AL-BALQA APPLIED UNIVERSITY

AL-HUSON UNIVERSITY COLLEGE

Electrical Engineering Department

Graduation Project

"ESP32-Based Cyber Threat Detection And Monitoring in Wireless Local Area Networks"

Authors:

Yazan Waleed Taha
Mohammed Al Mahdi Al Najjar

Supervisors:

Nasr Gharaibeh

January 15, 2026

Contents

List of Figures

Abstract

n this project, a system was developed based on the ESP32 microcontroller to function as an advanced monitoring device within local networks. The system operates in Promiscuous Mode to capture and analyze Wi-Fi frames in real time. It is capable of detecting several common cyberattacks, such as deauthentication attacks, fake access point attacks (Beacon Flood), authentication flood attacks, and reconnaissance activities including Probe Request floods and internal network mapping (ARP Scanning). Additionally, the system includes a network monitoring feature that tracks device connectivity to identify and alert on new devices joining the network.

The system relies on multiple algorithms that analyze traffic behavior by counting frames within a fixed time window and identifying abnormal patterns indicative of security threats. When any attack or suspicious activity is detected, an instant alert is sent through the Telegram platform. The system was tested on a real Wi-Fi network and demonstrated high accuracy in capturing most frame types and reliably detecting attacks compared with traditional sniffing tools on computers.

This project aims to provide a low-cost, scalable solution that can be easily integrated into home networks and small business environments.

Chapter 1

Introduction

1.1 Overview

In recent years, the security of local area networks (LANs) has become a growing concern due to the increasing number of connected devices and the rise in cyberattacks targeting smaller networks. LANs are commonly used in homes, educational institutions, small offices, as well as commercial spaces such as restaurants and cafés, often without strong security measures in place [1]. This makes them vulnerable to various forms of cyber threats such as unauthorized access, spoofing, denial-of-service (DoS) attacks, and port scanning [2–5]. Ensuring early detection of these threats is critical to maintaining the integrity, availability, and confidentiality of the network.

To address these security challenges, particularly in cost-sensitive environments, this project proposes a lightweight and low-cost Intrusion Detection System (IDS) based on the ESP32 microcontroller. Unlike traditional enterprise security solutions that require expensive hardware or complex configurations, the proposed system leverages the ESP32's Wi-Fi capabilities to monitor wireless traffic in real-time. It is designed to detect specific Layer 2 attacks, including Deauthentication, Beacon Flooding, and Authentication Flooding, while simultaneously monitoring network connectivity to identify unauthorized devices. This approach provides a practical, scalable, and accessible security layer for home and small business networks.

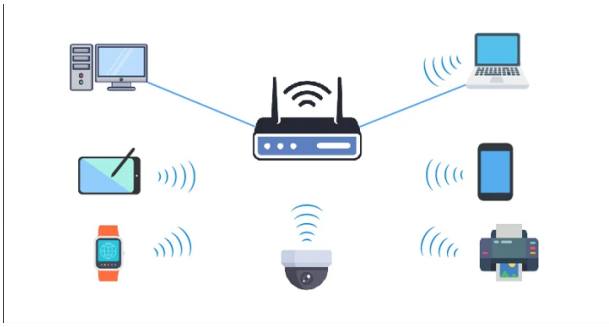

Fig. 1.1: Local area networks

1.2 Problem Statement

Wireless networks (Wi-Fi) have become the backbone of modern connectivity, yet the security tools available for Small Office/Home Office (SOHO) environments have failed to keep pace with evolving threats. While enterprise networks rely on sophisticated Intrusion Detection Systems (IDS) like Snort or Suricata, these solutions demand expensive hardware and high technical expertise, making them inaccessible for average users [6, 7].

A critical vulnerability in standard Wi-Fi protocols (specifically WPA2) is the lack of integrity protection for Management Frames. Unlike data packets, these frames—responsible for network authentication and connection maintenance—are often transmitted in clear text. Consequently, adversaries can easily spoof these frames to launch Layer 2 attacks, such as Deauthentication and Beacon Flooding, effectively disconnecting users or creating fake access points without needing the network password.

Compounding this issue is the limitation of consumer-grade routers. Standard routers are designed for connectivity, not surveillance; they lack the capability to monitor raw traffic (Promiscuous Mode) or analyze management frame headers. As a result, SOHO networks operate with significant "security blind spots."Users remain unaware of ongoing surveillance or jamming attacks until connectivity is completely lost. This creates an urgent need for a dedicated, low-cost, and standalone monitoring system capable of detecting these invisible threats in real-time.

1.3 Aim of Study

The primary aim of this study is to design and implement a cost-effective, embedded Intrusion Detection System (IDS) using the ESP32 microcontroller, specifically targeting threats at the **OSI Data Link Layer (Layer 2)**. The system is designed to monitor wireless traffic in real-time by analyzing management and control frames to secure the physical and data link boundaries of the network. Specifically, it aims to:

- Detect Denial-of-Service (DoS) attacks targeting the Wi-Fi infrastructure, such as Deauthentication and Authentication Floods.
- Identify wireless scouting and spoofing attempts, including Probe Request Floods and Beacon Flooding (Fake APs).
- Detect internal network reconnaissance and mapping activities, specifically **ARP Scanning** and Broadcast Flooding used to identify active hosts.
- Enhance network visibility by monitoring device connectivity and alerting administrators of unauthorized or new devices joining the network via DHCP analysis.

This approach focuses on providing an accessible, modular, and easy-to-deploy security solution for Small Office/Home Office (SOHO) networks that typically lack advanced monitoring infrastructure [8].

1.4 Report Structure

This report is organized into five main chapters, detailing the development, implementation, and evaluation of the proposed network monitoring and intrusion detection system. The structure is outlined as follows:

- **Chapter 1: Introduction** establishes the context of the project. It defines the problem statement regarding wireless network security, outlines the specific aims of the study, and presents the overall organization of the report.
- **Chapter 2: Background and Literature Review** provides the theoretical foundation necessary to understand the project. It reviews the state-of-the-art in wireless security, explains the theoretical background of IEEE 802.11 protocols, and examines relevant literature and similar studies to contextualize the proposed solution.
- **Chapter 3: Methodology** details the practical implementation of the system. It covers the hardware architecture (ESP32), the development environment, and the core software logic, including packet sniffing, frame parsing, and the specific algorithms developed for threat detection and DHCP analysis. It also outlines the testing procedures and acknowledges the system's limitations.
- **Chapter 4: Results and Discussion** presents the experimental findings. It demonstrates the system's performance in establishing connectivity, classifying traffic, and detecting devices. Crucially, it visualizes the detection of specific attack scenarios—such as Deauthentication, Beacon Floods, and Scanning—using real-time logs. The chapter concludes with a critical discussion of detection accuracy, system latency, and a cost-benefit analysis.
- **Chapter 5: Conclusions and Future Work** summarizes the project's key achievements and findings. It also provides recommendations for future enhancements, including potential hardware upgrades and feature expansions to overcome current limitations.

Chapter 2

Background & Literature Review

2.1 State of the Art

Local Area Networks (LANs) have become an integral part of modern communication infrastructures, utilized extensively in homes, offices, and commercial venues such as restaurants and cafés. The proliferation of connected devices and the rising sophistication of cyberattacks have heightened the need for effective security measures in these networks [1]. Current trends in network security emphasize the importance of real-time monitoring and intrusion detection to protect against unauthorized access, spoofing, denial-of-service (DoS) attacks, and other malicious activities [2–5].

Traditional software-based Intrusion Detection Systems (IDS), such as Snort [7] and Suricata [9], are widely deployed in enterprise environments. These systems are highly effective but demand considerable computational resources and storage, making them impractical for small-scale or embedded networks [6].

Recently, embedded systems and microcontrollers like the ESP32 have gained popularity as cost-effective platforms for implementing security solutions in resource-constrained environments [8, 10, 11]. Their native support for Wi-Fi Promiscuous Mode allows them to capture and analyze raw IEEE 802.11 frames similarly to desktop tools like Wireshark [12], making them suitable candidates for lightweight intrusion detection systems (IDS) tailored for SOHO (Small Office/Home Office) networks.

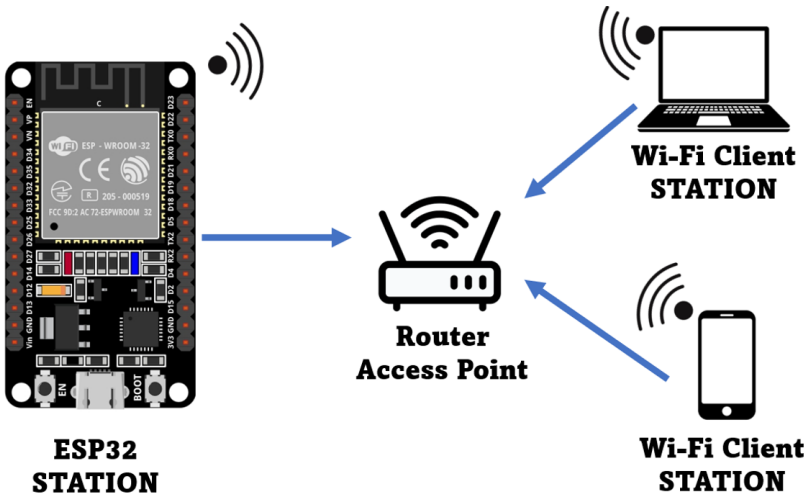

Fig. 2.1: ESP32 implementation in wireless LAN environment

2.2 Theoretical Background

Before reviewing related literature, it is essential to understand the specific Layer 2 attacks that threaten wireless networks, which form the basis of this project's detection logic.

- **Deauthentication Attack:** A type of Denial-of-Service (DoS) attack where adversaries send spoofed deauthentication frames to force a client to disconnect from the Access Point (AP). This is often a precursor to capturing the WPA2 handshake [13].
- **Beacon Flooding (Fake APs):** An attack involving the injection of thousands of fake Beacon frames to create non-existent Access Points, causing confusion for wireless clients and jamming the Wi-Fi scanner [14].
- **Probe Request Flooding:** A surveillance and resource-exhaustion attack where a device floods the network with requests to discover hidden SSIDs, indicating active network scanning [15].
- **Authentication Flood:** A resource-exhaustion DoS attack where the attacker floods the Access Point with a high volume of authentication requests using random spoofed MAC addresses. The goal is to overflow the AP's client association table, preventing legitimate users from connecting [16].
- **Internal Network Mapping (ARP Scanning):** Unlike wireless probing which scans the air, this technique targets the local subnet. Attackers broadcast high volumes of Address Resolution Protocol (ARP) requests to discover active IP addresses and map live hosts. Detecting surges in ARP broadcast traffic is critical as it serves as the reconnaissance phase for Man-in-the-Middle (MitM) attacks [17].
- **Network Asset Discovery (DHCP Analysis):** Beyond attack detection, network visibility relies on the Dynamic Host Configuration Protocol (DHCP). When a device connects to a network, it initiates the **DORA** process (Discover, Offer, Request, Acknowledge) to obtain an IP address. The *DHCP Request* packets are of particular interest as they are typically broadcast at Layer 2 and contain unencrypted metadata, including the device's **Hostname** (Option 12) and **MAC address**. Monitoring this traffic provides the theoretical basis for identifying unauthorized devices immediately upon connection [18].

