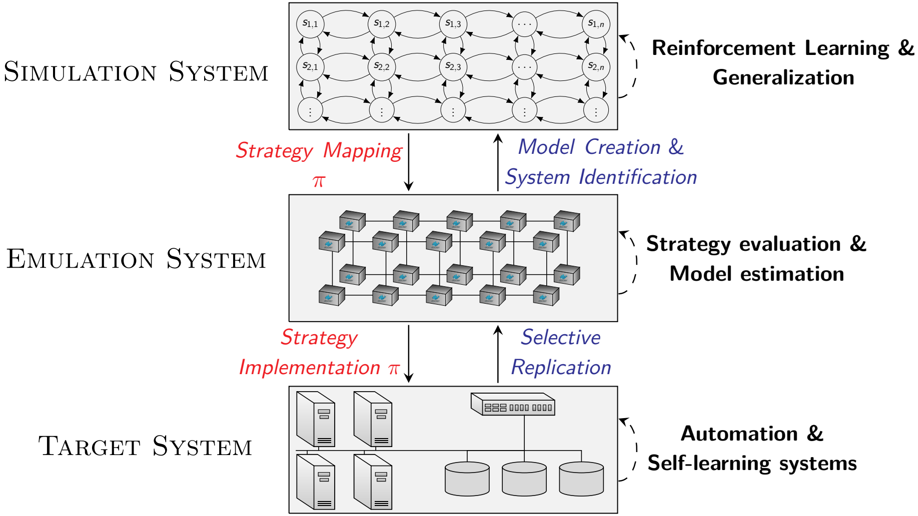


Approach: Self-Learning Security Systems

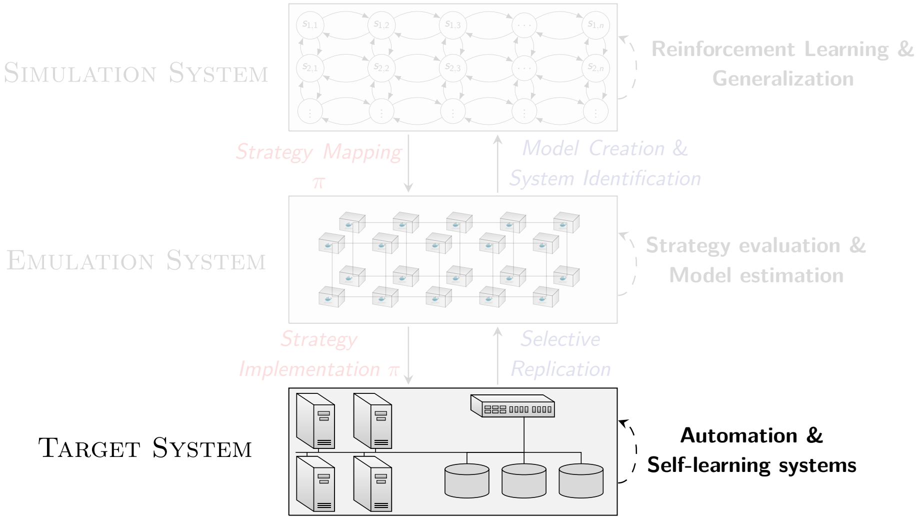
- **Challenges**
  - Evolving & automated attacks
  - Complex infrastructures
- **Our Goal**:
  - Automate security tasks
  - Adapt to changing attack methods
- **Our Approach: Self-Learning Systems**:
  - real-time telemetry
  - stream processing
  - theories from control/game/decision theory
  - computational methods (e.g. dynamic programming & reinforcement learning)
  - automated network management (SDN, NFV, etc.)



