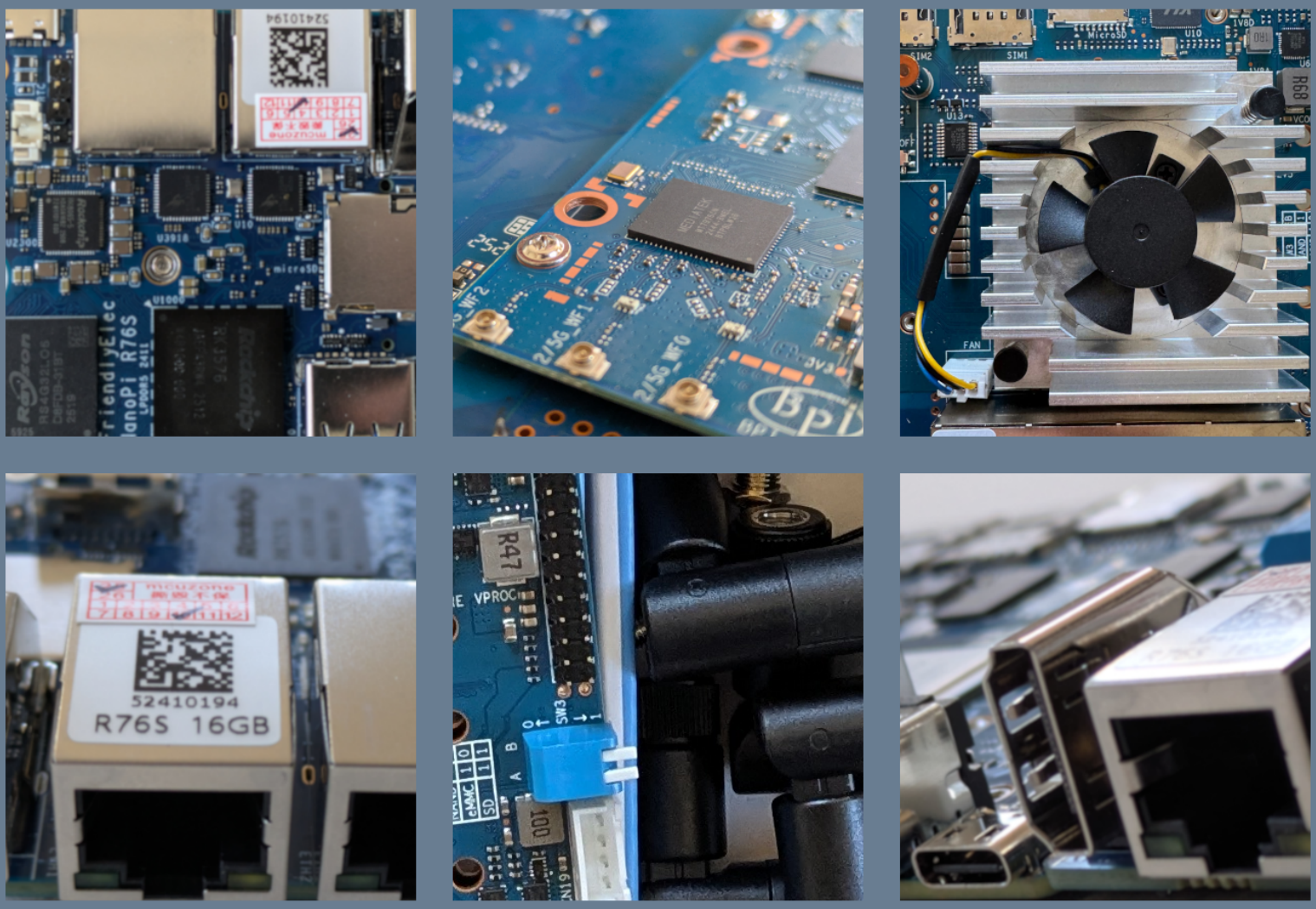

Project 2

Network Swiss Army Knife

Course of study	Bachelor of Science in Computer Science
Author	Frank Gauss (gausf1) and Lukas von Allmen (vonal3)
Advisor	Wenger Hansjürg

Version 1.0 of January 16, 2026

Abstract

We describe Network Swiss Army Knife (NSAK) as a modular, open-source, scenario-based network security framework, which is designed to be deployed on embedded hardware. We provide an overview of the system design, emphasizing the functionality of a Swiss Army Knife in a networking context. A NSAK scenario describes a common red or blue team activity for specific network environments and provides an automated and configurable implementation, based on multiple drills. Drills are the modular and configurable technical building blocks that can be used across multiple scenarios to configure the NSAK device or execute a specific task for reconnaissance or exploitation in the target network. In the analogy of the Swiss Army Knife, the right knife with the proper drills can be selected to solve a certain task. The NSAK framework is based on a core backend that implements an API to manage the framework configuration and resources such as environments, scenarios, and drills. The provided core-API is used by the CLI implementation to enable user interaction and could be used for other frontends and systems in the future. Another important concept of the framework is containerization, where each scenario can be built as an OCI container and executed on a provisioned NSAK device in a network environment. To prove that the concepts are working in the real world, we evaluated two different embedded ARM-based systems and executed experiments with the implemented scenarios in physical lab environments. The results of these experiments indicate that NSAK is well suited for security experimentation and controlled attack simulations on constrained systems. On the other hand, it showed that the automation of such scenarios requires a lot of assumptions about the target environment, which potentially renders them very brittle. The framework provides a practical foundation for network security research and can be extended to support automated testing and future attack scenarios.

Keywords: Modular Security Automation Framework, Network Reconnaissance, Exploitation, Red/Blue Team

Contents

1 Introduction

1.1 Introduction

According to the World Economic Forum's Global Risk Report 2025, the categories "Crime and illicit economic activity incl. Cyber" and "Cyber espionage and warfare" are both ranked among the top 10 global risks in the next two to ten years [1]. These risks are expected to intensify even further because the economic and operational costs of launching cyberattacks will decrease due to AI automation [2]. This underlines the need for cost-effective and easy-to-use security tools, methods and frameworks (conceptional and software) to identify and defend against cyberattacks.

Information security methods and conceptional frameworks
In practice, the method of combining red team activities and blue team observation techniques is widely adopted within the cybersecurity community and industry [3]. While the red team focuses on emulating adversarial behavior, the primary objective of the blue team is to detect such activities through non-invasive monitoring and analysis of system behavior [4]. Further, we can observe approaches from the community and the industry to evolve this approach in to the so-called "InfoSec color wheel" [3]. In the proposed InfoSec color wheel, the author splits the six colors into primary and secondary colors, where the primary colors are teams on their own and the secondary colors are cooperation between two primary color teams. The primary colors are represented by the red team, the Blue Team, and a newly introduced the Yellow Team, which represents the "builders" of software and systems. The secondary colors are represented by the purple team (red and blue), the Orange Team (red and yellow), and the Green Team (yellow and blue). Where "purple teaming" is actually an already established praxis as it evolved naturally from the cooperation between red and blue teams [4].

Information security tools and software framworks
One approach to reduce operational security costs is to adopt multiple modular frameworks that can be easily extended, configured, and executed continuously in a controlled manner [5]. The threat emulation frameworks analyzed by Zilberman et al. evaluate multiple attack phases, including lateral movement, persistence, and attack execution.

Network Swiss Army Knife (NSAK)

The Network Swiss Army Knife focuses on containerized, orchestrated scenarios that execute specific attack drills in a controlled environment. Future extensions will focus on enriching the assessment layer by systematically capturing and evaluating defensive responses of multiple scenarios.

This proof of concept comprises the design and implementation of a modular, isolated open-source security framework that focuses on extensibility and the controlled execution of attack-based scenarios.

The objective of this work is to investigate whether such a framework can provide a flexible, extendable, and safe foundation for modular and automated security testing in a network environment. .

