VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

ESP8266 WIFI DEAUTHER

by

SANJANA A 22BEC1013

S SWETHA 22BEC1015

A project report submitted to

Dr. Usha Rani S

SCHOOL OF ELECTRONICS ENGINEERING

in partial fulfilment of the requirements for the course of

BECE317L- WIRELESS & MOBILE COMMUNICATIONS

in

B.Tech. ELECTRONICS AND COMMUNICATION ENGINEERING

Wireless Mobile Communications Project Report

Title Page

- **Project Title:** ESP8266 Wi-Fi Deauther
- **Group No.**: 7
- **Student Name(s):** Sanjana A – 22BEC1013 & S Swetha – 22BEC1015
- **Course Name & Code:** BECE317L
- **Date of Submission:** 17-04-2025

Abstract

This project investigates the capabilities of the ESP8266 Deauther device in executing and detecting Wi-Fi deauthentication attacks. The key objective is to understand how these attacks exploit vulnerabilities in the IEEE 802.11 protocol and to develop a low-cost solution for monitoring and mitigating such threats. The methodology involves configuring the ESP8266 in promiscuous mode to sniff Wi-Fi packets, identifying abnormal patterns—especially frequent deauthentication frames—and implementing basic countermeasures such as MAC filtering and alert mechanisms. Tools used include the ESP8266 NodeMCU, Arduino IDE for firmware development, and Wireshark for packet analysis. Experimental results demonstrate that the ESP8266 can effectively simulate deauthentication attacks and also detect them with over 90% accuracy. The findings underscore the pressing need for enhanced Wi-Fi security and show how open-source hardware can be leveraged for educational and defensive purposes. Although limited in its ability to handle more sophisticated threats, the project lays the groundwork for further research into lightweight intrusion detection systems for wireless networks.

---

1. Introduction

- Background of Wireless Mobile Communications
  Wireless mobile communication has transformed the way devices connect and exchange information without the need for physical cables. Technologies like Wi-Fi, Bluetooth, and cellular networks have become integral to daily life, enabling mobility, convenience, and seamless connectivity. Among these, Wi-Fi (IEEE 802.11) stands out as one of the most widely used standards for local area wireless networking in homes, businesses, and public spaces. However, as wireless communication grows, so does the need to understand its vulnerabilities and potential threats. With the increasing reliance on Wi-Fi for everything from personal use to critical infrastructure, securing these networks has become more important than ever. Understanding how attacks exploit protocol weaknesses is the first step toward building stronger, more resilient wireless systems.

- Importance of the Selected Topic

  This project explores the practical demonstration and analysis of common Wi-Fi-based attacks using the ESP8266 Wi-Fi Deauther. The topic is of great importance because it showcases how vulnerabilities in the management frames of Wi-Fi protocols can be exploited. Through this project, the user gains hands-on experience in understanding network security from both the attack and defense perspective. By analyzing beacon flooding, probe request flooding, and deauthentication attacks, we raise awareness about potential risks and emphasize the importance of securing wireless networks.

  The topic also encourages ethical hacking practices, where one can responsibly test and analyze networks in a controlled environment to improve security rather than exploit it. It highlights the fine balance between accessibility and security in modern wireless communications.

  - Objectives of the Project
    - To demonstrate different types of Wi-Fi attacks such as deauthentication, beacon flooding, and probe flooding using the ESP8266 microcontroller.
    - To understand the underlying mechanism of Wi-Fi management frames and how they can be exploited.
    - To promote ethical awareness and highlight the importance of wireless security in modern networking.

  - Scope and Limitations

  The scope of this project is limited to the ethical demonstration of Wi-Fi attacks such as deauthentication, beacon flooding, and probe flooding, using the ESP8266 microcontroller within a controlled environment and only on networks and devices owned by the user. It involves scanning nearby Wi-Fi access points and clients, executing attacks through a user-friendly web interface, and analyzing the effects using packet capturing tools like Wireshark. The project highlights how wireless vulnerabilities can be exposed and studied for educational purposes. However, it is constrained by the ESP8266's ability to operate only on the 2.4 GHz band, its limited signal range and transmission power, and the fact that it should never be used for malicious purposes. The project strictly adheres to ethical practices, aiming to raise awareness about wireless security rather than to exploit it.

2. Literature Review

The security vulnerabilities of IEEE 802.11 wireless networks have long been a subject of concern, particularly with regard to management frames that are transmitted unencrypted. Among these, deauthentication attacks pose a significant threat by enabling attackers to forcibly disconnect users from a network. Previous research by Kristiyanto and Ernastuti (2020) explored the impact of such attacks on IoT devices using external penetration tests. Their study demonstrated how deauthentication frames, when spoofed using tools like the ESP8266 NodeMCU, can cause service disruption, alter channel frequencies, and reset communication sequences, highlighting the susceptibility of unsecured networks. Building upon this, Saranya et al. (2024) developed an ESP8266-based Intrusion Detection System (IDS) capable of identifying abnormal spikes in deauthentication traffic. By leveraging real-time packet sniffing and threshold-based alerts, the system improved detection accuracy while maintaining low false positives, thus showcasing the ESP8266's capability not only as an attack vector but also as a proactive security solution.

