Project 2

Swiss Army Knife Network Sniffer

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 1, 2026

Abstract

We describe NSAK as an embedded, modular, open source, scenario-based network sniffing and security framework. We provide an overview of the system design, emphasizing the functionality of a Swiss Army Knife in a networking context. The attack scenario includes drills that configure or target a specific task to observe or open an attack vector in the system. NSAK is based on a core back-end part that manages specific environments, scenarios, and drills. The whole concept is based on containerization, where each container builds or prepares a scenario that can be triggered in a network environment. In the analogy of the Swiss Army Knife, the right knife with the proper drills for the necessary task can be selected. Modularity comes into play when a drill is selected across multiple scenarios.

Keywords Network intrusion detection, Network Monitoring, Red/Blue Team

Mit Regeln von Abstract gegenchecken und querlesen

Rules to apply

One-paragraph summary of the entire study typically no more than 250 words in length (and in many cases it is well shorter than that), the Abstract provides an overview of the study.

Inhalt Abstract Hintergrundinformationen wie Ausgangslage, Relevanz, Forschungskontext in ein bis zwei Sätzen zusammenfassen. Fragestellung und Ziel explizit formulieren. Die wichtigsten Eckpunkte zum methodischen Vorgehen angeben, bei empirischen Studien auch Angaben zu den Daten wie etwa die Charakteristika der Stichprobe. Im Hauptteil des Abstracts die relevanten Ergebnisse und deren Bedeutung mit wichtigen Kennzahlen aufführen (ca. zwei Drittel des Abstracts). Mit wichtigen Schlussfolgerungen oder Anwendungsmöglichkeiten das Abstract abrunden. Das Abstract enthält keine Quellenverweise

Contents

Abstract iii

1 First Part Thesis: Evaluation 1
1.1 Introduction . . . 1
1.1.1 Information security methods and conceptional frameworks 1
1.1.2 Information security tools and software framworks . . . 1
1.1.3 Lower the bar with modularity and automation . . . 2
1.1.4 ?? (??) . . . 2
1.2 Current State of Research . . . 3

2 Evaluation 5
2.1 Hardware Selection . . . 5
2.1.1 Hardware Requirements . . . 5
2.1.2 Evaluated Boards . . . 6
2.1.3 Decision . . . 7
2.1.4 Hardware Specification . . . 7
2.2 Software Selection . . . 9
2.2.1 Framework Technology Stack . . . 9
2.2.2 System Dependencies . . . 9

3 Architecture and Design 11
3.1 Framework Concepts . . . 11
3.1.1 Devices . . . 11
3.1.2 Environments . . . 12
3.1.3 Drills . . . 12
3.1.4 Scenarios . . . 13
3.1.5 Operator . . . 13
3.1.6 Operation . . . 13
3.2 Use-Cases . . . 14
3.3 Component-diagram . . . 20
3.4 Sequence-Diagram . . . 21

4 Second Part Thesis: Implementation 23
4.1 Method . . . 23
4.1.1 Research Approach . . . 23
4.1.2 System Design Strategy . . . 23
4.1.3 Scenario Oriented Orchestration . . . 24

4.1.4 Modular Drill-Based Architecture 24
4.1.5 Experimental Setup 24
4.1.6 Delimitation 25
4.1.7 Project Management 25
4.2 Implementation 25
4.3 Conclusion 25

Bibliography 29

List of Figures 31

List of Tables 33

Listings 35

Glossary 37
.1 First Appendix Chapter 38
.1.1 Project 2 Proposal 38

1 First Part Thesis: Evaluation

1.1 Introduction

Braucht es mehr quellen? sind alle Punkte erfüllt

According to the World Economic Forums 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.

1.1.1 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 []. 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 [3]. Further, we can observe approaches from the community and the industry to evolve this approach in to the so-called "InfoSec color wheel" [4]. 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 [].

Add security methods and conceptional framworks such as MITRE attack, NIST, OWASP etc.