2.3 Review of Relevant Literature

2.3.1 Academic Approaches to Lightweight IDS

Several studies have explored various approaches to intrusion detection in LAN environments. Smith and Doe [6] investigated the challenges of deploying IDS in small networks, highlighting limitations related to cost and computational resources. Their research aimed to develop lightweight detection algorithms that can operate efficiently on embedded hardware.

Espressif Systems [8] provided a technical reference for the ESP32 microcontroller, demonstrating its potential in network monitoring applications. Their work showed how ESP32's dual-core processor and built-in Wi-Fi capabilities enable efficient packet capture and analysis, supporting the feasibility of embedded IDS.

Other researchers have focused on specific threat types, such as ARP spoofing [3] and wireless DoS attacks [2], proposing detection mechanisms adapted for lightweight systems. Man-in-the-Middle attacks [4] and wireless LAN vulnerabilities [5] have also been studied, emphasizing the importance of timely detection.

Regarding detection methodologies, recent research has explored Machine Learning (ML) for small networks. Rahman and Silva [11] applied ML algorithms for detecting DoS attacks. However, while ML offers high adaptability, it often requires significant processing power and training datasets. Consequently, hybrid models or **Statistical Threshold-based approaches**—like the one proposed in this project—are often preferred for microcontrollers to balance detection speed with low computational overhead [11]. Figure 2.2 illustrates the architectural differences and resource trade-offs between complex ML-based systems and the streamlined statistical approach adopted in this study.

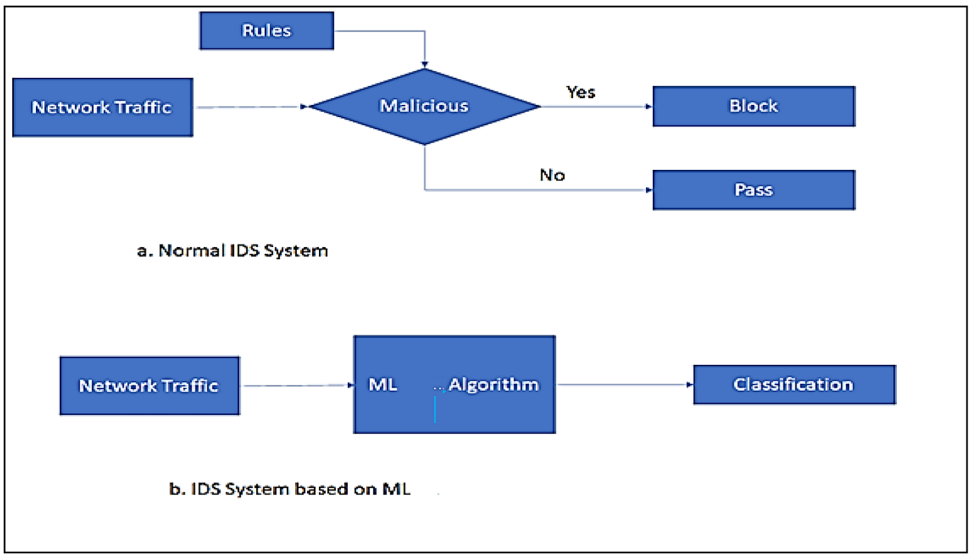

Fig. 2.2: Comparison: ML-Based IDS vs. Statistical/Rule-Based IDS approaches

2.3.2 Existing Wireless Security Tools

Beyond academic theory, several practical tools have shaped the landscape of wireless security. This project draws inspiration from these standard utilities but adapts them for an embedded architecture:

- **Kismet (Desktop-based Passive Monitoring):** Considered the "gold standard" for wireless network detection, Kismet is a passive sniffer operating at Layer 2 [19]. While highly effective, it requires a dedicated laptop or high-performance computer. This project aims to replicate Kismet's core passive monitoring logic within the resource constraints of the ESP32.

- **Aircrack-ng Suite:** Widely used for network auditing, tools like *airodump-ng* are used to capture packets and detect deauthentication frames manually [20]. This project automates these manual auditing processes into a continuous firmware loop without human intervention.

- **ESP8266 Deauther (Spacehuhn):** A popular open-source project that uses the ESP8266 to generate Deauthentication frames [21]. **Contrast with Proposed System:** It is crucial to distinguish this project from the Deauther. While they share similar hardware architecture, the ESP8266 Deauther is an **offensive tool** designed to *launch* attacks. Conversely, the system proposed in this study is a **defensive IDS** designed to *detect* such attacks, acting as a "Blue Team" solution.

2.4 Project Description in Theory

This project aims to design and implement a modular embedded cyber threat detection system using the ESP32 microcontroller. Unlike complex ML-based systems, this project utilizes a **deterministic algorithm** based on traffic analysis and frame counting to identify threats in real-time with minimal latency.

The system leverages the ESP32's **Promiscuous Mode** to sniff raw Management Frames (Deauth, Beacon, Probe) and Data Frames (EAPOL). It applies a **Statistical Anomaly Detection** method, where traffic behavior is compared against predefined thresholds within fixed time windows. If the counter for a specific frame type (e.g., Deauthentication frames) exceeds the safety threshold, the system flags it as an attack.

Additionally, the system incorporates a **Network Connectivity Monitor** (DHCP Sniffer) to track device associations. This theoretical hybrid approach—combining Layer 2 attack detection with network visibility—ensures a comprehensive security layer for networks without requiring expensive hardware or complex configurations.

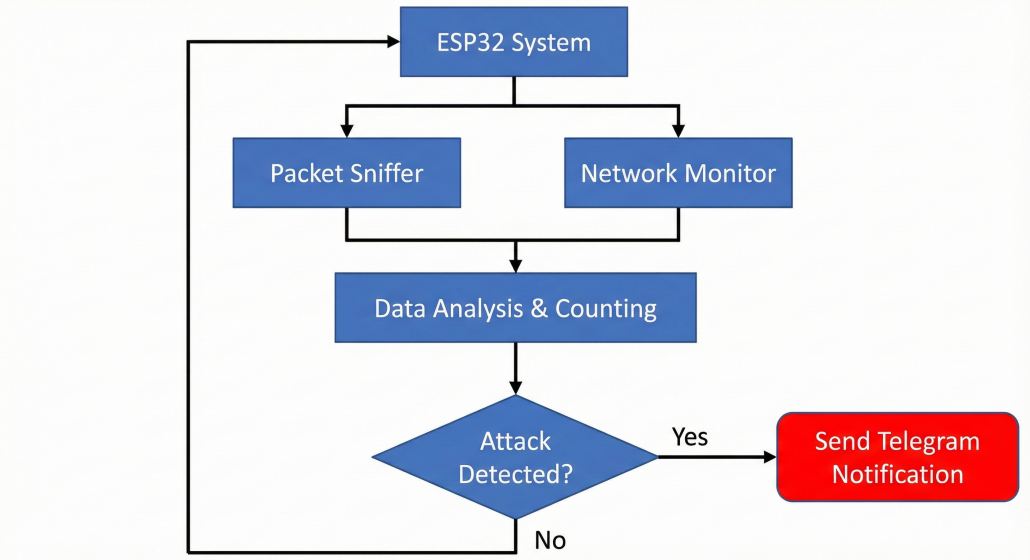

Fig. 2.3: Simplified System Operational Logic

Chapter 3

Methodology

This chapter details the practical steps taken to design and build the network monitoring and threat detection system. The implementation process is divided into three main phases: hardware configuration, firmware development, and system integration. The following sections will discuss the hardware setup, the software algorithms used for packet parsing and threat detection, and the testing scenarios conducted to validate the system's efficiency.

3.1 Project Implementation Overview

The system architecture utilizes the ESP32 microcontroller to perform two complementary security functions: **wireless intrusion detection** and **network connectivity monitoring**.

First, the detection engine operates in Promiscuous Mode to capture raw IEEE 802.11 frames without establishing a standard connection. As illustrated in Figure 3.1, this module continuously analyzes specific frame headers, focusing exclusively on **Management and Data frames**. Crucially, the system parses the **Frame Subtype** fields within Management frames to distinguish between various network activities. Furthermore, the analysis extends to Data frames by monitoring the rate of Broadcast traffic, which enables the system to detect internal network reconnaissance activities such as ARP Scanning. By counting these specific subtypes within fixed time windows, the system identifies abnormal patterns indicative of cyberattacks.

In parallel, the system incorporates a DHCP inspection mechanism to enhance network visibility. This function monitors DHCP transactions to detect new devices joining the network, extracting critical identity details such as Hostnames, IP addresses, and MAC addresses, as shown in Figure 3.2. By integrating these two data streams, the system can both flag suspicious traffic patterns and identify unauthorized devices, triggering real-time alerts via the Telegram API.