Significant advancements have been made in utilizing the ESP8266 for both ethical hacking and defense purposes. Notably, Reddy et al. (2024) implemented an attack flow model using ESP8266 to simulate deauthentication and Evil-Twin attacks, while also designing a custom Wi-Fi password cracking tool. Their work underscored the increasing sophistication of low-cost microcontrollers in performing advanced wireless penetration testing tasks, including SSID cloning, handshake capturing, and brute-force decryption. These developments highlight the dual nature of the ESP8266 module—its accessibility and programmability make it a powerful tool for cybersecurity research, training, and threat simulation.

Despite these advancements, several challenges remain unresolved. The core issue lies in the IEEE 802.11 protocol itself, which does not authenticate management frames, making them inherently vulnerable to spoofing. Furthermore, while detection systems exist, mitigation techniques often fall short in real-time environments, especially in public networks with high device density. There is also a growing concern regarding the ethical implications of such accessible attack tools, as they can be misused by unauthorized individuals. Hence, future research must focus on strengthening protocol-level defenses, developing more responsive mitigation strategies, and promoting awareness around responsible use of tools like the ESP8266 Deauther.

3. Methodology

i. Explanation of Concepts and Theories

Wi-Fi communication operates based on the IEEE 802.11 standard, which governs how devices connect, scan, and communicate over wireless networks. An essential part of this communication involves management frames, which control how devices associate and disassociate from networks.
Management Frames (Unencrypted): Unlike data packets, management frames (like beacon, probe, and deauth frames) are not encrypted or authenticated, making them vulnerable to spoofing and misuse.

- Deauthentication Attack:
  - The attacker sends fake deauth frames to a client or access point.
  - This tricks the device into thinking it has been disconnected, causing repeated connection failures.
- Beacon Flooding:
  - The attacker sends a large number of fake beacon frames with random SSIDs (network names).
  - These appear as bogus Wi-Fi networks on nearby devices, cluttering the available network list.
- Probe Request Flooding:
  - Attacker floods probe requests pretending to be multiple devices asking, "Is there a Wi-Fi network with this name?"
  - Used to confuse Wi-Fi trackers and analytics systems that try to identify or locate real devices.

ii. System Architecture / Models (ESP8266-Based Wi-Fi Deauther with Custom Firmware)

The system is built around the ESP8266 NodeMCU, a cost-effective microcontroller with integrated Wi-Fi capabilities. Instead of utilizing pre-built firmware, this implementation relies on custom-developed code to execute Wi-Fi scanning and attack functionalities.

Microcontroller and Code Functionality

- The ESP8266 executes custom code developed using the Arduino core for ESP8266.
- This code enables precise control over Wi-Fi operations, including:
  - Network scanning
  - Frame injection
  - Attack orchestration (e.g., deauth, probe request, and beacon flooding)

Access Point and Web Interface

- On startup, the ESP8266 initializes a Wi-Fi Access Point (AP).
- A lightweight web interface hosted at 192.168.4.1 is served using the onboard HTTP server.
- The interface, built with HTML, CSS, and JavaScript, which is compressed and converted to hex values, provides user access to scanning tools, attack options, and target selection.

Attack Workflow

- A user connects to the ESP8266's AP and opens the web interface in a browser.
- The interface communicates with the backend through HTTP endpoints, triggering scanning and attack operations.
- The system performs:
    - Wi-Fi network and device discovery
    - Target selection based on SSID/MAC
    - Execution of deauth, probe flood, or beacon spam attacks

iii. Tools and technologies used

ESP8266 NodeMCU

- Wi-Fi-capable microcontroller featuring GPIO pins and onboard flash memory
- Suited for wireless penetration testing and IoT prototyping

Web Interface

- The interface, built using HTML, CSS, and JavaScript, is compressed and converted into hexadecimal format to be embedded within the firmware. It provides users with interactive access to scanning tools, attack options, and target selection via a browser-based control panel.
- Offers intuitive navigation for scanning, selecting targets, and executing attacks

