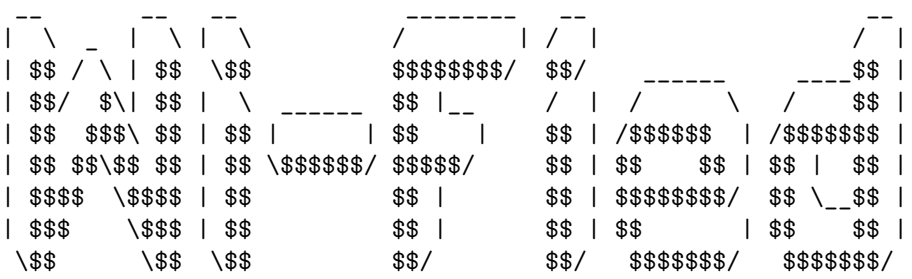
OST
Ostschweizer
Fachhochschule

BA 2025

Wi-Fied

An Educational Platform for Hands-On Wi-Fi Security



Version 1.0
June 12, 2025
Mario Burger, Alice Glaus

Abstract

Wi-Fi networks are as popular as ever and find daily use in private households as well as in corporate environments. Awareness around vulnerabilities and dangerous exploits in wireless communication networks is vital to understand the security implications of operating such infrastructure securely. The Wi-Fied Platform is a modular, Python-based demonstration tool developed to support and enhance the practical education of Wi-Fi security concepts. This work presents the design and implementation of the platform as a complementary tool to existing theoretical content, enabling learners and IT professionals to gain hands-on experience with wireless network vulnerabilities and exploits, utilizing Raspberry Pi's for physical lab-setups.

Inspired by prior research into the classification and impact of Wi-Fi security threats, Wi-Fied supports a controlled, extensible environment for experimenting with attacks within an isolated lab setup. The architecture of the platform follows a structured design, aiming to facilitate future development. Rather than offering a standalone or comprehensive solution, the Wi-Fied Platform is intended to enrich existing curricula by bridging the gap between academic study and applied understanding of Wi-Fi threat scenarios and mitigation techniques.

The architecture of the Wi-Fied Platform evolved iteratively alongside its implementation, enabling flexible adaptation to specific requirements; such as the generation and transmission of Wi-Fi packets. As a result the Wi-Fied Platform project produced a base for future expansion, already providing central functionalities like packet creation, automation of lab-device setups, and a unified CLI-based user interface. Many of the challenges encountered during the project stemmed from library-specific implementation details, ambiguities in Wi-Fi standard definitions, or evolving requirements driven by feature progression.

In conclusion, building a robust but simultaneously flexible and extensible tool with critical dependencies towards physical hardware, Wi-Fi standards, and dissimilar runtime environments is not trivial. However, the practical benefits of an auxiliary tool to enrich existing curricula by bridging the gap between academic study and applied understanding of Wi-Fi security are advantageous. Future work may include but not be limited to the expansion of exploit scenarios and feature sets for packet manipulation, the support of more complex exploit demonstrations with real-time actions, and the addition of a graphical user interface.

Management Summary

Wi-Fi networks are as popular as ever and find daily use in private households as well as in corporate environments. Awareness around vulnerabilities and dangerous exploits in wireless communication networks is vital to understand the security implications of operating such infrastructure securely. The Wi-Fied Platform is a lightweight, modular tool designed to facilitate practical education in Wi-Fi security. It supports a hands-on environment for demonstrating common vulnerabilities and attack techniques in IEEE 802.11 networks, with particular emphasis on reproducibility and controlled experimentation.

This platform was developed with three primary stakeholder groups in mind: Educators, students, and developers or maintainers. Each group brings distinct interests, ranging from instructional integration of hands-on learning into existing curricula and long-term codebase evolution, while also sharing several important interests. The following diagram in Figure 1 illustrates the relationships and overlapping interests among these target audiences:

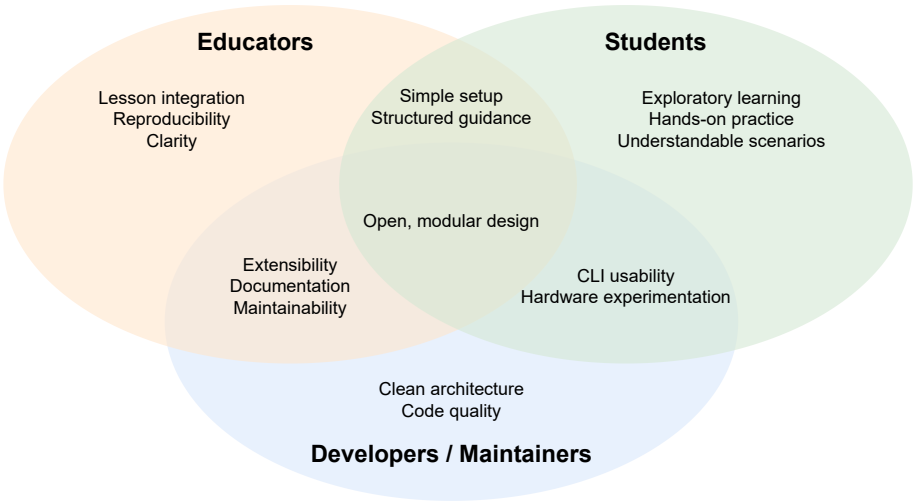


Figure 1.: Interests of Primary Stakeholder Groups

Developed for use in academic and lab-based training, the platform integrates familiar tools like Ansible, Scapy, and hostapd into a cohesive CLI that enables instructors and students to set up and run Wi-Fi security scenarios with minimal effort. The platform leverages affordable hardware, such as Raspberry Pi boards and compatible Wi-Fi adapters, making it accessible to educational institutions without requiring costly infrastructure. The modular codebase allows further development and adaptation to emerging threats or learning objectives. The project lays foundational groundwork for a growing suite of Wi-Fi exploit demonstrations and can evolve into a broader learning and testing platform over time.

Key technical challenges, such as hardware compatibility and software fragility, are acknowledged transparently and mitigated where possible. While the platform prioritizes maintainability and extensibility over short-term feature completeness, it already supports the necessary steps to demonstrate a Key Reinstallation Attack (KRACK) or a Deauthentication Attack (refer to Figure 2).

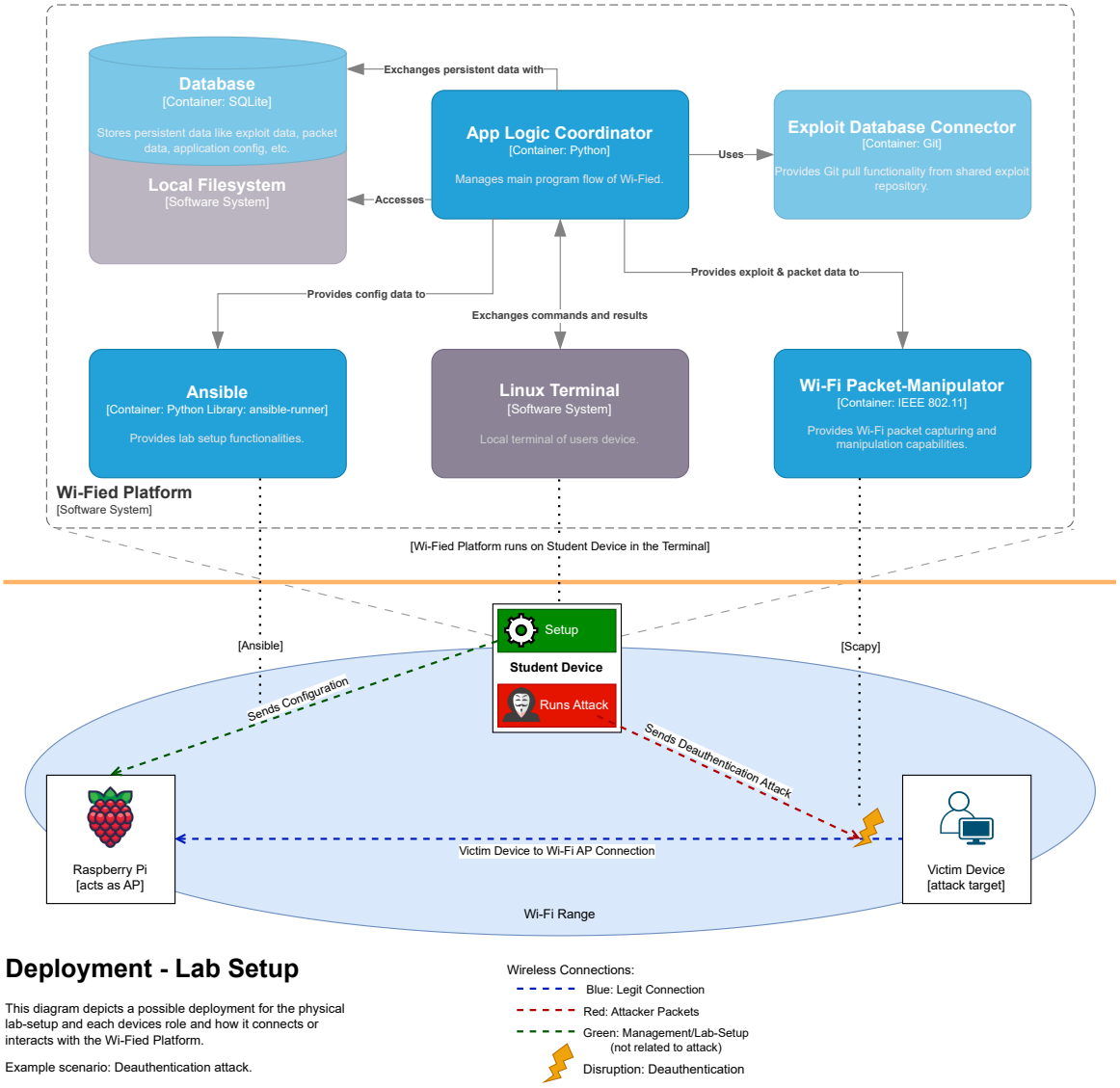


Figure 2.: Conceptual Lab Setup for Deauthentication Attack

In conclusion, building a robust but simultaneously flexible and extensible tool with critical dependencies towards physical hardware, Wi-Fi standards, and dissimilar runtime environments is not trivial. However, the practical benefits of an auxiliary tool to enrich existing course material by bridging the gap between academic study and applied understanding of Wi-Fi security are advantageous. Future work may include but not be limited to the expansion of exploit scenarios and feature sets for packet manipulation, the support of more complex exploit demonstrations with real-time actions, and the addition of a graphical user interface.

Disclaimer - Intended Use

Wi-Fied is intended exclusively for educational use. It complements existing Wi-Fi security courses and materials by offering a practical tool for demonstrations and experimentation. Its purpose is to help learners and professionals better understand Wi-Fi security risks. Any malicious or unauthorized use is strictly prohibited and not supported by the authors.

Introduction

Understanding the security of wireless communication systems is an increasingly relevant topic, as Wi-Fi continues to serve as the backbone for both personal and professional connectivity. This report introduces the Wi-Fied Platform, a tool designed to support hands-on experimentation with Wi-Fi vulnerabilities in an educational context. The project combines prior academic insights with practical implementation efforts and follows a structured development approach to ensure modularity, usability, and future extensibility. The following sections provide background on the motivation and context of the work, before detailing its architectural and technical foundations.

Background and Motivation

Wireless networking is a cornerstone of modern connectivity, enabling access to digital services in homes, businesses, schools, and public spaces. While convenient and widely adopted, Wi-Fi networks are susceptible to various security vulnerabilities due to legacy protocols, misconfigurations, and evolving attack techniques. As these threats become more complex, the need for practical education in Wi-Fi security increases.

This project is motivated by the desire to close the gap between theoretical knowledge and real-world experimentation in this domain. The idea emerged from the team's earlier academic work titled *"Wi-Fi Security Threats - an Integrative Review"* [1], in which the project team systematically examined and categorized vulnerabilities in Wi-Fi networks. That literature review contributed a structured understanding of attack types, their technical impacts, and mitigation strategies, while also exposing a lack of accessible tools for hands-on learning.

The Wi-Fied Platform was developed to address this gap: providing a modular, reproducible lab environment that supports educational exploration of Wi-Fi attacks and defenses in a controlled setting. By building directly on the team's own research, the project translates conceptual insight into tangible, actionable learning experiences.

Project Aim and Scope

The Wi-Fied Platform was developed to address this gap by creating a modular, extensible toolkit that enables controlled experimentation with Wi-Fi exploits in a local, lab-based environment. The platform supports some common attack scenarios out of the box, such as deauthentication or key reinstallation (KRACK), and is specifically designed to be extensible and easily used in educational settings.

The primary stakeholders include educators, learners, and developers. Educators benefit from the tools integration potential into courses and labs, students gain hands-on experience with realistic network behaviors, and developers or maintainers can extend the platform with new tools, features, and learning modules.

While the Wi-Fied Platform focuses on a specific hardware and software setup to reduce complexity, it is built with extensibility in mind. It does not aim to replace existing security frameworks or enterprise-grade simulation environments, but to provide a didactic tool for security education in a self-contained, reproducible way.

Methods

At the beginning of the project, an initial evaluation phase was conducted to establish foundational knowledge in Wi-Fi packet capturing and manipulation using Python. This phase built upon the team's prior academic work on Wi-Fi vulnerabilities [1], and aimed to assess the practical feasibility of implementing low-level operations such as packet injection.

Key tools evaluated included Scapy for packet-level control and initially Nornir (later switched to Ansible) for automation of system-level configuration tasks. The insights gained during this period informed an early architectural vision, initially shaped by principles from Robert C. Martin's *Clean Architecture* [2]. While this model provided valuable guidelines regarding modularity and separation of concerns, strict adherence to its layered structure proved impractical in the face of rapid prototyping and evolving requirements.

As a result, the team adopted a dynamic, iterative development strategy. Features were first built in self-contained, experimental prototypes to ensure viability and feasibility. Subsequently, these proof-of-concept implementations were formalized; typically by wrapping system interactions in structured automations and gradually integrating them into the main codebase. Finally, new additions were tested, refined, and aligned with the platform's architectural direction.

This cyclical approach (prototype, formalize, integrate, refactor) allowed the project to remain agile and responsive, especially when dealing with evolving technical constraints or newly discovered user needs. Though the architecture underwent significant revisions during development, this flexible methodology ultimately led to a coherent, purpose-built design that balanced maintainability with practical usability.

Results and Lessons Learned

The Wi-Fied Platform demonstrates that a hands-on, modular toolkit for exploring Wi-Fi security threats is both feasible and valuable in educational settings. Throughout development, the project revealed important lessons about balancing extensibility with simplicity, especially when operating in constrained environments such as Raspberry Pi hardware and limited time budgets.

Working with existing technologies like Python, Scapy, and Nornir (later switched to Ansible) allowed for faster iteration and a lower entry barrier for learners.

The user testing phase confirmed that the current CLI-based interface is understandable, reasonably usable, and offers sufficient guidance for achieving meaningful results. Minor usability feedback was collected and implemented, but no critical flaws or design revisions were required.

Overall, the project delivered a foundational platform that can be integrated into practical Wi-Fi security courses and lays the groundwork for future, more feature-rich iterations. Additionally, an instructional lab document was written, aiming to be used in a cryptography module introducing key derivation and KRACK, utilizing the Wi-Fied Platform.

Outlook

While the current implementation fulfills its intended role as a prototype platform for educational Wi-Fi exploit demonstrations, several enhancements and future directions have been identified.

Enhanced Lab Scenarios and Instructions: Future work may involve refining the instructional materials and lab exercises to include preconfigured scenarios, sandboxed environments, and even automated assessment or logging of student results. Scenario-based learning could be expanded to simulate real-world security audits or Red Team vs. Blue Team activities.

Improved User Interface: While the current CLI is sufficient for early adopters and technically proficient users, adding an optional graphical interface could improve accessibility, especially

for students unfamiliar with command-line workflows. Such an interface could visualize attack flows, capture results in real time, or assist in scenario configuration.

Hardware Compatibility and Deployment Flexibility: At present, the platform targets a narrow set of tested hardware. Future development could extend support for a broader range of USB Wi-Fi adapters and platforms (including virtualized environments or cloud-hosted simulation) to improve scalability and ease of adoption. Integrating hardware checks or compatibility tests could also reduce setup friction for instructors.

Open Source and Community Involvement: Eventually, the project could be made available as an open-source resource, inviting contributions from educators, students, and researchers. This would not only promote sustainability and reuse, but also help in continuously expanding attack coverage and pedagogical utility.

In conclusion, the Wi-Fied Platform provides a solid starting point for Wi-Fi security education and experimentation, but its greatest potential lies in future iterations, guided by classroom experience, evolving threat landscapes, and community feedback.

Structure of This Report

This report is divided into three main parts: An architectural and product-focused documentation based on the arc42 methodology (Part I), a comprehensive project documentation (Part II), and various supplementary materials; lab-exercise document, user test protocols (appendix). It begins with a high-level overview of constraints, goals, and decisions made during the design and implementation of the platform, and later reflects on lessons learned, evaluation results, and future directions.

Acknowledgements

We want to thank our advisor Dr. Daniel Tschudi for his guidance and support throughout the project, Dr. Olaf Zimmermann for his valuable inputs on architectural design, and Dr. Bernhard Tellenbach for kindly offering his time and expertise.

Further, we would also like to extend our thanks to the people who supported us during our project. Special thanks go to our proofreaders; Dr. Arno Wagner and Jan Untersander, as well as to the test candidates, who helped us improve our tool: Urs Baumann, Ramon Bister, Stefanie Jäger, Dominic Klinger, Fabio Marti, Laura Thoma.

Additionally, we thank the Institute for Network and Security for supporting us with the acquisition of lab-devices and other hardware.