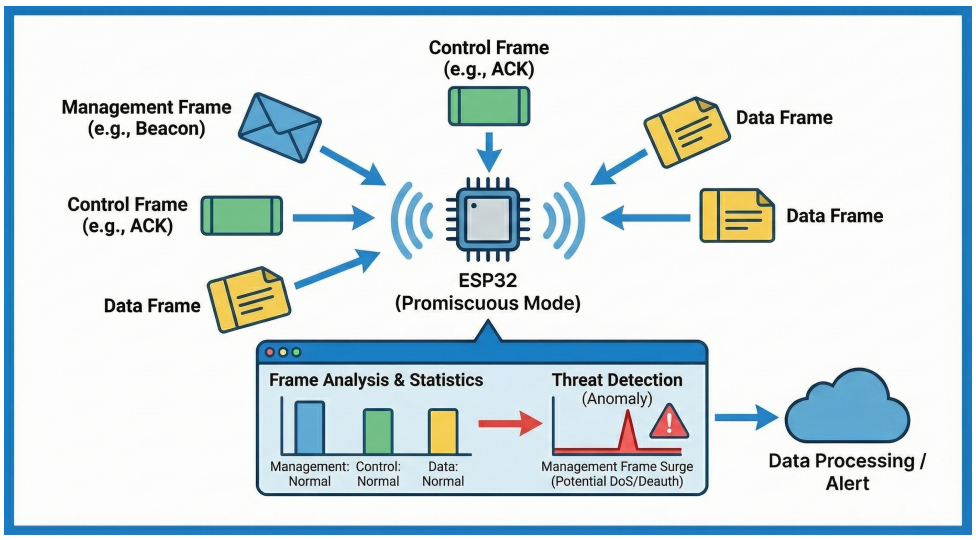

Fig. 3.1: Real-time frame capture and analysis process

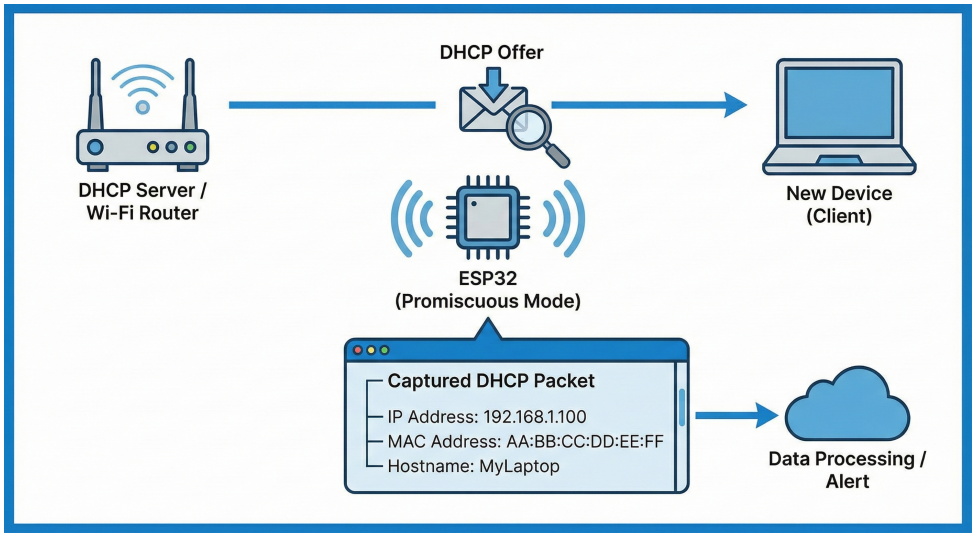

Fig. 3.2: Captured DHCP packet showing Hostname and IP extraction

3.2 Hardware Components

The hardware architecture of the project is designed to be modular and cost-effective. The system consists of the following primary components:

- **ESP32 Development Board:** The ESP32 serves as the main processing unit of the system. It features built-in Wi-Fi capabilities, a dual-core processor, and low power consumption, making it suitable for embedded network security applications.

  Deployment Note: To achieve simultaneous operation of both the Intrusion Detection System and the Network Connectivity Monitor, the implementation utilizes two separate ESP32 units. One unit is dedicated to Promiscuous Mode for packet analysis, while the second unit operates in Station Mode for DHCP monitoring.

- **Power Supply (USB or External Adapter):** Used to provide stable power (5V) to the ESP32 units during continuous monitoring and packet capturing operations.

- **Optional Expansion Components:** Future enhancements may include integrating additional hardware such as a Raspberry Pi for centralized logging and advanced data processing, or visual indicators (LEDs or LCD displays) to show system status and immediate alerts.

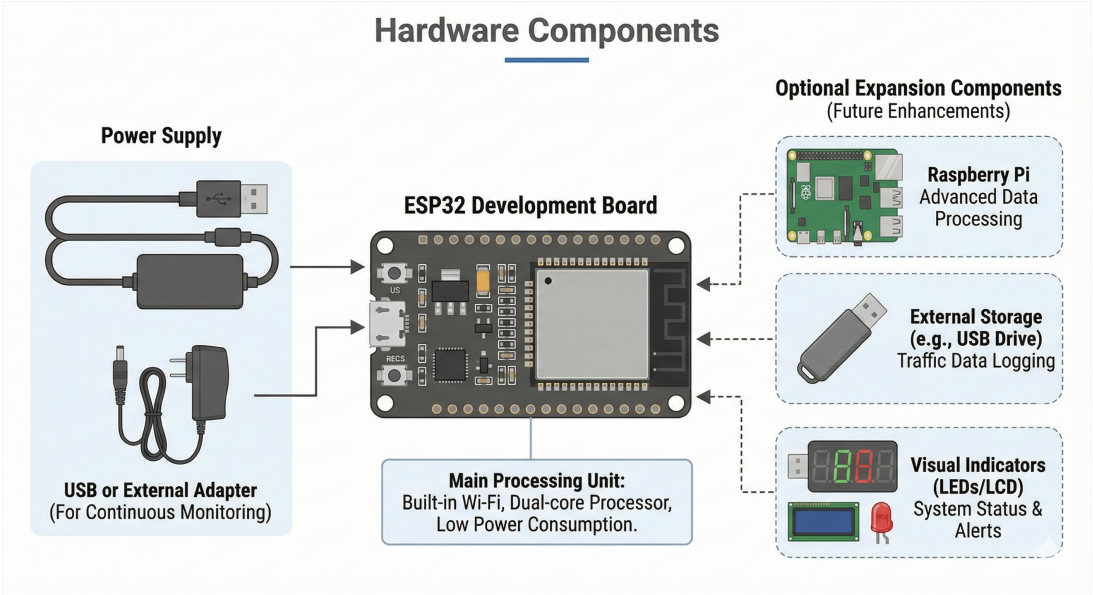

Fig. 3.3: Hardware Components

3.3 Development Environment and Tools

The software implementation of the proposed system was developed to ensure real-time monitoring and efficient threat detection on the ESP32 microcontroller. The software architecture consists of three main stages: packet capturing, frame parsing, and threshold-based anomaly detection.

The firmware was developed using the following tools and libraries:

- **Arduino IDE:** Used as the primary Integrated Development Environment for writing, compiling, and uploading the C/C++ code to the ESP32.
- **ESP32 Wi-Fi Promiscuous Library:** Essential low-level libraries were utilized to switch the ESP32 Wi-Fi interface into *Promiscuous Mode*. This mode allows the controller to listen to all wireless traffic on a specific channel, regardless of the destination MAC address.
- **Telegram Bot API:** Integrated via HTTP requests to send real-time alerts to the network administrator's smartphone.

3.4 Packet Sniffing and Parsing Logic

The core software module is the **Packet Sniffer**. Once a Wi-Fi frame is captured, the software executes a callback function to parse the raw data. The parsing process involves:

1. **Header Extraction:** Reading the IEEE 802.11 header to identify the Source MAC, Destination MAC, and Frame Control Field.
2. **Frame Classification:** The system inspects the first byte of the captured packet, specifically the **Frame Control Field**, to determine the packet type. The classification logic relies on filtering specific hexadecimal identifiers:
    - **Management Frames:** Identified by the Type bits `00`. The system further parses the **Subtype** bits to distinguish specific network activities:
        - **Beacon Frames (0x80):** Broadcast frames sent by APs to announce their presence.
        - **Probe Requests (0x40):** Frames sent by clients scanning for available networks.
        - **Authentication (0xB0) & Deauthentication (0xC0):** Frames managing the connection state between client and AP.
    - **Data Frames:** Identified by the Type bits `10`. While the payload is usually encrypted, the system inspects the **Packet Headers** to detect critical signatures:
        - **EAPOL Handshakes:** Identified by the EtherType `0x888E` in the LLC header, indicating a 4-way handshake attempt (password exchange).
        - **Internal Reconnaissance (ARP):** Identified by the **Broadcast Destination MAC** (`FF:FF:FF:FF:FF:FF`). Since ARP requests must be broadcasted to reach all hosts, monitoring the rate of these frames allows the system to detect internal network mapping scans (ARP Floods) even without decrypting the payload.

3. **DHCP Traffic Inspection (Layer 4 Analysis):** Unlike the attack detection logic which focuses on Layer 2 frames, this module extends monitoring to the **Transport Layer (Layer 4)**. It operates by actively listening on **UDP Port 67** to intercept broadcast datagrams sent by clients during the **DORA (Discover, Offer, Request, Acknowledge)** process. The parsing algorithm specifically targets **DHCP Request** packets and iterates through the payload to locate specific **DHCP Options** identified by their hexadecimal codes:

   - **Option 0x0C (Hostname):** Used to extract the device's human-readable name (e.g., "iPhone-User").
   - **Option 0x32 (Requested IP):** Used to retrieve the specific IP address the client is attempting to lease from the router.

   This mechanism allows the system to identify and log new devices immediately as they join the network.

3.5 Threat Detection Algorithm

To detect cyber attacks without decrypting WPA2/3 traffic, the system implements a **Threshold-based Detection Algorithm**. The software maintains counters for specific frame types within a **three-second time window**. If the frequency of a specific frame type exceeds a predefined threshold during this interval, the system triggers an alert.

It is important to note that these threshold values are not absolute; they vary depending on the network density and baseline traffic. The conditions presented in Table 3.1 have been empirically calibrated for a typical small-scale network environment containing approximately **5 to 10 connected devices**.

Frame Type	**Subtype**	**Condition (per 3s)**	**Detected Threat**
Management	Deauth (0xC0)	> 40 frames	DoS / Deauth Attack
Management	Beacon (0x80)	> 400 frames	Beacon Flood / Fake AP
Management	Probe Request	> 50 frames	Active Wi-Fi Scouting
Management	Auth (0xB0)	> 40 frames	Auth Flood (DoS)
Data	EAPOL (0x888E)	> 40 packets	Password Cracking
Data	Broadcast(ARP)	> 100 frames	Internal Network Mapping

Table 3.1: Frame-Based Threat Detection Logic (Updated with ARP Detection)

This rule-based approach ensures that the system remains lightweight and effective, identifying threats based on traffic behavior rather than payload content.

3.6 Alert Management and System Resilience

To ensure the system remains practical and reliable during active attacks, two critical logic mechanisms were implemented in the firmware:

- **Alert Rate Limiting (Anti-Spam Mechanism):** During high-intensity attacks (e.g., a Beacon Flood), generating a notification for every detected violation would overwhelm the Telegram API and the administrator. To prevent "notification fatigue," the system implements a **Non-Blocking Cooldown Timer**. Once an alert is triggered, a flag is set to suppress further identical alerts for a predefined interval (e.g., 15 seconds), ensuring the administrator receives timely warnings without being flooded by redundant messages.
- **Resilience Against Deauthentication (Auto-Reconnection Loop):** A unique challenge of the Deauthentication attack is that it targets all devices, including the monitoring ESP32 itself. To prevent alert loss during disconnection, a **"Store-and-Forward" logic** was implemented. When an attack is detected, the system immediately enters a strictly timed loop to attempt reconnection with the Access Point. The alert is buffered internally and dispatched to the Telegram server only after connectivity is successfully restored.

3.7 Data Collection and Testing

Data collection was conducted by deploying the ESP32 within a real or simulated local area network environment, where the device continuously captured wireless traffic exchanged between network nodes.

The testing process included:

- Monitoring normal network behavior to establish baseline traffic patterns.
- Adding new devices to the network and disconnecting existing devices to evaluate the system's ability to detect changes in network membership.
- Verifying that the system correctly identified newly connected devices and generated real-time notifications via the Telegram bot.
- Observing variations in Wi-Fi frame type statistics during device connection and disconnection events.
- Analyzing abnormal frame distribution patterns to assess the system's capability to detect potential attacks such as Denial of Service (DoS) and Deauthentication (Deauth) attacks.

