Description

Lightweight framework to train and test cyber agents (benign, attacker, defender) in real-like environments using Docker and standard DevOps toolchains. Motivated by the need to evaluate defensive tactics, automate detection/response, and stress-test agents against realistic attack patterns while keeping experiments reproducible and containerized.

Main goal

Build and validate defender agents that operate reliably in containerized real-world stacks, while supporting benign/attacker agents for realistic training and evaluation.

Milestones

1. Short- term For the group to finish the task of a class
2. Mid-term Something working to send to Ekoparty (3-4 months)
3. Long-term nlnet

Tasks

- **Environment**
    - Define minimal Dockerized target stack(s) (web server, DB, auth).
      Add CI/CD pipeline for spin-up/tear-down (Terraform/Ansible/Make + Git actions).
- **Agents**
    - Implement a lightweight **defender** agent (monitoring, rule-based response, feedback loop).
    - Implement a benign agent (normal user behavior)
    - Implement attacker agent (parameterized adversary scenarios).
- **Training & Testing**
    - Create training scenarios focused on detection, containment, and recovery for the defender.
      Define metrics: detection time, false positives, containment success, and recovery time.
- **Evaluation**
    - Run scripted red-team vs blue-team runs; collect logs, metrics, and replayable traces.
    - Iterate defender policy based on failures (automated retrain/deploy).
- **Ops & Repro**
    - Container images, config as code, and a README with runbook (start, stop, reproduce).
    - Minimal observability: central logging + simple dashboard/alerts for results.

Priority note: keep defender development and evaluation first-class — every task should include at least one defender-specific deliverable (rules, telemetry, or retraining).

TASK 1 — Lab Infrastructure

The first phase focuses on standing up the core lab using Docker Compose. The lab must include a victim machine and a server, both with SSH access and OpenCode installed, SLIPS running to generate alerts, a small forwarder service to send alerts to the defender machine, and a minimal FastAPI service on the defender side to receive them.

During this period, you ensure the containers can communicate, logs rotate properly, services restart cleanly, and alerts flow reliably from SLIPS to the defender. By the end of this step, running the lab stack should consistently result in alert messages arriving at the defender endpoint.

Deliverables: A Docker Compose stack and associated Dockerfiles for victim, server, SLIPS, and defender. a short readme.md with build/start/verify steps, test scripts that validate container networking, SSH reachability, log rotation, restart policies, and end-to-end alert flow.

TASK 2 — ATTACKER Integration

Integrate ARACNE on an external, operational host and use it to perform lightweight, non-destructive actions on the target network (for example basic enumeration and benign persistence checks) in order to validate that ARACNE activity reliably triggers SLIPS alerts and that those alerts propagate correctly through the defender pipeline. The goal is not to emulate a full, sophisticated compromise but to produce reproducible, measurable events that exercise detection, alerting, and ingestion workflows. Conceptually this should resemble aspects of NetSecGame — ARACNE acting from an external host to generate controlled network activity — while keeping all actions safe, constrained, and focused on detection validation.

Deliverables: a short set of natural language ARACNE prompts,SLIPS alert JSON that those prompts produced, a compact results CSV summary of runs (run id, prompt, alert id, latency.

Example ARACNE prompts:

“Scan the whole network to find any machines offering SSH.”

“Check which computers respond on port 80 and tell me their IPs.”

TASK 3 — Defender Prompt Builder

In this phase, an LLM will design the prompt for the defender. This module takes in SLIPS alerts and produces well-scoped defensive prompts for OpenCode. It does not execute anything yet it only returns text prompts.

Deliverables: the prompt-builder code, plus examples of received alerts and the text prompts/output the defender produces from them.

TASK 4 — Remote Execution Path

This step connects everything: the defender now takes the generated prompt, SSHes into the victim, and runs it with OpenCode. Output must be collected, timeouts handled, and failures surfaced clearly.

Deliverables: An executor component that performs SSH, uploads prompts, invokes OpenCode in a safe, sandboxed way, and returns structured results with exit codes, stdout/stderr, and diagnostics.

TASK 5 — End-to-End Baseline Runs

With execution wired in, you conduct structured runs across a small set of baseline attacker behaviors, such as reconnaissance and simple persistence. For each run, you record alerts, defender actions, system responses, and success or failure.

The emphasis is reliability. You refine error handling, improve logs, and verify that the defender's decisions are consistent across multiple executions. A minimum consistent success rate across the scenarios is required before moving on.

Deliverables: A set of scenario definitions (recon, persistence, etc.), a runner to orchestrate ARACNE runs and defender flows, and structured results (CSV/JSON) capturing timestamps, alert IDs, actions, success/failure and latencies. Include analysis scripts that compute success rates and latency distributions, plus an autogenerated summary report showing acceptance thresholds and observed reliability.