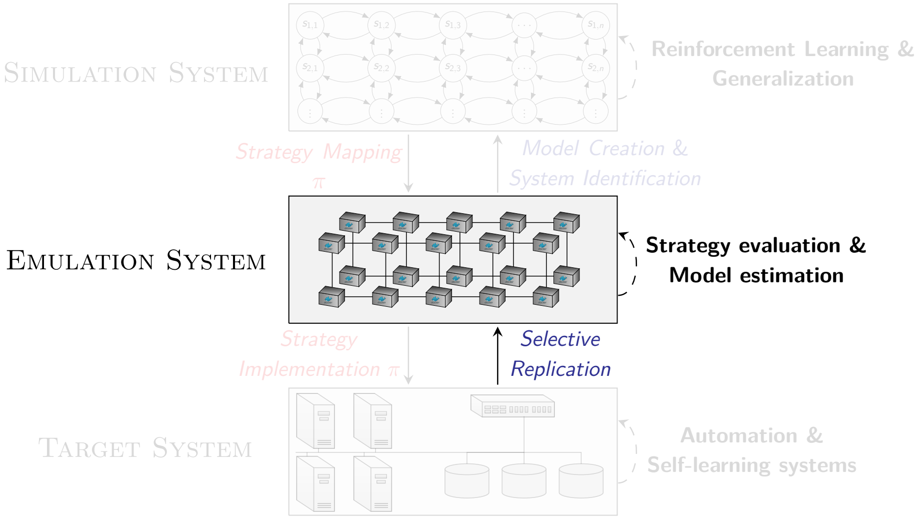
Our Framework for Automated Network Security



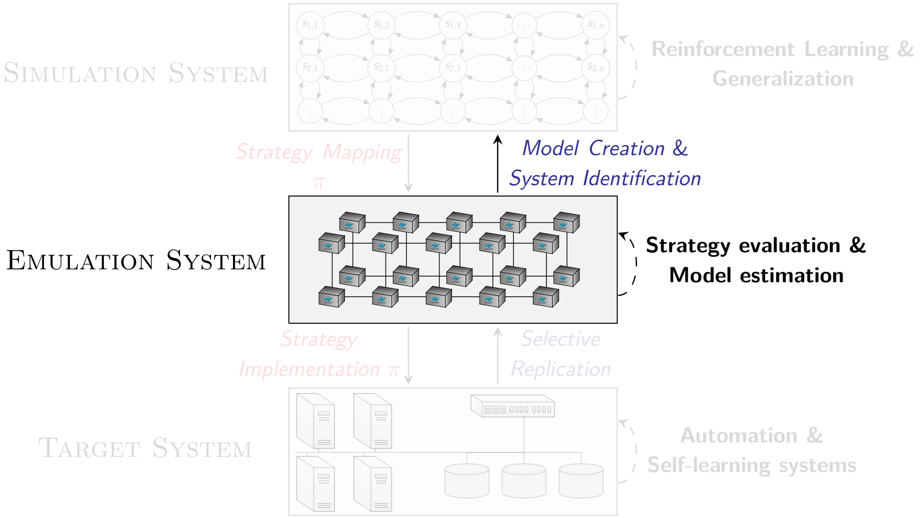
Our Framework for Automated Network Security



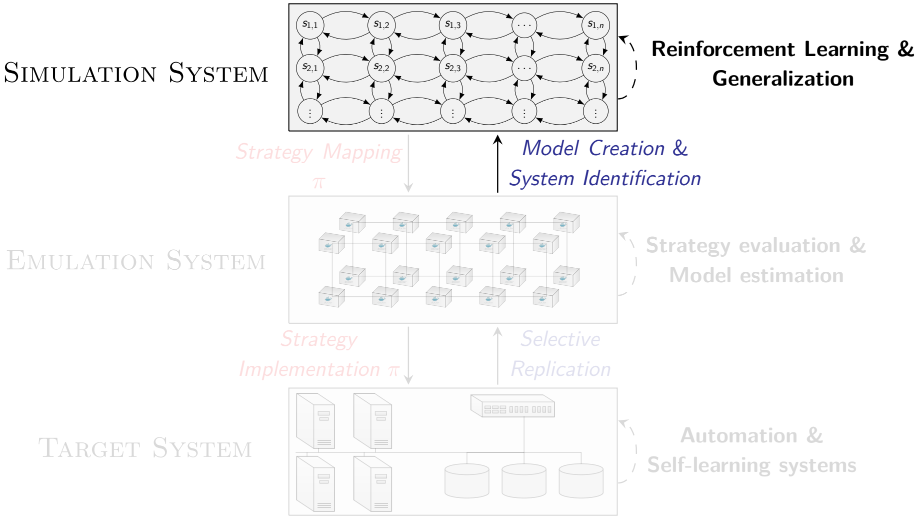
Our Framework for Automated Network Security