Contents

Part I.

Product Documentation

Architectural Documentation Approach

This product documentation outlines the architecture of the Wi-Fied Platform based on the well-established arc42 template [3], by Dr. Peter Hruschka, Dr. Gernot Starke and contributors. All twelve chapters of the template were adapted accordingly to fit the educational scope and technical scale of the project. Given Wi-Fied's focused domain, some architectural views are treated with less depth to maintain clarity and relevance.

This pragmatic use of arc42 supports the platform's goal: To educate users about Wi-Fi vulnerabilities through a modular, extensible, Python-based system.

Note: The product documentation is self-contained and serves as a standalone reference for the architectural design of the Wi-Fied Platform.

1. Introduction and Goals

The primary goal of the project around Wi-Fied Platform is to develop an educational tool that complements existing Wi-Fi security courses and materials by adding a practical tool for demonstrations and experimentation. Its purpose is to help learners and professionals better understand Wi-Fi security risks.

By following a modular and extendable design approach, the Wi-Fied Platform aims to support practical cybersecurity awareness using accessible, open-source technologies on affordable hardware like the Raspberry Pi while being adaptable and extendable in future settings.

1.1. Quality Goals

The top three quality goals are briefly described in Table 1.1. These have the highest importance for the success of the project and are distilled from the project assignment and requirements. For more details refer to chapter 10.

Prio	Quality	Description
1	Usability & Learnability	The educational context for the Wi-Fied Platform defines the most important quality goal: Ease of use and focus on learning. Functionalities of the Wi-Fied Platform should be comprehensive and students familiar with CLI-tools should be able to use it within 10 minutes.
2	Extensibility	The architecture of the platform should follow design principles to support future extension.
3	Modularity	The platform should follow a modular design, allowing future modification.

Table 1.1.: Top quality goals

1.2. Stakeholders

All the important stakeholders of the Wi-Fied Platform project are listed in Table 1.2. Furthermore, prior to implementation of the project proto personas were defined to further specify the needs and requirements of users in more details.

1.2.1. Stakeholder Overview

List of all relevant stakeholders with their respective role, interests, goals, and requirements.

Stakeholder	Role / Description	Interests / Goals
Educators / Instructors	Use the tool to teach Wi-Fi security topics	Hands-on tool to demonstrate attacks with integration into teaching plans (simplicity, reproducibility, clear documentation)
Students / Learners	Target audience for education on Wi-Fi vulnerabilities and defenses	Practical experience, easy setup, exploratory learning (usability, understandable scenarios, potential for experimentation)
Tool Maintainers / Developers *	Extend or maintain the codebase	Code quality, extensibility, maintainability (clean architecture, modularity, documentation)

Table 1.2.: Stakeholder Overview

*Tool Maintainer / Developer roles may also be taken up by Educators / Instructors or Students / Learners, when they adapt or change the tool to suit their needs.

1.2.2. Proto Personas

The following proto personas describe the main target users of the Wi-Fied Platform. These complement the overview of the stakeholders. The first proto persona in Table 1.3 characterizes a standard user.

Proto Persona 1 – Learner	
Name	Steve Smith
Role/Occupation	Student or course participant with IT background
Goals	Persona wants to run their own experiments with real hardware and receive instructions on how to do so.
Challenges/Pain-points	Persona is interested to learn more about Wi-Fi security but struggles to find praxis-oriented resources encouraging experimentation with realistic scenarios.
Learning preferences	Prefers experiment-like, hands-on learning and concise, structured instructions.
Usage context	Studies or takes part in a course with high information density over a short time.

Proto Persona 1 – Learner (continued)

Table 1.3.: Proto Persona 1

A more specialized user is characterized by the second proto persona in Table 1.4.

Proto Persona 2 - Educator	
Name	Danny Davis
Role/Occupation	Lecturer or course instructor at educational institution
Goals	Persona wants to complement their theoretical course materials with experiment-like, hands-on exercises.
Challenges/Pain-points	To conduct realistic and insightful exercises requires a lot of time and preparation.
Work preferences	Prefers to have a collection of ready-to-run scenarios at their disposal, with the ability of easy modification and flexible future extension.
Usage context	Works in a fast-paced institution in need of multi-modal teaching aids.

Table 1.4.: Proto Persona 2

1.3. Requirements Overview

Use cases were specified to describe key features of the Wi-Fied Platform project. Each describes a specific need of a user. The projects *Context and Scope* which also outlines the *Scope & Minimum Viable Product (MVP)* is based on those use cases.
Note: The use cases were defined at the beginning of the project and were gradually updated to reflect the evolution of the project.

1.3.1. Use Case Diagram

The use case diagram in Figure 1.1 shows the user interaction with the application and how it should respond.

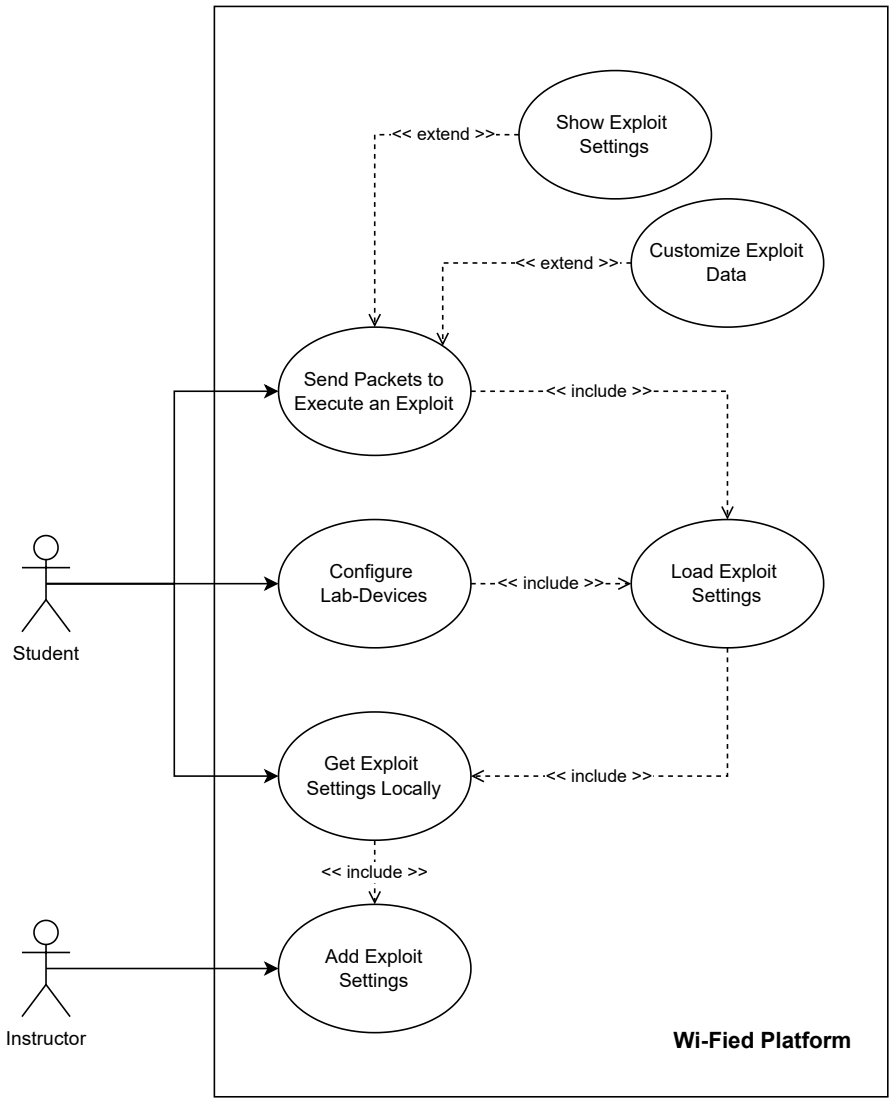


Figure 1.1.: Use Case Diagram

1.3.2. Fully Dressed Use Cases

All use cases were defined as fully dressed use cases. They describe the basic user interactions with the application and define the criteria for the minimum viable product.

Each use case describes the main success scenario in the form of a step-by-step flow. Some require specific pre-conditions. Alternative flows are additionally formulated in use cases, where the main success scenario might not always be expected to work out. In those cases, the alternative flows are intended to provide alternate ways to fulfil (possibly via detour) the user needs.

Use Case 1 - Basic Use of the Wi-Fied Platform	
Applies to	Wi-Fi Educational Platform
Level	User goal
Primary actor	Student
Stakeholders & Interests	Student: Wants to launch the Wi-Fied Platform and learn about its features.
Pre-conditions	None.
Success guarantee (post-conditions)	Help command intuitively guides the user.
Main success scenario	1. Student launches the Wi-Fied Platform. 2. CLI shows up. 3. The help-command is used by the student. 4. Student learns about the possible commands.
Special requirements	None.
Open issues	None.

Table 1.5.: UC1

Use Case 2 - Enable Monitor Mode	
Applies to	Wi-Fi Educational Platform
Level	User goal
Primary actor	Student
Stakeholders & Interests	Student: Wants to configure a Wi-Fi interface in monitor mode.
Pre-conditions	The platform is running (*UC1*).
Success guarantee (post-conditions)	A Wi-Fi interface is set up in monitor mode and able to send raw IEEE 802.11 traffic.
Main success scenario	1. User provides a monitor mode capable interface. 2. User uses the command to set the interface in monitor mode. 3. The system processes the config and sends it to the devices.

Use Case 2 - Enable Monitor Mode (continued)	
	4. The devices apply the config and return the result to the system. 5. The system provides the result of the tasks to the user.
Alternative flows	**Alt3a:** The config does not exist: 1. An error message is shown to the user explaining the issue. 2. The user identifies the problem and possibly returns to step 1 or 2 of the main success scenario. **Alt3b:** A connection error occurs: 1. An error message is shown to the user explaining the issue. 2. The user identifies the problem and possibly returns to step 1 or 2 of the main success scenario. **Alt4a:** The config is invalid or cannot be applied: 1. An error message is shown to the user explaining the issue. 2. The user corrects the config and returns to step 2 of the main success scenario.
Special requirements	Wi-Fi adapter that supports monitor mode.
Open issues	Check if the Wi-Fi adapter supports monitor mode.

Table 1.6.: UC2

Use Case 3 - Distribution of Exploits	
Applies to	Wi-Fi Educational Platform
Level	User goal
Primary actor	Instructor & Student
Stakeholders & Interests	Instructor: Wants to distribute predefined exploits through a centralized system. Student: Wants to have new exploits.
Pre-conditions	The platform is running (*UC1*).
Success guarantee (post-conditions)	Exploits can be distributed.
Main success scenario	1. Instructor uploads exploits to a centralized system. 2. User issues a command in Wi-Fied to get exploits. 3. The system connects to the centralized system and downloads exploits locally.
Alternative flows	**Alt3a:** Connection fails:

Use Case 3 - Distribution of Exploits (continued)	
	1. The system informs the user about the issue and returns to await further commands by the user.
Special requirements	Device on which student runs Wi-Fied Platform has internet access.
Open issues	None.

Table 1.7.: UC3

Use Case 4 - Configure Lab-Devices	
Applies to	Wi-Fi Educational Platform
Level	User goal
Primary actor	Student
Stakeholders & Interests	Student: Wants to set up the experiment and configure the devices. Instructor: Wants to create a new setup and test it beforehand.
Pre-conditions	Exploits were downloaded locally (*UC3*) and an exploit was selected to load.
Success guarantee (post-conditions)	The experiment is set up and configured.
Main success scenario	1. User sets up the experiment and connects all devices. 2. User uses the command to start the setup. 3. The system processes the config and sends it to the connected devices. 4. The devices apply the config and return the result to the system. 5. The system provides the result of the tasks to the user.
Alternative flows	**Alt3a:** The config does not exist: 1. An error message is shown to the user explaining the issue. 2. The user identifies the problem and possibly returns to step 1 or 2 of the main success scenario. **Alt3b:** A connection error occurs: 1. An error message is shown to the user explaining the issue. 2. The user identifies the problem and possibly returns to step 1 or 2 of the main success scenario. **Alt4a:** The config is invalid or cannot be applied: 1. An error message is shown to the user explaining the issue. 2. The user corrects the config and returns to step 2 of the main success scenario.
Special requirements	The hardware required for the experiment setup is available.

Use Case 4 - Configure Lab-Devices (continued)	
Open issues	Incompatibilities with hardware, drivers, service- or OS-versions. Raspberry Pi recommended, others are out of scope.

Table 1.8.: UC4

Use Case 5 - Send Packets	
Applies to	Wi-Fi Educational Platform
Level	User goal
Primary actor	Student
Stakeholders & Interests	Student: Wants to execute an exploit.
Pre-conditions	Exploits were downloaded locally (*UC3*) and an exploit was selected to load which contains packet data.
Success guarantee (post-conditions)	The packets are sent out, possibly executing an exploit.
Main success scenario	1. User uses the command to send packets including the packets' id. 2. The system loads the packets from the database, builds them and sends them out. 3. The system reports to the user that the packets were sent successfully.
Alternative flows	**Alt2a:** Packet ids do not exist: 1. The system ignores the ids which do not exist and only continues with the ones that do. **Alt3b:** The packets cannot be built: 1. An error message is shown to the user explaining the issue. 2. The user corrects the packet data and returns to step 1 of the main success scenario. **Alt4a:** The packets cannot be sent: 1. An error message is shown to the user explaining the issue. 2. The user identifies the problem and returns to step 1 of the main success scenario.
Special requirements	Root privileges are needed to send packets.
Open issues	Sending of raw Wi-Fi packets on Windows is very limited and therefore not supported by Wi-Fied Platform (out of scope).

Table 1.9.: UC5

2. Constraints

This chapter outlines boundary conditions that significantly influenced architectural decisions throughout the project. These include both organizational factors and technical limitations that were either externally imposed or emerged during development.

2.1. Organizational Constraints

This section describes non-technical conditions that impacted the project, such as institutional requirements, collaboration structures, and time or resource limitations. These factors were often outside the development team's control but shaped the scope and pace of the work.

2.1.1. Educational Scope

The tool is designed strictly for educational purposes, not for offensive or production use. The main focus is on didactic value and low barriers of entry. Wi-Fied Platform should enable junior IT professionals or students to easily familiarize themselves with the tool's usage.
The platform should also be focussing on extensibility. The number and complexity of predefined exploits may be limited, because targeting the future expansion as the overreaching goal is prioritized.

2.1.2. Time and Resource Limitations

The project is developed by a two-person team, each with a time budget of around 360 hours. As a bachelor thesis, the project is bound by academic scope and requirements. In addition to implementation work, it involves substantial non-technical tasks such as presentations, supplementary hand-ins (e.g., brochure abstract, poster), and written self-reflection. Limited time and team size naturally restrict the overall project scope.

Due to those limitations, the team focuses on creating a platform that is extendable in the future.

2.2. Technical Constraints

Here, we detail technology-related limitations such as library or protocol constraints, and availability of supported hardware. These technical boundaries restricted design choices and affected implementation decisions directly.

2.2.1. Programming Language and Feature-Set

A commonly known ecosystem, actively maintained and equipped with strong functional abilities (or given appropriate extensibility of such through purpose-made libraries) should be chosen for the project. Exotic, highly specialized tools should be avoided, since the project is set out to minimize barriers for learners using the Wi-Fied Platform, or educators who intend to expand its features.

More details on the selected tools and their benefits as well as drawbacks are listed in chapter 4, *Solution Strategy*.

2.2.2. Supported Hardware

To reduce complexity and ensure a manageable development scope, the project team has deliberately limited hardware support to a predefined set of lab devices. This artificial constraint focuses the initial implementation on cost-effective, widely available hardware (such as Raspberry Pi boards and compatible Wi-Fi adapters) commonly used in educational setups. While the tool is designed to be extensible and not inherently tied to specific hardware, broader compatibility across different devices, operating systems, or platforms was not evaluated or tested within the scope of this first iteration.

For details on the supported hardware refer to the described hardware in chapter 4, *Solution Strategy*.

2.3. Legal and Ethical Constraints

The project is considered to take place in an educational and non-malicious context only and adheres to these rules:

- **No Malicious Use:** Wi-Fied Platform must not be used outside controlled lab environments.
- **Compliance with Laws:** Wi-Fied Platform must not violate Wi-Fi communication laws or ethical guidelines.

Wi-Fied is intended exclusively for educational use. It complements existing Wi-Fi security courses and materials by offering a practical tool for demonstrations and experimentation. Its purpose is to help learners and professionals better understand Wi-Fi security risks. Any malicious or unauthorized use is strictly prohibited and not supported by the authors.

3. Context and Scope

3.1. Business Context

The business context of the application is depicted in Figure 3.1 with external systems and interactors. The scope of the system and any interacting neighboring systems are shown. Users and their interactions with the system complete the context diagram. In general, the system is isolated from any critical outside factors (such as existing legacy-systems or adjacent prod-environments), due to the nature of being an experimental, controlled lab-environment in which it operates.