TASK 6 — Benign Activity Introduction

Next, you bring realism to the environment by simulating benign user behavior through BUDA or a similar generator (https://github.com/cmu-sei/GHOSTS). You calibrate this activity so the environment looks more like a real system, with everyday user actions happening alongside attacker behavior.

The goal is to understand false positives and response noise. You observe how your defender behaves under benign load, validate that SLIPS remains useful, and make adjustments to avoid unnecessary responses during quiet or legitimate activity windows.

Deliverables: Integration configs and compose/run scripts for BUDA/GHOSTS or an equivalent benign workload generator, calibrated activity profiles (light/medium/heavy), and calibration scripts to collect SLIPS alert counts and false-positive metrics.

TASK 7 — Manual Red-Team Investigation

At this stage, you shift to probing the system for weakness. You manually attempt prompt injection and environment-level manipulations, such as poisoning banners, file names, outputs, or other text sources that may be reflected into the defensive prompts or OpenCode context.

You document which attempts succeed or fail and capture precise examples along with mitigation ideas. The goal is to harden the system through realistic adversarial pressure.

Deliverables: A catalog of manual test cases (vectors, payloads, steps), and reproducible evidence (captured prompts, logs, screenshots). Include a concise runbook for performing safe manual tests.

TASK 8 — Automated Red-Team Campaign

Finally, you automate the successful manual injection methods via ARACNE. This step validates whether the system can defend against programmatic attempts that apply the learned injection strategies and whether the mitigations hold under scripted pressure.

Deliverables: Automated campaign scripts/playbooks that reproduce successful manual injection vectors at a controlled pace, a campaign runner that tags and aggregates per-run metrics, and a final analysis/report showing automated success rates, time-to-detect, mitigation effectiveness, and recommended fixes.

TASK 9 — Traffic Capture and Dataset Generation

Objective: produce reproducible, privacy-conscious network dataset artifacts that map SLIPS alerts to packet/flow data for offline analysis and ML experiments.

As part of validation and reproducibility, the lab will capture network traffic and produce dataset artifacts that link SLIPS alerts to the underlying network sessions. These artifacts are intended for analysis, debugging, and offline ML/forensic experiments and will be treated as sensitive data.

ATT&CK tagging: Captured artifacts must be enriched with MITRE ATT&CK labels. For each SLIPS alert and/or correlated flow, include `attack_ids` (MITRE technique IDs). Provide an `attack_mapping` artifact (CSV/YAML) that documents mapping rules used and the ATT&CK matrix version.

Deliverables: raw packet capture (pcap), Zeek logs, labels

Year long plan (Actually just some ideas)

TASK 10 — New network topology

Here you scale the environment to multiple victim machines and a small services node behind a simulated gateway. The stack should remain manageable and provide an environment closer to a small network rather than a single host.

For this, 5 clients and 5 different servers will be created, with an external computer for the attacker, and an extra one for the defender

Your focus is ensuring SLIPS still functions meaningfully across hosts, the defender can correctly identify and act on the right target, and the execution path remains stable in a slightly more complex topology.

Another option is to instead of increasing the network size, create more specialized networks for critical infrastructure, hospitals, industrial control systems, satellites, etc.

TASK 11 — Defensive Agent New Architecture

A coding agent as a defender is a good starting point, however, a specialized architecture could improve it on several aspects. On this step, a defensive agent will be created, having as inspiration, the ARACNE architecture, or even modifying it to adapt it as a defender agent.

TASK 14 — Dataset Generation New Architecture

Task 9 must be re-executed to ensure data consistency and coverage. This repeat phase includes redeploying the capture instrumentation on all new network segments, regenerating pcaps and flow logs under the updated topology, and validating that SLIPS alerts, ATT&CK mappings, and label schemas remain accurate.

TASK 15 — Temporal GNN NIDS Evaluation

Evaluate a Network Intrusion Detection System (NIDS) based on **t**emporal Graph Neural Networks (GNNs) using the lab’s captured traffic and labels (including MITRE ATT&CK tags).

Future Task — Host-Level Detection Integration

Objective: Incorporate host-based intrusion detection alongside SLIPS’s network monitoring to capture malicious behaviors that pure network analysis might miss. While SLIPS analyzes network traffic, adding a lightweight host agent on each victim (e.g., monitoring system logs, processes, or file changes) will provide a more holistic view of attacks.