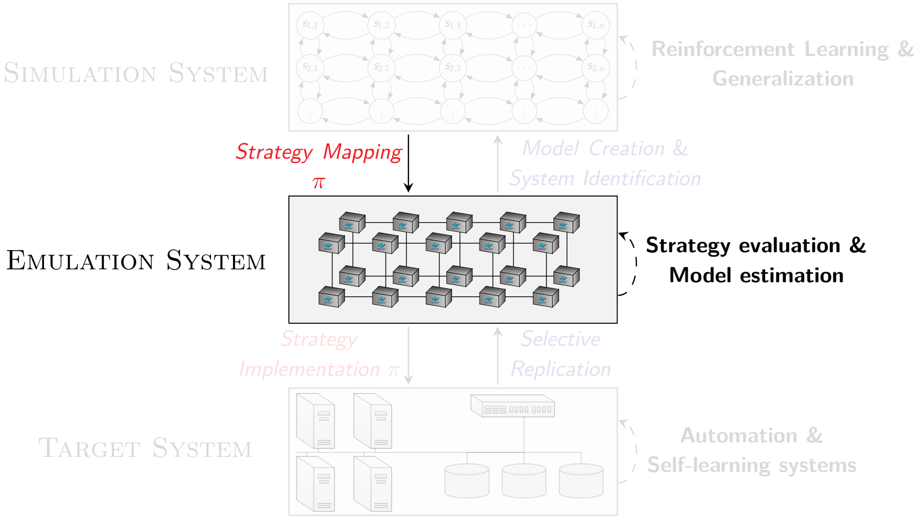
Our Framework for Automated Network Security



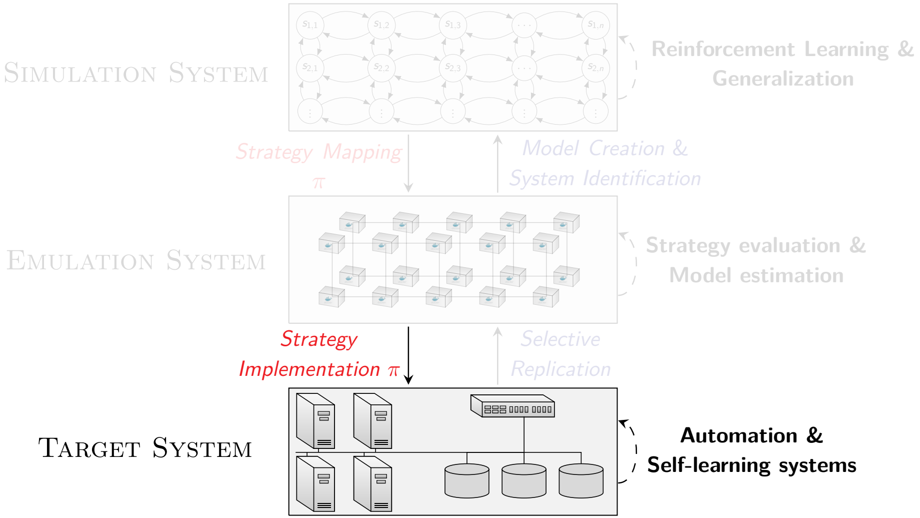
Our Framework for Automated Network Security



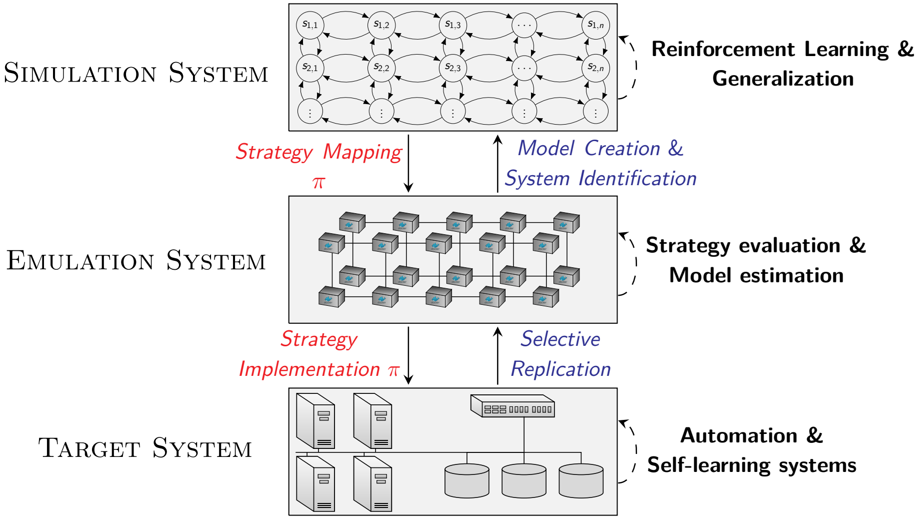
Our Framework for Automated Network Security