  The system's output was compared with the expected network behavior to validate detection accuracy. Performance evaluation was based on response time, detection reliability, and resource utilization.

3.8 Limitations and Challenges

Despite successful implementation, the project faced several limitations and challenges, including:

- **Limited Hardware Resources:** The ESP32 has constrained memory and processing power, which restricts the use of complex analysis algorithms or machine learning techniques.
- **Single-Band Operation (2.4 GHz Only):** The ESP32's internal radio hardware is physically limited to the 2.4 GHz frequency band. Consequently, the Intrusion Detection System is blind to attacks or rogue devices operating exclusively on the 5 GHz bands.
- **Alert-Induced Blind Spots:** Due to the **half-duplex** nature of the single Wi-Fi radio, the ESP32 cannot transmit data and capture packets simultaneously. When the system detects a threat and switches to *Station Mode* to send a Telegram alert, the *Promiscuous Mode* sniffer is temporarily suspended. This switching process creates unavoidable "blind spots" (typically lasting 1-3 seconds) where the system is effectively blind to any subsequent attacks until the transmission is complete.
- **Concurrency Constraint (IDS vs. Asset Discovery):** The single-radio architecture limits the system's ability to perform full-scale Intrusion Detection (IDS) and Network Asset Discovery (DHCP Analysis) simultaneously with equal efficiency. The IDS module requires continuous scanning to detect threats, whereas the DHCP module requires staying on a fixed channel to reliably intercept the ephemeral "DORA" handshake packets. Consequently, the system must prioritize one function over the other, as it cannot dedicate the radio to continuous scanning and fixed-channel monitoring at the same exact moment.
- **Fixed Channel Monitoring:** To ensure a stable connection for sending real-time Telegram alerts, the current prototype is configured to monitor a specific static channel (e.g., Channel 11). This limitation prevents the detection of threats occurring on other channels unless a "Channel Hopping" mechanism is implemented in future iterations.
- **Wireless Traffic Visibility:** Packet capture effectiveness depends on signal strength, network congestion, and the wireless channel being monitored.
- **Encrypted Traffic Analysis:** The system cannot inspect the payload of encrypted packets and therefore relies mainly on metadata and traffic patterns for threat detection.
- **False Positives:** Some legitimate network activities may be mistakenly flagged as malicious, requiring further tuning and optimization of detection rules.

Chapter 4

Results and Discussions

This chapter presents the practical results obtained from the system implementation and testing phases. The discussion begins with the verification of system connectivity and initialization, followed by the monitoring of normal network traffic baselines. Finally, the chapter evaluates the system's efficiency and accuracy in detecting the five targeted Wi-Fi attacks.

4.1 System Connectivity

Upon powering up the system, the initial phase involves initializing the ESP32 microcontroller and establishing a connection to the local Wi-Fi network. Once the network link is secured, the system initiates a handshake with the Telegram Bot API. This successful connection is verified through two distinct outputs: the visual confirmation on the administrator's mobile interface, as shown in Figure 4.1, and the corresponding debug logs on the Serial Monitor, displayed in Figure 4.2. These combined indicators validate that the alerting mechanism is active and ready to transmit real-time security warnings.

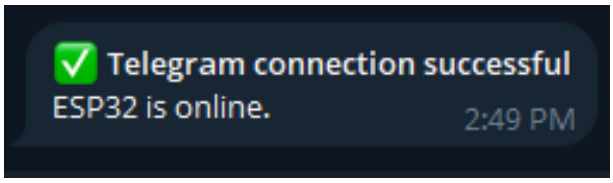

Fig. 4.1: Telegram Connection Notification

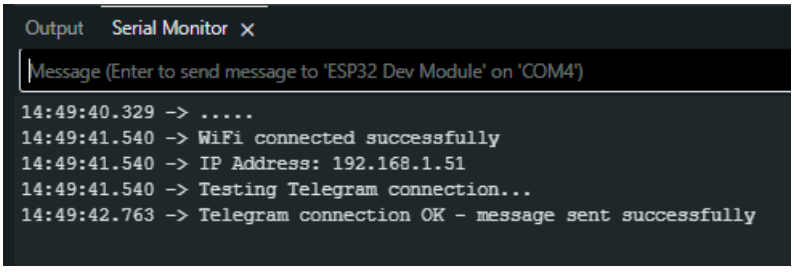

Fig. 4.2: Telegram Connection Serial Monitor

4.2 Baseline Monitoring and Traffic Classification

Following the connectivity verification, the system transitions into promiscuous mode to begin passive monitoring of the wireless spectrum. In this operational state, the ESP32 captures raw IEEE 802.11 frames, specifically filtering for **Management and Data frames**, while intentionally excluding Control frames to prioritize relevant network activity. The system detailedly parses Management frames into distinct subtypes (Beacon, Probe Request/Response, Authentication, and Deauthentication) and inspects Data frames to identify two critical traffic signatures: **EAPOL** packets used in the authentication handshake, and **Broadcast Data frames** (typically ARP requests) used for internal network mapping. Figure 4.3 presents the Serial Monitor output during this phase, displaying the real-time incrementation of these specific counters. This continuous data stream establishes a normal traffic baseline, demonstrating that the packet sniffing algorithm is correctly distinguishing frame types and detecting security-critical packets prior to any active attack scenarios.

```
01:43:13.602 -> ================ 📡 IDS Frame Statistics ================
01:43:13.635 ->  Management Frames : 151
01:43:13.635 ->    ├─ Beacon      : 132
01:43:13.635 ->    ├─ Probe Req   : 2
01:43:13.635 ->    ├─ Auth        : 0
01:43:13.635 ->    └─ Deauth      : 1
01:43:13.635 -> ------------------------------------------------------------
01:43:13.635 ->  Data Frames       : 495
01:43:13.635 ->    ├─ EAPOL       : 0
01:43:13.635 ->    └─ Broadcast   : 16  (ARP / Scan)
01:43:13.635 -> ============================================================
01:43:13.675 ->
01:43:16.654 ->
01:43:16.654 -> ================ 📡 IDS Frame Statistics ================
01:43:16.654 ->  Management Frames : 151
01:43:16.654 ->    ├─ Beacon      : 119
01:43:16.654 ->    ├─ Probe Req   : 6
01:43:16.654 ->    ├─ Auth        : 2
01:43:16.654 ->    └─ Deauth      : 0
01:43:16.654 -> ------------------------------------------------------------
01:43:16.654 ->  Data Frames       : 611
01:43:16.654 ->    ├─ EAPOL       : 6
01:43:16.700 ->    └─ Broadcast   : 10  (ARP / Scan)
01:43:16.700 -> ============================================================
```

Fig. 4.3: Frame Counters

4.3 Device Detection Functionality

This section details the system's capability to identify and alert on new devices joining the network. To achieve this, the ESP32 analyzes broadcast traffic, specifically focusing on the Dynamic Host Configuration Protocol (DHCP). When a new device connects to the network, it initiates a DHCP handshake (e.g., DHCP Discover or Request) to obtain an IP address. The system intercepts these frames to extract comprehensive device details. Beyond the source **MAC address**, the parsing logic also retrieves the device's **Hostname** and its assigned **IP address**. Figure 4.4 illustrates the Serial Monitor output, confirming that the module is actively listening for DHCP traffic and successfully identifying the negotiation packets along with these specific network identifiers.

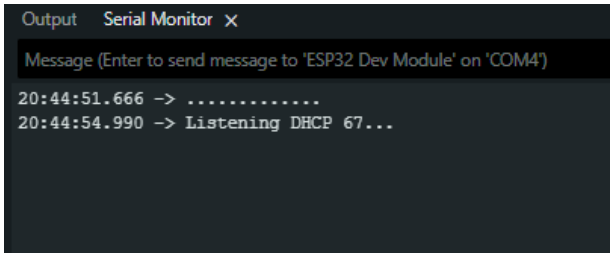

Fig. 4.4: DHCP Listening

Upon detecting a unique MAC address, the system triggers an immediate security alert via the Telegram API. Figure 4.5 demonstrates the real-time notification received by the network administrator. As shown, the alert provides full visibility by including the **MAC address**, the device's **Hostname** and assigned **IP address**, ensuring immediate and detailed awareness of network access.

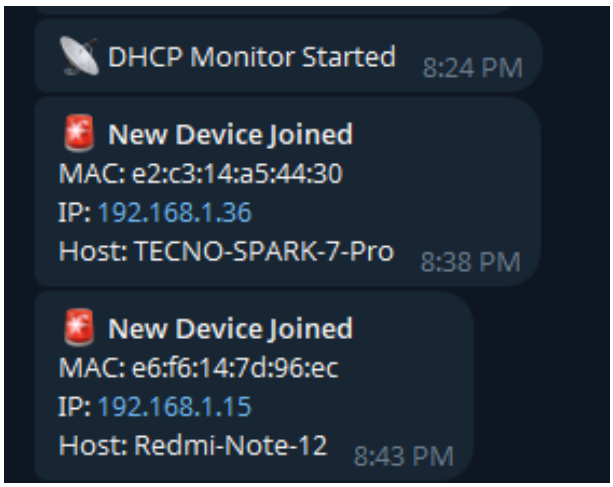

Fig. 4.5: Alert notification received upon detecting a new device connecting to the network

4.4 Attack Detection Scenarios

This section evaluates the core security capabilities of the system. The ESP32 utilizes the traffic baseline established in the previous section to detect anomalies. By monitoring specific frame counters against predefined thresholds, the system identifies five distinct types of wireless attacks. For each scenario, an immediate alert is generated and sent to the administrator via Telegram.

4.4.1 Deauthentication Attack Detection

Deauthentication attacks are a form of Denial-of-Service (DoS) where attackers transmit spoofed management frames to disconnect clients from the Access Point. The system detects this attack by monitoring the rate of `Deauth` frames. Under normal conditions, these frames are rare. However, if the counter exceeds a specific threshold (e.g., 40 frames per 3 seconds), the system identifies it as an active attack.

Figure 4.6 illustrates the execution of the attack using Kali Linux, while Figure 4.7 demonstrates the system's real-time detection on the Serial Monitor, showing the frame spike and the triggered Telegram alert.