In the summary of their paper, Zilberman et al. are highlighting the necessity of the following design requirements [5]:

- Cleanup and configurability are important in order to repeat and automate the execution of attack scenarios during security tool assessment and what-if analysis.
- An emulator should support cleanup after the completion of the attack scenario, like CALDERA, Atomic Red Team, and Infection Monkey do, rather than after each individual procedure.
- An API, currently provided by Atomic Red Team, CALDERA, and Metasploit, facilitates integration between the threat emulators and organizational security array, thus enabling periodic and systematic security assessment.
- It is important to provide a GUI and ready to execute multi-procedure attacks for novice operators as well as a CLI to support automation and advanced customization capabilities.

We reconsolidate the highlighted design requirements for the implementation of NSAK into the following list of features:

- CLI, GUI, and API to manage resources and execute scenarios.
- Configurability of the framework and the resources.
- Automatic cleanup procedures after the completion of a scenario.

Even though we agree with the importance of the highlighted design requirements, we are not able to implement them all in the time constraints of this project. Because we are planning to build upon this PoC, we will incorporate the design requirements in the chapter architecture and design 2 of this paper and list them in the future work 7.1 section.

1.2 Current State of Research

Recent studies highlight the importance of modularity, reproducibility, and automation to reduce the operational overhead of security assessments. Methodologies such as red and blue teaming, and their combinations within the InfoSec color wheel, show the complexity and overlapping disciplines in the security sector. [3, 4]

On a technical level, threat emulation frameworks such as CALDERA, Atomic Red Team, and Metasploit implement multi-stage attack scenarios to evaluate the detection of defensive systems. However, many existing frameworks focus primarily on large-scale enterprise environments and require significant setup effort, limiting their adaptability in resource-constrained networks. [5]

These aspects highlight the need for a lightweight and modular framework to reduce overall cyber thread risks in network infrastructures. [1]

2 Design and Architecture

2.1 Framework Concepts

This section describes the high-level concepts, resources and vocabulary needed to understand and work with the NSAK framework.

Overview of the NSAK resources and concepts:

- Devices
- Environments
- Drills
- Scenarios
- Operator
- Operation

2.1.1 Devices

Under a **device** we understand a physical or virtual machine, which is capable of running the NSAK framework. Even though we currently only work and describe the hardware devices evaluated in the section Hardware Selection 3.3, other devices or virtual machines could be used with NSAK.

The following list vaguely describes the minimum requirements for a device:

- Processor architecture: ARM and x86 should work equally well, as the NSAK framework is written in python and the scenarios are OCI images/containers, which are built on the NSAK device.
- Capable of running a Linux-based operating system, such as Debian.
- Enough memory and compute resources to run multiple OCI containers.
- Ideally, multiple physical network ports and Wi-Fi for covering many scenarios and environments.
- Optionally, additional bulk storage for data collection, such as PCAPs via T-Shark.

Provisioning a NSAK device usually consists of the following tasks:

1. Install and configure a Linux-based operating system
2. Set up a minimal network configuration and SSH access
3. Install system dependencies required for NSAK
4. Install and configure NSAK

After a device is provisioned, we refer to it as a **NSAK device**, which may or may not be prepared for an operation.

2.1.2 Environments

An **environment** is representing a specific network topology including infrastructure components, servers, clients and services. Ideally, an environment describes a part or a subset of a network and system infrastructure like you would encounter in a real organization.

Examples of environments:

- WLAN AP: Smartphone, WLAN AccessPoint, Router
- Client - server: Client, Server, Switch
- Home network: Router, WLAN, One Physical Network (Star Topology), Multiple Devices (Computers, Laptops, SmartPhones, SmartTVs)
- Business network: Firewall, Router, DC Server, Intranet, Multiple Subnets, Multiple WLAN Access points, Switches

2.1.3 Drills

A **drill**, initially called a module, is a sequence of actions with a specific goal. This goal can be an active or passive attack, network discovery, monitoring, analysis, data extraction, a hook for manual intervention or a device configuration.

Examples of drills:

- Network sniffing with TShark with a specific filter (http traffic)
- Data extraction on an internal bulk storage or external network file system
- Active or passive MITM (man in the middle) attack with a transparent TCP proxy
- ARP Spoofing
- WLAN SSID spoofing

- Network discovery with nmap or ARP-scan
- Network configuration, such as enabling IP-Forwarding or NAT

2.1.4 Scenarios

A **scenario** is designed for one or multiple environments, consists of a sequence of drills and describes a concrete use case for specific red or blue team activities.

Examples of Scenarios:

- WLAN SSID Spoofing:
  - Environment: WLAN AP
  - Drills: Network configuration for DHCP, NAT, SSID Spoofing, Packet Sniffing
- TCP MITM Attack:
  - Environment: TCP client - server
  - Drills: Automatic network discovery and configuration, ARP Spoofing, Transparent TCP Proxy, Packet manipulation

2.1.5 Operator

For simplicity and consistency we use the term **operator** for the person or team, which is planning and executing operations with NSAK. So an operator can refer to a single IT-specialist, a red, blue or purple team.

Examples of Operators:

- A single IT-Specialized or Security researcher
- System and network engineering teams
- Usually, red, blue and purple teams, but potentially all teams in the InfoSec color wheel [3]

2.1.6 Operation

An **operation** is the deployment of NSAK in a real network.

An operation explicitly excludes the development phase for scenarios, drills and environments, as these resources should be finalized and tested before being used in a real operation, otherwise the following conventions should be used:

- **Simulated Operation:** Simulating an operation in a virtualized environment.
- **Test Operation:** Testing an operation in a physical lab network.

Preparing and planning an operation usually has the following sequence of tasks, assessed and executed by an operator:

1. Provision a NSAK device.
2. Select one or multiple environments which are relevant for the target network and system infrastructure.
3. Configure and build all or a subset of scenarios which can be executed in the selected environments.
4. Ideally, simulate and test the operation in a virtual or lab network infrastructure.

2.2 Network Environments

Each environment represents a practical setup in which the NSAK device can be deployed. For traffic analysis, performance testing or security evaluation.

Category I — Inline: Diagram: Laptop ↔ NSAK device ↔ Router

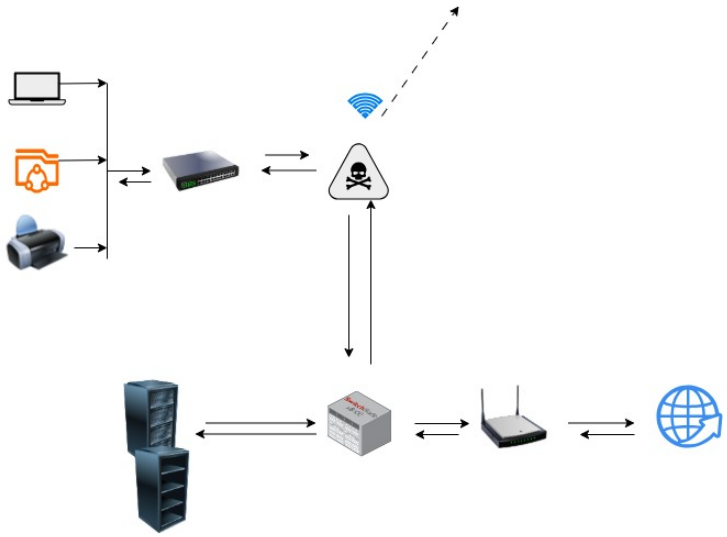

Figure 2.1: The network topology with inline NSAK-device

Description: Figure 2.1 shows direct inline bridge between a client or switch and router. Used for basic LAN capturing, latency, and throughput testing.