Our Framework for Automated Network Security



Our Framework for Automated Network Security



Our Previous Work

- *Finding Effective Security Strategies through Reinforcement Learning and Self-Play*[1]
- *Learning Intrusion Prevention Policies through Optimal Stopping*[2]
- *A System for Interactive Examination of Learned Security Policies*[3]
- *Intrusion Prevention Through Optimal Stopping*[4]
- *Learning Security Strategies through Game Play and Optimal Stopping*[5]

---

[1] 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.

[2] 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.

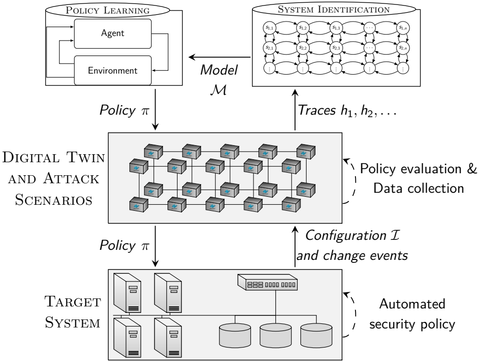
[3] Kim Hammar and Rolf Stadler. "A System for Interactive Examination of Learned Security Policies". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 2022, pp. 1–3. DOI: `10.1109/NOMS54207.2022.9789707`.

[4] 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`.

[5] 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.

This Paper: Learning in Dynamic IT Environments[6]

- **Challenge:** operational IT environments are dynamic
  - Components may fail, load patterns can shift, etc.
- Contribution: we present a framework for learning and updating security policies in dynamic IT environments



[6] Kim Hammar and Rolf Stadler. "An Online Framework for Adapting Security Policies in Dynamic IT Environments". In: *International Conference on Network and Service Management (CNSM 2022)*. Thessaloniki, Greece, 2022.

Learning in Dynamic IT Environments

Algorithm 1: High-level execution of the framework

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

Algorithm (*emulator*, φ, ϕ)
  do in parallel
    DigitalTwin(*emulator*)
    SystemIdProcess(φ)
    LearningProcess(ϕ)
  end

Procedure DigitalTwin(*emulator*)
  Loop
    $\pi \leftarrow$ ReceiveFromLearningProcess()
    $h_t \leftarrow$ CollectTrace(π)
    SendToSystemIdProcess(h_t)
    UpdateDigitalTwin(*emulator*)
  EndLoop

Procedure SystemIdProcess(φ)
  Loop
    $h_1, h_2, \ldots \leftarrow$ ReceiveFromDigitalTwin()
    $\mathcal{M} \leftarrow \varphi(h_1, h_2, \ldots)$ // estimate model
    SendToLearningProcess($\mathcal{M}$)
  EndLoop

Procedure LearningProcess(ϕ)
  Loop
    $\mathcal{M} \leftarrow$ ReceiveFromSystemIdProcess()
    $\pi \leftarrow \phi(\mathcal{M})$ // learn policy π
    SendToDigitalTwin(π)
  EndLoop

Learning in Dynamic IT Environments

Algorithm 2: High-level execution of the framework

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

Algorithm (*emulator*, φ, ϕ)
    do in parallel
        DigitalTwin(*emulator*)
        SystemIdProcess(φ)
        LearningProcess(ϕ)
    end

Procedure DigitalTwin(*emulator*)
    Loop
        $\pi \leftarrow$ ReceiveFromLearningProcess()
        $h_t \leftarrow$ CollectTrace(π)
        SendToSystemIdProcess(h_t)
        UpdateDigitalTwin(*emulator*)
    EndLoop

Procedure SystemIdProcess(φ)
    Loop
        $h_1, h_2, \ldots \leftarrow$ ReceiveFromDigitalTwin()
        $\mathcal{M} \leftarrow \varphi(h_1, h_2, \ldots)$ `// estimate model`
        SendToLearningProcess($\mathcal{M}$)
    EndLoop

Procedure LearningProcess(ϕ)
    Loop
        $\mathcal{M} \leftarrow$ ReceiveFromSystemIdProcess()
        $\pi \leftarrow \phi(\mathcal{M})$ `// learn policy` π
        SendToDigitalTwin(π)
    EndLoop