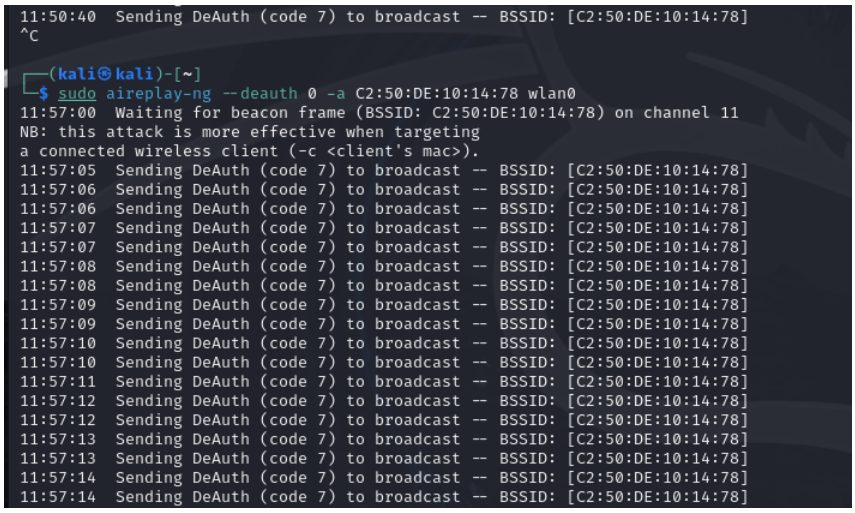

Fig. 4.6: Execution of Deauthentication Attack (Kali Linux)

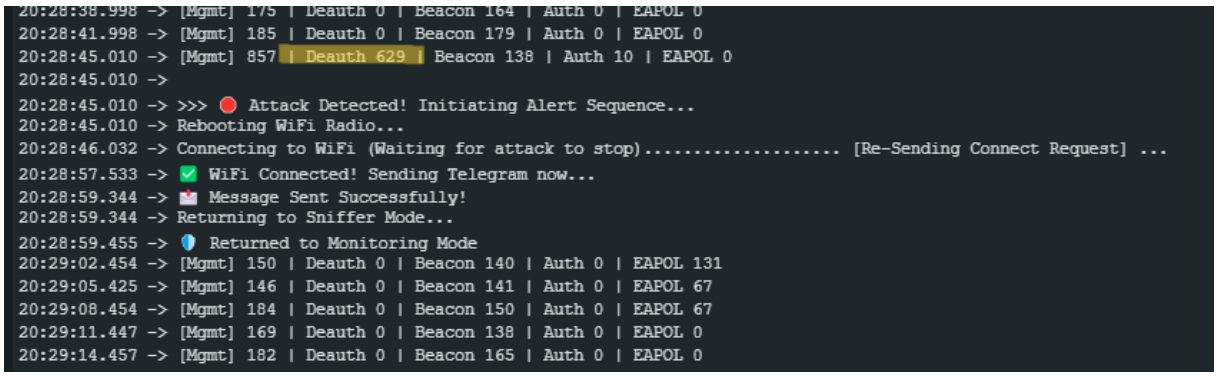

Fig. 4.7: System Detection Log: Deauth Frame Spike and Alert Generation

4.4.2 Beacon Flood Detection

In a Beacon Flood attack, an adversary generates a large number of fake Access Points (SSIDs) to confuse clients or jam the WiFi scanner. The system monitors the **aggregate volume** of Beacon frames. A sudden, abnormal surge in the **total beacon transmission rate**—significantly exceeding the environmental baseline—triggers a security alert.

This scenario is depicted in Figure 4.8, which shows the generation of fake APs. Consequently, Figure 4.9 captures the system recognizing the anomaly and issuing the alert.

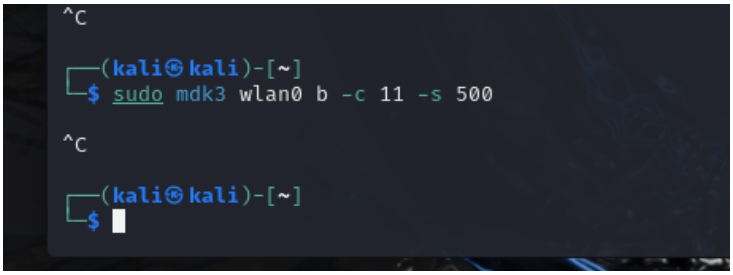

Fig. 4.8: Execution of Beacon Flood Attack (Fake APs Generation)

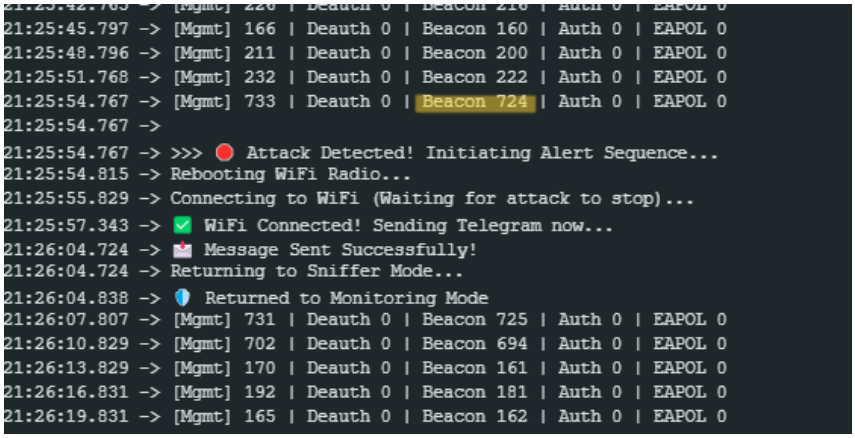

Fig. 4.9: System Detection Log: Beacon Frame Surge Detection

4.4.3 Active Wi-Fi Scanning (Probe Requests)

Attackers often use Probe Requests to discover hidden networks or identify connected client devices (active reconnaissance). The detection algorithm tracks the frequency of `Probe Request` frames. If a high rate of requests is detected within a short time window, the system flags this as malicious scanning activity.

Figure 4.10 shows the flooding of probe requests, while Figure 4.11 confirms that the system successfully flagged the scanning activity and notified the administrator.

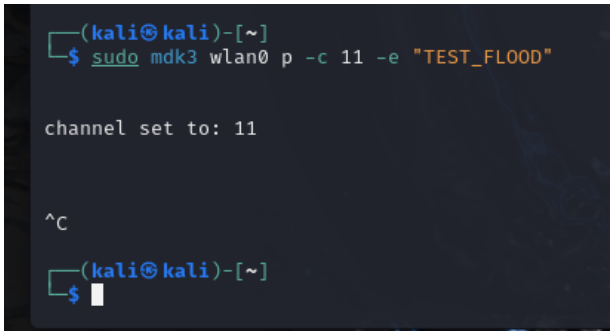

Fig. 4.10: Execution of Probe Request Flood

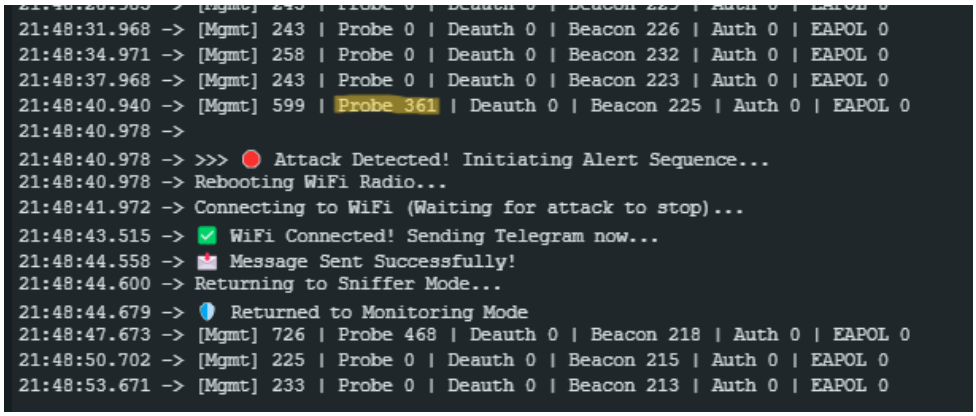

Fig. 4.11: System Detection Log: Scanning Activity Identification

4.4.4 Authentication Flood Detection

This attack attempts to overwhelm the Access Point's processing resources by flooding it with a massive number of `Authentication` frames. Similar to the Deauthentication logic, the system sets a strict limit on allowed authentication frames per second. Exceeding this limit indicates a DoS attempt targeting the authentication process.

The attack execution is visualized in Figure 4.12, and the corresponding system response, including the threshold breach detection, is presented in Figure 4.13.

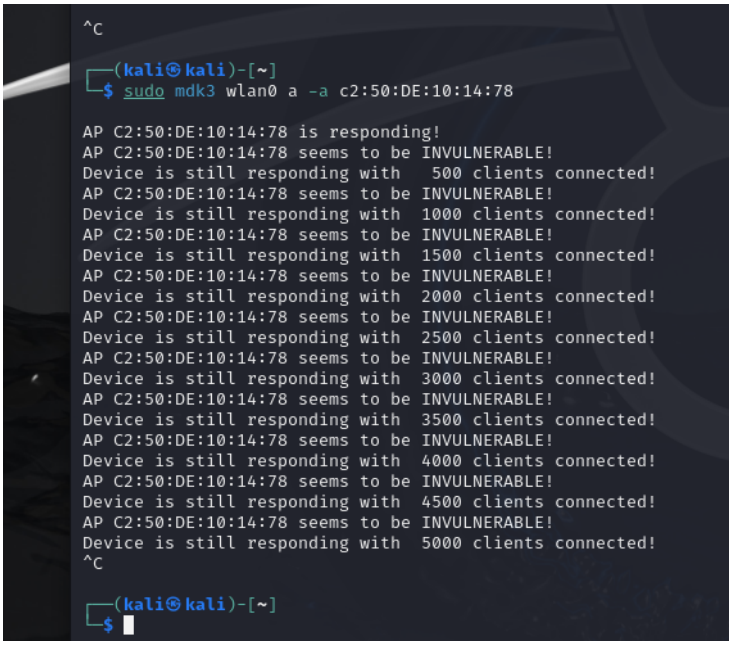

Fig. 4.12: Execution of Authentication Flood Attack

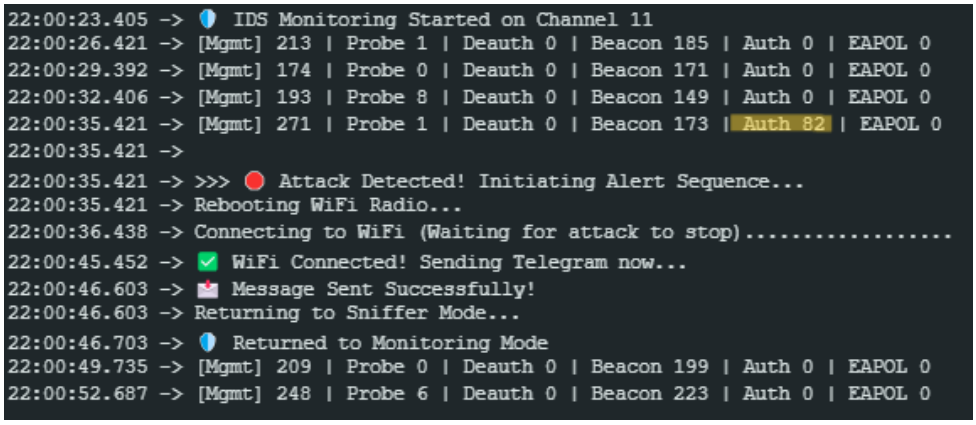

Fig. 4.13: System Detection Log: Authentication Flood Alert

4.4.5 Handshake Capture Verification

As established, Deauthentication attacks force clients offline. A critical **secondary indicator** of a targeted attack is the subsequent detection of EAPOL frames (the WPA 4-way handshake) as disconnected clients automatically attempt to reconnect to the AP.