Category II — Wireless: Diagram: Laptop, Smart Devices, Printer ↔ NSAK device (inline) ↔ Router

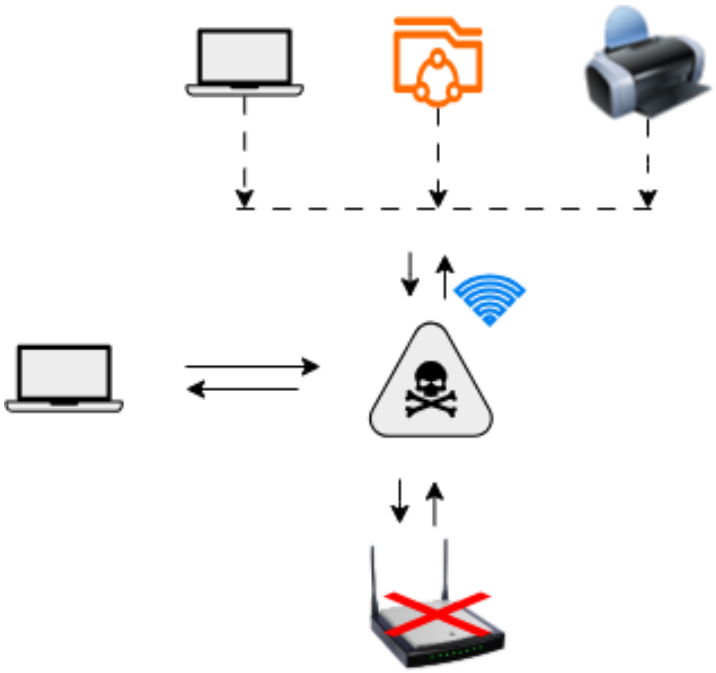

Figure 2.2: The network topology for RAP and Wifi scenarios

Description: The NSAK device is inline and lets traffic pass but intercepts as Rogue Access Point (RAP) and capture data Figure 2.2.

2.3 Use-Cases

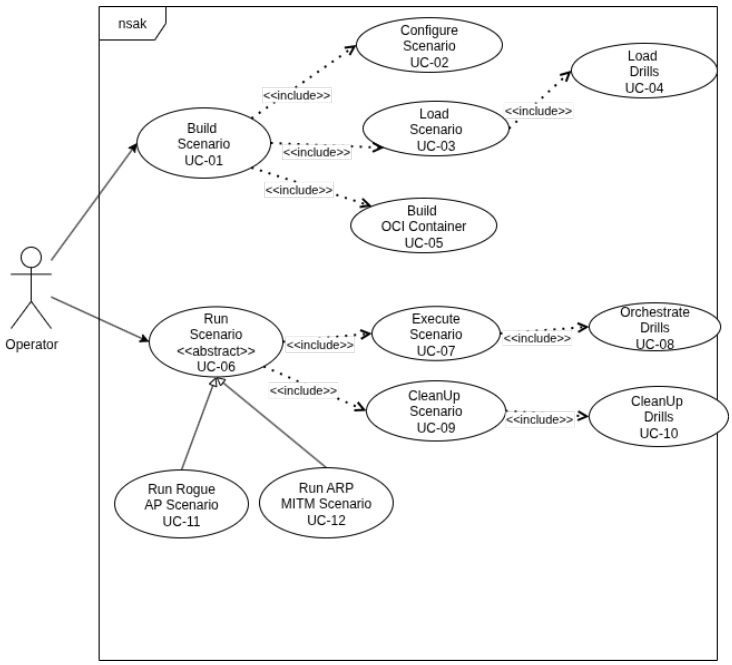

Figure 2.3: User-Story UML Diagram

Figure 2.3 illustrates the use case structure of the proposed NSAK modular framework. The operator interacts with NSAK primarily through two high-level commands: Build Scenario (UC-01) and Run Scenario (UC-06). During the Build Scenario use case (UC-01), the system builds a scenario container for execution. In complex network infrastructures, additional configuration parameters, such as network interface mappings may be required and need to be provided as build-time arguments (UC-02).

The system loads the selected scenario (UC-03). At this stage, the scenario orchestrates the required drills necessary to perform the intended attack.

The build process concludes with the creation of an OCI compliant container image (UC-05), which encapsulates the fully configured scenario.

The Run Scenario use case (UC-06) represents an abstract execution phase. In this phase, the previously built container image is run, and the configured attack drills are executed within the containerized environment (UC-08).

Finally, the system performs a cleanup procedure which needs to be optimized in future work, in which all scenario-specific resources, processes, and drills are terminated. This step minimizes side effects, reduces system noise, and prevents interference with other scenarios that may reuse the same drills.

Table 2.1: Use Cases Specification (NSAK)

NR & **Details**	
UC-01	**Use-Case:** Build Scenario **Description:** Builds a scenario container based on a selected scenario configuration. **Actor:** Operator **Trigger:** Operator initiates a scenario build via the command-line interface. **Preconditions:** NSAK initialized; scenarios available. **Main Scenario:** 1. Operator selects a scenario to build using the command-line interface. 2. System validates the selected scenario. 3. System executes the included use cases: • Configure Scenario (UC-2) • Load Scenario (UC-3) • Load Drills (UC-4) • Build OCI Container (UC-5) **Alternative Scenarios:** No scenarios available → inform the operator. **Error Scenarios:** Conflicting scenario configuration detected → build aborted. **Result:** Scenario container successfully built. **Postconditions:** Scenario container stored and ready to run.
UC-02	**Use-Case:** Configure Scenario **Description:** Defines scenario-specific build parameters such as network interfaces and execution options. **Actor:** System **Trigger:** Scenario selected for build (UC-01). **Preconditions:** Scenario selection is available. **Main Scenario:** 1. System applies scenario-specific configuration parameters. **Result:** Scenario configuration created. **Postconditions:** Scenario configuration available for loading.

NR & **Details**	
UC-03	**Use-Case:** Load Scenario **Description:** Loads and validate the selected scenarios **Actor:** System **Trigger:** Scenario configuration available (UC-02). **Preconditions:** Scenario configuration created. **Main Scenario:** 1. System retrieves the scenario definition files (scenario.yaml, scenario.py, README.md). 2. System validates the scenario structure and resolves declared dependencies. **Error Scenarios:** Validation or dependency failure, preparation aborted with Error Log. **Result:** Scenario is successfully loaded. **Postconditions:** Scenario representation available for drill loading.
UC-04	**Use-Case:** Load Drills **Description:** Loads the attack drills required by the selected scenario. **Actor:** System **Trigger:** Scenario loaded (UC-03). **Preconditions:** Scenario representation is available. **Main Scenario:** 1. System resolves drill references defined in the scenario configuration. 2. System instantiates drill objects and loads associated metadata. **Error Scenarios:** Invalid drill definition, drill not found, or ambiguous drill reference. **Result:** Required drill objects loaded. **Postconditions:** Drills available for container build.

NR & **Details**	
UC-05	**Use-Case:** Build OCI Container **Description:** Builds an OCI compliant container image for the loaded scenario. **Actor:** System **Trigger:** Scenario and drills loaded (UC-03, UC-04). **Preconditions:** Scenario representation and drill objects available. **Main Scenario:** 1. System generates the container build context. 2. System builds the scenario container image with required privileges and network configuration. **Error Scenarios:** Container build failure — build aborted with an error message. **Result:** OCI compliant scenario container image built. **Postconditions:** Scenario container image stored and ready for execution.
UC-06	**Use-Case:** Run Scenario **Description:** Executes a previously built scenario container. Specific scenarios such as Rogue AP or ARP MITM represent specialized configurations of this use case. **Actor:** Operator **Trigger:** Operator initiates scenario execution via the command-line interface. **Preconditions:** Scenario container image available (UC-05). **Main Scenario:** 1. System starts the scenario container with the required execution parameters. 2. System executes the included use cases: • Execute Scenario (UC-07) • Cleanup Scenario (UC-09) **Result:** Scenario container execution started. **Postconditions:** Scenario execution context active.

NR & **Details**	
UC-07	**Use-Case:** Execute Scenario **Description:** Orchestrates the execution of a previously built scenario container and coordinates the execution of the associated attack drills. **Actor:** System **Trigger:** Run Scenario (UC-6) **Preconditions:** Scenario Image available and started **Main Scenario:** 1. System Scenario Manager executes for the selected scenario 2. System Drill Manager executes drill UC-8 include use-case **Error Scenarios:** Scenario not found or scenario container was not available. **Result:** Scenario execution initiated and drill execution orchestrated. **Postconditions:** Scenario container is running and drills are being executed.
UC-08	**Use-Case:** Execute Drills **Description:** Executes the attack drills defined in the scenario configuration within the running scenario container. **Actor:** System **Preconditions:** Scenario execution context initialized. **Main Scenario:** 1. System Drill Manager retrieves the list of configured drills. 2. System Drill Manager executes the drills according to the defined order and parameters. **Error Scenarios:** Drill execution failure or missing drill definition. **Result:** Configured attack drills executed.