The Digital Twin

Algorithm 3: High-level execution of the framework

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

Algorithm (*emulator*, φ, ϕ)
    do in parallel
        $\textsc{DigitalTwin}(\textit{emulator})$
        $\textsc{SystemIdProcess}(\varphi)$
        $\textsc{LearningProcess}(\phi)$
    end

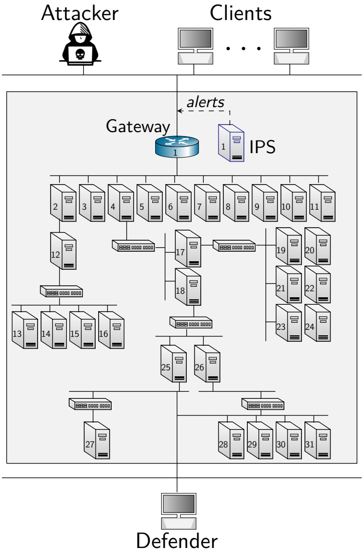
Procedure $\textsc{DigitalTwin}(\textit{emulator})$
    Loop
        $\pi \leftarrow \textsc{ReceiveFromLearningProcess}()$
        $h_t \leftarrow \textsc{CollectTrace}(\pi)$
        $\textsc{SendToSystemIdProcess}(h_t)$
        $\textsc{UpdateDigitalTwin}(\textit{emulator})$
    EndLoop

Procedure $\textsc{SystemIdProcess}(\varphi)$
    Loop
        $h_1, h_2, \ldots \leftarrow \textsc{ReceiveFromDigitalTwin}()$
        $\mathcal{M} \leftarrow \varphi(h_1, h_2, \ldots)$ `// estimate model`
        $\textsc{SendToLearningProcess}(\mathcal{M})$
    EndLoop

Procedure $\textsc{LearningProcess}(\phi)$
    Loop
        $\mathcal{M} \leftarrow \textsc{ReceiveFromSystemIdProcess}()$
        $\pi \leftarrow \phi(\mathcal{M})$ `// learn policy` π
        $\textsc{SendToDigitalTwin}(\pi)$
    EndLoop

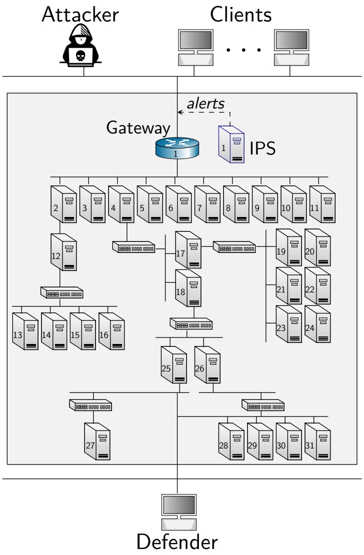
Creating a Digital Twin of the Target System

- Emulate **hosts** with docker containers
- Emulate **IPS and vulnerabilities** with software
- Network isolation and **traffic shaping** through NetEm in the Linux kernel
- Enforce **resource constraints** using cgroups.
- Emulate **client arrivals** with Poisson process
- **Internal connections** are full-duplex & loss-less with bit capacities of 1000 Mbit/s
- **External connections** are full-duplex with bit capacities of 100 Mbit/s & 0.1% packet loss in normal operation and random bursts of 1% packet loss



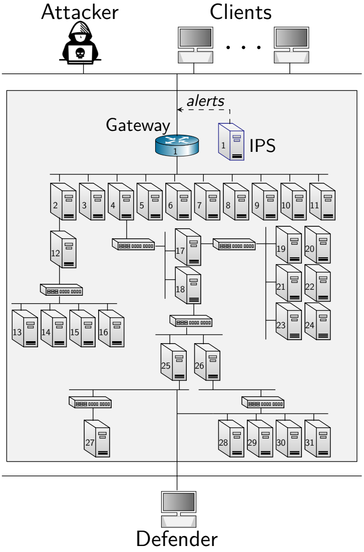
Creating a Digital Twin of the Target System

- Emulate **hosts** with docker containers
- Emulate **IPS and vulnerabilities** with software
- Network isolation and **traffic shaping** through NetEm in the Linux kernel
- Enforce **resource constraints** using cgroups.
- Emulate **client arrivals** with Poisson process
- **Internal connections** are full-duplex & loss-less with bit capacities of 1000 Mbit/s
- **External connections** are full-duplex with bit capacities of 100 Mbit/s & 0.1% packet loss in normal operation and random bursts of 1% packet loss