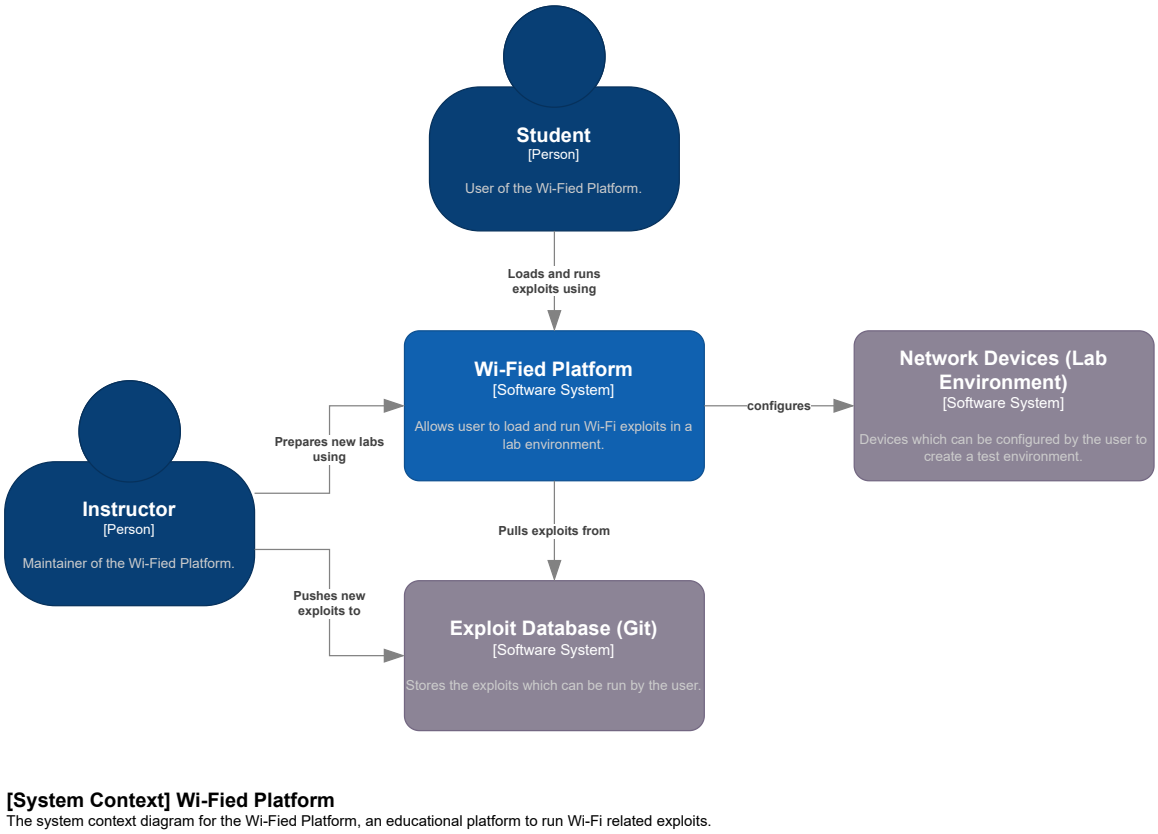


Figure 3.1.: System Context Diagram - C4 model diagram-style, level 1

The interactors and external systems as seen in the diagram are described in more detail in the following Table 3.1.

Neighbor	Description
Student	Uses Wi-Fied Platform to experiment with Wi-Fi exploits and vulnerabilities. Wants to try out new exploits in a controlled environment.
Instructor	Extends the available exploits and adjusts configurations as necessary. Is able to push new exploits to the Exploit Database. Wants to have an extendable platform.
Exploit Database	Provides a set of predefined exploits to the Wi-Fied Platform.
Network Devices	Provides the necessary hardware-infrastructure for the controlled environment in which the Wi-Fied Platform executes exploits and experiments.

Table 3.1.: Interactors and External Systems of the Business Context

3.2. Technical Context

Apart from the inner workings of the Wi-Fied Platform, the context with the lab-setup including physical devices is important to mention when highlighting the technical context. In the following an exemplary lab-setup for the Wi-Fied Platform is described.

The setup is heavily influenced by the exploit scenario that is being experimented on, which can require different devices and configurations. For the understanding of the technical context of the Wi-Fied Platform it is sufficient to illustrate the lab-setup without every specific detail. Because of that, the diagram only hints towards the lab-setup depicted as a black box.

The following Table 3.2 provides an overview of the different items depicted in Figure 3.2 below.

Neighbor	Description
Internet Services	Wi-Fied Platform requires an internet connection on the system it is used. The Exploit Repository (Git-based) and package managers may be used in certain scenarios.
Student Device	A computer which is Linux-based to run the tool on. Root privileges are required to use all features.
Example of a Lab-Setup	Raspberry Pi computers fulfill the roles required for the experiments. They need to be reachable from the Student Device via SSH. The involved physical devices vary depending on the given scenario.

Table 3.2.: Exemplary Technical Lab-Setup Context

Figure 3.2 illustrates how the context of the Wi-Fied Platform may look like, when deployed in a lab constellation.

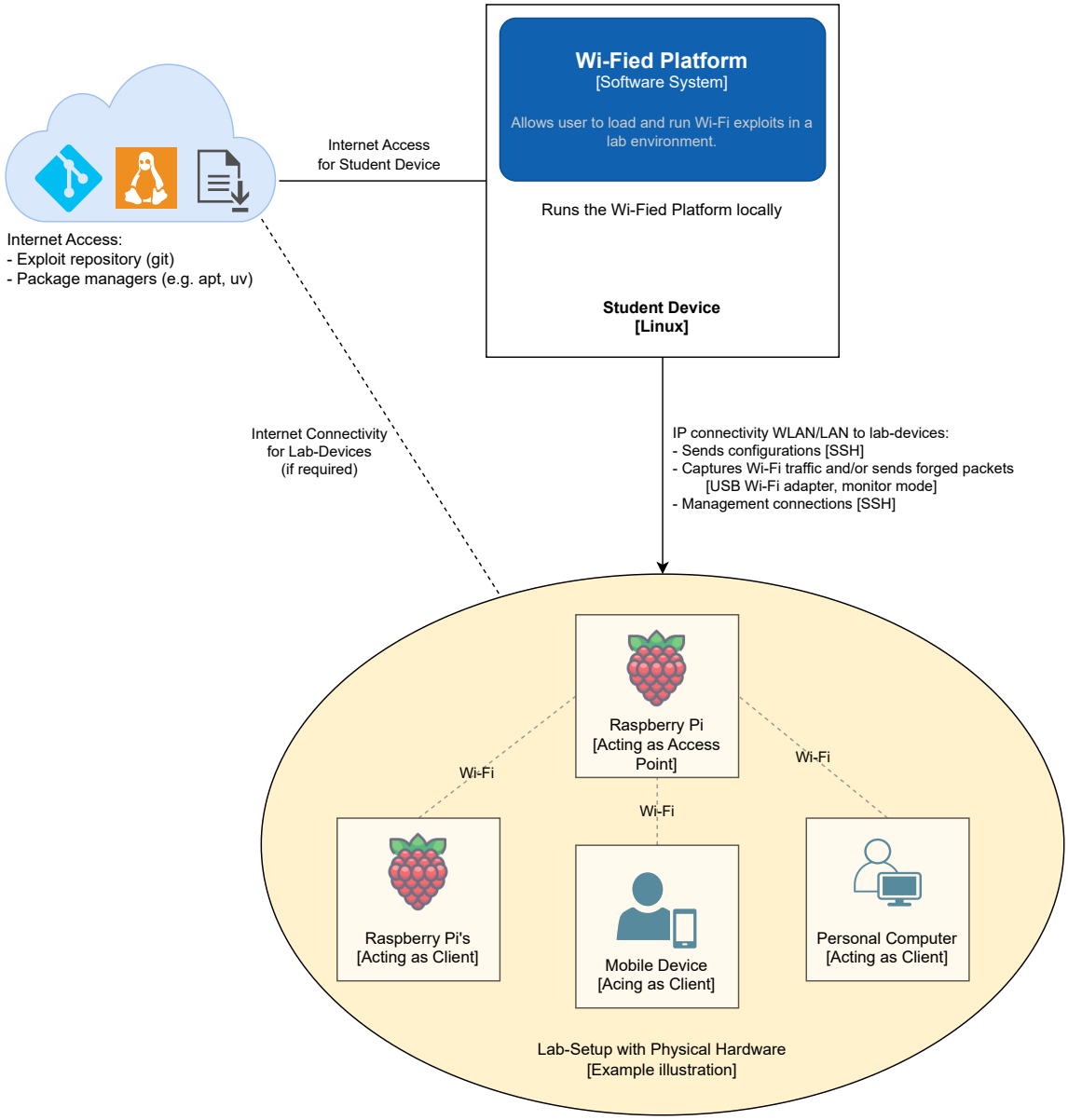


Figure 3.2.: Lab-Setup Context Diagram

3.3. Scope & Minimum Viable Product (MVP)

The initial release of the Wi-Fied Platform is considered feature-complete and usable once the following Minimum Viable Product (MVP) criteria are met. These criteria reflect a baseline of functional and architectural capabilities required to support the educational goals of the project and form the foundation for future extensibility.

3.3.1. MVP Criteria

1. User Interaction
   - The Wi-Fied Platform must provide a user-friendly user interface (e.g., CLI) allowing users to set up lab-devices, run exploits, and manage configurations.
   - Interaction must support guided help (e.g., –help) and structured argument input.
2. Lab-Setup Automation
   - Users must be able to define custom scripts / instructions to automatically set up lab-devices (e.g., Nornir / Anisble).
   - The Wi-Fied Platform must support executing these setup scripts / instructions directly against connected lab hardware, with success/failure feedback visible to the user.
3. Assistance for Monitor Mode Support
   - The Wi-Fied Platform must assist the user in enabling monitor mode on supported interfaces.
4. Predefined Exploit Scenarios
   - One working exploit scenarios must be available out-of-the-box.
   - These scenarios must be executable with minimal configuration and serve demonstrative educational purposes.
5. Extensibility for Custom Exploits
   - Users must be able to define and integrate new exploit scenarios without modifying core application code (e.g., dynamic packet building).
   - The tool must support pulling externally maintained exploit configurations from a Git repository and loading them for use in the app.
6. Extensible and Modular Architecture
   - The system architecture must follow a modular and loosely coupled design to support future enhancements.
   - Each major functional domain (e.g., packet manipulation, and setup automation) must be encapsulated in a separate, well-defined module or component.
   - The architectural structure must be documented (e.g., via arc42 and C4 Model diagrams).

3.3.2. Related Use Cases

The aforementioned criteria for MVP can be linked directly to use cases derived from the requirements. In the following Table 3.3 this mapping is completed by also adding relevant related quality attributes to each entry.

For details on use cases refer to section 1.3, or for quality attributes refer to column *Quality* in Table 10.5.

MVP Criterion	**Related Use Cases**	**Related Quality Attributes**
1. User Interaction (CLI Interface)	*UC1*: Basic Use of Wi-Fied Platform	Ease of Use, Learnability, Consistency
2. Lab-Setup Automation	*UC4*: Configure Lab-Devices	Idempotence, Scriptability, Portability
3. Monitor Mode	*UC2*: Enable Monitor Mode	Reliability, Performance
4. Predefined Exploit Scenarios	*UC5*: Execute Exploit	Extensibility, Simplicity
5. Custom Exploits via Git	*UC3*: Pull from Exploit Repository	Extensibility, Maintainability
6. Modular Architecture	Cross-cutting (all use cases)	Modularity, Reusability

Table 3.3.: MVP Criteria Mapping

3.4. Out of Scope

The project primarily confines itself to the criteria listed in section 3.3.

Out of scope are:

- Hardware for lab-setups, other than mentioned in subsection 4.1.8, and subsection 4.1.9. Due to the project constraints as well as due to the challenging nature of the environment, which the Wi-Fied Platform operates in (hardware proximity), no guarantees regarding compatibility or feature-availability are covered by the project.
- Host systems, other than Debian or Ubuntu based Linux, to run the Wi-Fied Platform on. This is because of limitations of available package managers required by the Wi-Fied Platform.
- All-in-one solution for teaching, experimenting, and execution of exploits. Wi-Fied Platform serves as a helpful tool to complement existing teaching materials. It is not expected to provide a rich catalog of ready-to-go lab experiences for experimenting with exploits, because the project mainly aims to build an extensible platform.

4. Solution Strategy

High level summary of the fundamental technological decisions and an explanation of the general approach to the project's implementation, with broad architectural consequences.

4.1. Technology Selection

The selection of technologies for the Wi-Fied Platform was guided by practical considerations and prior experience. Whenever feasible, tools and languages already familiar from earlier coursework (such as Python and the Scapy library) were preferred to reduce overhead and speed up prototyping. This allowed the team to focus on solving domain-specific challenges rather than spending time learning new frameworks. On the hardware side, technology choices were shaped by the constraints outlined earlier in subsection 2.2.2, prioritizing accessible, well-documented components to ensure reproducibility in educational lab environments.

4.1.1. Programming Language Python

The project requires a programming language with libraries that support packet creation and capturing. Python [4] was chosen due to its regular update cycles and wide community supported libraries.

Benefits

- Accessibility for learners:
  The target audience is likely familiar with Python programming. This lowers the barrier of entry and allows users to focus on understanding security concepts rather than language quirks.
- Wide adoption and rich ecosystem:
  Users benefit from learning in a widely-used ecosystem. Several key functionalities of the project depend on specialized libraries, for example Scapy for packet manipulation (more details below).
- Strong community and resources:
  Documentation and community support are abundant, which helps when extending the tool in the future.

Drawbacks

- Performance limitations:
  Python is slower than compiled languages like Go. This might be noticeable in time-critical operations, though it is unlikely to be a major issue in an educational setup.
- Packaging and deployment complexity:
  Native dependencies (installed packages on target system) or privileged operations like monitor mode might be challenging to predict, due to limited control over target system hardware. Although, project and dependency managers (such as UV) greatly simplify the setup.

4.1.2. Scapy

Scapy [5] is required to capture, decode, and forge packets. In the context of the Wi-Fied Platform, it handles all interactions with actual Wi-Fi packets.

Benefits

- Supports 802.11 (Wi-Fi):
  Full control over packets down to the byte level, which is ideal to demonstrate vulnerabilities. Native support for crafting and parsing 802.11 frames makes it suitable for Wi-Fi exploits and attack simulations.
- Flexible and extensible:
  Existing layers can be modified to explore advanced network behavior. Integrates seamless into Python workflows, allowing automation.

Drawbacks

- Requires root privileges:
  Sending and sniffing raw packets usually needs elevated permissions, which can be an additional hurdle.
- Inconsistent support for complex Wi-Fi setups:
  Some features or encrypted frames require manual effort or external tools (e.g., aircrack-ng). Differences between Scapy and Wireshark interpretations can lead to trial and error when analyzing packet fields.

4.1.3. Ansible Runner

The anisble-runner [6] library offers a stable and consistent interface abstraction for Ansible. See subsection 4.1.6 for details.

4.1.4. SQLAlchemy

SQLAlchemy [7] is a Python library that abstracts database interactions. In the Wi-Fied Platform, it is used to manage entity storage and retrieval utilizing a SQLite database. For details see subsection 4.1.5.

4.1.5. SQLite

To persist data inside the application, a simple file-based database system with low overhead like SQLite [8] proofed to be a sufficient solution.

Benefits

- Lightweight and self-contained:
  No server setup required and easy to integrate (just a file, no services to manage). Ideal for simple deployment on local machines.
- Sufficient for small-scale use:
  Perfect for local state, configs, or storing exploits and user settings in a lab setup.
- Good Python support:
  Comes with the standard library (sqlite3) and works well with ORMs like SQLAlchemy.

Drawbacks

- Unsuited for concurrent access, limited scalability, fewer advanced features, all data is local:
  All drawbacks are outweighed by the benefits for this specific application in the Wi-Fied Platform. Performance may be impacted on bigger scales, but this should not be a relevant factor in this project.

4.1.6. Ansible

Ansible [9] automates the configuration, deployment, and management of systems using agentless playbooks. In the context of the Wi-Fied Platform, it handles the setup and configuration of the associated hardware components.

Benefits

- Agent-less:
  Requires no software to be installed on the managed devices, SSH connectivity is enough.
- Idempotent:
  Ensures that tasks can be run multiple times without changing the system unless necessary.
- Extensible and well-integrated:
  Supports custom modules and plugins, suitable for tailored educational setups. It is Python-based and uses the declarative markup language YAML to describe the desired state of devices, which is both readable and again familiar to the target audience.

Drawbacks

- Debugging:
  Troubleshooting can be hard, especially with complex custom playbooks.
- Limitations:
  Some native features may not be directly available because of the use of Python library ansible-runner.

4.1.7. Git and GitLab

Distributed source code management with git [10], utilizing GitLab instance of OST [11].
Git was used to version control code and pull playbooks and exploit configurations from GitLab. This integration supports easy extensibility and a clear separation between application logic and content.

Benefits

- Distributed version control:
  Supports powerful workflows (e.g., branching and merging) and integrations (CI/CD pipelines) both during development and for the exploit repository.
- Well-established and widely adopted:
  Git is a well established tool for version control, ensuring broad compatibility and developer familiarity. The target audience is expected to be familiar with it.

Drawbacks

- Not ideal for large binary files:
  Should pose no problems for the Wi-Fied Platform currently, because all exploit configs are text-based files (e.g., YAML). May become relevant with future expansion, depending on features (e.g., deploying different revisions of PDF guides through the exploit repository).

4.1.8. Raspberry Pi 5

Raspberry Pi 5 for hardware lab-devices [12].
Chosen for their affordability, availability, and strong community support. Their small form factor and Linux-based OS make them ideal for flexible, reproducible lab setups involving low-level networking tasks.

Benefits

- Cost-effective:
  Raspberry Pi 5 is an affordable hardware solution, making it ideal for educational projects.
- Compact form factor:
  Its small size allows for easy deployment in tight spaces or mobile setups.
- Community support:
  A large user community provides a wealth of resources, guides, and troubleshooting help. The target audience is likely to be familiar with Raspberry Pi as well.

Drawbacks

- Lack of enterprise-level features:
  Not suitable for large-scale, enterprise-level Wi-Fi security testing. Network or Wi-Fi specific features not natively supported requires software substitution of such functionalities.