1.1.2 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 [?]. The threat emulation frameworks analyzed by Zilberman

et al. evaluate multiple attack phases, including lateral movement, persistence, and attack execution.

Add security tools and software frameworks such as MITRE Caledra, Atomic Red Team Metasploit, Infection Monkey, etc.

1.1.3 Lower the bar with modularity and automation

1.1.4 ?? (??)

Describe gaps and issues with the existing tools

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 features [?]:

- 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.

Which gap are we closing, how we plan to distinguish NSAK from MITRE Caledra and other Tools, how can it be incoperated into existing security methods and conceptional frameworks

- Hypothese: Mit einem modularen framework kan das erstellen von automatisierten und reproduzierbaren Red und Blue Team Szenarien ermöglicht werden - Experimente: - Das implementieren von Szenarien mithilfe von modularen und wiederverwendbaren drills funktioniert - Szenarien können in simulierten und realen Netzwerkumgebungen ausgeführt werden

2.2 Einleitung Die Einleitung führt einerseits zum Thema hin (Ausgangslage), und informiert andererseits darüber, warum (Fragestellung/Problem) und wozu

(Ziel/Zweck) es die Arbeit gibt sowie ggf. wie sie zustande gekommen ist (methodisches Vorgehen). Die Einleitung kann je nach Umfang und Thema mit oder ohne Unterkapitel verfasst werden. Sie umfasst ca. 510 Einen Überblick über die Kapitel der Arbeit gibt es höchstens bei sehr langen Arbeiten (beispielsweise Masterarbeit). Die Ausgangslage beschreiben Relevanz: Warum ist dieses Thema überhaupt bedeutsam? Schon hier gilt es, nicht einfach etwas zu behaupten, sondern Fakten und Aussagen mit Fachliteratur zu belegen. Aktualität: Gibt es einen aktuellen Bezug? Zusammenhang: Wie lässt sich das Thema einordnen? In welchem fachlichen Kontext steht die Arbeit? Forschungsstand: Was ist schon erforscht? Gibt es bereits Untersuchungen? (Ist-Zustand und Zusammenhang mit dem eigenen Thema.) Je nach Art und Umfang der Arbeit gibt es zum Wissensstand ein separates Kapitel (siehe 2.3). Wenn vorhanden externe Auftraggeber, Auftraggeberinnen: Wer sind die beteiligten Partnerinnen oder Stakeholder? Den Kurs bzw. das Modul, in dem die Arbeit entsteht, erwähnt man auf dem Titelblatt (siehe 4.1). 8 Das Problem und das Ziel darstellen Zweck der Arbeit: Welche Aufgabe, Herausforderung, welches Problem soll gelöst werden? Warum sollte man die Arbeit lesen? Fragestellung: Auf welche konkrete Hauptfrage (evtl. mit konkretisierenden Unterfragen) soll im Schlusskapitel eine fundierte Antwort gegeben werden? Ist die Fragestellung genügend eingegrenzt? Gibt es Hypothesen? Abgrenzung: Wo liegen die Grenzen der Untersuchung (zeitlich, geografisch, thematisch, methodisch, in der Auswahl der Hilfsmittel usw.)? Was wird nicht untersucht? Was kann die Arbeit nicht leisten? Ziel: Was soll die Untersuchung genau bewirken? Was ist die Absicht hinter der Arbeit? Erwartung: Was möchte die Arbeit leisten (Nutzen der Untersuchung, Soll-Zustand)? Für welche konkrete Zielgruppe sind die Ergebnisse der Arbeit von Nutzen? Was ist zu erwarten? Was ist nicht zu erwarten? Das methodische Vorgehen andeuten Wie man methodisch vorgeht, wird ausführlich im Methodenkapitel beschrieben (siehe 2.4). Oft erwähnt man aber in der Einleitung bereits in ein bis zwei Sätzen, mit welcher Methode man arbeitet (Literaturarbeit, Umfrage, Entwickeln eines Prototyps, Variantenstudium usw.).