Success Criteria: By the end of this task, the lab should generate alerts from host-level sensors in addition to SLIPS. In testing, simulate a simple malicious action (e.g., ARACNE creating a new user or running a suspicious binary) and confirm that a host sensor alert is triggered and received by the defender. This ensures “by combining network and host logs, we are able to detect malicious behavior that cannot be detected by either log alone.”

Future steps

- Defender Hardening
- Give ARACNE access to tools like exploitdb

Defender Possible Actions

- Firewall rules
- Process killing
- Account suspending
- File deletion

-

Trident: Realistic cyber range for Benign, Attacker and Defender Autonomous agents

Summary: Can autonomous benign, attacker, and defender agents be truly trusted to operate on real systems?. This can be answered with the creation of a realistic cyber range where these agents execute unrestricted code, interacting with each other on virtual networks, built with dockers on top of StratoCyberLab, which already trains thousands of students each year.

Summary: To assess and improve the trustworthiness of autonomous benign, attacker, and defender agents by designing a realistic and easily deployable cyber range where these agents execute real, unrestricted code, on virtual networks. The evaluation of agent safety, robustness, and misuse is carried out in a containerized environment built on top of StratoCyberLab, which already trains thousands of students each year.

Outcomes

- **Enable trustworthy and safe evaluation of autonomous agents**
  This includes letting benign, attacker, and defender agents operate with real code inside a controlled cyber range, validating that they can safely execute unrestricted code within isolated containers, and benchmarking attacker behaviors for systematic comparisons.

- **Provide a reproducible and realistic environment for experimentation**
  The environment should mirror real networks, support larger topologies, and allow safe experimentation without risk to production systems.

- **Improve defensive automation and agent reliability**
  This covers generating high quality prompts, structured execution paths, and consistent response policies for defender agents, as well as measuring robustness, misuse resistance, and reliability through baseline runs, adversarial testing, and automated red team campaigns.

- **Reduce false positives and model operational noise realistically**
  This involves introducing calibrated benign activity that resembles real user behavior to help reduce false positives in defender agents and better understand noise within operational environments.

- **Generate high quality datasets and support next generation detection models**
  The system should produce rich network and host level datasets enriched with MITRE ATT&CK

tags, supporting offline analysis, ML experimentation, future NIDS research, and evaluation of advanced detection models such as temporal GNN based NIDS.

- **Provide an extensible and modular framework for sensors, agents, and detection modules**
  New sensors, autonomous agents, or detection components should be pluggable without redesigning the environment, enabling studies of mixed benign, attacker, and defender activity and revealing emergent behaviors or failure modes.

Abstract: Can you explain the whole project and its expected outcome(s). (1200 chars)

"Trident" is a realistic cyber range built on dockers using **StratoCyberLab**. It enables benign, attacker, and defender LLM agents to interact and execute unrestricted code on virtual networks.

The workflow evolves through distinct phases:

1. **Infrastructure & Integration:** Establishing a closed-loop system where an Attacker triggers alerts, prompting the Defender to act.
2. **Realism & Calibration:** Injecting benign user traffic to measure false positives and refine detection logic.
3. **LLM Red Teaming:** conducting manual and automated red-teaming to test the Defender against prompt injections.
4. **Expansion:** Scaling to complex network topologies to test advanced detection models (Temporal GNNs).

Expected Outcomes

- **Trustworthy Evaluation:** A safe, isolated sandbox for benchmarking unrestricted command benign, attacker, and defender code.
- **Modular Framework:** An extensible platform to test future sensors and agent architectures on simulated critical infrastructure.
- **High-Quality Datasets:** Reproducible network artifacts (PCAP/Zeek) enriched with **MITRE ATT&CK** tags for offline ML research.

Have you been involved with projects or organisations relevant to this project before? And if so, can you tell us a bit about your contributions?

Since 2018, we have been collaborating closely with the Stratosphere Laboratory, the research group behind the SCL cyber-range platform used in the CTU Introduction to Cybersecurity MOOC, which serves thousands of students worldwide. Our work with Stratosphere has focused primarily on developing advanced tools for malware and network-based threat detection.

Our contributions include the development of a Domain Generation Algorithm (DGA) detection model in 2016, followed by a DNS tunneling detection model in 2019. These efforts supported Stratosphere's broader mission to create open, effective, and accessible cybersecurity solutions.

More recently, we have been part of the next generation of SLIPS Immune, a state-of-the-art Network Detection and Intrusion System (NDIS) being developed by Stratosphere. SLIPS Immune is also supported by NLnet and is currently under active development. Our participation has focused on research, design, and implementation of new detection capabilities within the platform.

Through these long-term collaborations, we have gained extensive experience in open-source cybersecurity research, machine-learning-based threat detection, and the development of tools that align strongly with NLnet's vision and values.