4.1.9. Linksys AE3000 Wireless USB adapter

Wi-Fi USB-adapters from Linksys (model AE3000) [13] are used for traffic capture and monitor mode.
For the lab setup with Raspberry Pi's some additional Wi-Fi interfaces are required (e.g., when configured as access point or in monitor mode).

Benefits

- Availability:
  Linksys AE3000 is already in stock at the network lab of OST. Because of this availability it is suited well for this educational purpose.
- Specifications:
  Supports 802.11 a/b/g/n wireless standards and operates on 2.4 or 5 GHz frequencies. Works in monitor mode.

4.2. Iterative Approach to Implementation

As a precursor to the implementation of the project an evaluation period was held to work out the basics of Wi-Fi packet capture and manipulation in Python. This evaluation was based on prior research which the team already had a chance to familiarize themselves with in a previous work [1]. After the completed evaluation a rough idea of a suited software architecture for the project was formed based on the book 'Clean Architecture: A Craftsman's Guide to Software Structure and Design' by Robert C. Martin [2].

However, as the project evolved and the architecture changed drastically multiple times, the team developed a more dynamic iterative approach. In the case of the Wi-Fied Platform it was found, that the general ideas of architecture design are a good guideline. But in order to keep rapid prototyping and continuous improvement of the Wi-Fied Platform codebase viable, the attempt to follow strict patterns was abandoned.

Many of the features where implemented in a few rudimentary steps; First, getting the thing up and running in a standalone / prototype(-ish) setting. Then formalization (e.g., creating repeatable automations through Ansible) of the ingredients and integration into the Wi-Fied Platform. Finally, testing and refactoring of the new additions.

This approach proved its value over time. An initially high effort of getting the project off the ground and intense refactoring of core components have lead to a purpose-built and well-suited architecture.

5. Building Block View

The building block view describes the architecture of the Wi-Fied Platform in more detail. Here the decomposition of the overall system is documented, breaking it up in layers.

For the visual representation of the architecture in the form of diagrams the style of the C4 model [14] was chosen, because it offers a clean design template. Due to this choice the different levels of the architecture diagrams primarily follow the naming notation of the C4 model. However, since this documentation is oriented alongside the arc42 template, references to the corresponding nomenclature in arc42 are given consistently.

The container- (level 1 in arc42) and component-diagrams (level 2 in arc42) follow in their hierarchical order.

5.1. Whitebox Overall System

The top-level Context view (or Scope & Context in arc42) has already been addressed with the diagram in Figure 3.1, which for readers convenience is here shown again (Figure 5.1). It contains the Wi-Fied Platform software system as a blackbox and shows its external software system dependencies.

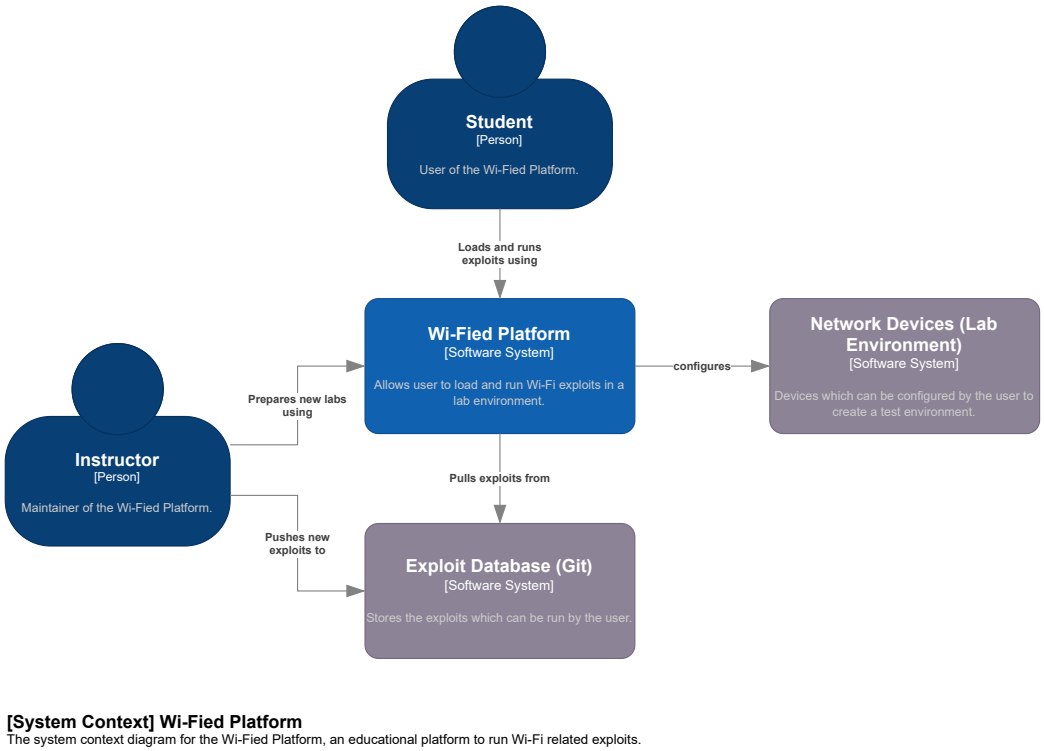


Figure 5.1.: System Context Diagram (repetitive) - C4 model diagram-style, level 1

The overall system whitebox contains the following blackboxes. External software systems (gray boxes in Figure 5.1) will not be covered in more details.

Blackbox Name	**Responsibility**
Wi-Fied Platform*[Software System]*	Extensible platform for educational Wi-Fi security learning.
Network Devices*[ext. Software System]*	Physical hardware setup for lab-devices.
Exploit Database*[ext. Software System]*	Git repository stores predefined exploits.

Table 5.1.: Contained Blackboxes in the Overall System

5.2. Containers (arc42 level 1)

The containers (or building block view level 1 in arc42) of the Wi-Fied Platform are shown in the following diagram, styled according to the C4 model. It is a zoomed-in view into the Wi-Fied Platform software system as depicted in Figure 3.1 and reveals it as a whitebox.

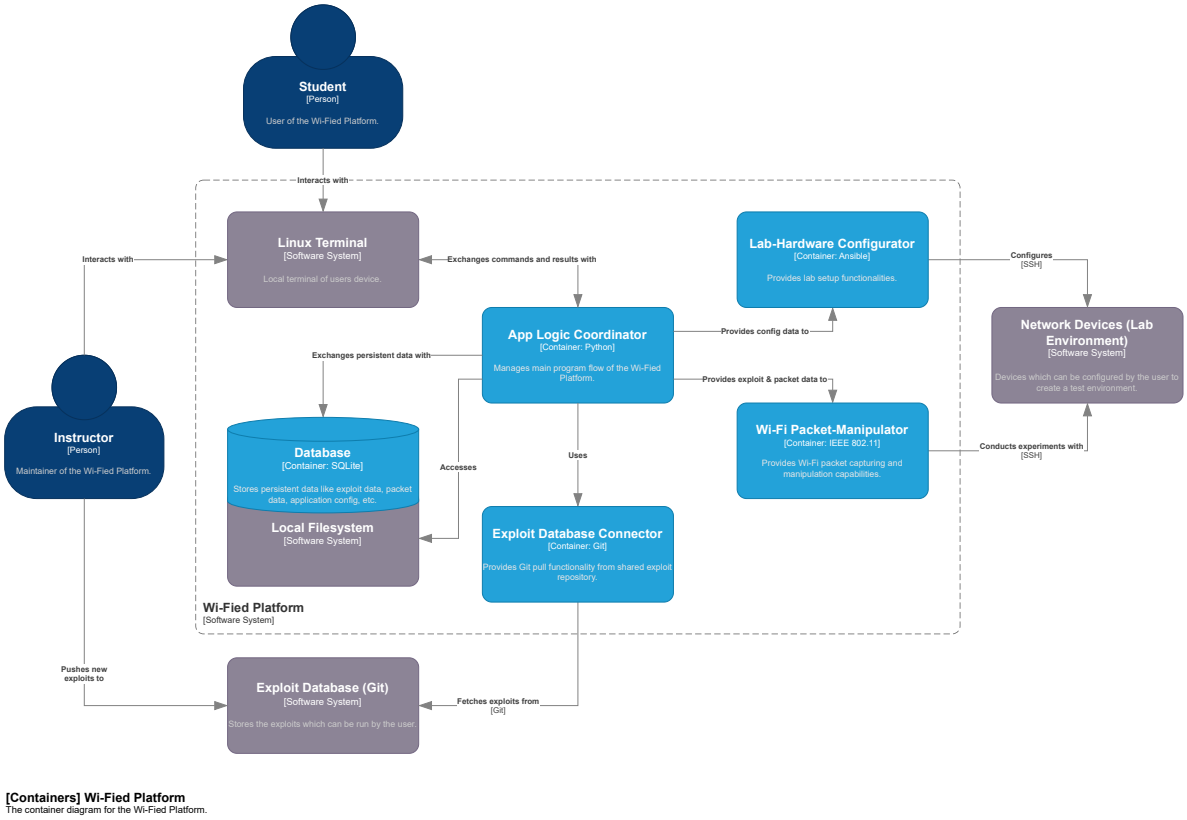


Figure 5.2.: Container Diagram - C4 model diagram-style, level 2

The container diagram (level 1 in arc42) Figure 5.2 contains the following containers as blackboxes. External software systems (gray boxes) will not be covered in more details.

Blackbox Name	**Responsibility**
App Logic Coordinator	Essential central container of Wi-Fied.
Lab-Hardware Configurator	Controls and configures physical hardware (Ansible).
Wi-Fi Packet-Manipulator	Provides Wi-Fi packet capture and manipulation (Scapy).
Exploit Database Connector	Stores predefined exploits (GitHub/GitLab).
Linux Terminal*[ext. Software System]*	Host where the Python app runs and user interacts.
Local Filesystem*[ext. Software System]*	On the host where the Python app runs.

Table 5.2.: Blackboxes in Wi-Fied Platform Software System

5.2.1. App Logic Coordinator

The App Logic Coordinator container is the most important part of the Wi-Fied Platform software system and is broken up further into its components in section 5.3.

Purpose/Responsibility: The App Logic Controller is responsible for the processing of user inputs, the controlling of program flows, the management of configurations, and the interactions with neighboring containers (such as the Lab-Hardware Configurator or the Wi-Fi Packet-Manipulator).

5.2.2. Lab-Hardware Configurator

The Lab-Hardware Configurator interfaces with the physical hardware for the lab-setup.

Purpose/Responsibility: The Lab-Hardware Configurator is responsible for automated configuration and setup of the devices. It sends configs and controls matching the desired behavior/roles of the devices depending on the selected exploits/usecases. This is achieved with Ansible (see subsection 4.1.6).
Because this container builds up on Ansible features and mechanisms (such as playbooks, inventories), there will be no further breaking up into components in this documentation. See Ansible [9] and ansible-runner [6] documentations for further technical details.

5.2.3. Wi-Fi Packet-Manipulator

The Wi-Fi Packet-Manipulator interacts with Wi-Fi (IEEE 802.11) packets.

Purpose/Responsibility: The Wi-Fi Packet-Manipulator is responsible for capturing, manipulating, and sending Wi-Fi packets on the lab wireless network. It utilizes Scapy (see subsection 4.1.2) to fulfill its Responsibilities.
Because this container builds up on Scapy, there will be no further breaking up into components for it in this documentation. See the Scapy documentation [5] for further technical details.

5.2.4. Exploit Database Connector

The Exploit Database Connector provides Git pulling abilities to download predefined exploits from a shared repository.

Purpose/Responsibility: The Exploit Database Connector is responsible for pulling existing exploits from a repository. It utilizes Git (see subsection 4.1.7) and the GitPython library. Because this container builds up on Git and GitHub/GitLab technology, it will not be broken down into components for this documentation. See Git [10] and GitPython [15] for technical details.

5.3. Components (arc42 level 2)

All the components (or building block view level 2 in arc42) of the Wi-Fied Platform are shown in the following diagram, styled according to the C4 model. Zooming further into the App Logic Coordinator container of the previous diagram Figure 5.2 reveals it as a whitebox.

The App Logic Coordinator is the functionally most important and architecturally interesting container of the Wi-Fied Platform software system. Because of its central position it was broken down into its components in this section. All distinct parts of the Wi-Fied Platform functionalities become visible at this level and an overview is given. The components and their respective interactions within Wi-Fied Platform and to external software systems are explained.

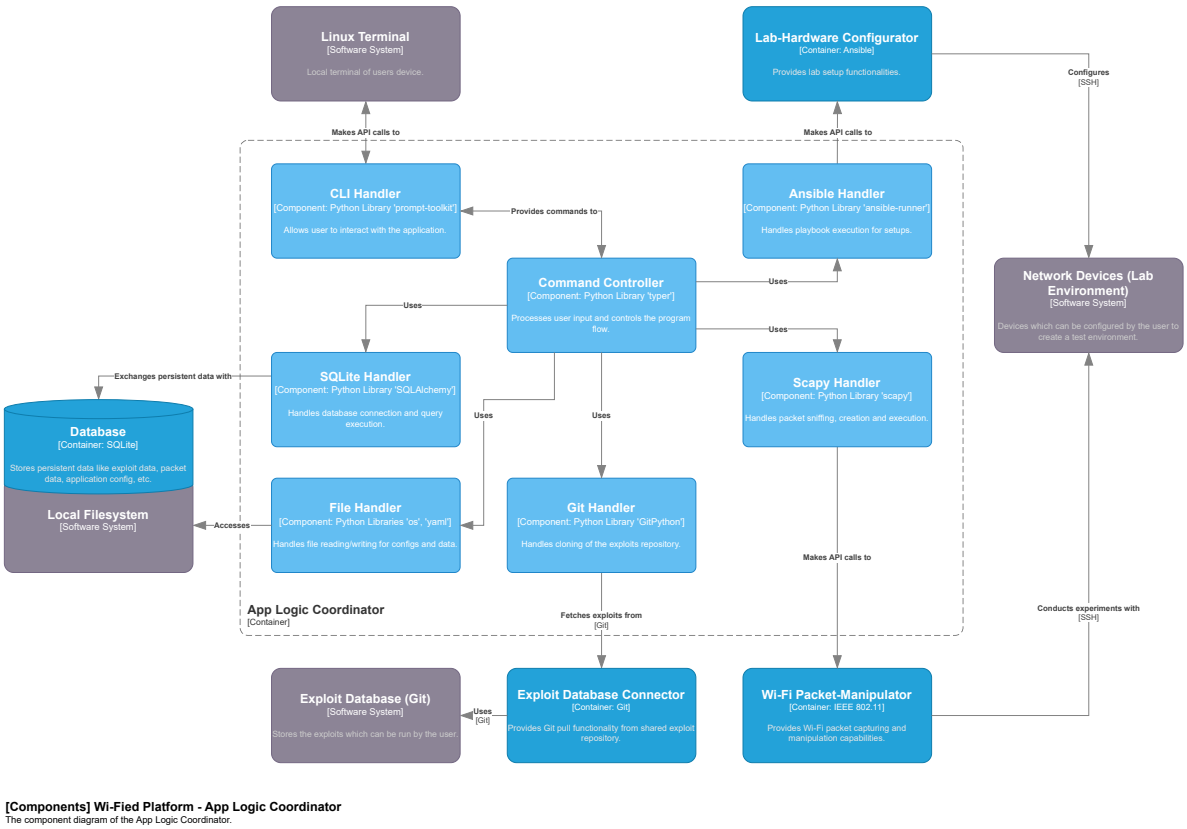


Figure 5.3.: Component Diagram - C4 model diagram-style, level 3

The component diagram (level 2 in arc42) contains the following components as blackboxes from the App Logic Coordinator container.

Blackbox Name	Responsibility
CLI Handler	Enables user interactions.
Ansible Handler	Handles playbook execution.
Command Controller	Process user input and controls program flows.
SQLite Handler	Handles database connections and queries.
Scapy Handler	Handles packet sniffing, creation and sending.
File Handler	Handles file I/O.
Git Handler	Handles cloning of Exploits.

Table 5.3.: Components as Blackboxes in the Component Diagram

5.3.1. CLI Handler

The CLI Handler component controls the commands in the CLI and enables user interactions with the Wi-Fied Platform.

- **Purpose/Responsibility:** The CLI Handler is responsible for the processing of user inputs. Providing autocompletion, handling session states, parsing commands. It is built with the Python library 'prompt-toolkit' [16].
- **Directory/File Location:** Located in `app/handlers/` from the projects root.
- **Fulfilled Requirements:** Helps with *UC1*, by enabling CLI interaction with help texts and meaningful warnings/errors.

5.3.2. Ansible Handler

The Ansible Handler component controls the automated device setup and configuration for the lab-devices required by the Wi-Fied Platform.

- **Purpose/Responsibility**
  The Ansible Handler is responsible for automated setup of lab-devices. Allowing the user to specify target hosts in an inventory file and defining (or reusing existing) roles and configs for the lab-devices. It utilizes the Python library 'ansible-runner' [6].
- **Directory/File Location**
  Located in `app/handlers/` from the projects root.
- **Fulfilled Requirements**
  Helps with *UC4*, by supporting the automated setup of lab-devices and idempotent updating of their configurations.

5.3.3. Command Controller

The Command Controller component controls the processing of user inputted commands and coordination of related actions.

- **Purpose/Responsibility**
  The Command Controller is responsible for mapping user commands and inputs to features/actions of the Wi-Fied Platform. It keeps track of the program flow and exchanges results of other handlers. The Python library 'typer' [17] is essential here.

- **Directory/File Location**
  Located in `app/cli_contexts/` from the projects root.

- **Fulfilled Requirements**
  Helps with most usecases, since Command Controller component plays a key role in managing the various program flows.

5.3.4. SQLite Handler

The SQLite Handler component handles actions related to persistent data.