1.2 Current State of Research

Der ńStand der Forschungż beantwortet folgende Fragen:

Was wurde zum Thema der Arbeit bereits erforscht (Forschungsstand)? Auf welche Forschungsarbeiten stützt sich die Arbeit ab? Welche Theorien oder Konzepte sind für die Beantwortung der Fragestellung relevant? Welche Begriffe müssen definiert werden? Welche Normen spielen für die Untersuchung eine Rolle

2 Evaluation

2.1 Hardware Selection

2.1.1 Hardware Requirements

The following requirements were defined 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)

2.1.2 Evaluated Boards

Several boards were considered as potential variants. Their main specifications relevant to the project are listed in Table 2.1.

Table 2.1: 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 Œ 2.5 GbE	57 W	MT7976C, Wi-Fi 6 (AP/Client/Monitor)
Banana Pi R3	MT7986A, Quad-core ARM Cortex-A53 @ 1.3 GHz	24 GB DDR4, eMMC, microSD	1 Œ 1 GbE, 2 Œ 2.5 GbE, 4 Œ 1 GbE	710 W	MT7976C, Wi-Fi 6
Banana Pi R4	MT7988A, Quad-core ARM Cortex-A73	4 GB DDR4, NVMe option	4 Œ 2.5 GbE, 2 Œ 10 GbE (SFP+)	1015 W	None (M.2 Wi-Fi module required)
Banana Pi R5	MT7988B, Quad-core ARM Cortex-A73	4 GB DDR4, NVMe option	2 Œ 10 GbE, 2 Œ 2.5 GbE	1218 W	None (M.2 Wi-Fi module required)
Raspberry Pi 4	BCM2711, Quad-core ARM Cortex-A72 @ 1.5 GHz	28 GB LPDDR4, microSD	1 Œ 1 GbE (second via USB dongle)	68 W	Wi-Fi 5 (AP/Client only)
Raspberry Pi 5	BCM2712, Quad-core ARM Cortex-A76 @ 2.4 GHz	48 GB LPDDR4X, microSD	1 Œ 1 GbE (second via PCIe card)	812 W	Wi-Fi 5 (AP/Client only)
NanoPi R76S	Rockchip RK3588S, Octa-core (4Œ Cortex-A76 @ 2.4 GHz + 4Œ Cortex-A55 @ 1.8 GHz)	16 GB LPDDR4X / LPDDR5, NVMe (option via M.2)	3 Œ 2.5 GbE (RJ45)	1015 W	None (M.2 Wi-Fi 6E module recommended)

Table 2.2: 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	✓ (2Œ)	✓ (2Œ)	✓✓ (4Œ)	✓ (2Œ)	✗	✗	✓
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

2.1.3 Decision

Based on the defined requirements and the evaluation of alternatives, the Banana Pi R4 and theNanoPI R76S are the most suitable hardware platforms for this prototype implementation.

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. Raspberry 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.