NR & **Details**	
UC-09	**Use-Case:** Clean Up Scenario **Description:** Terminates the running scenario container and restores the system to a defined baseline state. **Actor:** System **Trigger:** Stop Scenario (UC-06) **Preconditions:** Scenario container is running. **Main Scenario:** 1. System stops the running scenario container. 2. System invokes the included use case Clean Up Drills (UC-10). **Error Scenarios:** Scenario container cannot be terminated. **Result:** Scenario execution terminated. **Postconditions:** Scenario container stopped and removed.
UC-10	**Use-Case:** Clean Up Drills **Description:** Cleans up artifacts and state changes introduced by executed attack drills. **Actor:** System **Preconditions:** Drill execution completed or aborted. **Main Scenario:** 1. System Drill Manager terminates active drill processes. 2. System Drill Manager removes temporary artifacts and resets modified parameters. **Error Scenarios:** Incomplete cleanup due to failed drill termination. **Result:** Drill-related artifacts removed and state reset.

2.4 Component-diagram

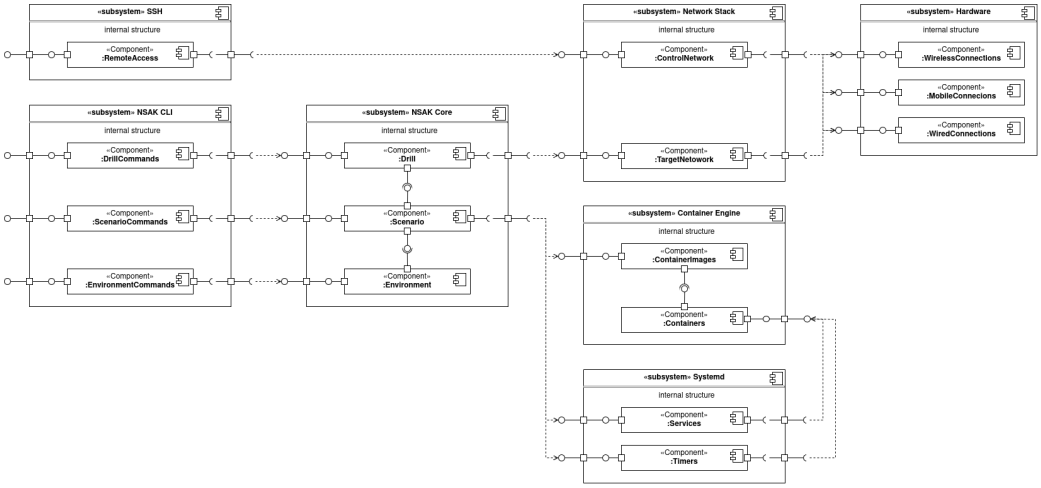

Figure 2.4: The Component diagram illustrated how the systems interact which each other

The component diagram (Figure **2.4**) illustrates how the different subsystems and their respective components should interface each other. The **Network Stack** subsystem abstracts away the **hardware** layer and provides a **ManagementNetwork** and **TargetNetwork** component. The components **WIFIConnection**, **EthernetConnection** and **MobileConnection** are physical interfaces in the **Hardware** subsystem. Operators would connect to the NSAK device via **SSH** over the **management network** to open a remote shell. Afterward, they use the **NSAK CLI**, which exposes **DrillCommands**, **ScenarioCommands** and **EnvironmentCommands**. These commands interact with the API provided by the **NSAK core**, which is structured into a management component per resource type: **Drills**, **Scenarios** and **Environments**. In the **NSAK core** subsystem we can see how the management components are responsible for their respective resources and interfacing with each other. On the right side of we can see how the resources are interacting with the broader system context:

- **Drills:** Ability to access the **TargetNetwork** via the **NetworkStack** subsystem.
- **Scenarios:** Responsible for interacting with the **Container Engine** and **SystemD** subsystems.
- **Environments:** Interfaces with other subsystems on the right side, but provides important information for the scenarios.

As scenarios are built as OCI images, we need an **OCI Container Engine** subsystem consisting of the **OCIImage** and **OCIContainer** components. While planned

and already displayed in the diagram, the **SystemD** subsystem is not yet integrated into the framework. As most Linux distributions already ship with SystemD, it would make sense to implement the possibility to provide unit files to schedule and manage scenarios with the **Timer** and **Service** components respectively.

2.5 Sequence-Diagram

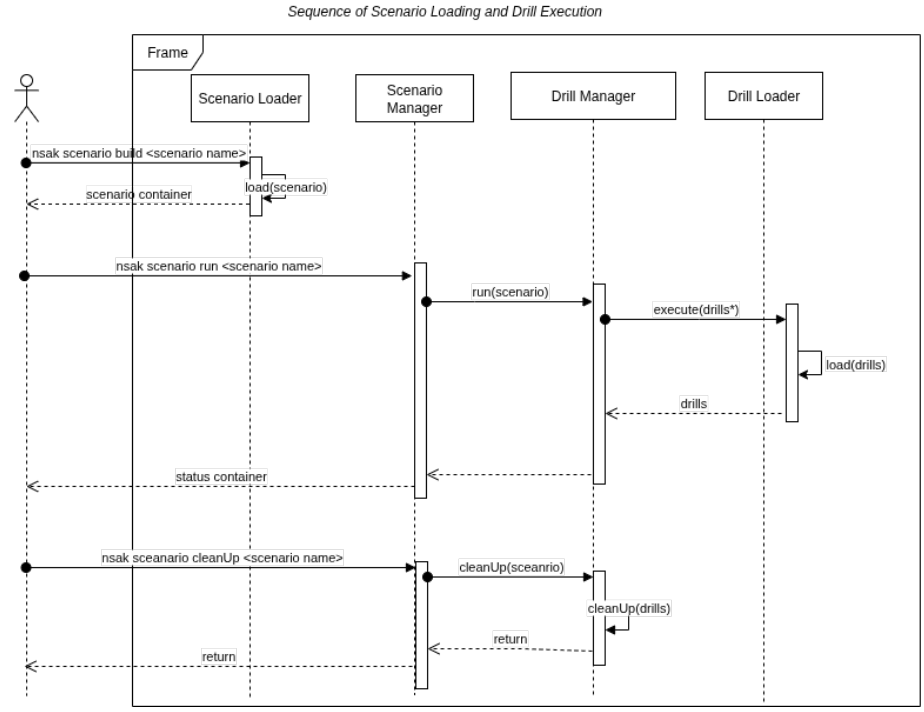

Figure 2.5: Sequence of a Scenario loading and drill execution

Figure 2.5 shows the interaction sequence for building, loading, and executing a scenario within NSAK. The diagram focuses on the main modularity concept and describes the orchestration flow between scenarios and drills without interface details, error-handling, and drill or scenario clean-up mechanisms.

The process begins with the operator triggering the build command for a scenario container. During this phase, the selected scenario is loaded and returned as a containerized representation. In the execution phase, the scenario manager runs the container image and orchestrates the Drills order. The Drill Manager executes the required drills.

A scenario may contain multiple drills; therefore, the * signalize various drills can be executed from a Drill Manager in a one scenario. Each drill is resolved and executed individually, while the scenario manager maintains complete control over the scenario lifecycle.

Finally, each drill is intended to provide a clean-up procedure that can be triggered from the scenario manager. All drill-specific artifacts should be removed from the host system to enable the sequential execution of multiple scenarios. The clean-up process is a planned extension of the framework.

3 Hardware Evaluation

3.1 Hardware Requirements

Based on the vaguely described minimum requirements in the section Devices 2.1.1, we defined the following requirements for the hardware platform used in this project:

- At least two native Ethernet interfaces for inline packet sniffing
- Support for 2.5 GbE or higher
- Onboard Wi-Fi with access point (AP) and monitor mode support
- Low power consumption suitable for 24/7 operation
- Compact form factor for laboratory and prototype setups
- Strong community and software support
- Affordable cost (below 150 CHF)

3.2 Evaluated Boards

Several boards were considered as potential variants. Their main specifications relevant to the project are listed in Table 3.1.