Requested **Amount** (in Euro)

Explain what the requested budget will be used for? Does the project have other funding sources, both past and present? (If you want, you can in addition attach a budget at the bottom of the form)

The estimated duration of the project is one year. The budget will be used for salaries of the researchers, and will be distributed as follows:

- Project coordinator, senior AI expert, researcher: 1,800 EUR gross per month
- Main developer: 1300 EUR gross per month .
- Junior developer and docker specialist: 1000 EUR gross per month.

- Total amount gross per month: 4,100 EUR
- Total duration of the project: 12 months

Total budget requested: 4,100 * 12 = **49,200 EUR**

Compare your own project with existing or historical efforts. 4000 chars

Trident evaluates whether autonomous LLM-based benign, attacker, and defender agents can operate safely and effectively on real systems. It gives agents direct access to actual shells and services inside reproducible Docker containers. The goal is not only to check if an agent reaches an objective, but to observe how it behaves when every decision directly controls a real system: its commands execute exactly as generated, with no abstraction layer correcting or filtering them.

Earlier platforms operate at lower fidelity. CyberBattleSim (https://github.com/microsoft/CyberBattleSim) and Yawning Titan (https://github.com/dstl/YAWNING-TITAN) simplify networks into graphs where nodes flip between abstract states like “compromised,” and actions are symbolic choices with no underlying system execution. These are efficient for RL, but strategies learned in such models rarely transfer to real operating conditions.

NetSecGame (https://github.com/stratosphereips/NetSecGame) goes a step further by requiring agents to identify services and follow logical network interactions. However, actions remain structured messages applied to a simulated environment. The agent specifies intentions; the engine pretends the effects happened. Since no commands are actually run, there are no real logs, errors, or unexpected consequences to learn from.

CybORG (https://github.com/cage-challenge/CybORG) is the closest predecessor. Its emulation mode maps predefined agent actions to concrete commands on cloud machines, producing real effects. But those commands come from a controlled action set that the agent never writes them itself. This maintains safety but limits what can be studied: behaviour is bounded by designer assumptions. Cloud execution overhead also leads many researchers to remain in its simplified simulation mode.

Trident removes the predefined action set entirely. Agents type the commands they choose, and those commands run directly on containerized hosts. The platform does not interpret, translate, or correct actions; it executes them. Safety comes from controlled reproducibility: Docker containers can be reset at any time, allowing experiments to freely observe both effective and flawed behaviours without risk outside the environment.

This enables datasets no other platform currently produces: records of **real reconnaissance, real interactions, real alerts, and real responses** driven end-to-end by the agent’s own decisions. Traffic generated during scanning, attempted intrusions, and defensive interventions all become part of the captured evidence. The dataset reflects what the agent truly did; not what it intended or what a simulator assumed.

What are significant technical **challenges** you expect to solve during the project, if any? (optional but recommended, 5000 chars)

We expect to solve the following technical challenges

1. How to ensure safe and realistic execution of unrestricted attacker, defender, and benign agent actions.
2. How to emulate convincing benign user behavior and background traffic to support calibration of detection quality and false-positive control.
3. How to coordinate stable multi-agent interactions so attacker, defender, and benign LLM agents behave consistently with their intended roles.
4. How to conduct systematic LLM red-teaming to evaluate the defender against prompt injections and evasion strategies.
5. How to collect high-quality network telemetry (PCAP/Zeek) enriched with reliable MITRE ATT&CK annotations for reproducible research.
6. How to scale the platform to complex and dynamic network topologies required to evaluate advanced detection models.

Describe the ecosystem of the project, and how you will engage with relevant actors and promote the outcomes? (2500 chars)

The project operates within an ecosystem that includes **CTU/Stratosphere Laboratory,** their open-source tools such as **SLIPS** and **ARACNE**, and the wider community of security **practitioners, educators, students, and researchers** who need safe, close-to-real environments for testing new defensive tools. This ecosystem already values reproducibility, open development, and realistic experimentation.

We will engage these groups by contributing improvements and integrations back to the Stratosphere toolchain, ensuring full compatibility with SLIPS alerts, ARACNE scenarios, and existing workflows. Practitioners and educators will have access to ready-to-run Docker environments, baseline attack and defense scenarios, and clear documentation that facilitates adoption in training and classroom settings. Researchers will be able to use the environment and its labeled datasets to run controlled, repeatable experiments.

All project outputs, including code, datasets, runbooks, and evaluation scripts, will be openly released and promoted through Stratosphere channels, community events, and public repositories, ensuring broad visibility and long-term impact.