- **Purpose/Responsibility**
  The SQLite Handler is responsible for the file-based database. It is required to store customized configurations persistently. It utilizes the Python library 'SQLAlchemy' [7].

- **Directory/File Location**
  Located in `app/handlers/` from the projects root.

- **Fulfilled Requirements**
  Helps with *UC5* and *UC3*, because configuring and running exploits as well as pulling predefined exploits requires local persistent storage of data.

5.3.5. Scapy Handler

The Scapy Handler component handles all Wi-Fi packet related actions.

- **Purpose/Responsibility**
  The Scapy Handler is responsible for capturing, creating, manipulating, and sending of Wi-Fi packets. It utilizes the Python library 'scapy' [5].

- **Directory/File Location**
  Located in `app/handlers/` from the projects root.

- **Fulfilled Requirements**
  Helps with *UC5*, because it enables creating and executing exploits.

5.3.6. File Handler

The File Handler component handles local files relative to `app/data/`.

- **Purpose/Responsibility**
  The File Handler is responsible verifying file paths and reading/writing file contents.

- **Directory/File Location**
  Located in `app/handlers/` from the projects root.

5.3.7. Git Handler

The Git Handler component handles cloning of the Exploit repository.

- **Purpose/Responsibility**
  The Git Handler is responsible for pulling predefined exploits from the Exploit repository. It utilizes the Python library 'GitPython' [15].
- **Directory/File Location**
  Located in `app/handlers/` from the projects root.
- **Fulfilled Requirements**
  Helps with *UC3*, by supporting the cloning of predefined exploits.

5.4. Code

The Code view (or building block view level 3 in arc42) will be omitted for the Wi-Fied Platform, because it would not add value to this documentation and to the understanding of the architecture.

6. Runtime View

This chapter outlines selected runtime scenarios of the Wi-Fied Platform to illustrate the dynamic behavior of its components during execution. Rather than exhaustively documenting all possible interactions, it focuses on representative sequences that highlight key architectural responsibilities, such as executing Ansible playbooks, capturing wireless traffic, or running exploit scenarios. These interactions correspond to the most important use cases, include interactions with critical external interfaces, and represent central program flows.

The scenarios listed in Table 6.1 help clarify how components collaborate at runtime. The described interactions serve both as architectural validation and as a foundation for further refinement or troubleshooting.

ID	Scenario Name	Description
RS1	Lab-Devices Setup	Focuses on lab-device setup and configuration, central component is Ansible Handler.
RS2	Pulling Exploits	Showcases cloning of predefined content for use in Wi-Fied Platform, central component is Git Handler.
RS3	Running Exploit	Demonstrates an attack via execution of exploit in Wi-Fied Platform, central component is Scapy Handler.

Table 6.1.: List of Runtime Scenarios

6.1. Runtime Scenario 1 — Pulling Exploits

This runtime scenario describes the flow of pulling exploits from the Exploit Database (GitHub/GitLab repository) into the Wi-Fied Platform.

The relevant components used for this scenario are:
- CLI Handler
- Command Controller
- Git Handler

The following steps define this runtime scenario. The flow diagram in Figure 6.1 visualizes the program flow below:

1. User launches Wi-Fied Platform and interacts with the CLI (CLI Handler involved).
2. User selects commands to pull exploits from the Exploit Database.
3. The CLI Handler receives the command and sends it to the Command Controller which then processes it.
4. The Command Controller determines the Git Handler is responsible and calls it.
5. The Git Handler pulls the Exploit Database repository into local storage (SQLite Handler and File Handler omitted from this flow for simplicity).
6. A pull status is reported back and displayed to the user.

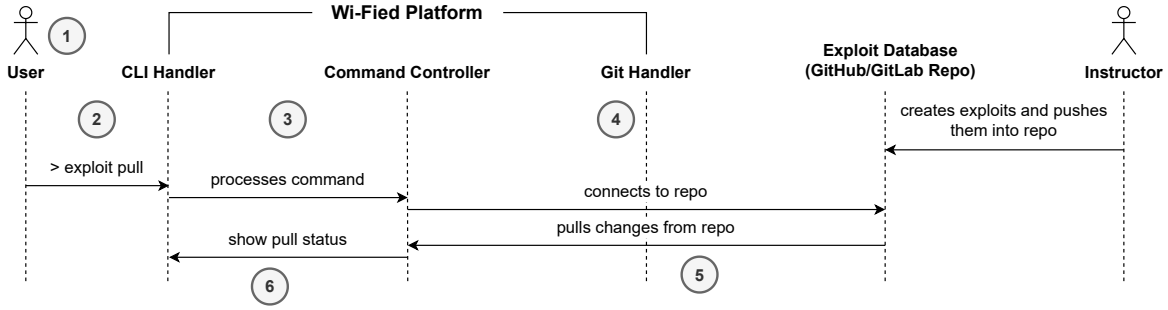


Figure 6.1.: Pulling Exploits Runtime Scenario

6.2. Runtime Scenario 2 — Lab-Devices Setup

This runtime scenario describes the flow of setting up lab-devices through the Wi-Fied Platform. It requires a loaded exploit from which the relevant configuration details for the lab-devices can be accessed.

The relevant components used for this scenario are:

- CLI Handler
- Command Controller
- Ansible Handler

The following steps define this runtime scenario. The flow diagram in Figure 6.2 visualizes the program flow below:

1. User launches Wi-Fied Platform and interacts with the CLI (CLI Handler involved).
2. User selects commands to run the setup (customization of configs omitted for simplicity, assume they are predefined).
3. The CLI Handler receives the command and sends it to the Command Controller which then processes it.
4. The Command Controller determines the Ansible Handler is responsible and calls it.
5. The Ansible Handler runs the specified playbooks on the specified hosts and configures the lab-devices.
6. A task status is reported back and displayed to the user.

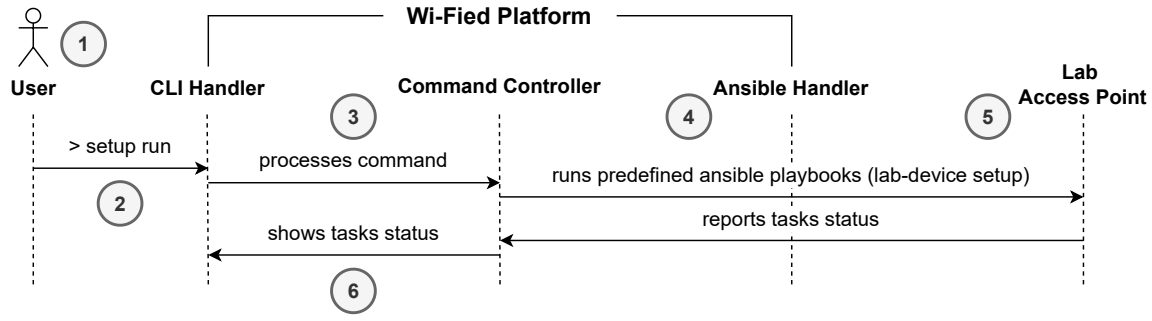


Figure 6.2.: Lab-Devices Setup Runtime Scenario

6.3. Runtime Scenario 3 — Sending Packet

This runtime scenario describes the flow of sending a packet with the Wi-Fied Platform. It requires a loaded exploit from which the packet data can be accessed. In this specific instance a deauthentication attack is being conducted.

The relevant components used for this scenario are:
- CLI Handler
- Command Controller
- Scapy Handler

The following steps define this runtime scenario. The flow diagram in Figure 6.3 visualizes the program flow below:

1. User launches Wi-Fied Platform and interacts with the CLI (CLI Handler involved).
2. User selects commands to run the exploit (customization of configs omitted for simplicity, assume they are predefined).
3. The CLI Handler receives the command and sends it to the Command Controller which then processes it.
4. The Command Controller determines the Scapy Handler is responsible and calls it.
5. The Scapy Handler loads the specified Wi-Fi packet from the database and sends it to the specified address to deauthenticate the victim.
6. Successful sending of packet is reported back and displayed to the user.

The Wi-Fied Platform does not indicate whether this was successful. However, users may observe the victim device being disconnected from the Wi-Fi network.

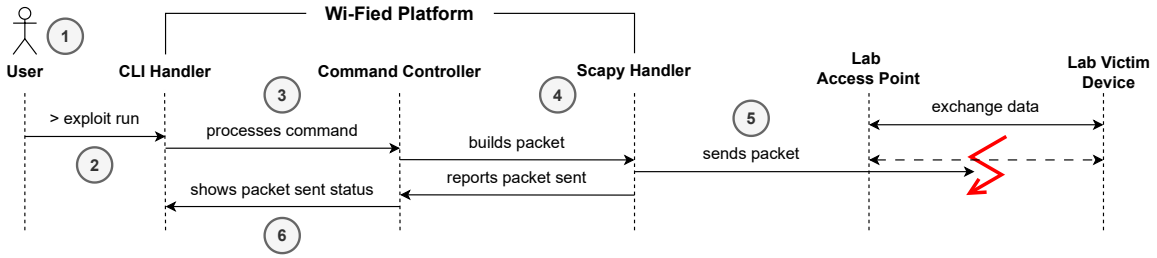


Figure 6.3.: Running Exploit (deauthentication attack) Runtime Scenario

7. Deployment View

This section describes the technical infrastructure involved and how the hardware components of the lab setup interact with each other.

7.1. Deployment Diagram of a Lab Setup

The diagram in Figure 7.1 shows an exemplary setup of lab-devices and how the interaction with the Wi-Fied Platform works.

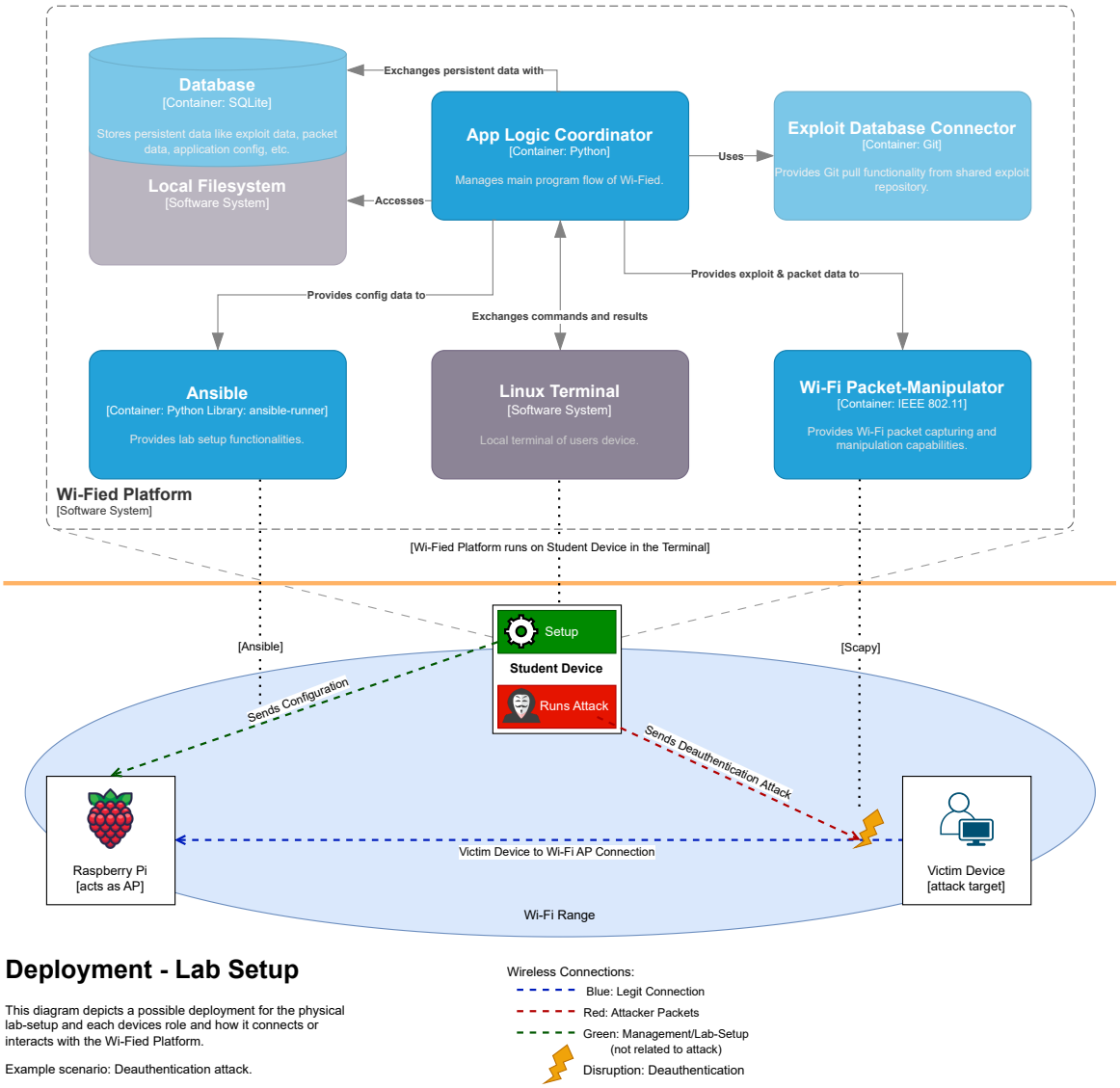


Figure 7.1.: Deployment Diagram — Lab Setup for deauthentication attack

7.1.1. Deployment Diagram Explained

Figure 7.1 depicts a lab setup (lower half) in which one **Raspberry Pi** acts as an access point and a **Victim Device** is connected to that access point will be targeted by the **Student Device**. It also maps relevant components of the Wi-Fied Platform (upper half) to their respective use cases in this attack scenario.

7.1.2. Setup and Attack-Phase

There are 2 phases combined in this diagram, which in practice will often be executed by a user in sequence:

1. **Setup phase** (green box and arrow from the Student Device): Allows to configure the access point Raspberry Pi with the help of Ansible playbooks (SSH connectivity between Student Device and Raspberry Pi required).

2. **Attack phase** (red box and arrow from the Student Device): Student Device runs the attack and effectively disrupts the Wi-Fi connectivity of the Victim Device.

7.2. Hardware Deployment

The Wi-Fied Platform will be deployed in an educational lab setup, and the CLI tool itself runs locally on the users system. Deployment is done with a shell script, which installs prerequisites and clones the repository from GitLab. Python's project management tool 'uv' is utilized to create a virtual Python environment with all dependencies for the Wi-Fied Platform. Lab-devices can vary in number depending on the desired experiment scenario.

Additional hardware components which might be needed for a lab setup can be:
- **Raspberry Pi:** Fulfill different roles, depending on selected exploit/vulnerability (e.g., configured as access point).
- **Victim devices, such as smartphones or tablets:** Targeted by attack and/or to validate/demonstrate effectiveness (e.g., deauthentication attack).
- **USB Wi-Fi adapters:** Used to monitor or capture Wi-Fi traffic.

Other attacks, exploits, or vulnerability demonstrations may have different requirements towards the amount, configuration, and build versions of services or software on the lab-devices. Similar to this example, lab-devices would always have to be prepared first (Phase 1) before the demonstration can be performed (Phase 2). The design of the Wi-Fied Platform allows users to accommodate their specific requirements for an experiment with the flexibility of creating custom Ansible playbooks. Educators are able to prepare setup configurations ahead of time. Without the need to understand the technical details of those setup configurations, students are able to pull them from the exploit repository of the Wi-Fied Platform. This allows quick and easy setup of the lab-devices.

7.3. Internet Connectivity

The Wi-Fied Platform due to its nature could very well be used in a standalone configuration without any outside connection. In order to utilize some features, a basic internet connectivity from the user device (on which the Wi-Fied Platform runs) is required. Pulling exploits with Git from an exploit database, or configuring lab-devices with Ansible playbooks are examples for such features that require internet access.

8. Crosscutting Concepts

This chapter is dedicated to overarching (crosscutting) patterns, rules, and decisions affecting multiple building blocks. Focus is on technical decisions.

8.1. Extensibility Mechanisms

The Wi-Fied Platform is built with extensibility in mind. This lead to a plugin-like design for the content (i.e., exploits) allowing easy expansion. Exploits are defined in YAML-files. These contain all information necessary for an exploit to be loaded, configured, and executed. Packet information for crafting a custom Wi-Fi packet is also included in YAML-files, should an exploit require specific packets.

In order to cater towards the lab-devices requirements for each exploit, file structures of Ansible (e.g., hosts files, playbooks) are also paired with each exploit. This greatly reduces the workload of preparing and configuring the lab-setup before each experiment. Exploits obtained through the Exploit Repository (refer to section 8.3), or ones that have been created manually can be loaded through Wi-Fied Platform's CLI-commands and customized for the local environment.

Wi-Fied in summary provides a uniform structure to store and load exploits and related information. User interactions with the Wi-Fied Platform, as well as interactions of the platform with interfacing systems like physical hardware and wireless networks are streamlined and consistent.

8.2. Tooling Integration

The Wi-Fied Platform is designed to abstract invocations of third-party tools behind Python code to provide a streamlined and consistent user experience. Although commands are simplified through the Wi-Fied Platform's command line, tools like Ansible (see subsection 4.1.6) or Scapy (see subsection 4.1.2) are still integrated according to their respective mechanisms in the background. This ensures flexibility and preserves the reliability of those established tools, while also enabling the aforementioned separation of content (exploit data and Ansible instructions) into separate YAML-files.

8.3. Persistence Strategy

For all data that is loaded into or configured within Wi-Fied Platform through the CLI-commands are stored in a local, file-based SQLite database. SQLite has characteristics that perfectly fulfill the requirements towards persisting data in the context of the Wi-Fied Platform. All benefits and drawbacks of SQLite are described in more details in subsection 4.1.5.