Table 3.1: Comparison of Board Variants

Board	**SoC / CPU**	**RAM / Storage**	**Ethernet Ports**	**Power (typ.)**	**Wireless (onboard)**
Banana Pi R3 Mini	MT7986A, Quad-core ARM Cortex-A53 @ 1.3 GHz	2 GB DDR4, 8 GB eMMC, microSD	2 × 2.5 GbE	5–7 W	MT7976C, Wi-Fi 6 (AP/Client/Monitor)
Banana Pi R3	MT7986A, Quad-core ARM Cortex-A53 @ 1.3 GHz	2–4 GB DDR4, eMMC, microSD	1 × 1 GbE, 2 × 2.5 GbE, 4 × 1 GbE	7–10 W	MT7976C, Wi-Fi 6
Banana Pi R4	MT7988A, Quad-core ARM Cortex-A73	4 GB DDR4, NVMe option	4 × 2.5 GbE, 2 × 10 GbE (SFP+)	10–15 W	None (M.2 Wi-Fi module required)
Banana Pi R5	MT7988B, Quad-core ARM Cortex-A73	4 GB DDR4, NVMe option	2 × 10 GbE, 2 × 2.5 GbE	12–18 W	None (M.2 Wi-Fi module required)
Raspberry Pi 4	BCM2711, Quad-core ARM Cortex-A72 @ 1.5 GHz	2–8 GB LPDDR4, microSD	1 × 1 GbE (second via USB dongle)	6–8 W	Wi-Fi 5 (AP/Client only)
Raspberry Pi 5	BCM2712, Quad-core ARM Cortex-A76 @ 2.4 GHz	4–8 GB LPDDR4X, microSD	1 × 1 GbE (second via PCIe card)	8–12 W	Wi-Fi 5 (AP/Client only)
NanoPi R76S	Rockchip RK3588S, Octa-core (4× Cortex-A76 @ 2.4 GHz + 4× Cortex-A55 @ 1.8 GHz)	16 GB LPDDR4X / LPDDR5, NVMe (option via M.2)	3 × 2.5 GbE (RJ45)	10–15 W	None (M.2 Wi-Fi 6E module recommended)

3.3 Hardware Selection

Based on the defined requirements and the evaluation of alternatives, compared in table 3.2, the **Banana Pi R4** and the **NanoPI R76S** are the most suitable hardware platforms for this prototype implementation.

Table 3.2: Requirements Fulfillment by Candidate Boards

Requirement	R3 Mini	R3	R4	R5	RPi 4	RPi 5	NanoPi R76S
≥2 native Ethernet interfaces	✓	✓	✓	✓	✗	✗	✓
RAM > 4GB	✗	✗	✓	✓	✗	✓	✓
≥ 2.5 GbE support	✓ (2×)	✓ (2×)	✓✓ (4×)	✓ (2×)	✗	✗	✓
Onboard Wi-Fi with AP & Monitor mode	✓	✓	✗	✗	✗	✗	✗
Low power consumption (<10 W)	✓	✓/▲	✗	✗	✓	▲	✓
Compact form factor	✓	✗	✗	✗	✓	✓	✓
Strong community & software support	✓	✓	▲	▲	✓ (general)	✓ (general)	✓
Suitable for inline packet sniffing	✓	✓ (overkill)	▲ (overkill)	▲ (expensive)	✗	✗	✓

Legend: ✓ = Requirement fulfilled, ✗ = Requirement not fulfilled, ▲ = Partially fulfilled / limited

The **Banana Pi R4** offers two native 2.5 GbE interfaces for inline sniffing the board is compact, affordable, and supported by a strong community. In Addition, the two 10 GbE SFP+ ports provide flexibility for extensions as fiber-based packet capturing. A drawback of the R4 is the weaker CPU and a larger size compared to the NanoPI R76S

The **NanoPi R76S** is more compact and provides up to 16GB of RAM, which is advantageous for memory-intensive processing and buffering tasks. While it lacks built-in Wi-fi, it can be expanded via the M.2 Wi-Fi 6E module. It cannot host both a Wi-Fi card and NVMe SSD simultaneously. Consequently, data storage must be provided via microSD card or external USB SSD

Alternative boards such as the **Banana Pi R3 Mini**, R3 are limited overall performance. **PI 4 or 5** offer higher single core performance but were ultimately discarded because they provide only a single native Ethernet interface, requiring external adapters that reduce performance for inline sniffing scenarios.

4 Methods

4.1 Research Approach

This work follows a design-oriented research approach and presents a proof of concept of a modular network sniffing framework named NSAK (Network Swiss Army Knife). The objective is not to introduce a new type of network attack techniques, but to design and implement a modular framework that enables reproducibility and encapsulation in network security systems.

The Swiss Army Knife inspires the conceptual design of the NSAK device: Instead of providing a single-purpose tool, the framework offers multiple small specialized components. that can be used depending on the operation. For the NSAK device, the operational environment is the network. Situations are represented as scenarios, and Individual tools for performing a task are implemented as drills.

The research is primarily based on existing scientific literature, including journal articles, conference papers, and established open source networking tools. The focus lies on system integration, modularization, architectural design, reproducibility, and experimental validation rather than theoretical innovation.

4.2 System Design Strategy

The NSAK framework is structured in three main layers: a core backend, a CLI package, and a library package. The NSAK device comprises three components: environments, scenarios, and drills. The library provides reusable, small-component packages that are used by the core's central logic. loads, manages, and executes. The click CLI provides all the user handling over the command line.

Scenario Oriented Orchestration Scenarios are responsible for orchestrating drills and defining the drill order in which they are executed. Therefore, it needs specific parameters to be passed to the drill. Finally, the scenario is responsible for managing the cleanup process of the drills.

Each scenario is designed to run inside a containerized environment, to ensure reproducibility and isolation. While the scenario runs in the container, the drills it orchestrates execute privileged operations on the host system.

A scenario consists of:

- a scenario.py file containing the orchestrating scenario
- a scenario.yaml file describing metadata and dependencies
- and a README.md file providing configuration, tips, and documentation

Modular Drill-Based Architecture A drill represents the smallest functional unit within the NSAK framework. Each drill is responsible for a specific task.

A drill consists of:

- a drill.py file containing the execution and cleanup logic
- a drill.YAML file describing metadata
- and a README.md file providing configuration, tips, and documentation

By design, drills are independent, allowing them to be reused across multiple scenarios. This modularity enables flexible composition and contribution while keeping the components focused and straightforward.

Experimental Setup

The experimental setup was conducted on an arm-based embedded system equipped with a wireless interface. The following criteria were used to assess the framework:

- successful execution of individual drills,
- correct orchestration of multiple drills within a scenario,
- and reproducibility of experimental results.

The evaluation demonstrates that the NSAK framework enables structured, modular, and repeatable experimentation in network security research environments.

The evaluation of the ARP MITM Scenario requires a controlled test environment consisting of a Layer 2 network switch and cables, 2x Raspberry Pi: Alice (Client) and Bob (Server), Banana PI R4 or Nano PI: Malcom (NSAK) and three SD Cards for the operating systems

The evaluation of the Rogue Access Point scenario requires a controlled test environment consisting of a gateway host system, an embedded NSAK device (NanoPi or Banana Pi R4), and multiple Wi-Fi client devices, including tablets, laptops, or smartphones.

Delimitation This work does not aim to evaluate attack success rates in real-world environments. The focus is limited to architectural design and functional validation.

4.3 Project Management

The development process used GitLab for version control and issue tracking. An issue board was used to structure development tasks, track progress, and enable the project for future contributions and further development. This approach improves traceability and enables the review of design decisions in the repository.

5 Implementation

In this chapter we will describe what and how we implemented the NSAK framework, the first two scenarios, their drills and environments. To efficiently describe the implementation, we reference paths and files relative to the repositories root directory [6]. Additional technical information about the framework installation and development setup can be found in the `README.md`.

5.1 NSAK framework

In this section we describe which technologies we choose to implement the NSAK framework and the resource library.

As this project is a so-called monorepo, the logical primary driver for the technology stack and dependencies is the core component. The following system components such as the resource library and the CLI will inherit the technology stack, the python, and system dependencies. For this reason, the components following the core will only describe what they introduce additionally.