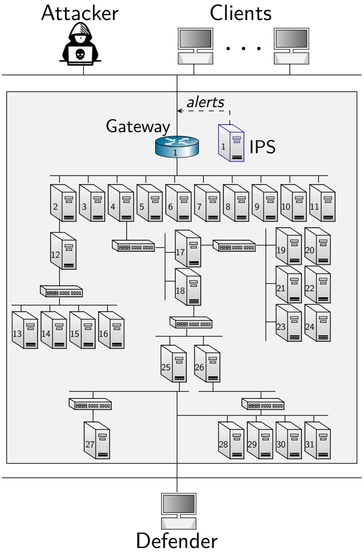
Creating a Digital Twin of the Target System

- Emulate **hosts** with docker containers
- Emulate **IPS and vulnerabilities** with software
- Network isolation and **traffic shaping** through NetEm in the Linux kernel
- Enforce **resource constraints** using cgroups.
- Emulate **client arrivals** with Poisson process
- **Internal connections** are full-duplex & loss-less with bit capacities of 1000 Mbit/s
- **External connections** are full-duplex with bit capacities of 100 Mbit/s & 0.1% packet loss in normal operation and random bursts of 1% packet loss



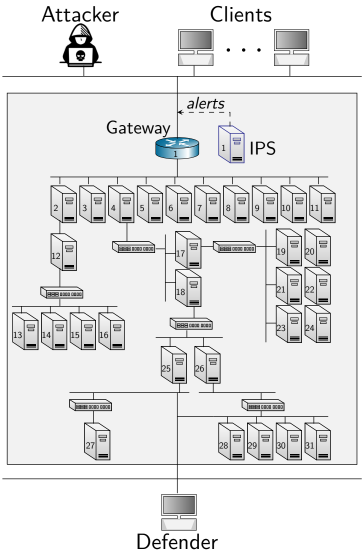
Creating a Digital Twin of the Target System

- Emulate **hosts** with docker containers
- Emulate **IPS and vulnerabilities** with software
- Network isolation and **traffic shaping** through NetEm in the Linux kernel
- Enforce **resource constraints** using cgroups.
- Emulate **client arrivals** with Poisson process
- **Internal connections** are full-duplex & loss-less with bit capacities of 1000 Mbit/s
- **External connections** are full-duplex with bit capacities of 100 Mbit/s & 0.1% packet loss in normal operation and random bursts of 1% packet loss



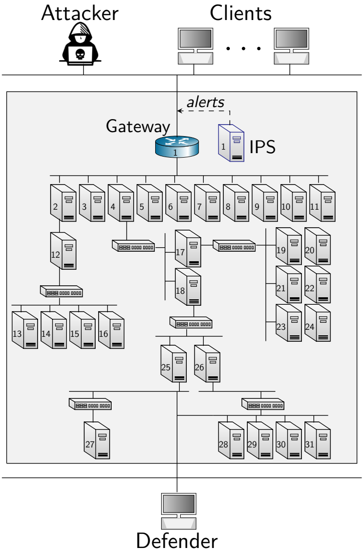
Creating a Digital Twin of the Target System

- Emulate **hosts** with docker containers
- Emulate **IPS and vulnerabilities** with software
- Network isolation and **traffic shaping** through NetEm in the Linux kernel
- Enforce **resource constraints** using cgroups.
- Emulate **client arrivals** with Poisson process
- **Internal connections** are full-duplex & loss-less with bit capacities of 1000 Mbit/s
- **External connections** are full-duplex with bit capacities of 100 Mbit/s & 0.1% packet loss in normal operation and random bursts of 1% packet loss



Creating a Digital Twin of the Target System

- Emulate **hosts** with docker containers
- Emulate **IPS and vulnerabilities** with software
- Network isolation and **traffic shaping** through NetEm in the Linux kernel
- Enforce **resource constraints** using cgroups.
- Emulate **client arrivals** with Poisson process
- **Internal connections** are full-duplex & loss-less with bit capacities of 1000 Mbit/s
- **External connections** are full-duplex with bit capacities of 100 Mbit/s & 0.1% packet loss in normal operation and random bursts of 1% packet loss