2.1.4 Hardware Specification

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

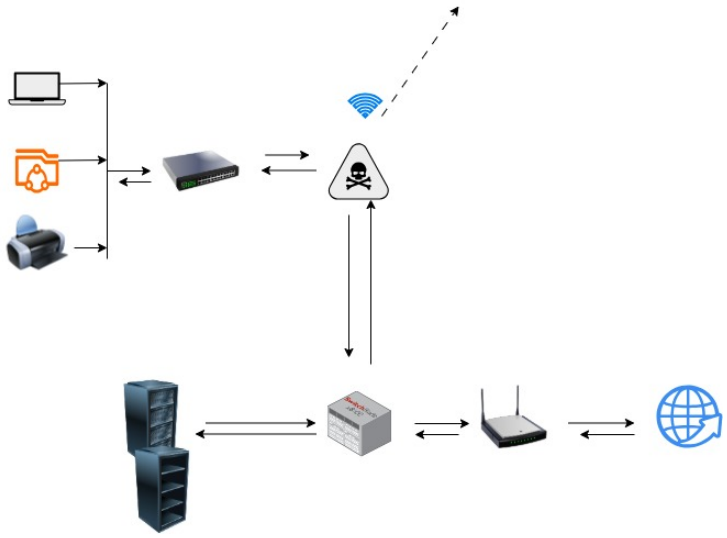

Figure 2.1

Description: 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

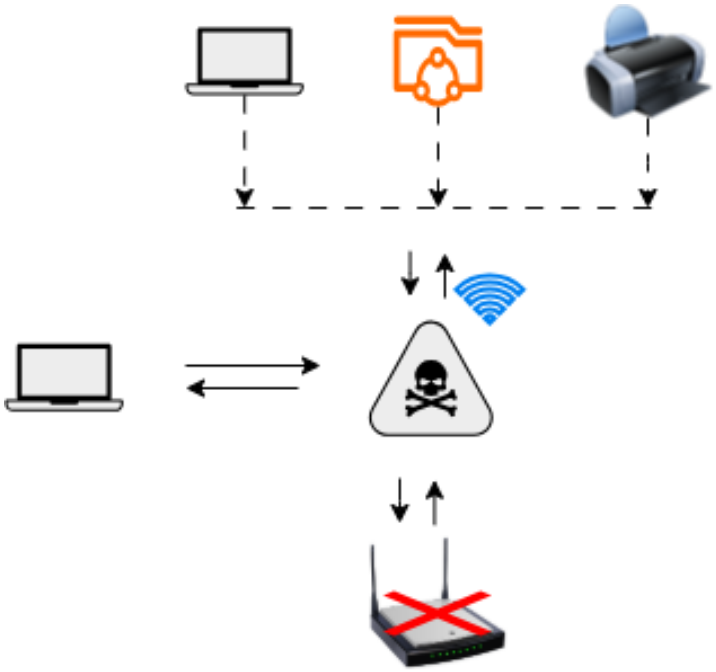

Figure 2.2

Description: The NSAK device is inline and lets traffic pass but intercepts as Rouge AP and capture data

2.2 Software Selection

2.2.1 Framework Technology Stack

Core / CLI

Programming language: Python Dependency manager: uv Virtual environment manager: uv Package build tool: uv Linter: ruff Formatter: ruff Type checker: mypy Testing framework: pytest

Dependencies: click: Library for building CLIs pyyaml: Library for loading, validating and reading yaml files scapy: Library for red team operations pre-commit: Package to enforce code quality tools for each commit

Describe why we choose this technology, maybe we find references which underline the ease of use and advantages for modularity which are comming with python

2.2.2 System Dependencies

Write this section nicer and explain what the advantages are of such a design is, especially in relation to modularity

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 Network tooling: iptables (we should switch to nf_tables) OCI container manager: podman OCI container orchestrator: podman-compose Programming languages: python3, python3-pip, uv Utilities: curl, sudo

3 Architecture and Design

3.1 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

3.1.1 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 2.1, 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.

3.1.2 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

3.1.3 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

3.1.4 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

3.1.5 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 [4]

3.1.6 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.