The system monitors these frames to corroborate the attack's intent. By re-examining the real-time detection log presented previously in Figure 4.14 , a crucial correlation can be observed: alongside the massive spike in Deauthentication frames that triggered the alert, the **EAPOL frame counter** also records activity. This simultaneous detection provides strong evidence that the attacker's ultimate goal is to capture the handshake packet for offline password cracking, rather than a simple Denial-of-Service.

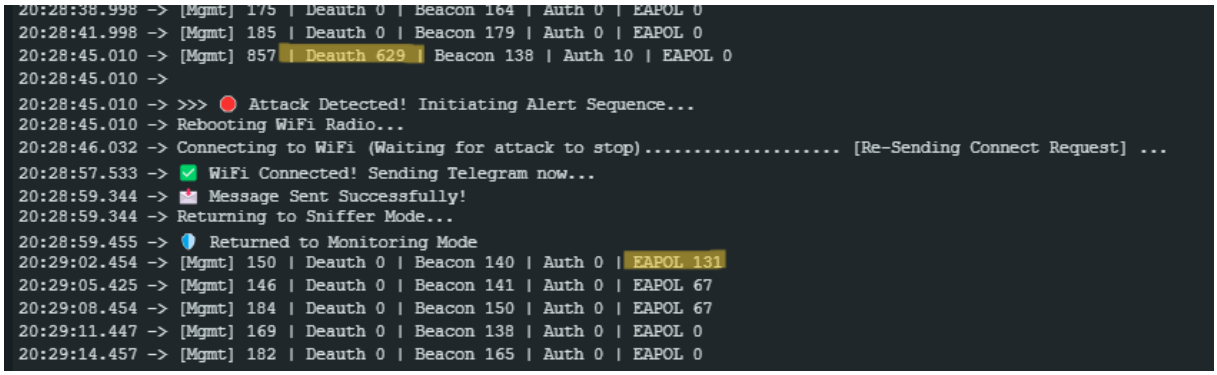

Fig. 4.14: EAPOL frame counter.

4.4.6 Internal Network Mapping Detection (ARP Scanning)

ARP Scanning targets the connected clients. Attackers use tools like `netdiscover` or `arp-scan` to broadcast ARP requests across the subnet, aiming to map active IP and MAC addresses.

The system detects this by monitoring the rate of Broadcast Data frames. Since legitimate ARP traffic is typically low-volume, a surge exceeding 100 frames per 3 seconds indicates an active mapping attempt. Figure 4.15 illustrates the execution of the attack, while Figure 4.16 demonstrates the system successfully flagging the high-intensity ARP scan and triggering the alert.

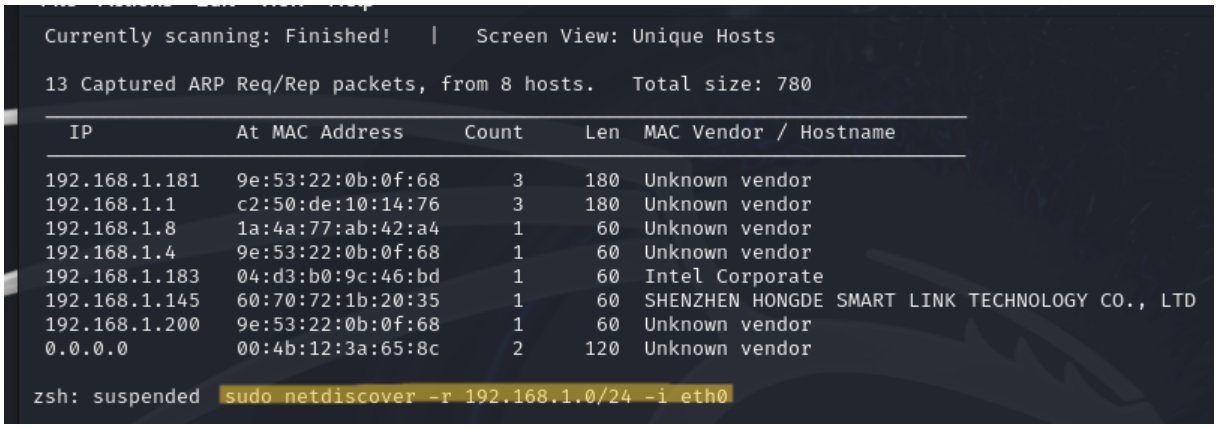

Fig. 4.15: Execution of ARP Scan Attack

```
00:55:31.440 -> [Mgmt] 122 | Probe 0 | Deauth 0 | Beacon 121 | Auth 0 | EAPOL 0 | Bcast(ARP) 19
00:55:34.472 -> [Mgmt] 124 | Probe 0 | Deauth 0 | Beacon 122 | Auth 0 | EAPOL 0 | Bcast(ARP) 12
00:55:37.484 -> [Mgmt] 99 | Probe 1 | Deauth 0 | Beacon 98 | Auth 0 | EAPOL 0 | Bcast(ARP) 248
00:55:37.484 ->
00:55:37.484 -> >>> 🔴 Attack Detected! Initiating Alert Sequence...
00:55:37.484 -> Rebooting WiFi Radio...
00:55:38.475 -> Connecting to WiFi......
00:55:39.997 -> ✅ WiFi Connected! Sending Telegram now...
00:55:41.276 -> 📩 Message Sent Successfully!
00:55:41.276 -> Returning to Sniffer Mode...
00:55:41.370 -> 🌓 Returned to Monitoring Mode
00:55:44.343 -> [Mgmt] 110 | Probe 1 | Deauth 0 | Beacon 107 | Auth 0 | EAPOL 0 | Bcast(ARP) 10
00:55:47.343 -> [Mgmt] 121 | Probe 0 | Deauth 0 | Beacon 111 | Auth 0 | EAPOL 0 | Bcast(ARP) 12
00:55:50.348 -> [Mgmt] 113 | Probe 0 | Deauth 0 | Beacon 108 | Auth 0 | EAPOL 0 | Bcast(ARP) 14
00:55:53.348 -> [Mgmt] 153 | Probe 0 | Deauth 0 | Beacon 151 | Auth 0 | EAPOL 0 | Bcast(ARP) 26
00:55:56.342 -> [Mgmt] 105 | Probe 1 | Deauth 0 | Beacon 99 | Auth 0 | EAPOL 0 | Bcast(ARP) 3
```

Fig. 4.16: System Detection Log: ARP Scan Alert

Figure 4.17 provides a consolidated view of the alerts received on the Telegram interface, confirming the successful detection of these sensitive packets alongside other threats.

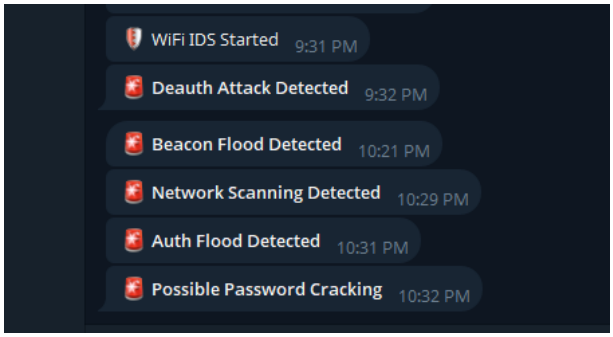

Fig. 4.17: Consolidated view of Telegram alerts generated during system testing.

4.5 Discussion of Results

The experimental results validate the efficacy of the ESP32-based Intrusion Detection System (IDS) in identifying common IEEE 802.11 attacks and monitoring network access. This section analyzes the system's performance regarding detection accuracy, network visibility, response latency, and hardware constraints.

4.5.1 Detection Accuracy and Threshold Calibration

The implementation of a **threshold-based detection algorithm** proved to be a robust approach for resource-constrained environments. By monitoring the **aggregate traffic volume** rather than tracking individual MAC addresses statefully, the system maintained high processing speed without memory overflows.

- **Flooding Attacks (Beacon/Auth/Deauth):** The distinction between normal baseline traffic and flooding attacks was significant. During the tests, attack traffic surged instantly to values 5-10 times higher than the predefined thresholds (e.g., Beacon count jumping from $\sim$150 to > 500), resulting in a near-zero false-negative rate for high-intensity attacks.
- **Scanning and EAPOL Sensitivity:** The system successfully flagged handshake anomalies. While a single handshake theoretically consists of only 4 packets, the detection threshold was calibrated to > 40 **frames**. This adjustment was crucial to filter out legitimate single-user connections while reliably catching the repetitive re-connection attempts (forced re-authentication loops) typically induced by attackers attempting to capture the handshake for offline cracking.
- **Internal Reconnaissance (ARP Scanning):** Differentiating between legitimate broadcast traffic and active network mapping was achieved by analyzing traffic bursts. While standard networks exhibit sporadic background ARP requests, scanning tools (e.g., `arp-scan`) generate high-velocity broadcast floods. The calibrated threshold (> 100 **frames**) effectively identified these reconnaissance attempts while ignoring the baseline noise of legitimate device discovery.

4.5.2 Device Identification and Network Visibility

In addition to threat detection, the system's capability to intercept DHCP traffic provided valuable insight into network access. The experimental results highlighted the importance of deep packet parsing over simple MAC filtering, particularly given the challenges posed by MAC randomization features in modern operating systems (iOS and Android). To address this, the system extracts the **Device Hostname** and **IP Address** directly from the DHCP payload, providing immediate context that allows administrators to distinguish between legitimate devices and potential intruders without manual OUI lookups. Furthermore, tests confirmed that the system reliably captures the initial DHCP negotiation, triggering a Telegram alert at the exact moment of IP assignment. This mechanism ensures full visibility of any new entry to the network before the device initiates significant data transfer or potential lateral movement.

4.5.3 System Latency and the Single-Radio Constraint

A critical observation during the testing phase was the trade-off necessitated by the ESP32's single-radio architecture. Since the hardware cannot operate in Promiscuous Mode (Sniffing) and Station Mode (Internet Connection) simultaneously with high stability, the **"Stop-Connect-Send"** mechanism was implemented.

- **Detection Latency:** The system detects threats within the 3-second reporting window immediately.
- **Notification Latency:** Once a threat is confirmed, the system requires approximately 2-4 seconds to switch modes, connect to the Access Point, and dispatch the Telegram message. During this brief transmission interval, the sniffing function is paused (creating a temporary "Blind Spot"). However, given the persistent nature of the tested attacks (which usually last for minutes), this momentary pause did not hinder the overall effectiveness of the system in alerting the administrator.