Git is utilized for the distribution of the exploit definitions (YAML-files) and the associated Ansible instructions for the lab-devices. A separate Exploit Repository is used to collect, store, and provide exploits. The Wi-Fied Platform as a tool allows pulling from that Exploit Repository, saving a local copy.

9. Architectural Decisions

This chapter is concerned with architectural decisions. We decided to integrate Y-statements [18] to justify and explain certain central architectural decisions.

9.1. Y-Statement 1 — Python as Implementation Language

This Y-Statement justifies the choice of using the Python ecosystem for the implementation.

Justifying the Decision to Use Python:

In the context of selecting a programming language for the Wi-Fied Platform,
facing the challenge to make the tool accessible to students and the need to make it modular and extensible,

we decided to use Python for the implementation of a local CLI-based tool
and not a statically typed, compiled language or a browser-centric framework

because Python supports rapid prototyping, has a low entry barrier for learners, and offers mature libraries for packet manipulation and networking tasks,
accepting certain drawbacks*, such as lower runtime performance and limited type safety, in favor of development speed and educational accessibility.

*Python (list of benefits and drawbacks) is covered in subsection 4.1.1.

9.2. Y-Statement 2 — Standalone CLI Architecture

This Y-Statement justifies the choice of opting for a standalone CLI architecture without reliance on any backend server infrastructure.

Justifying the Decision to Use a Standalone CLI Architecture:

In the context of deciding on the architectural infrastructure for the Wi-Fied Platform,
facing the challenge to make the tool flexible, educational, and portable,

we decided to go with serverless by design
and not rely on any backend server infrastructure

because this reduces operational complexity and eliminates costs and effort associated with remote services,
accepting to have no central shared state and to require manual distribution of the software.

As a result, the tool is especially suited for lightweight environments and scenarios where minimal infrastructure overhead is a priority.

9.3. Y-Statement 3 — Ansible for Automated Setup

In this Y-Statement the selection of Ansible for automated lab-devices setup is covered.

Note: At first we planned for an implementation with nornir, based on previous knowledge of the tool. Oversights regarding shortcomings of its capabilities have been discovered in early stages of implementation. This Y-Statement clarifies the decision to move away from nornir.

Justifying the Decision to Integrate Ansible:

In the context of requiring a solution to automate lab-device setup with the Wi-Fied Platform,
facing the challenge of making the process user-friendly, repeatable, and agent-less,

we decided to run Ansible via ansible-runner*
and not nornir (as first envisioned and implemented in the evaluation phase),

because nornir lacks idempotence (nornir_netmiko [19]) and we cannot use nornir_napalm [20] on plain Linux OS,
accepting limitations in Ansible's features due to ansible-runner limitations.

*Ansible Runner (list of benefits and drawbacks) is covered in subsection 4.1.3.

9.4. Y-Statement 4 — Scapy for Packet Manipulation

We decided to use Scapy for packet manipulation, in order to have full control over IEEE 802.11 frames and implement flexible attack scenarios.

Justifying the Decision to Integrate Scapy:

In the context of adding features for capturing, crafting, and manipulating of Wi-Fi packets to the Wi-Fied Platform,
facing the challenge to work closely at the IEEE 802.11 standard specifications,

we decided to utilize Scapy*
instead of looking at other libraries,

because it has been introduced in module exercises, and it also sufficed in our evaluation,
accepting the (subjectively) hard to read and sometimes spotty documentation and community support of it.

*Scapy (list of benefits and drawbacks) is covered in subsection 4.1.2.

No alternatives to Scapy were considered, because the teams' familiarity with the library was prioritized.

9.5. Y-Statement 5 — SQLite for Persistent Storage

We decided to store configuration in SQLite, in order to provide a lightweight, file-based persistence layer without requiring external services.

Justifying the Decision to Persist Data in SQLite:

In the context of requiring persistent storage of some sort to save configurations of the Wi-Fied Platform,
while aiming to keep the overhead of the locally running tool low,

we decided to persist local data in a SQLite* file-based database
and not to use a database system that required a background-service and credentials,

because no concurrent, multi-origin database transactions will occur,
accepting the drawbacks of file-based databases and not being able to run concurrent app instances.

*SQLite (list of benefits and drawbacks) is covered in subsection 4.1.5.

10. Quality Requirements

This chapter lists the measures that ensure code and project quality and efficiency. In addition, these measures indirectly reduce the listed risks: High code quality ensures code maintainability and expendability.

10.1. Non-Functional Requirements (NFR)

The following non-functional requirements where chosen with a school environment in mind. The application should meet the users' and instructors needs.

NFR-1 - Extensibility	
Description	The application should have a modular structure so that it can be easily expanded.
Measurability	The architecture will be designed accordingly and reviewed in team sessions.
Priority	High
Review	During the creation of the architecture with periodic verification during development.

Table 10.1.: NFR1

NFR-2 - Usability	
Description	The user interface should be designed user-friendly, this includes (but is not limited to): 1. Comprehensive user manual (e.g., help page), 2. Clear progress indication, 3. Detailed error messages.
Measurability	Will be tested through user tests at the end of the development phase.
Priority	High
Review	See user tests in appendix.

Table 10.2.: NFR2

NFR-3 - Maintainability	
Description	The used libraries should be up-to-date and well maintained to make sure that the application can be maintained and updated easily.
Measurability	Will be verified when choosing the library.
Priority	Medium
Review	During development.

Table 10.3.: NFR3

10.2. Quality Scenarios

Quality scenarios are concrete examples that describe how the Wi-Fied Platform should respond to specific situations related to non-functional requirements. They help clarify abstract quality attributes like performance, extensibility, or usability by turning them into actionable, testable statements.

ID	Scenario
SC1 *Change Scenario (see NFR1)*	The architectural design should allow extensions without affecting current functionality or requiring rebuilding core components.
SC2 *Usage Scenario (see NFR2)*	An untrained user should be able to understand and use the Wi-Fied Platform with the help command provided in the CLI within a lab-setting.
SC3 *Change Scenario (see NFR3)*	The modular architecture should enable flexibility in adapting to other environments or making replacing 3rd-party libraries possible without requiring changes to other components.

Table 10.4.: Quality Scenarios (SC = Scenario)

10.2.1. Change Scenario SC1

This change scenario (SC1) supports *NFR1*:

- **Context/Background:** Given the limited project resources, future extension should be as simple as possible. Wi-Fied Platform will be used in an educational context and shall be reflective of an extensible architectural design.
- **Source/Stimulus:** Future work may include extension of the Wi-Fied Platform's functionalities. This could be initiated by other students in the form of a project or by the advisors/tutors of modules utilizing Wi-Fied Platform in their curriculum.
- **Metric/Acceptance Criteria:** The architectural design should allow extensions without affecting current functionality or requiring rebuilding core components. Specifically the existing features should not be impacted by changes not affecting their modules directly. No full-scale refactoring should be required to make an extension work.

10.2.2. Usage Scenario SC2

This usage scenario (SC2) supports *NFR2*:

- **Context/Background:** Wi-Fied Platform is targeted to an educational context for Wi-Fi security in a lab-setting for learning IT professionals and/or students. It is important to enable users to quickly operate the Wi-Fied Platform without much background knowledge. Although users can ask questions in such an environment, it is expected from Wi-Fied Platform to be as simple to use as possible.

- **Source/Stimulus:** Any user (learning IT professionals or students, and tutors) interacting with the Wi-Fied Platform. Usually first-time users of the Wi-Fied Platform with limited experience in the field of Wi-Fi security will fall under this usage scenario.

- **Metric/Acceptance Criteria:** An untrained user should be able to understand and use the Wi-Fied Platform with the help command provided in the CLI within a lab-setting. It is aimed towards users who previously operated other CLI-based tools. For users with no prior experience with CLI-based tools, personal instructions or further handouts are expected to be required.

10.2.3. Change Scenario SC3

This change scenario (SC3) supports *NFR3*:

- **Context/Background:** Adaption of Wi-Fied Platform's features of the initial release may be desired in the future, given the limited project scope regarding supported hardware, predefined exploits and user interface. Wi-Fied Platform should aim towards an architectural design that is maintainable and can be approached and understood by other coders.

- **Source/Stimulus:** Tutors or advisors of a lecture/module may want to implement future changes to make Wi-Fied Platform better fit into their curriculum. Future student projects may be another scenario in which changes to Wi-Fied Platform may occur.

- **Metric/Acceptance Criteria:** The modular architecture should enable flexibility in adapting to other environments or making replacing 3rd-party libraries possible without requiring changes to other components.

10.3. Quality Tree

The projects main quality goals (1), (2), (3) in the following table refer to the top level quality requirements from Table 1.1. The following Table 10.5 lists and categorizes quality attributes to give an overview in form of a quality tree. Where applicable, separate specialized quality scenarios are referenced.

Category	Quality	Description	Scenario
Usability	Ease of Use (1)	Wi-Fied Platform's functionalities should be comprehensive and students familiar with CLI-tools should be able to use it within 10 minutes.	*SC2*
	Learnability	The CLI-interface should be intuitive and adhere to familiar practices (e.g., help command).	
	Simplicity	The platform and its features should be easy to use and complexity should be simplified for a novice user without deep understanding of Wi-Fi security.	
Extensibility & Maintainability	Extensibility (2)	The architecture of the platform should follow design principles to support future extension.	*SC1*
	Modularity (3)	The platform should follow a modular design, allowing future modification.	*SC3*
	Reusability	Code and configuration should aim to be flexibly reusable for different (future) scenarios.	
	Maintainability	The platform should aim to be low maintenance, without unnecessary hard dependencies.	
Automation	Scriptability	Reoccurring setup tasks and configurations should be automated and safe the users time.	
	Portability	Lab- and test-devices should be interchangeable.	
	Idempotence	Configurations consistently should only be applied once or if parameters changed.	
Performance	Robustness	Wi-Fied Platform should work reliable under all specified use cases and produce comprehensive outputs (e.g., error handling).	
	Consistency	Operations should perform consistently in time (progress indication).	
	Reliability	Functionalities should be working as expected and yield dependable results.	

Table 10.5.: Quality Tree

10.4. Quality Assurance

This section discusses the measures to achieve quality goals and methods to verify the achievement of quality goals.

10.4.1. Code Quality

Measures were put in place to try and keep a certain code quality standard. Ruff was used as a Python code formatter and linter, ensuring uniform code formatting. It was implemented using a pre-commit hook.

Python pytests were added for important modules to help verify feature compatibility between iterations throughout the project. The tests are run in the CI&CD pipeline and their results can be viewed in GitLab.

Additionally, the code gets analyzed by SonarQube, which scans for bugs, security vulnerabilities, bad practices and more. It also evaluates the test coverage; although, complete coverage is not achieved due to the team's prioritization of research, focus on usability, and proof of concept through the creation of an instructional lab. SonarQube is also part of the CI/CD pipeline.

10.4.2. User Tests

User tests were conducted to validate the implementation and confirm that the platform's usability meets the expectations of its intended audience; primarily students. Six participants with professional IT backgrounds, spanning roles from network infrastructure administration to frontend development, took part in the evaluation.

The special emphasis on user tests aligns with the project's primary quality goal (Table 1.1) to achieve high usability & learnability standards. Usage scenario *SC2* defines the general direction of the user tests.

All feedback gathered through the user tests did not reveal any critical issues affecting the architecture or core design of the Wi-Fied Platform. While minor usability suggestions were collected, they led only to small refinements that did not require architectural adjustments. The tests therefore served as a confirmation of both the tool's usability and the stability of its architectural decisions.

11. Risks and Technical Debt

This chapter is concerned with identified technical risks or technical debts in descending priority.

11.1. Technical Risks

The following technical risks have been identified. Priority is descending, starting with the highest priority technical risk.

11.1.1. Toolchain Fragility

Reliance on external tools like hostapd, dnsmasq, and wpa_supplicant could break with OS updates. Also, tools and libraries such as ansible-runner pose similar risks when functionality may change with updates.

Mitigation: Many exploits require specific versions of tools like hostapd to be effectively vulnerable, because fixes have been applied. In order to reduce the risk of breaking Wi-Fied exploits with changing software releases, specific versions may be pinned or built from an archived source to ensure compatibility with the desired vulnerabilities for the demonstration purpose inside the Wi-Fied Platform. Ansible supports not only automation of current releases through apt, but also allows automating custom installations from different sources through direct shell commands. With strategies like this, and less reliance on individual packet distributors definition of 'latest' or different versioning schemes, the Wi-Fied Platform ensures a certain degree of autonomy.

11.1.2. Hardware Compatibility

Raspberry Pi models or Wi-Fi chipsets may behave inconsistently (e.g., monitor mode support varies).

Mitigation: A strict recommendation regarding hardware selection is given, where tested hardware during development is limited (refer to subsection 4.1.8 and subsection 4.1.9). This risk was an early concern and broader hardware support was considered out of scope for this project.

Future work may involve automated compatibility checks (e.g., whether and USB Wi-Fi adapter supports monitor mode) or a wider test envelope to verify additional hardware models.

11.1.3. Security of Local Execution

Executing CLI-based tools from user-controlled environments may be abused or tampered with. Inexperienced users could also accidentally damage their system by misconfiguring the environment.

Mitigation: This risk is not mitigated, because of impracticalities. The tool is not considered to be presented to complete beginners. Also, not restricting the local execution can be viewed as a necessity for the effectiveness of the tools features. In the educational context, it should be possible to provide sandboxed / imaged lab-computers for students, which greatly weakens this risk.

11.2. Technical Debts

Table 11.1 lists technical debts that have been identified. Priority is descending, starting with the highest priority technical debt.

Description	Future Direction
Limited Immediate Utility*: The current version primarily establishes a structured, extensible foundation for future capabilities. Limited advantages over direct use of Ansible or Scapy are presently offered. This initial design prioritizes architectural groundwork over short-term feature completeness.	Incrementally expand the platform's functionality, usability, and exploit scenario coverage to deliver increasing value to users over time.
No (Central) Persistent State or Logging: Current CLI design may not persist useful run-time data (e.g., session logs, telemetry). The reason for this are simplicity, serverless constraints.	Optional local state caching or remote logging.
Missing Automated Tests for Hardware-Dependent Code: Most testing is manual due to hardware requirements.	Future work may involve automated compatibility checks and a wider test envelope to verify additional hardware models.
Scalability Assumptions: The Wi-Fied Platform assumes single-user, single-device environments.	Consider support for concurrent testing setups or remote execution agents.

Table 11.1.: Technical Debts

*** More on Foundational Phase with Limited Immediate Utility:**
Although the current version of the Wi-Fied Platform offers only limited advantages compared to directly using existing tools such as Ansible or Scapy, this is an intentional and strategic decision made during the project's early development phase. The primary focus has been to design and implement a flexible, modular architecture that can serve as a solid foundation for future expansion.

By emphasizing extensibility and maintainability over immediate utility, the project has laid the groundwork for a more comprehensive and user-friendly tool. This foundational approach enables the platform to evolve organically, allowing new features, scenarios, and integrations to be added with minimal refactoring effort. However, this architectural focus introduces a temporary trade-off: The platform's current user-facing value may appear limited in scope.

To address this, future development should concentrate on incrementally building out high-value features, improving usability, and demonstrating clear benefits through practical use cases. As the platform matures, these enhancements will progressively increase its appeal and relevance to target users.

12. Glossary

List of Terms

access point A device or software service that provides wireless network access to clients. In this project, a Raspberry Pi acts as a software-based access point. 22, 35

ansible An automation tool used for system configuration and deployment. In this project, it automates setup of access points and services for the lab-devices. 20, 21, 22, 25, 27, 30, 32, 35, 36, 38, 45, 46

deauthentication attack A denial-of-service technique in Wi-Fi networks that sends forged deauthentication frames to forcibly disconnect clients from an access point. 33, 34, 35, 67

KRACK Key Reinstallation Attack (KRACK) is a vulnerability in WPA2 that exploits key reinstallation during the 4-way handshake to decrypt and manipulate Wi-Fi traffic. v, vi, 63

monitor mode A wireless network interface mode that enables passive capture of all 802.11 frames in range, including management and control frames. 7, 8, 17, 19, 22, 45

scapy A Python-based packet manipulation tool used for crafting, sending, sniffing, and dissecting network packets. 19, 20, 25, 28, 30, 33, 36, 38, 46

References of Part *Product Documentation*

[1] A. Glaus and M. Burger, *Wi-fi security threats - an integrative review*, https://eprints.ost.ch/id/eprint/1241/, Accessed: 2025-05-03, 2024.

[2] R. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design* (Robert C. Martin series). Prentice Hall, 2017, ISBN: 9780134494272.

[3] D. G. S. Dr. Peter Hruschka *et al.*, *Arc42 - architecture documentation template*, https://arc42.org, Accessed: 2025-05-03, 2025.

[4] Python Software Foundation, *Python language reference, version 3.11*, https://docs.python.org/3/reference/, Accessed: 2025-05-04, 2023.

[5] *Scapy documentation, python library*, https://scapy.readthedocs.io/en/latest/, Accessed: 2025-05-11, 2025.

[6] *Ansible runner documentation, python library*, https://ansible.readthedocs.io/projects/runner/en/latest/, Accessed: 2025-05-11, 2025.

[7] *Sqlalchemy documentation, python library*, https://docs.sqlalchemy.org/en/20/, Accessed: 2025-05-11, 2025.

[8] *Sqlite documentation*, https://www.sqlite.org/docs.html, Accessed: 2025-05-27, 2025.

[9] *Ansible community documentation*, https://docs.ansible.com/, Accessed: 2025-05-11, 2025.

[10] *Git documentation*, https://git-scm.com/doc, Accessed: 2025-05-11, 2025.

[11] *Gitlab instance of ost*, https://gitlab.ost.ch/, Accessed: 2025-05-27, 2025.