The System Identification Process

Algorithm 4: High-level execution of the framework

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

Algorithm (*emulator*, φ, ϕ)
  do in parallel
    DigitalTwin(*emulator*)
    SystemIdProcess(φ)
    LearningProcess(ϕ)
  end

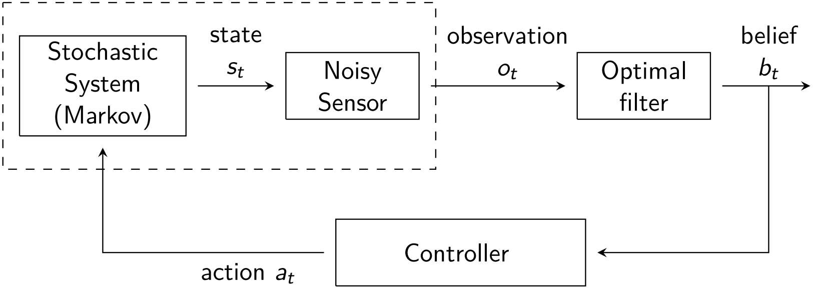
Procedure DigitalTwin(*emulator*)
  Loop
    $\pi \leftarrow$ ReceiveFromLearningProcess()
    $h_t \leftarrow$ CollectTrace(π)
    SendToSystemIdProcess(h_t)
    UpdateDigitalTwin(*emulator*)
  EndLoop

Procedure SystemIdProcess(φ)
  Loop
    $h_1, h_2, \ldots \leftarrow$ ReceiveFromDigitalTwin()
    $\mathcal{M} \leftarrow \varphi(h_1, h_2, \ldots)$ // estimate model
    SendToLearningProcess($\mathcal{M}$)
  EndLoop

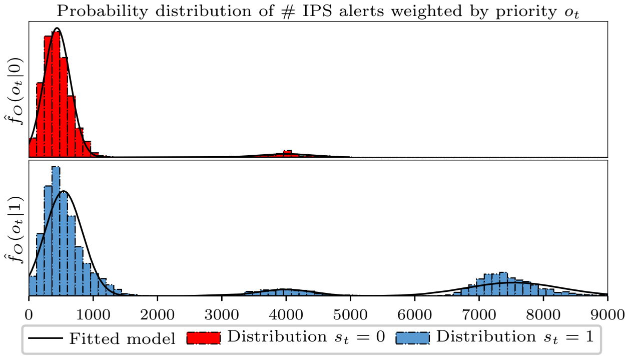
Procedure LearningProcess(ϕ)
  Loop
    $\mathcal{M} \leftarrow$ ReceiveFromSystemIdProcess()
    $\pi \leftarrow \phi(\mathcal{M})$ // learn policy π
    SendToDigitalTwin(π)
  EndLoop

System Model

- We model the evolution of the system with a discrete-time dynamical system.
- We assume a Markovian system with stochastic dynamics and partial observability.



System Identification



- The distribution f_O of defender observations (system metrics) is unknown.
- We fit a Gaussian mixture distribution $\hat{f}_O$ as an estimate of f_O in the target system.
- For each state s, we obtain the conditional distribution $\hat{f}_{O|s}$ through expectation-maximization.

The Policy Learning Process

Algorithm 5: High-level execution of the framework

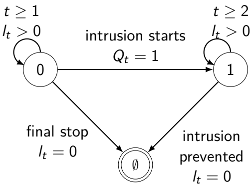
Input: *emulator*: method to create digital twin
φ: system identification algorithm
ϕ: policy learning algorithm

```
Algorithm (emulator, φ, ϕ)
    do in parallel
        DigitalTwin(emulator)
        SystemIdProcess(φ)
        LearningProcess(ϕ)
    end
Procedure DigitalTwin(emulator)
    Loop
        π ← ReceiveFromLearningProcess()
        h_t ← CollectTrace(π)
        SendToSystemIdProcess(h_t)
        UpdateDigitalTwin(emulator)
    EndLoop
Procedure SystemIdProcess(φ)
    Loop
        h_1, h_2, ... ← ReceiveFromDigitalTwin()
        M ← φ(h_1, h_2, ...)                    // estimate model
        SendToLearningProcess(M)
    EndLoop
Procedure LearningProcess(ϕ)
    Loop
        M ← ReceiveFromSystemIdProcess()
        π ← ϕ(M)                                // learn policy π
        SendToDigitalTwin(π)
    EndLoop
```