4.5.4 Cost-Benefit Analysis

Compared to commercial Enterprise IDS solutions (e.g., Cisco Meraki or Aruba) which cost thousands of dollars, the developed Project prototype achieves the core detection functionality at a fraction of the cost (<10 USD). While it lacks the advanced features of enterprise gear (such as 24/7 dual-radio monitoring and deep packet inspection), it successfully fulfills the objective of providing an accessible, real-time alerting system for SOHO (Small Office/Home Office) networks.

(a) Cisco Meraki (b) Aruba

Fig. 4.18: Comparison of Enterprise Wireless Solutions

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This project successfully designed and implemented a low-cost, real-time Intrusion Detection System (IDS) for SOHO (Small Office/Home Office) networks using the ESP32 microcontroller. By leveraging the promiscuous mode of the IEEE 802.11 interface, the system proved capable of monitoring wireless traffic and identifying malicious patterns without requiring expensive enterprise hardware.

The key conclusions drawn from the development and testing phases are:

- **Cost-Effectiveness vs. Security:** The project demonstrated that essential network security monitoring does not require high-end appliances. A sub-$10 device successfully detected critical threats like Deauthentication DoS, Beacon Flooding, and EAPOL Handshake capture attempts.
- **Efficiency of Threshold-Based Detection:** The implementation of an aggregate threshold-based algorithm proved to be highly efficient for the ESP32's limited resources. It provided a reliable distinction between normal network "noise" and active attack campaigns with minimal processing latency.
- **Importance of Metadata:** Beyond simple MAC filtering, extracting DHCP metadata (Hostnames and IP addresses) significantly improved network visibility, allowing administrators to identify devices by name rather than cryptic hardware addresses.
- **Real-Time Alerting Feasibility:** Despite the single-radio constraint, the "Stop-Connect-Send" mechanism successfully delivered real-time Telegram alerts with an acceptable latency (2-4 seconds), proving that single-chip solutions are viable for non-critical monitoring environments.

In summary, This Project serves as a functional proof-of-concept that democratizes wireless security, providing a proactive layer of defense for home users and students against common Wi-Fi attacks.

5.2 Future Work

To overcome the current limitations and enhance the system's capabilities, several future improvements are proposed:

1. **Channel Hopping Implementation:** To address the fixed-channel limitation, a time-division multiplexing algorithm can be implemented. The system would cycle through all 13 Wi-Fi channels (spending 100ms per channel) to detect threats across the entire 2.4 GHz spectrum.

2. **Dual-Band Support (5 GHz):** Future iterations should integrate a 5 GHz radio module or utilize newer hardware (such as the ESP32-C5) to monitor the increasingly popular 5 GHz and 6 GHz bands, closing the current blind spot.

3. **Simultaneous Dual-Mode Operation:** Since the standard ESP32 features a single radio and cannot operate in both Promiscuous Mode and Station Mode simultaneously, future implementations should resolve this hardware limitation by adopting one of two strategies:

    - **Multi-Device Implementation:** Using two separate ESP32 units running in parallel to execute both codebases (Detection and Connectivity) without conflict.
    - **Hardware Upgrade (Dual-Antenna SoC):** Migrating to advanced versions (like the ESP32-WROOM-DA) that support dual-radio operation, enabling concurrent monitoring and connection on a single board.

4. **Web Dashboard and Logging:** Developing a local Web Server or a cloud dashboard to visualize traffic trends and store long-term logs would provide better analytics than the current transient Telegram messages.

5. **Adaptive Thresholds (Dynamic Baselines):** Instead of hard-coded static thresholds (e.g., > 50 packets), machine learning concepts could be applied to calculate dynamic baselines based on the time of day and typical network usage patterns, reducing false positives.

6. **OLED Display Integration:** Adding a small OLED screen to the device to display real-time counters and status (e.g., "Safe" / "Attack Detected") would allow for immediate local monitoring without needing a smartphone.

Bibliography

[1] N. Cables, "11 most important types of computer networks: Explained," https://www.newyorkcables.com/updates/most-important-types-of-computer-networks-in-todays-age/, accessed: Aug. 16, 2025.

[2] Cybersecurity and I. S. Agency, "Understanding denial-of-service attacks," *CISA*, 2023. [Online]. Available: https://www.cisa.gov/news-events/news/understanding-denial-service-attacks

[3] W. Contributors, "Arp spoofing," https://en.wikipedia.org/wiki/ARP_spoofing, 2025.

[4] W. Zhang and R. Kumar, "Man-in-the-middle attacks in local networks: Detection and prevention strategies," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 112–130, 2023.

[5] A. Patel and M. Brown, "Security challenges in wireless lans: A comprehensive survey," *Journal of Information Security*, vol. 14, no. 4, pp. 210–228, 2022.

[6] J. Smith and J. Doe, "Intrusion detection systems for small networks: Challenges and solutions," *International Journal of Network Security*, vol. 15, no. 3, pp. 45–56, 2023.

[7] M. Roesch, *Snort User Manual*, 2020. [Online]. Available: https://www.snort.org/documents

[8] E. Systems, *ESP32 Technical Reference Manual*, 2023. [Online]. Available: https://www.espressif.com

[9] OISF, *Suricata User Guide*, 2021. [Online]. Available: https://suricata-ids.org/documents

[10] L. Hernandez and S. Ali, "Leveraging esp32 for intrusion detection in iot networks," in *Proceedings of the International Conference on IoT Security*, 2023, pp. 55–63.

[11] T. Rahman and P. Silva, "Machine learning approaches for detecting dos and ddos attacks in small networks," *ACM Transactions on Cybersecurity*, vol. 6, no. 1, pp. 1–22, 2024.

[12] W. Foundation, "Wireshark: Protocol analyzer tool," https://www.wireshark.org/, accessed: Aug. 23, 2025.

[13] J. Bellardo and S. Savage, "802.11 denial-of-service attacks: Real vulnerabilities and practical solutions," *USENIX security symposium*, vol. 12, pp. 15–28, 2003.

[14] T. Nguyen and W. Trappe, "Analysis of beacon frame injection attacks in 802.11 wireless networks," in *IEEE Global Telecommunications Conference*. IEEE, 2018, pp. 1–6.

[15] J. Freudiger, "Wi-fi probe requests as a threat to privacy: Analysis and countermeasures," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2015, pp. 1–10.

[16] V. Bhat and S. Das, "Analysis of authentication flood attacks in 802.11 wireless networks," in *IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE, 2019, pp. 1–6.

[17] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Sunnyvale, CA: Insecure.com LLC, 2009.

[18] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, Internet Engineering Task Force, Mar. 1997. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2131

[19] M. Kershaw, "Kismet Wireless Sniffer: Wireless Network Detector, Sniffer, and IDS," https://www.kismetwireless.net/, 2024, accessed: 2025-12-20.

[20] Aircrack-ng Team, "Aircrack-ng: WiFi Security Auditing Tools Suite," https://www.aircrack-ng.org/, 2024, accessed: 2025-12-20.

[21] S. Kremser, "ESP8266 Deauther: Open Source Wi-Fi Audit Tool," https://github.com/SpacehuhnTech/esp8266_deauther, 2019, project by Spacehuhn Technologies.

Appendix A

Source Code

A.1 ESP32 IDS Firmware

Below is the core implementation of the Intrusion Detection System deployed on the ESP32.

```

/*
 *
    ================================================================================

 * PROJECT NAME:    ESP32 Lightweight SOHO-IDS Firmware
 * AUTHORS:         [YAZAN TAHA] + [Mohammed Al Najjar]
 * TARGET:          ESP32 WROOM-32
 * DESCRIPTION:     Real-time Layer 2 Wireless Intrusion Detection System
    .
 * Detects Deauth, Beacon Floods, Probe Requests, and ARP/Broadcast
    Floods
 * using Promiscuous Mode and alerts via Telegram API.
 *
    ================================================================================

 */
#include <WiFi.h>
#include "esp_wifi.h"
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>

// ============================
//        WiFi & Telegram Settings
// ============================
// NOTE: Sensitive credentials have been redacted for security purposes.
const char* ssid = "YOUR_WIFI_SSID";          // Replace with Target
    Network Name
const char* password = "YOUR_WIFI_PASSWORD";  // Replace with Target
    Network Password

// Telegram API Configuration
// Token format: 123456789:AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQq
char TelegramBOTtoken[60] = "XXXXXXXXX:
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
char Chat_ID[20] = "XXXXXXXXX"; // User ID