3.2 Use-Cases

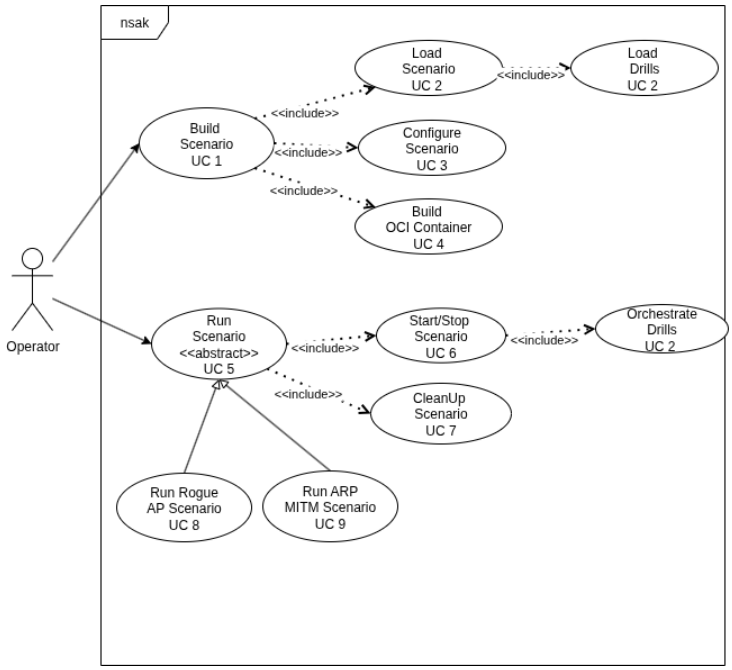

Figure 3.1

Figure 3.1 illustrates the use case structure of the proposed NSAK modular framework. The operator interacts with the 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 parameterssuch 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 runed, and the configured attack drills are been executed within the containerized environment (UC-08).

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

drüber lesen und mit UC final ableichen

Table 3.1: 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 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 operator. Error Scenarios: Conflicting scenario configuration detected build aborted. Result: Scenario container successfully built. Postconditions: Scenario container stored and ready to run.

NR & Details	
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 available. Main Scenario: 1. System applies scenario-specific configuration parameters. Result: Scenario configuration created. Postconditions: Scenario configuration available for loading.
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 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 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) ▶ Clean Up 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 execute drill UC-8 include use-case Error Scenarios: Scenario not found or scenario container 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.