Core
We did not make an extensive evaluation of which technologies are best suited for building developing the software and decided to use to go with tools that we like and are familiar with. The location of the core in the GIT repository is under `src/nsak/core`, there are files for setup and configuration, a subfolder for network utilities and for each resource type. All resource subfolders contain files for a dataclass representing the resource itself, a loader, and a manager class.

- Programming language: Python [7]
- Dependency manager: uv [8]
- Virtual environment manager: uv [8]
- Package build tool: uv [8]
- Linter: ruff [9]
- Formatter: ruff [9]
- Type checker: mypy [10]

- Testing framework: pytest [11]

Python dependencies:

- pyyaml: Library for loading, validating, and reading YAML files [12], [13]
- scapy: Library for red team operations [14]
- pre-commit: Package to enforce code quality tools for each commit [15]

System Dependencies
As we leverage the abstraction of OCI containers to run scenarios in an encapsulated environment, we have only a minimal set of system dependencies. All system dependencies that are required for running a drill or a scenario are installed into the scenario image, during the build process.

- Version control: git [16]
- Network tooling: iptables, nftables [17], [18]
- OCI container manager: podman [19]
- OCI container orchestrator: podman-compose [20]
- Programming languages: python3, pip, uv [7], [21], [8]
- Utilities: curl, sudo [22], [23]

Resource Library
All resources included in the monorepo are located in their respective resource type subfolders under `lib/{drills,scenarios,environments}`. This path can be adjusted with the environment variable `NSAK_LIBRARY_PATH` if a framework user or developer wants to load his own resource library. In the additional `devices` subfolder are instructions to set up a specific NSAK device, currently this resource type is not implemented in the core and only contains one description for the Banana PI R4 as an example. Resources with code implementations, namely scenarios and drills, manage their own python and system dependencies in their YAML files.

CLI
The CLI frontend is located under `src/nsak/cli` with a file per-resource type, implementing and exposing the functionality to the user. The steps required to enable autocompletion for some shells are described in the `README.md`. Here is a list of common commands:

- `nsak --help`: A description of all commands
- `nsak <resource-type> --help`: A description of the subcommand
- `nsak environment list`: Lists all described environment resources
- `nsak drill list`: Lists all implemented drills

- `nsak scenario list`: Lists all available scenarios
- `nsak scenario build <scenario-name>`: Build the OCI image for the scenario
- `nsak scenario run <scenario-name>`: Run a scenario as a OCI container

Python dependencies:

- click: Library for building CLIs [24]

5.2 MITM (ARP-spoofing / transparent TCP Proxy Server)

Network Topology / Environment

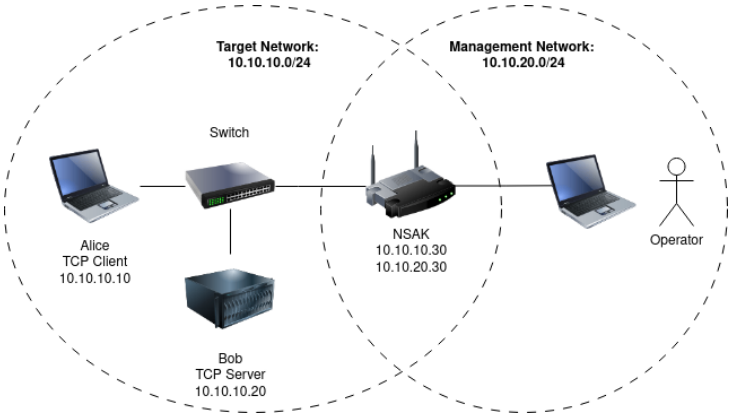

Figure 5.1: MITM scenarios in the simple TCP client server environment

Figure 5.1 illustrates the network topology used for the Man-in-the-Middle (MITM) scenario. It consists of two dotted circles separating the network boundaries of two networks. The left circle is describing a simple TCP client - server environment consisting of Alice (client), Bob (server), a network switch, and the NSAK device which was placed in the target network. The right circle is sketching the management network, which is used by a red team operator to remotely execute the MITM scenario on the NSAK device.

MITM scenario implementation
This section describes the implementation and simulation of the MITM scenario. It represents a classic TCP Man-in-the-Middle (MITM) attack, where traffic is intercepted with ARP poisoning and a transparent TCP Proxy Server. After an initial network discovery phase, the scenario creates a transparent Proxy Server by starting a TCP client and TCP server and redirecting traffic. Then it's time to use MAC Address spoofing to convince all other network participants that they

should send all packets to the NSAK device. If everything worked correctly, the NSAK device is able to read, modify, or drop packets sent from one participant to another.

I choose the following separation of the scenario, which describes the drills, which have to be implemented:

- Network Hosts Discovery: `lib/drills/discover_hosts`
- Transparent TCP Proxy Server: `lib/drills/transparent_tcp_proxy`
- ARP-spoof: `lib/drills/arp_spoof`

Drills
The **Discover Host Drill** iterates over all target network interfaces and uses ARP-scan to collect a list of all hosts in all subnets. While this approach is certainly good enough for the PoC, it could possibly take a lot of time and yield huge lists of hosts. So it would make sense to provide further configration options such as subnet filters or limits. In the context of this drill, a new dataclass `NetworkDiscoveryResultMap` was added to the core to provide a common interface for similar network discovery drills and drills requiring such results. The drill then returns an instance of the `NetworkDiscoveryResultMap` dataclass.

The **Transparent TCP Proxy Server** requires a `NetworkDiscoveryResultMap` instance, e.g., generated with the host discovery drill. It starts a transparent TCP proxy for each client - server connection in the `NetworkDiscoveryResultMap` and sets up the necessary target network configuration with iptables. Then the drill reads and modifies the intercepted traffic, which is the goal of a MITM attack. The packet manipulation is currently only a simple string replacement, it would be a nice future work to make it possible to specify a custom payload. Maybe it could make sense to split up this drill even further, as maybe as TCP client, server, or the network interface configration could be reused in other drills. The **ARP-spoof Drill** also requires a `NetworkDiscoveryResultMap` instance, which provides target network interfaces and subnets to iterate over. It then uses the “arpspoof” command to poison the ARP tables of all target hosts.

Simple TCP client - server environment
For development and testing purposes the simple TCP client - server environment was added as a resource to the NSAK repository. The resource includes a virtualized lab setup with a podman/docker compose file and provides an extensive readme file, describing the setup of this environment as a physical lab environment.

This physical lab setup builds the basis for the experiment, verifying that the MITM scenario works in a real-world environment. The compose file provides a possible representation of the target network with containers acting as Alice and Bob. The containers entrypoints are two Python scripts, which are implementing

the behavior as a TCP client for Alice and as a TCP server for Bob. Because the default network driver used by podman and docker compose is abstracting away the data link of the OSI model, the configuration of the macvlan network driver is required to simulate the network behavior on layer 2 correctly. The MITM scenario resource contains a subfolder containing another readme file, describing its integration in this physical lab environment. A video demonstrating this physical setup in a real-world experiment can be found in this repository. This subfolder also provides a podman/docker compose file which extends the compose file of this resource, providing a complete setup for running the scenario in a simulation. The containerized simulation can be executed with `nsak environment simulate simple_tcp_client_server mitm`. The NSAK framework was then extended with the ability to simulate a scenario in a compatible environment which provides a podman/docker compose implementation.