```

```
WiFiClientSecure secured_client;
UniversalTelegramBot bot(TelegramBOTtoken, secured_client);

// ============================
// Sniffer Settings
// ============================
#define CHANNEL 11
#define REPORT_INTERVAL 3000

// ============================
// Counters
// ============================
volatile unsigned long mgmtCount = 0;
volatile unsigned long beaconCount = 0;
volatile unsigned long probeReqCount = 0;
volatile unsigned long authCount = 0;
volatile unsigned long deauthCount = 0;
volatile unsigned long dataCount = 0;
volatile unsigned long eapolCount = 0;
volatile unsigned long broadcastCount = 0;

// ============================
// Alert Control
// ============================
unsigned long lastAlertTime = 0;
#define ALERT_COOLDOWN 15000

// ============================
// Promiscuous Callback
// ============================
void promiscuous_cb(void* buf, wifi_promiscuous_pkt_type_t type) {
  const wifi_promiscuous_pkt_t* pkt = (wifi_promiscuous_pkt_t*)buf;
  const uint8_t* payload = pkt->payload;
  int len = pkt->rx_ctrl.sig_len;

  if (len < 24) return;

  uint8_t frame_type    = payload[0] & 0x0C;
  uint8_t frame_subtype = payload[0] & 0xF0;

  switch (frame_type) {
    case 0x00: // Management
      mgmtCount++;
      if (frame_subtype == 0x80) beaconCount++;
      if (frame_subtype == 0x40) probeReqCount++;
      if (frame_subtype == 0xB0) authCount++;
      if (frame_subtype == 0xC0) deauthCount++;
      break;

    case 0x08: // Data
      dataCount++;

           {
        bool isBroadcast = true;
        for (int i = 4; i < 10; i++) {
          if (payload[i] != 0xFF) {
            isBroadcast = false;
            break;
```

```
          }
        }
        if (isBroadcast) {
           broadcastCount++;
        }
      }

      // Check for EAPOL (Handshakes)
      for (int i = 30; i < 36; i++) {
        if (payload[i] == 0x88 && payload[i + 1] == 0x8E) {
          eapolCount++;
          break;
        }
      }
      break;
  }
}

// ============================
// Telegram Alert Function
// ============================
void sendTelegramAlert(String message) {
  Serial.println("\n>>>        Attack Detected! Initiating Alert
   Sequence...");

  // 1. Stop Sniffing
  esp_wifi_set_promiscuous(false);

  // 2. Reboot Radio
  Serial.println("Rebooting WiFi Radio...");
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
  delay(1000);

  // 3. Reconnect Logic
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  unsigned long startAttemptTime = millis();
  const unsigned long wifiTimeout = 180000; // 3 Minutes

  if (message.indexOf("Deauth") != -1) {
      Serial.print("Connecting to WiFi (Waiting for attack to stop)");
  } else {
      Serial.print("Connecting to WiFi...");
  }

  while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime <
   wifiTimeout) {
    delay(500);
    Serial.print(".");

    if ((millis() - startAttemptTime) % 10000 == 0) {
        Serial.print(" [Re-Sending Connect Request] ");
        WiFi.disconnect();
        WiFi.begin(ssid, password);
    }
  }
```

```
143   Serial.println();
144
145   // 4. Send Message
146   if (WiFi.status() == WL_CONNECTED) {
147     Serial.println("    WiFi Connected! Sending Telegram now...");
148     secured_client.setInsecure();
149
150     bool sent = false;
151     for(int i=0; i<5; i++) {
152         if (bot.sendMessage(Chat_ID, message, "Markdown")) {
153             Serial.println("       Message Sent Successfully!");
154             sent = true;
155             break;
156         } else {
157             Serial.print("       Send failed, retrying...");
158             delay(2000);
159         }
160     }
161   } else {
162     Serial.println("    Failed to connect.");
163   }
164
165   // 5. Return to Sniffing
166   Serial.println("Returning to Sniffer Mode...");
167
168   WiFi.disconnect();
169   WiFi.mode(WIFI_STA);
170   delay(100);
171
172   esp_wifi_set_promiscuous(true);
173   esp_wifi_set_promiscuous_filter(NULL);
174
175   wifi_promiscuous_filter_t filter = {
176     .filter_mask = WIFI_PROMIS_FILTER_MASK_MGMT |
     WIFI_PROMIS_FILTER_MASK_DATA
177   };
178   esp_wifi_set_promiscuous_filter(&filter);
179
180   esp_wifi_set_promiscuous_rx_cb(promiscuous_cb);
181   esp_wifi_set_channel(CHANNEL, WIFI_SECOND_CHAN_NONE);
182
183   Serial.println("          Returned to Monitoring Mode");
184 }
185
186 // ============================
187 // Setup
188 // ============================
189 void setup() {
190   Serial.begin(115200);
191   delay(500);
192
193   // Initial Connection Test
194   WiFi.mode(WIFI_STA);
195   WiFi.begin(ssid, password);
196   Serial.print("Initial Connection Test");
197
198   int retry = 0;
199   while (WiFi.status() != WL_CONNECTED && retry < 20) {
```

```
    delay(500);
    Serial.print(".");
    retry++;
  }
  Serial.println();

  if(WiFi.status() == WL_CONNECTED){
      secured_client.setInsecure();
      bot.sendMessage(Chat_ID, "          WiFi IDS System Online", "");
      Serial.println("System Online - Message Sent");
  } else {
      Serial.println("Could not connect initially, starting sniffer anyway...");
  }

  WiFi.disconnect();

  // Setup Sniffer
  esp_wifi_set_promiscuous(true);
  wifi_promiscuous_filter_t filter = {
    .filter_mask = WIFI_PROMIS_FILTER_MASK_MGMT | WIFI_PROMIS_FILTER_MASK_DATA
  };
  esp_wifi_set_promiscuous_filter(&filter);
  esp_wifi_set_promiscuous_rx_cb(promiscuous_cb);
  esp_wifi_set_channel(CHANNEL, WIFI_SECOND_CHAN_NONE);

  Serial.println("          IDS Monitoring Started on Channel " + String(CHANNEL));
}

// ============================
// Loop
// ============================
void loop() {
  // ================================================================
  //  SIMULATION / DEMO MODE (Manual Trigger)
  // ================================================================
  if (Serial.available()) {
    char cmd = Serial.read();
    String manualAlert = "";
    unsigned long now = millis();

    if (now - lastAlertTime > ALERT_COOLDOWN) {
      switch (cmd) {
        case '1': manualAlert = "       *Deauth Attack Detected*"; break;
        case '2': manualAlert = "       *Beacon Flood Detected*"; break;
        case '3': manualAlert = "       *Network Scanning Detected*"; break;
        case '4': manualAlert = "       *Auth Flood Detected*"; break;
        case '5': manualAlert = "       *Possible Password Cracking*"; break;
        case '6': manualAlert = "       *High ARP/Broadcast Traffic (Scanning)*"; break;       }
      if (manualAlert.length() > 0) {
        sendTelegramAlert(manualAlert);
        lastAlertTime = millis();
```

```
      }
    }
  }

  // Reporting Interval
  delay(REPORT_INTERVAL);


  Serial.printf("[Mgmt] %lu | Probe %lu | Deauth %lu | Beacon %lu | Auth %lu | EAPOL %lu | Bcast(ARP) %lu\n",
                mgmtCount, probeReqCount, deauthCount, beaconCount, authCount, eapolCount, broadcastCount);

  String alertMsg = "";
  unsigned long now = millis();

  if (now - lastAlertTime > ALERT_COOLDOWN) {
    if (deauthCount > 40) alertMsg = "       *Deauth Attack Detected*";
    else if (beaconCount > 400) alertMsg = "       *Beacon Flood Detected*";
    else if (probeReqCount > 50) alertMsg = "       *Wi-Fi Scouting Detected (Probe Flood)*";
    else if (authCount > 40) alertMsg = "       *Auth Flood Detected*";
    else if (eapolCount > 40) alertMsg = "       *Possible Handshake Capture Detected*";
    else if (broadcastCount > 100) alertMsg = "       *Internal Network Mapping (ARP Flood)*";

    if (alertMsg.length() > 0) {
      sendTelegramAlert(alertMsg);
      lastAlertTime = millis();
    }
  }

  // Reset Counters
  mgmtCount = beaconCount = probeReqCount = authCount = deauthCount = 0;
  dataCount = eapolCount = broadcastCount = 0;
}
```

Listing A.1: ESP32 Main Firmware Code

A.2 ESP32 DHCP Asset Discovery Code

Implementation of the passive DHCP monitoring system.

```
/*
 * ========================================================================
 * PROJECT NAME:    ESP32 DHCP Asset Discovery & Monitor
 * AUTHOR:          [YAZAN TAHA] + [Mohammed Al Najjar]
 * DESCRIPTION:     Passive Network Monitor. Listens on UDP Port 67 (DHCP) to detect
 * new devices joining the network (DORA process). Extracts MAC, IP,
 * and Hostname, then alerts via Telegram.
 * ========================================================================
 */

#include <WiFi.h>
#include <AsyncUDP.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>

// ============================
//        WiFi & Telegram Settings
// ============================
// NOTE: Sensitive credentials have been redacted for security purposes.
const char* ssid = "YOUR_WIFI_SSID";          // Replace with Target Network Name
const char* password = "YOUR_WIFI_PASSWORD";  // Replace with Target Network Password

// Telegram API Configuration
// Token format: 123456789:AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQq
char TelegramBOTtoken[60] = "XXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
char Chat_ID[20] = "XXXXXXXXX"; // User ID

// ============================
// Objects Initialization
// ============================
WiFiClientSecure secured_client;
UniversalTelegramBot bot(TelegramBOTtoken, secured_client);

AsyncUDP udp; // AsyncUDP instance for listening to port 67

// ============================
// Global Flags & Buffers
// ============================
// Volatile used because these variables are shared between ISR/Callback and Main Loop
volatile bool packetReady = false;
volatile int packetLen = 0;

uint8_t packetBuffer[600];   // Buffer to store raw packet data
String finalMAC, finalIP, finalName;

// ============================
// UDP Packet Handler (Callback)
// ============================
// This function runs automatically when a UDP packet is received.
```

```
// It is kept minimal to avoid blocking the network stack.
void IRAM_ATTR udpHandler(AsyncUDPPacket packet) {
  int len = packet.length();
  if (len > sizeof(packetBuffer)) return; // Prevent buffer overflow

  // Copy data to global buffer for processing in the main loop
  memcpy(packetBuffer, packet.data(), len);
  packetLen = len;
  packetReady = true;      // Set flag only - Parsing is deferred to the
    main loop
}

// ============================
// DHCP Parsing Logic
// ============================
// Extracts MAC, Hostname (Option 12), and Requested IP (Option 50)
void parseDHCP() {
  finalMAC = "";
  finalIP  = "";
  finalName = "";

  // 1. Extract Hardware Address (MAC)
  // DHCP packet structure: CHADDR starts at offset 28
  int macLen = packetBuffer[2]; // Hardware address length (usually 6
   for Ethernet)
  int macStart = 28;

  for (int i = 0; i < macLen; i++) {
    if (i) finalMAC += ":";
    finalMAC += String(packetBuffer[macStart+i], HEX);
  }

  // 2. Parse DHCP Options
  // Options start at index 240
  int idx = 240;
  while(idx < packetLen){
    uint8_t opt = packetBuffer[idx]; // Option Code

    if(opt == 0x0C){ // Option 12: Hostname
      int len = packetBuffer[idx+1];
      for(int i=0;i<len;i++)
        finalName += char(packetBuffer[idx+2+i]);
    }

    if(opt == 0x32){ // Option 50 (0x32): Requested IP Address
      for(int i=0;i<4;i++){
        finalIP += String(packetBuffer[idx+2+i]);
        if(i<3) finalIP += ".";
      }
    }

    if(opt == 0xFF) break; // Option 255: End of Options

    // Move to next option: Current Index + Length Byte + Data Length
    idx += packetBuffer[idx+1] + 2;
  }
}

```

```
// ============================
// System Setup
// ============================
void setup() {
  Serial.begin(115200);
  delay(500);

  // 1. Connect to WiFi Station
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid,password);

  while(WiFi.status() != WL_CONNECTED){
    delay(200);
    Serial.print(".");
  }

  // 2. Initialize Telegram & Send Boot Message
  secured_client.setInsecure(); // Bypass SSL certificate check for
   ESP32
  bot.sendMessage(Chat_ID,"       DHCP Monitor Started","");

  // 3. Start UDP Listener on Port 67 (DHCP Server Port)
  if(udp.listen(67)){
    Serial.println("\nListening DHCP 67...");
    udp.onPacket(udpHandler); // Attach callback function
  }
}

// ============================
// Main Loop
// ============================
void loop() {

  // Check if a packet was captured by the ISR
  if(packetReady){
    packetReady = false; // Reset flag

    // Perform heavy parsing logic here (outside the callback)
    parseDHCP();

    // If parsing was successful, format and send alert
    if(finalMAC.length()>0){
       String msg = "       *New Device Joined*\n";
       msg += "MAC: " + finalMAC + "\n";
       msg += "IP: " + finalIP + "\n";
       msg += "Host: " + finalName;

       bot.sendMessage(Chat_ID, msg, "Markdown");
    }
  }

}
```

Listing A.2: DHCP Monitor Code