3.3 Component-diagram

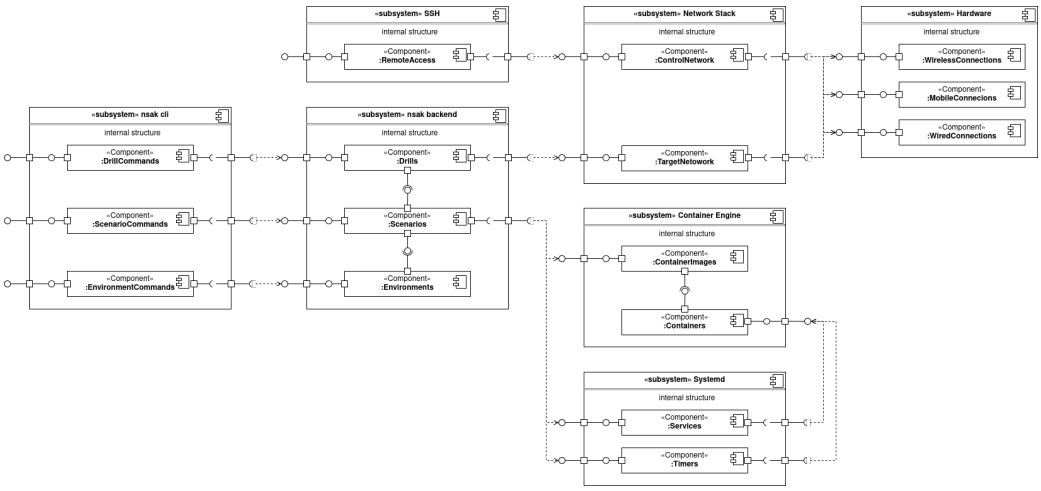

Figure 3.2

3.4 Sequence-Diagram

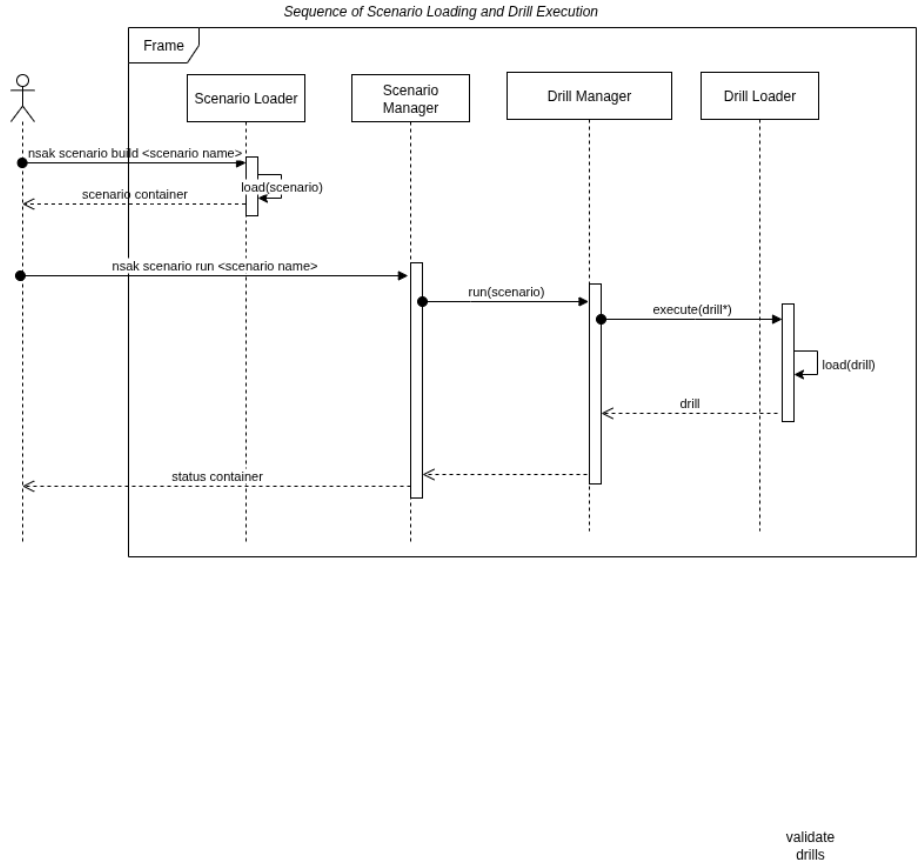

Figure 3.3

Figure 3.3 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.

4 Second Part Thesis: Implementation

4.1 Method

4.1.1 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.

4.1.2 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.

sollen wir das rausnehmen

From a contributors perspective, extending the framework requires answering three guiding questions:

- In which network environment is the NSAK device operating?
- Which scenario should be executed in that environment?
- Which drills are required to implement the scenario?

end

4.1.3 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

4.1.4 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.

4.1.5 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.

4.1.6 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.

4.1.7 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.

4.2 Implementation

4.3 Conclusion

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 1, 2026

Frank Gauss (gausf1) and Lukas von Allmen (vonal3)

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] 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.

[4] 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-security` November 2018. Accessed: 2025-12-30.

List of Figures

2.1 8
2.2 9

3.1 14
3.2 20
3.3 21

List of Tables

2.1 Comparison of Board Variants 6
2.2 Requirements Fulfillment by Candidate Boards 6

3.1 Use Cases Specification (NSAK) 15

Listings

Glossary

This document is incomplete. The external file associated with the glossary 'main' (which should be called `documentation.gls`) hasn't been created.

Check the contents of the file `documentation.glo`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

If you don't want this glossary, add `nomain` to your package option list when you load `glossaries-extra.sty`. For example:

```
\usepackage[nomain]{glossaries-extra}
```

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

  ```
  \usepackage[automake]{glossaries-extra}
  ```

- Run the external (Lua) application:

  ```
  makeglossaries-lite.lua "documentation"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "documentation"
  ```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

.1 First Appendix Chapter

.1.1 Project 2 Proposal