Learning Effective Defender Policies

- **Optimization problem:**
  - Each stopping time = one defensive action
  - Maximize reward of stopping times $\tau_L, \tau_{L-1}, \ldots, \tau_1$:

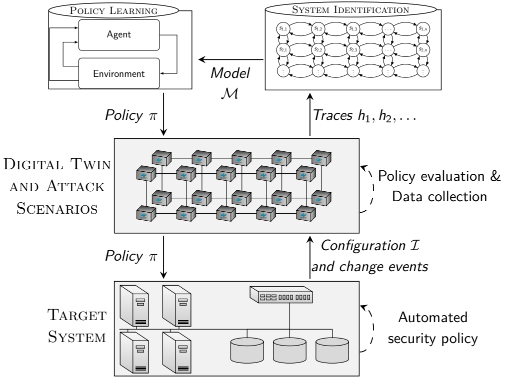
$$
\pi_l^* \in \arg\max_{\pi_l} \mathbb{E}_{\pi_l}\Bigg[\sum_{t=1}^{\tau_L-1} \gamma^{t-1}\mathcal{R}^C_{s_t,s_{t+1},L} + \gamma^{\tau_L-1}\mathcal{R}^S_{s_{\tau_L},s_{\tau_L+1},L} + \ldots + \sum_{t=\tau_2+1}^{\tau_1-1} \gamma^{t-1}\mathcal{R}^C_{s_t,s_{t+1},1} + \gamma^{\tau_1-1}\mathcal{R}^S_{s_{\tau_1},s_{\tau_1+1},1}\Bigg]
$$



- **Optimization methods:** Reinforcement learning, dynamic programming, computational game theory, etc.

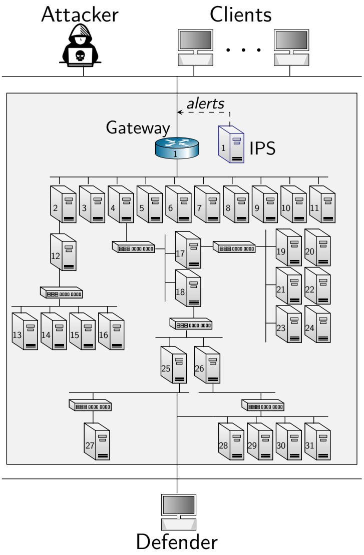
Putting it all together: Learning in Dynamic Environments

1. Changes in the target system are monitored.
2. When changes are detected, the emulation is updated.
3. Attack and defense scenarios are run in the emulation to collect data.
4. The system model and the defender policy are updated periodically with the new data.

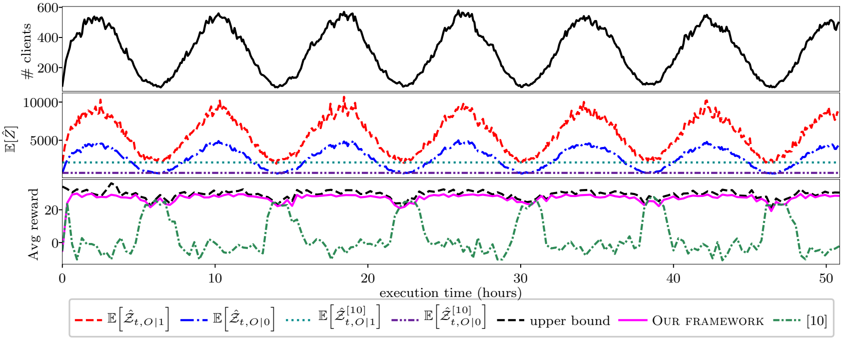


Use Case: Intrusion Prevention

- A **Defender** owns an infrastructure
  - Consists of connected components
  - Components run network services
  - Defender defends the infrastructure by monitoring and active defense
  - Has partial observability
- An **Attacker** seeks to intrude on the infrastructure
  - Has a partial view of the infrastructure
  - Wants to compromise specific components
  - Attacks by reconnaissance, exploitation and pivoting



Results: Learning in a Dynamic IT Environment



Results from running our framework for 50 hours in the digital twin/emulation.

Conclusions

- We present a framework for learning and updating security policies in dynamic IT environments
- We apply the method to an **intrusion prevention use case**.
- We show numerical results in a realistic emulation environment.
- We design a solution framework guided by the theory of optimal stopping.