References:

- Simple TCP Client - Server Environment:
  - Folder: `lib/environments/simple_tcp_client_server`
  - Podman compose file: `simple_tcp_client_server/compose.yaml`
  - Readme file: `simple_tcp_client_server/README.md`
- MITM Scenario in the Smiple TCP Client - Server Environment:
  - Subfolder: `lib/scenarios/mitm/environments/simple_tcp_client_server/`
  - Podman compose file: `simple_tcp_client_server/compose.yaml`
  - Readme file: `simple_tcp_client_server/README.md`
- Demo video: `docs/presentation/assets/nsak_mitm_scenario.mp4`

5.3 Rogue Access Point

Rogue Access Point

Network Topology

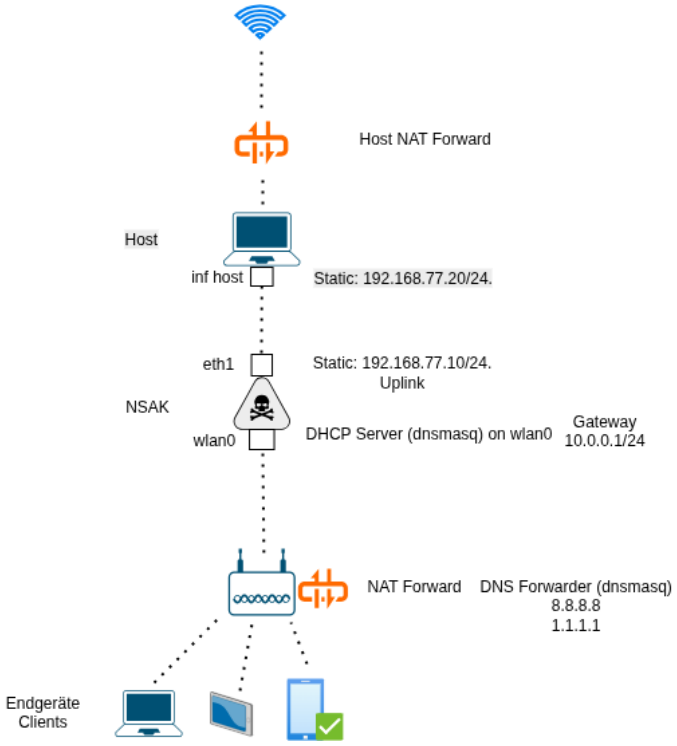

Figure 5.2: Network set up RAP

Figure 5.2 illustrates the network topology used for the Rogue Access Point scenario. The NSAK device is positioned between the wireless clients and an upstream host system acting as an internet gateway. Two network interfaces are used on the NSAK device: a wireless interface (wlan0) operating in access point mode, and a wired uplink interface (eth1) connected to the host system.

Rogue Access Point Implementation This section describes the technical realization of the Rogue Access Point scenario within the NSAK framework. The focus lies on the integration of wireless access point functionality, traffic forwarding, and packet capture.

Within the NSAK framework, the Rogue Access Point scenario is implemented as a composition of multiple drills. Each drill encapsulates a single operational responsibility, allowing the scenario to orchestrate the drills separately, ant to remain modular and extensible.

Scenario The Rogue Access Point scenario consists of several drills that are executed sequentially to establish a functional wireless access point capable of intercepting and forwarding network traffic.

- Network interface preparation
- Wireless access point initialization

- Traffic forwarding and network address translation
- Packet capture and monitoring

Drills

The Hostapd Drill is responsible for configuring the wireless network interface of the NSAK device in access mode. This includes assigning network parameters to the interfaces and enabling beacon transmission to allow the client devices to connect to the rogue access point

The access point functionality is implemented using standard Linux networking services running in an isolated subprocesses. The controlled interaction with the operating system allows reliable startup and shutdown behavior. Furthermore, the current process state can be tracked.

Traffic Forwarding and Network Integration drills are providing network connectivity for clients. The NSAK device establishes an uplink connection to an external network interface. Traffic forwarding is enabled between the wireless and uplink interfaces, enabling transparent internet access.

Network address translation and packet forwarding are configured dynamically during scenario execution. This enables the NSAK device to operate as an intermediary between wireless clients and the upstream network.

In parallel, a **traffic capture drill** on the connected interface captures traffic passing through. The pcap files can be used for later analysis, enabling the evaluation of client behavior and the network. interactions.

By separating packet capture into an independent drill, the framework allows traffic monitoring to be reused across different scenarios without modification.

The scenario manager orchestrates the execution of all drills involved in the Rogue Access Point scenario. Drills are executed in a predefined order, ensuring that the required network services are available before dependent components are started.

Error handling and cleanup
To prevent persistent system modifications, each drill defines a cleanup routine that can be executed after scenario completion or upon failure. This ensures that network interfaces and system services are restored to their original state. In the current state of the POC the cleanup functionality need to be adjusted for the broad diversity of the drills and covers momentarily not all possible edge cases.

But as mentioned in, a centralized cleanup mechanism ensures that partial execution states do not persist in the system in an inconsistent configuration. And helps to prevent uncontrolled behaviors of drills.

This section focused on the technical realization of the Rogue Access Point scenario. The effectiveness and behavior of the scenario under real network condi-

tions are evaluated in the subsequent evaluation chapter.

6 Evaluation and Discussion

6.1 Evaluation

6.1.1 Generel Functionality of a Rogue Access Point

The Rogue Access Point scenario was executed with different combinations of drills to test the modularity and the independent drill execution. The expected outcome is the creation of an open Wi-fi access point broadcasting the configured SSID, providing DHCP leases and forwarding traffic to the uplink interface. Additionally, traffic will be captured in PCAP format.

During execution, the access point was successfully created, and client devices could connect using the assigned IP address. via DHCP. The Network traffic was captured in a PCAP file.

6.1.2 Research Questions

RQ1 – Modularity RQ1: Can network security scenarios be composed of independent drills without modifying the core framework?

To ensure modularity, every drill should be able to work independently. When setting the DISABLE-DRILL environment variable, remaining drills should execute independently without failure. For example, if everything is disabled beside the ap-mode drill, the network should still be discoverable from clients. In T4 6.1, IP forwarding, uplink, and capture should not be executed and therefore are not working.

During the process, each drill has a specific attack goal or functionality. No implicit dependencies between drills were observed during execution.

Test	Description	Expectation
T1	Execute Rogue AP scenario with full drill set	Scenario executes successfully
T2	Deactivate packet capture drill	AP still functions
T3	Deactivate dnsmasq drill	All drills are working, tshark cannot capture traffic
T4	Only the AP-mode drill is enabled	Client can see the SSID and connect

Table 6.1: Modular execution test cases

RQ2 – Reusability RQ2: Can individual drills be reused across different scenarios with minimal configuration effort?

The hostapd drill was reused in two scenarios: a standalone access point and a rogue access point. access point scenario. In both cases, the drill implementation remained unchanged. Scenario-specific behavior was controlled through environment variables such as SSID, channel, and country code.

These results confirm that drills are scenario-orchestrated and can be composed flexibly with minimal modification.

RQ3 – Containerization RQ3: Does container-based execution provide sufficient isolation while maintaining required network functionality?

The evaluation focuses on whether container-based execution restricts or enables low-level network operations, required for the functionality of NSAK.

The Rogue Access Point scenario was executed inside a containerized NSAK environment on an embedded Linux device. The container was granted only the necessary capabilities for network configuration and packet capture.

During scenario execution, all network services were started in privileged mode within the container environment. During container-based execution, network interfaces, routing rules, and packet capture were successfully initialized. The drills ap-mode, network setup, dnsmasq, t-shark left certain config files and rules in iptables and filepaths.

RQ4 – Reproducibility RQ4: Can scenarios be executed repeatedly without changing the system behavior?

With the same configuration, the scenario execution was triggered multiple times. Devices could establish a connection with the access point. Network connectivity was enabled in all runs. And the NSAK framework was able to capture the traffic.

No deviations in system behavior were observed across repeated executions.

6.2 Discussion

This section discusses the results of the evaluation and the research questions defined in Section 6.1.2

Modularity of Drills (RQ1) 6.1.2

The evaluation demonstrates that network-composed scenarios can be built from different combinations of drills without requiring modifications to the core framework. All tested drill combinations executed successfully. Disabling individual or multiple drills did not cause unintended side effects and unwanted system behaviors. This indicates the loose coupling between drills.

However, the observed modularity is currently limited to the tested scenarios. More complex scenarios that target a more demanding network may introduce unintended dependencies that lead to unwanted behaviors, which should be investigated more thoroughly in future work.

Reusability Across Scenarios (RQ2) 6.1.2

The reuse of different drills in multiple scenarios suggests that drills can be reused with minimal configuration effort. Scenario-specific behavior was controlled exclusively through environment variables and build arguments. Drill implementation remained unchanged. This demonstrates the assumption that the NSAK framework is flexible and be orchestrated from a scenario perspective. The minimal configuration helps reduce complexity for network security teams. The flexibility between the inline and Wi-Fi attack scenarios demonstrates the spectrum and potential of network attacks.