[12] *Raspberry pi website*, https://www.raspberrypi.com/, Accessed: 2025-05-27, 2025.

[13] *Linksys ae3000, usb wi-fi adapter, page 73*, https://downloads.linksys.com/downloads/userguide/AE3000_UG_3425-01611A_Web.pdf, Accessed: 2025-05-27, 2025.

[14] *The c4 model for visualising software architecture*, https://c4model.com/, Accessed: 2025-05-11, 2025.

[15] *Gitpython documentation, python library*, https://gitpython.readthedocs.io/en/stable/, Accessed: 2025-05-11, 2025.

[16] *Prompt toolkit documentation, python library*, https://python-prompt-toolkit.readthedocs.io/en/master/, Accessed: 2025-05-11, 2025.

[17] *Typer documentation, python library*, https://typer.tiangolo.com/tutorial/, Accessed: 2025-05-11, 2025.

[18] *Architecture decision record template: Y-statements | zio's blog*, https://medium.com/olzzio/y-statements-10eb07b5a177, Accessed: 2025-05-13, 2020.

[19] *Nornir netmiko documentation, python library*, https://ktbyers.github.io/netmiko/docs/netmiko/index.html, Accessed: 2025-06-01, 2025.

[20] *Nornir napalm documentation, python library*, https://nornir.tech/nornir_napalm/html/api/index.html, Accessed: 2025-05-18, 2025.

Part II.

Project Documentation

1. Bachelor Project Assignment

The following section provides a partially summarized overview of the bachelor project assignment for the Wi-Fied Platform - An Educational Platform for Wi-Fi Security. The full original bachelor project assignment document is included in the Appendix **??**, **??**.

Problem Description
The goal of this project is to develop a modular platform that allows students to practically learn about different attack vectors on Wi-Fi protocols.

Task Description
The proposed tasks are as follows: Find a didactically meaningful approach for students to learn more about the vulnerabilities, e.g. in the form of exercises using the platform to execute an attack step by step.
Make the platform easily extendable, e.g., by providing a structure to add a UI.

- Identify a set of relevant Wi-Fi vulnerabilities and list the necessary hardware components.
- Build the necessary software components to manage the target and attacker systems of the platform.
- Find a didactically meaningful approach for students to learn more about the vulnerabilities.
- For example:
  - Define exercises that students can do with the platform.
  - For each exercise provide a how-to-guide and explanations.
  - Overall, this could come in the form of a book (presented as a webpage).
- Document your project.

Project Results
The documents mentioned in §5.5 of "Leitfaden für Bachelor- und Studienarbeiten, v1.3" must be submitted as part of the results. Note the requirements on the report mentioned in the Leitfaden. In addition, the results include the educational platform mentioned above.

Involved Parties
This project is as a Bachelorarbeit conducted by Alice Glaus and Mario Burger. The project is supervised by Daniel Tschudi. The expert is Bernhard Tellenbach. 'Gegenleser' is Olaf Zimmermann.

2. Project Plan

Due to limited team size and the scope of this project, a flexible approach inspired along SCRUM was used. Iterative sprints allowed the team to explore technical options and to react to evolving requirements. This flexible approach enabled project progress to be managed efficiently.

2.1. Target Group Declaration

The Wi-Fied Platform is developed for use in educational settings, focusing on practical exposure to Wi-Fi vulnerabilities and exploits. The project prioritizes architecture and extensibility over feature completeness. It should address the core needs of three target groups: Educators, students, and tool maintainers. The list of stakeholders in subsection 1.2.1 of the architecture documentation also reflects this target groups.

Educators / Instructors
Educators may use the platform as a lightweight teaching aid for introducing wireless security concepts. Rather than offering a polished lab framework or all-in-one suite, it should provide basic exploit scenarios and helpful tooling that could support lessons or demonstrations. Setup and compatibility may require preparation effort, and some exercises may depend on specific hardware or software versions. The tool is expected to be most useful where flexibility and openness are valued more than full stability or automation.

Students / Learners
Students are the primary audience. The platform aims to provide guided experimentation with real Wi-Fi attack techniques in controlled settings. It assumes some prior technical familiarity and is not aimed at complete beginners. The focus is set on clarity, simplicity, and hands-on understanding, rather than polished user interfaces or error-proof workflows. With suitable supervision or documentation, the platform may encourage exploration, failure, and learning by doing.

Tool Maintainers / Developers
Developers extending or maintaining the platform will find a modular codebase. As much of the platform interacts with hardware and external tools, some testing might remain manual, and state persistence or error handling can still be minimal. Nonetheless, the project aims to deliver a clean architectural foundation intended for gradual improvement. Its design should support contributions and future expansion without needing major rewrites.

2.2. Project Resources

The Wi-Fied Platform is developed as a student-led project with limited but focused time resources, approximately 360 hours per person (two people = 720 hours total) over 17 weeks. It operates without a formal budget, relying entirely on open-source software. Four Raspberry Pi's are purchased for this project via the university's procurement, further hardware components such as USB Wi-Fi adapters are already available at the INS [21] lab.

The student team is free to contact the dedicated advisor or other educators at OST for guidance at any time during the project.

2.3. Rough Planning

This section outlines the overall project timeline (see Table 2.1). The bold horizontal lines mark the end of a sprint in Scrum and the beginning of a new one. Dark-colored cells indicate the current work focus, while light-colored cells are secondary.

Week	Date of Mon.	Comments	Requirements Analysis	Architecture	Tooling / Infrastructure	Evaluation of Python	Implementation	Risk Buffer*	Usability Testing	Delivery	Presentation Preparation
W1	17. Feb.										
W2	24. Feb.										
W3	03. Mar.										
W4	10. Mar.										
W5	17. Mar.										
W6	24. Mar.										
W7	31. Mar.										
W8	07. Apr.	Spring Break									
W9	14. Apr.	Fri. holiday									
W10	21. Apr.	Mon. holiday									
W11	28. Apr.										
W12	05. May										
W13	12. May										
W14	19. May										
W15	26. May										
W16	02. Jun.										
W17	09. Jun.	Mon. holiday									
Hour Estimates		**Total 720h**	56h	77h	14h	84h	300h	48h	42h	84h	15h

Table 2.1.: Rough Planning Project Timeline

*A buffer in case one or multiple risks occur. In case that no risk (see section 3.4) needs to be mitigated, this time can be used for bug-fixing or further feature implementation.

2.4. Project Achievements

The following achievements highlight key progression and deliverables that heavily influenced the development of the Wi-Fied Platform.

ID	Name or Description	Achieved	Phase Completion & Remarks
1	Project setup, initial planning, and requirements definition	02. Mar.	Foundation established, risks and hardware constraints identified
2	Evaluation of tools	23. Mar.	Decision regarding tools that will be used to realize the project was made
3	Reach MVP	2. May	Minimal Viable Product reached
4	Usability testing	6. Jun.	Scenario walkthroughs validated with test users; identified guidance gaps
5	Research exploits and create a lab	9. Jun.	Suitable exploit for lab-exercise found, and described
6	Final delivery and documentation	13. Jun.	Submission complete; documentation includes risks, usage, and further directions

Table 2.2.: Project Achievements Overview

3. Project Risks

This chapter is concerned with risks associated with the project and its implementation. Technical risks are described in chapter 11 of the product documentation.

3.1. Risk Management

The following risks where determined at the beginning of the project, if not stated otherwise.

R.1 Technical Risks

R.1.1 Hardware Compatibility Issues: The app may not work consistently across different hardware models, leading to unexpected failures.
R.1.2 Unexpected Hardware Behavior: Unforeseen behavior of software components in used Hardware could lead to delay in the time plan.
R.1.3 No Vulnerable Firmware Available: Latest firmware of access points and clients were patched, and old vulnerable ones are not available anymore.
R.1.4 Latency & Performance Issues: Real-time Wi-Fi testing could be affected by delays in data collection or processing.
R.1.5 Infrastructure Disruptions: Important infrastructure becomes degraded or unavailable leading to delays or workarounds.
R.1.6 Idempotence not Guaranteed*: Reaching the goal of making automated lab-setups idempotent is recognized to be impossible with Nornir (risk added 7. April 2025).

*The risk R.1.6 was added on 7. April 2025, because this marks the realization of its lack of idempotence support on plain Linux OS. Switch to Ansible.

R.2 Project Objective Risks

R.2.1 Extensibility Challenges: Ensuring the application remains easily extendable could be difficult if the architecture is not well-designed.
R.2.2 Data Privacy Risks: As we sniff network traffic, we might be confronted with complaints regarding the privacy law.
R.2.3 Difficulties in Finding Suitable Exploits for Proof of Concept:** Researching a didactically valuable and doable exploit to formalize into instructional lab exercises for a proof of concept (risk added 18. May 2025).

**The risk R.2.3 was added on 18. May 2025, because at that time much effort went into research suitable exploits without acceptable results. Because of time restrictions, the risk was added to be documented.

R.3 Project Member related Risks

R.3.1 Limited Team Size: With only two people, workload distribution and expertise gaps could slow down development.
R.3.2 Miscommunication: Miscommunication in team syncs could lead to delays in the time plan.
R.3.3 Absence: If a team member is absence for a longer period of time (due to e.g., sickness, injury) the time plan might not be adhered.

3.2. Risk Matrix

The risks were graded regarding the two properties 'likelihood of occurrence' and 'impact on the time plan' by assigning an integer between one (lowest) and five (highest) as show in Table 3.1. The score is calculated by multiplying these integers. Depending on this score the risks can be categorized in three sections:

- **Scores up to 4 (green zone):** Moderate impact, can be dealt with in a reasonable amount of time.
- **Scores up to 9 (yellow zone):** Medium impact, can still be dealt with but need more time to adhere to especially when more than one risk of this category occurs.
- **Scores up to 25 (red zone):** High impact, should be mitigated ahead of time if possible.

Impact — — → Likelihood ↓	**1 Insignificant (4h)**	**2 Minor (8h)**	**3 Significant (16h)**	**4 Major (24h)**	**5 Severe (40h)**
5 Almost Certain			R.1.6	R.2.3	
4 Likely		R.1.4			
3 Moderate	R.1.5	R.1.2	R.1.1, R.1.3	R.2.1	
2 Unlikely		R.3.1	R.3.2	R.2.1-M*	
1 Rare		R.2.2		R.3.3	

Table 3.1.: Risk Matrix

*M = Mitigated, original risk-matrix severity changed.

3.3. Precautionary Actions

Risk R2.1 needs to be treated with caution, since it scores 12 on the Risk Matrix and is therefore in the red zone. In order to reduce the likelihood of this risk, the following measures were taken.
R2.1-M* Mitigation Strategy - Ensuring Extensibility and Solid Architecture

1. Reevaluate central design decisions before any major changes to the architecture. Cross-checks within project meetings.
2. Enforce adherence to good architectural principals throughout development. Code reviews will be an integral part of the implementation phase.
3. Add supportive tools, such as SonarQube, to automate and simplify code quality checks.

3.4. Risk Occurrence and Mitigation

Of the defined risks, two became a concern (and one that was previously not accounted for) during the development and one had to be revisited in time to plan adequate mitigation action in the risk-buffer phase:

- **R1.2 Unexpected Hardware Behavior:** Ansible was used to set up the labs, which means handling errors related to the tool. The time needed for this kind of error handling was included in the time calculated for implementing the Ansible handler. Unexpectedly, the Raspberry Pi's did not always behave as anticipated. After running a setup successfully, running it a second time to check idempotence exited in an error, as writing to the device was suddenly not possible anymore. This issue still persists, as we were not able to determine its source.
  Risk revision during risk-buffer phase: The team was able to fix the issue during the allocated time in the risk-buffer phase. The cause was a faulty configuration in one of the Ansible tasks, that lead to broken environments on the configured lab-devices.
- **R1.5 Infrastructure Disruptions:** Due to a scheduled maintenance OST's GitLab was down for two days, but complication occurred, and the service was unreliable for two more days. The impact was not critical, but we were losing time, as we first looked for the issue on our side.
- **R.1.6 Idempotence not Guaranteed:** Switch to Ansible was undertaken, allowing idempotence to be guaranteed for the lab-setups, accepting spending additional time for refactoring.
- **R.2.3 Difficulties in Finding Suitable Exploits for Proof of Concept:** Extra time was allocated to the research and implementation of a suitable exploit, cutting short the priority to achieve high test coverage.

4. Quality Measures

The following sections describe the measures taken to ensure the quality and usability of the written code.

4.1. Code Quality

Good code structure and adherence to best practices promotes maintainability and extensibility of the codebase.

4.1.1. Ruff

Ruff is used as a Python code formatter and linter, helping to verify and enforce that code is clean, consistent, and adheres to best practices. It is implemented as a pre-commit hook, which means it automatically checks and formats code before it is committed to version control, reducing the chances of style issues or errors being introduced. This integration into the development workflow promotes a higher standard of code quality by catching issues early. Using Ruff helps maintain a uniform coding style, making the codebase easier to read, understand, and maintain in the future.

4.1.2. SonarQube

To ensure the code quality, the code gets also evaluated by SonarQube. As part of the CI/CD pipeline, changes get synced to a SonarQube instance which scans for bugs, security vulnerabilities, bad practices and more. The feedback can be checked by team members and improved in the next update.

4.1.2.1. Quality Goals

SonarQube defines 'Quality Gates' to assess if the code provides sufficient quality. The built-in one, called 'Sonar way', is used for this project. It requires the code to align with certain measures:

- New code has 0 issues.
- All new security hotspots are reviewed.
- New code has sufficient test coverage: Coverage is greater than or equal to 80.0%.
- New code has limited duplications: Duplicated Lines (%) is less than or equal to 3.0%.

4.1.2.2. Final State

Figure 4.1 shows SonarQube's evaluation at the end of the project.

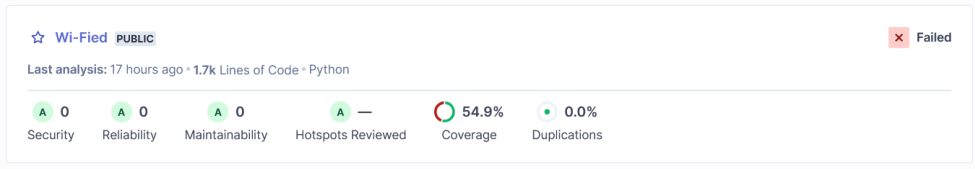


Figure 4.1.: SonarQube Final Evaluation

The code coverage is rated around ∼50% therefore the Quality Gate is marked as 'Failed'. As the project had a significant research part which entailed many code refactors, keeping the tests up to date was challenging. After reaching the MVP the team focused on usability tests and adding a proof of concept in form of a lab.

4.2. Usability Testing through User Tests

Usability tests were conducted, serving both as a validation of the implementation and confirmation of usability by the target group most involved with the tool; students and learners. A total of six test candidates were asked to take part and kindly offered their time. All testers work in the IT sector but in different fields reaching from network infrastructure administration to front end engineering.

4.2.1. Test Execution Strategy

Note: The team initially revealed very little information about features, in order to gain data based on truly inexperienced user actions. This decision helped to test the usability under extreme conditions that would normally not occur. In real scenarios, e.g., in a lab-session, users would always have access to instructional materials and an advisor.

Prior to the practical tasks, participants were asked some questions targeting their general know-how of Wi-Fi networking, preference and usage of CLI tools, and which CLI tools they would regularly use. These general questions helped the team to better comprehend the actions and thinking of candidates, and how to guide them through tasks in case they require support.
After the questionnaire, the following four tasks were given to the testers to determine, whether the flows are intuitive and the correct commands and options can be found without additional inputs:

- Discover how to list all available commands.
- You want to load an exploit, but you do not have any exploits on your system yet. How would you proceed?
- You want to set a connected Wi-Fi adapter in monitor mode on channel 1. How would you proceed?
- You want to execute a deauthentication attack. The hardware is already connected. Set up the hardware and execute the attack to deautheticate the client from the network.

After completing these tasks, all candidates were consulted with a second questionnaire to state their overall opinion of the tool, steps that felt unintuitive, and suggestions for improvement. There was also sufficient time planned to follow up on specific circumstances the team wanted to investigate deeper.

All tests were documented in test protocols, which are fully disclosed in Appendix **??**.

4.2.2. Test Results Summary

No critical issues or shortcomings that would severely impact the usability or core functionality of the tool were raised during the user tests. Participants were generally able to navigate the CLI and complete typical tasks without requiring extensive assistance or documentation. While a few minor suggestions were made – such as enhancing the clarity of certain help messages, improving error feedback, or refining command naming – these were considered quality-of-life improvements rather than essential fixes. As a result, no significant changes had to be implemented following the tests, and the overall structure and logic of the tool remained intact. The feedback primarily confirmed that the platform was already in a stable and usable state, especially within its intended lab and educational context.

The following items were condensed from all test feedbacks and deemed possible to improve:

1. Help message not automatically printed when unknown / bad command is entered.
2. Imprecise or misleading wording in help messages confused users, especially when looking for the `exploit pull` command.
3. Unintuitive use of positional and optional arguments (`count` and `interval`), especially for sending packets with `send` command in exploit mode.
4. Ctrl+c in exploit mode quits the tool entirely, instead of just exiting exploit mode.
5. Command `setup mon` stood out with abbreviated sub command 'mon', and the default value of the `channel` argument was unintuitive.

Only one minor bug was found: The tools custom autocompletion would continue suggesting the same filename for the `exploit load <file>` and `setup edit <file>` indefinitely. While having no critical impact, this could lead to confusion and presents a nuisance.

The changes prompted by the feedback and discoveries of the user tests are discussed in the next section.