Reusability was evaluated across a limited number of scenarios and drills. Further evaluation across a wider range of scenarios is required to validate in-depth.

Containerization and Isolation (RQ3) 6.1.2

The isolated scenario container can be pre-built and run independently with an individual drill set. The container execution in the test scenario was triggered solely from the command line, but could also be called similarly from a system process. This enables the NSAK device to remain latent in a network until the attack is most likely to succeed or to have the greatest impact.

At the same time, the evaluation revealed the necessity to manipulate the host-network configuration, such as Iptables rules and temporary files were not fully cleaned up after scenario execution. This highlights the importance of an improved cleanup mechanism to ensure isolation and reproducibility in repeated execution. The running containers are constantly running, increasing the likelihood of detection. In a more sophisticated approach, the NSAK device should be as stealthy as possible to remain hidden from blue teams.

Reproducibility or Scenarios (RQ4) 6.1.2 Repeated execution of identical scenarios resulted in consistent system behavior. This suggests that the current framework and implementation provide sufficient security reproducibility. experiments.

However, the reproducibility was assessed over a limited number of runs and devices. Long-term testing would be required to evaluate stability.

7 Future Work and Conclusion

7.1 Future work

While the presented proof of concept demonstrates the modularity and scenario-based network security framework, several aspects require extension and investigation.

REST API
A key area for future work is the introduction of a RESTful API that provides parity with the existing command-line interface. Such an HTTP-based API would enable the interoperability with many other systems and the addition of alternative frontends to the CLI. A good option which would integrate nicely into the current stack would be FastAPI [25].

Graphical User Interface (GUI)
Based on the proposed REST API, a browser-based GUI would provide a lower entry barrier and improved accessibility for a wider range of users and could also help in educational settings, security training and research labs. We thought of [26] as a possible library to implement this interface, but other technologies would be equally valid.

Cleanup Management
The evaluation revealed limitations in the current cleanup procedure. Temporary configuration artifacts and network rules were not removed, which affects the successive usage of multiple scenarios. Furthermore, the device would be more likely to be detected.

Configuration Management
We identified the possible brittleness of scenarios in different environments, as we have to make a lot of assumptions when designing a scenario or writing a drill. Implementing a well-designed configuration management for scenarios and drills could enable the operator to set up a scenario in a much flexible way and maybe also allow autoconfiguration for adoption during runtime. It would also render the drills much better suited for implementation in a wider range of scenarios.

Test Coverage
Increasing test coverage is essential to validate the correctness and stability of

the framework as it evolves. Automated testing would improve reliability and support long-term maintainability.

Advanced Scenario Management
As already proposed in the component diagram 2.4 the integration of SystemD capabilities for scenarios via unit files, would enable more complex management and scheduling of scenarios.

Automated Reporting and Analysis
Finally, integrating automated reporting and analysis of defensive responses would enable more comprehensive blue- and purple-team assessments.

7.2 Conclusion

This thesis presents a conceptual design and implementation of the Network Swiss Army Knife (NSAK). The NSAK framework is modular and scenario-based security software for controlled network security and experimentation.

The proposed architecture combines container-based isolation with reusable drills. These drills can be composed into high-level scenarios. The implementation demonstrated the core design objectives of the framework, and the evaluation emphasizes the modularity, reproducibility, and container-based isolation. Independent drills were successfully orchestrated in two differential environments, in line, on the data link layer, with an ARP spoofing MITM scenario and over a Wi-Fi network interface to capture traffic as a rogue access point.

The results indicate that the NSAK framework can reduce setup complexity for network security experiments while maintaining a high degree of flexibility.

At the same time, this work is merely a proof of concept with clear limitations. Supporting more sophisticated attack scenarios in complex networks requires further refinement and extension. Further improvements include advanced cleanup mechanisms, richer configuration models, and automated assessment of defensive responses.

In conclusion, the NSAK framework provides a promising foundation stone for future extension and open source contribution. The independent core backend and the isolated framework structure are supporting developing encapsuled drills for embedded devices without compromising other devices.

Declaration of Authorship

I hereby declare that I have written this thesis independently and have not used any sources or aids other than those acknowledged.

All statements taken from other writings, either literally or in essence, have been marked as such.

I hereby agree that the present work may be reviewed in electronic form using appropriate software.

January 16, 2026

L. von Allmen

F. Gauss

Bibliography

[1] World Economic Forum. The global risks report 2025, 2025.

[2] Vaibhav Garg and Jayati Dev. Artificial intelligence and the new economics of cyberattacks. USENIX ;login: online article, August 2024. Article shepherded by Rik Farrow.

[3] Louis Cremen. Introducing the infosec colour wheel — blending developers with red and blue security teams. https://hackernoon.com/introducing-the-infosec-colour-wheel-blending-developers-with-red-and-blue-securit November 2018. Accessed: 2025-12-30.

[4] National Institute of Standards and Technology. Red team/blue team approach. https://csrc.nist.gov/glossary/term/Red_Team_Blue_Team_Approach, 2012. Accessed: 2025-12-29.

[5] Polina Zilberman, Rami Puzis, Sunders Bruskin, Shai Shwarz, and Yuval Elovici. Sok: A survey of open-source threat emulators. arXiv preprint, 2020.

[6] Frank Gauss and Lukas von Allmen. Nsak (network swiss army knife). https://github.com/nsak-team/nsak, 2026. Accessed: 2026-01-10.

[7] Python programming language. https://www.python.org/.

[8] uv: An extremely fast python package and project manager, written in rust. https://docs.astral.sh/uv/.

[9] Ruff: a super fast python linter. https://github.com/charliermarsh/ruff.

[10] Mypy: Optional static typing for python. https://mypy-lang.org/.

[11] pytest: Simple powerful testing with python. https://docs.pytest.org/.

[12] Pyyaml — yaml parser and emitter for python. https://pyyaml.org/.

[13] Yaml ain't markup language (yaml) specification. https://yaml.org/.

[14] Scapy — packet manipulation tool. https://scapy.net/.

[15] pre-commit — a framework for managing git hooks. https://pre-commit.com/.

[16] Git — distributed version control system. https://git-scm.com/.

[17] iptables — linux firewall and nat administration. https://www.netfilter.org/projects/iptables/.

[18] nftables — linux packet classification and filtering framework. https://www.netfilter.org/projects/nftables/.

[19] Podman — containers without a daemon. https://podman.io/.

[20] podman-compose — docker-compose compatible tool for podman. https://github.com/containers/podman-compose.

[21] pip — the python package installer. https://pip.pypa.io/.

[22] curl — command line tool for transferring data with urls. https://curl.se/.

[23] sudo — execute commands as another user. https://www.sudo.ws/.

[24] Click — a python package for creating command-line interfaces. https://click.palletsprojects.com/.

[25] Fastapi — modern, fast (high-performance) python web framework. https://fastapi.tiangolo.com/.

[26] Vue.js — progressive javascript framework. https://vuejs.org/.

List of Figures

List of Tables

Listings

Glossary

API Application Programming Interface

ARP Address Resolution Protocol

Blue Team The blue team is responsible for protective measures within an organization; during an exercise, they detect and defend against red team engagements.

CLI Command Line Interface

Green Team The green team represents the cooperation between the blue and yellow teams to close knowledge gaps in both areas.

GUI Graphical User Interface

MAC Media Access Control

MITM Man-in-the-Middle

NSAK Network Swiss Army Knife

OCI Open Container Initiative

Orange Team The orange team represents the cooperation between the red and yellow teams, primarily for sharing insights, knowledge, and education.

Proxy Server An intermediary service between a client and a server.

purple team Purple teaming describes the cooperation between red and blue teams to enhance mutual understanding and defensive capabilities.

red team The red team consists of ethical hackers who simulate attacks against systems, networks, and software to test defensive measures.

REST Representational State Transfer

Rogue Access Point Rogue Access Point

TCP Transmission Control Protocol

YAML YAML Ain't Markup Language

Yellow Team The yellow team consists of system, network, and software architects and engineers who design and maintain the infrastructure.