4.2.3. Implemented Changes

The implementation of the following minor changes are the result of the most relevant feedbacks compiled from the user tests:

1. Help message now shows automatically when an unknown / bad command is entered.
2. Wording of help messages is now improved, especially for the `exploit` command, which now clearly indicates the option for pulling exploits.
3. The previous positional arguments `count` and `interval` can now be passed as options, like `send --count 3`.
4. Ctrl+c in exploit mode now just exits the mode, instead of quitting the tool entirely.
5. Command `setup monitor` is now spelled out, and the `channel` argument is now required.

4.2.4. Test Conclusion

Feedback gathered from the tests revealed a fairly matured tool with good usability. The test validated that most core functionalities could be completed with minimal guidance; such as setting up lab-devices, loading and running attack scenarios. Users appreciated the clear command structure and help messages, although some suggestions were made for improving minor details, none of which being critical. Overall, the results confirmed that the CLI is well-suited for its educational purpose, while also identifying minor areas for refinement in future iterations.

5. Team & Project Organization

This chapter discusses tools and methods to aid with project management and organization.

5.1. Project Management Methods

In this section we go into detail how the project itself was organized, how work items have been tracked and updated throughout the project, and which rules, standards, and conventions were followed.

5.1.1. Task Management

The tracking of todos and assignment of active work items to project team members is managed with Jira. The word ticket is used in the following listings, a ticket is a task or a bug created in Jira.

Definition of Ready (DoR)
The following properties must be satisfied by a ticket:

- The ticket has a title.
- The ticket was discussed in the team.
- If the ticket is of type bug, it also requires a description.

Definition of Done (DoD)
A ticket can be marked as "done" in Jira when:

- The ticket has been resolved; either through a feature implementation in the product, or a change in documentation.
- If the ticket concerns a change to the code: A corresponding merge request must be opened and reviewed by the other team member. An exception to this rule is if the ticket was realized through pair programming.

5.1.2. GitLab Workflow

The project is divided into several repositories on the OST GitLab with dedicated purposes:

- **Repo 'documentation':** Primary project documentation written in LaTeX*.
- **Repo 'Wi-Fied':** Source code for the Python-based CLI-tool Wi-Fied Platform.
- **Repo 'Wi-Fied-Exploits':** Collection of exploits to be pulled with git through the Wi-Fied Platform.
- **Repo 'lab':** Lab instructions / exercises for a predefined exploit scenario written in LaTeX*.
- **Repo 'project':** Pathfinder project for tooling evaluation and testing.

*Based on Institute of Network and Security at OST (INS) LaTeX templates.

CI/CD pipelines automatically create the PDF from the LaTeX source in the 'documentation' and 'lab' repos. For the 'Wi-Fied' repo, pytest and SonarQube checks are automated in the pipeline.

Merge requests should be used in the 'Wi-Fied' repository, ensuring proper reviews through the other team member before merging into main. In the 'documentation' repo, merge requests are optional, depending on the current work distribution and potential for conflicts.

5.2. Tools and Resources

Important tools and resources that were used for the project management and implementation of the product are listed below.

- **Course Materials:** Slides, scripts, and exercise handouts from previous modules and courses, in the field of network and security related ones were used for reference. Especially previous work from 'Studienarbeit' [1] and SE Project were valuable in regard to project management, planning, and documentation guidelines.
- **Technical documentations, web research:** References and relevant web resources were cited when referred to them. Research required for the understanding or implementation of the project was carried out online with search engines or relevant forums. Use of AI like OpenAI ChatGPT [22] (e.g., troubleshooting help, research and brainstorming) was used in compliance with OST's 'Leitlinie zum Umgang mit KI-basierten Hilfsmitteln in Lehre und Weiterbildung'.
- **Project Management:** Jira [23] was used for the task backlog and progress tracking during the sprints.
- **IDE:** Source code and most of the documentation (latex) were written using Visual Studio Code [24].
- **Drawing and Diagrams:** Diagrams and drawings for the figures in the documentation were created using Drawio [25] (where not stated otherwise).

5.3. Task Allocation, Meetings and Communication

The project team is responsible for planning, allocating and monitoring tasks and results. All work is equally assigned to both team members.

Regular sync meetings within the project team are negotiated on a daily basis on workdays. Meetings with the advisor are scheduled weekly, however it is up to the project team whether each occurrence is desired. It is possible to skip meetings if no relevant talking points come up. A meeting with the advisor serves primarily as a status call, giving all parties an update on the progress of the project. If a pressing situation should arise, ad-hoc meetings can be scheduled.

All involved parties (e.g., advisor, expert) are to be invited to the mandatory presentations. The project team is responsible for the coordination and communication. MS Teams and email / calendar invites shall be used for formal inquiries and invites. Communication within the project team is not regulated.

5.4. Time Tracking

This section is dedicated to documenting the targeted timeframe for the project and explaining deviations from the original target time. Each team member initially committed to contributing approximately 360 hours to the project, as mandated by the university's requirements. The following provides an overview of the actual time spent, broken down by contributor and key activity areas.

5.4.1. Target and Actual Timeframe

The Wi-Fied Platform project was scoped with an intended workload summing up to 720 hours total. Due to the project's exploratory and prototype-driven nature, both contributors exceeded the target time. In total, the team invested roughly 820 hours, reflecting an overrun of 40-60 hours per person:

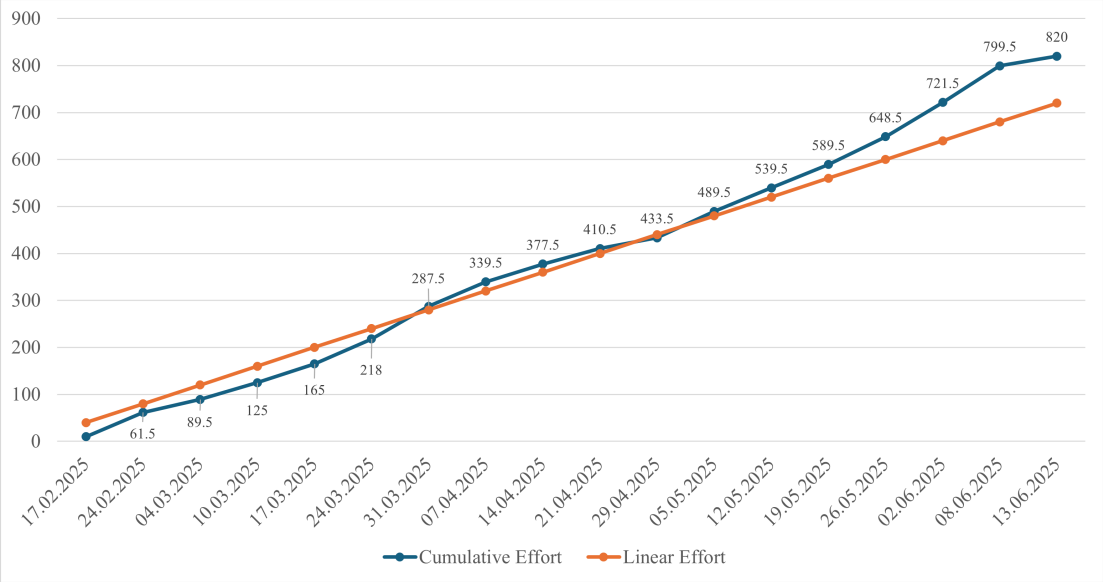


Figure 5.1.: Total Cumulative Effort

5.4.2. Causes of Overrun

The additional time can be attributed to several critical factors:

- **Prototype-Driven Development:** Early project phases involved frequent evaluation of tools, libraries, and platform architectures. This iterative process, while time-consuming, was essential to identify reliable components and technical feasibility.
- **Tool and Technology Evaluation:** Extensive time was spent researching and testing Wi-Fi packet manipulation, embedded system suitability (e.g., Raspberry Pi configurations), and automated lab setup.
- **Refactoring and Code Stabilization:** As requirements matured, significant refactoring efforts were needed to ensure code maintainability, modularity, and alignment with educational goals.
- **Educational Material Development:** Special effort went into crafting a didactically structured lab instruction document, including a theoretical prestudy and a practical lab that demonstrates a KRACK attack (CVE-2017-13082) using the Wi-Fied Platform. This material aims to support hands-on student engagement while grounding exercises in technical context.

5.4.3. Workload Distribution

The workload was collaboratively split but not strictly symmetrical. Both contributors were involved in all major tasks (evaluation, architectural design, coding, testing, documentation), with periodic pair sessions and individual focus areas.

Contributor	Target Hours	Actual Hours	Overrun
Alice	360	~420	+60
Mario	360	~400	+40
Total	**720**	**~820**	**+100**

Table 5.1.: Workload Distribution Among Team Members

The following diagram (see Figure 5.2) presents an overview of time spent by each team member across key project activities. These include efforts related to requirements engineering, software architecture, infrastructure and tooling, evaluation, implementation, user testing, lab scenario development, documentation, and final presentation. This breakdown highlights how work was distributed and which activities required particular focus during the course of the project:

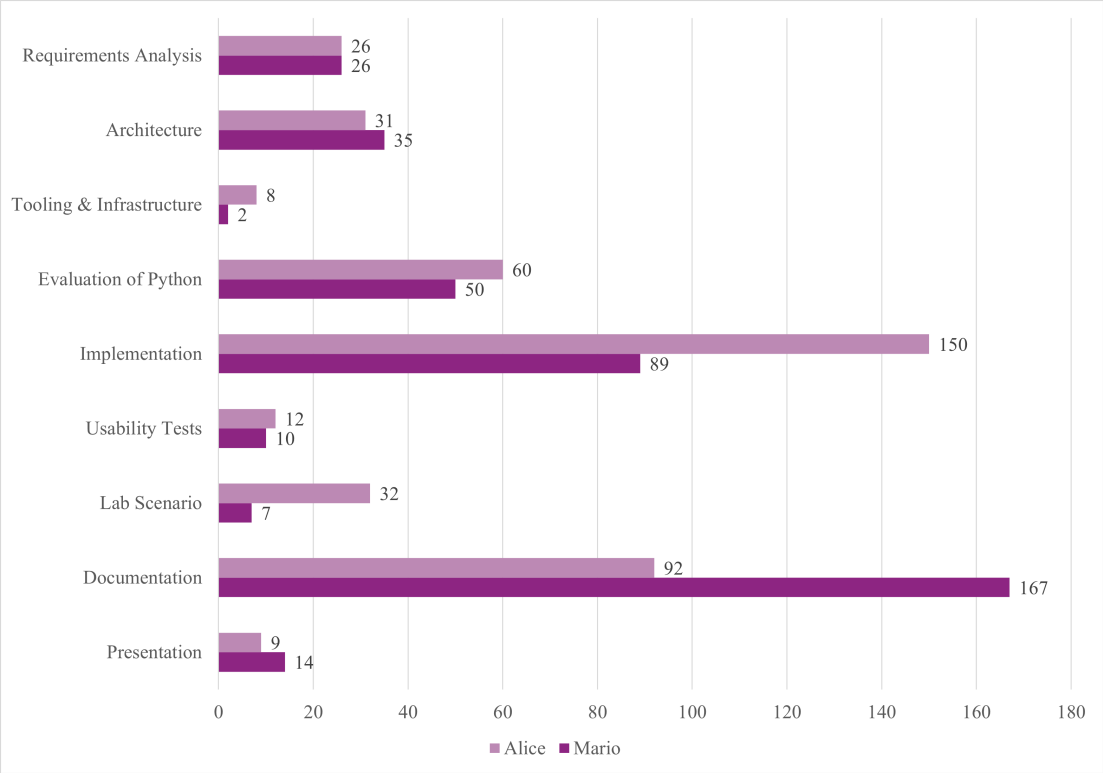


Figure 5.2.: Effort by Activity and Person

5.5. Roles

No discrete roles were defined due to the small size of the project team.

6. Conclusion

This chapter summarizes the key outcomes and insights gained throughout the project. It reflects on the results and challenges encountered, and outlines perspectives for the platform's future development.

6.1. Results and Lessons Learned

The Wi-Fied Platform demonstrates that a hands-on, modular toolkit for exploring Wi-Fi security threats is both feasible and valuable in educational settings. Its development provided important insights into how to balance extensibility and simplicity, especially when working with limited hardware capabilities such as those of the Raspberry Pi and under tight time constraints.

Using technologies already known from previous academic experience, including Python, Scapy, and Ansible, helped accelerate development. This also ensured that the tool remains accessible to students with varying levels of technical proficiency.

One key lesson was the complexity involved in researching, implementing, and formalizing a Wi-Fi exploit like KRACK for educational purposes. Understanding the exploit at a technical level, reproducing it reliably in a lab setting, and designing meaningful exercises around it required significant effort. Creating an instructional lab document that supports learning without oversimplifying the content proved especially challenging. The goal was to encourage student engagement and critical thinking, rather than providing step-by-step instructions that limit deeper understanding.

Despite these challenges, the project successfully delivered a functional platform and lab material that can be used in Wi-Fi security or cryptography courses. It serves as a starting point for future improvements, such as expanding the range of scenarios, refining exercises, and enhancing usability through additional features.

6.2. Outlook

As a foundational platform, Wi-Fied opens the door to many promising extensions and refinements. A central direction for future work lies in the development of additional exploit scenarios. Such additions could deepen the educational value of the platform by supporting more interactive and challenging lab exercises. In parallel, user interface improvements are envisioned to better visualize attack flows and packet sequences.

Enhanced Lab Scenarios and Instructions:

- Research WPA3 related exploits, to highlight the successor of WPA2.
- Create lab scenarios that support Man-in-the-Middle, as these would enable more dynamic and realistic demonstrations of vulnerabilities.
- More variation of exploit types, regarding different cryptographic cyphers and Wi-Fi standard amendments.

Improved User Interface:

- More visualization of attack flows, packet capture results (e.g. Wireshark-like) integrated into the platform.

- Integrated lab descriptions may improve usability and make the platform more accessible and interactive to learners.

Hardware Compatibility and Deployment Flexibility:

- Support for real hardware designated for Wi-Fi, such as access points and vulnerable clients.
- Extend support for a broader range of USB Wi-Fi adapters.
- Integrating hardware checks or compatibility tests.

Open Source and Community Involvement:

- Considering opening the Wi-Fied Platform to a broader community, making it open source.
- Strengthening the utility through more contribution.

List of Figures

List of Tables

Part I Product Documentation (arc42)

Part II Project Documentation

References

[1] A. Glaus and M. Burger, *Wi-fi security threats - an integrative review*, https://eprints.ost.ch/id/eprint/1241/, Accessed: 2025-05-03, 2024.

[2] R. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design* (Robert C. Martin series). Prentice Hall, 2017, ISBN: 9780134494272.

[3] D. G. S. Dr. Peter Hruschka *et al.*, *Arc42 - architecture documentation template*, https://arc42.org, Accessed: 2025-05-03, 2025.

[4] Python Software Foundation, *Python language reference, version 3.11*, https://docs.python.org/3/reference/, Accessed: 2025-05-04, 2023.

[5] *Scapy documentation, python library*, https://scapy.readthedocs.io/en/latest/, Accessed: 2025-05-11, 2025.

[6] *Ansible runner documentation, python library*, https://ansible.readthedocs.io/projects/runner/en/latest/, Accessed: 2025-05-11, 2025.

[7] *Sqlalchemy documentation, python library*, https://docs.sqlalchemy.org/en/20/, Accessed: 2025-05-11, 2025.

[8] *Sqlite documentation*, https://www.sqlite.org/docs.html, Accessed: 2025-05-27, 2025.

[9] *Ansible community documentation*, https://docs.ansible.com/, Accessed: 2025-05-11, 2025.

[10] *Git documentation*, https://git-scm.com/doc, Accessed: 2025-05-11, 2025.

[11] *Gitlab instance of ost*, https://gitlab.ost.ch/, Accessed: 2025-05-27, 2025.

[12] *Raspberry pi website*, https://www.raspberrypi.com/, Accessed: 2025-05-27, 2025.

[13] *Linksys ae3000, usb wi-fi adapter, page 73*, https://downloads.linksys.com/downloads/userguide/AE3000_UG_3425-01611A_Web.pdf, Accessed: 2025-05-27, 2025.

[14] *The c4 model for visualising software architecture*, https://c4model.com/, Accessed: 2025-05-11, 2025.

[15] *Gitpython documentation, python library*, https://gitpython.readthedocs.io/en/stable/, Accessed: 2025-05-11, 2025.

[16] *Prompt toolkit documentation, python library*, https://python-prompt-toolkit.readthedocs.io/en/master/, Accessed: 2025-05-11, 2025.

[17] *Typer documentation, python library*, https://typer.tiangolo.com/tutorial/, Accessed: 2025-05-11, 2025.

[18] *Architecture decision record template: Y-statements | zio's blog*, https://medium.com/olzzio/y-statements-10eb07b5a177, Accessed: 2025-05-13, 2020.

[19] *Nornir netmiko documentation, python library*, https://ktbyers.github.io/netmiko/docs/netmiko/index.html, Accessed: 2025-06-01, 2025.

[20] *Nornir napalm documentation, python library*, https://nornir.tech/nornir_napalm/html/api/index.html, Accessed: 2025-05-18, 2025.

[21] *Institute for network and security, ost*, https://www.ost.ch/en/research-and-consulting-services/computer-science/ins-institute-for-network-and-security, Accessed: 2025-05-31, 2025.

[22] *Openai chatgpt*, https://openai.com/chatgpt/overview/, Accessed: 2025-05-31, 2025.

[23] *Atlassian jira*, https://www.atlassian.com/software/jira, Accessed: 2025-05-31, 2025.

[24] *Visual studio code*, https://code.visualstudio.com/, Accessed: 2025-05-31, 2025.

[25] *Drawio*, https://www.drawio.com/, Accessed: 2025-05-31, 2025.