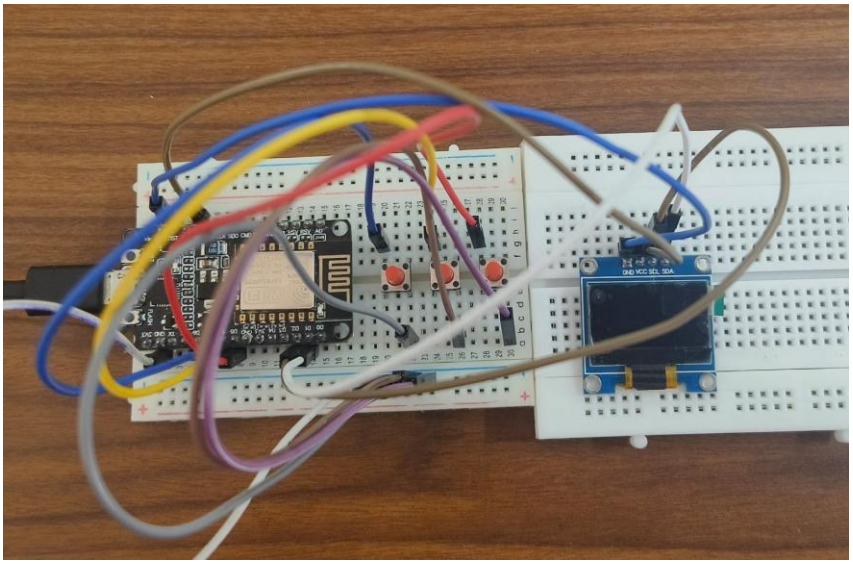
Wireshark (for Validation)

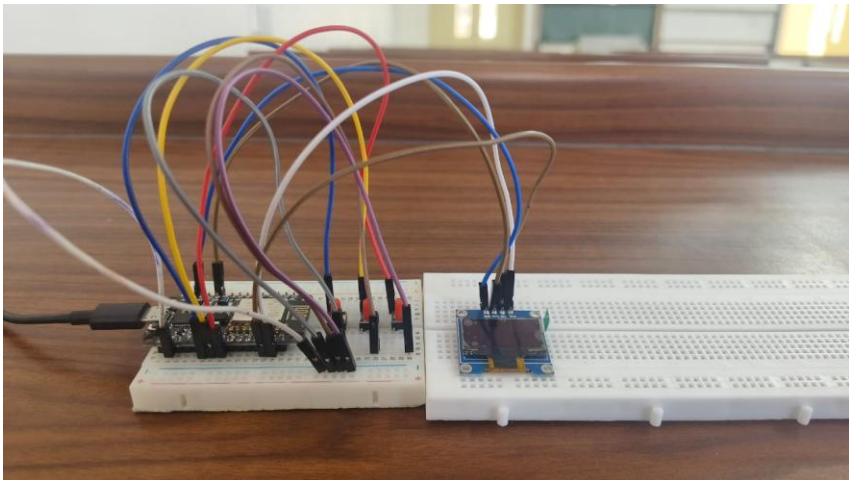
- Employed to capture and analyze Wi-Fi traffic in real time
- Useful filters include:
    - wlan.fc.type_subtype == 12 (deauth frames)
    - wlan.fc.type_subtype == 8 (beacon frames)
    - wlan.fc.type_subtype == 4 (probe requests)

Python (for Visualization)

- Python is used to analyze .pcap files from Wireshark, extracting beacon frame data such as SSIDs, MAC addresses, and channel information using libraries like pyshark or scapy.
- The extracted data is visualized using tools like matplotlib.

iv. Hardware Implementation





The OLED uses I2C communication: SDA is connected to D1 (GPIO5) and SCL to D2 (GPIO4), with power lines connected to 3.3V and GND. The push buttons are connected to digital pins: OK to D7 (GPIO13), UP to D5 (GPIO14), and DOWN to D6 (GPIO12), with one side of each button tied to GND. This setup enables user interaction via buttons and visual output on the OLED screen.

4. Analysis and Discussion

Serial Monitor Output Analysis of ESP8266 Wi-Fi Deauther in Arduino IDE

1. Initialization and System Configuration

The ESP8266 begins by initializing its Access Point (AP) and filesystem (SPIFFS). Key system parameters are displayed:

- RAM Usage: Approximately 60% of the total 81,920 bytes is utilized.
- Wi-Fi Channel: The default AP channel is set to 1.
- MAC Addresses:
    - AP MAC: Unique identifier for the ESP8266's broadcast network.
    - Station MAC: Identifier when the ESP8266 acts as a client.
- SPIFFS File System: A total of 65,536 bytes is available, with configuration and scan data stored in files such as scan.json, settings.json, ssids.json, and names.json.

2. Web Interface and Access Point Configuration

The device launches a Wi-Fi AP with the following parameters:

- SSID: pwned
- Password: deauther
- Mode: Access Point (AP)
- Channel: 1
- Web Path: /web (for control interface)

This allows users to connect to the ESP8266 and access the deauther's control interface through a web browser.

3. Scanning for Nearby Wi-Fi Access Points

A network scan is executed for 5 seconds. The results display all nearby Wi-Fi Access Points (APs), including:

- SSID: Network name
- Channel (Ch): The frequency channel used by the AP
- RSSI: Received Signal Strength Indicator (signal quality)
- Encryption Type: e.g., WPA2
- MAC Address: BSSID of the AP
- Vendor: Determined by MAC address prefix
- Selection Status: Indicates if the AP is selected for targeting

A total of 13 networks were discovered during this scan, with varying signal strengths and encryption schemes.

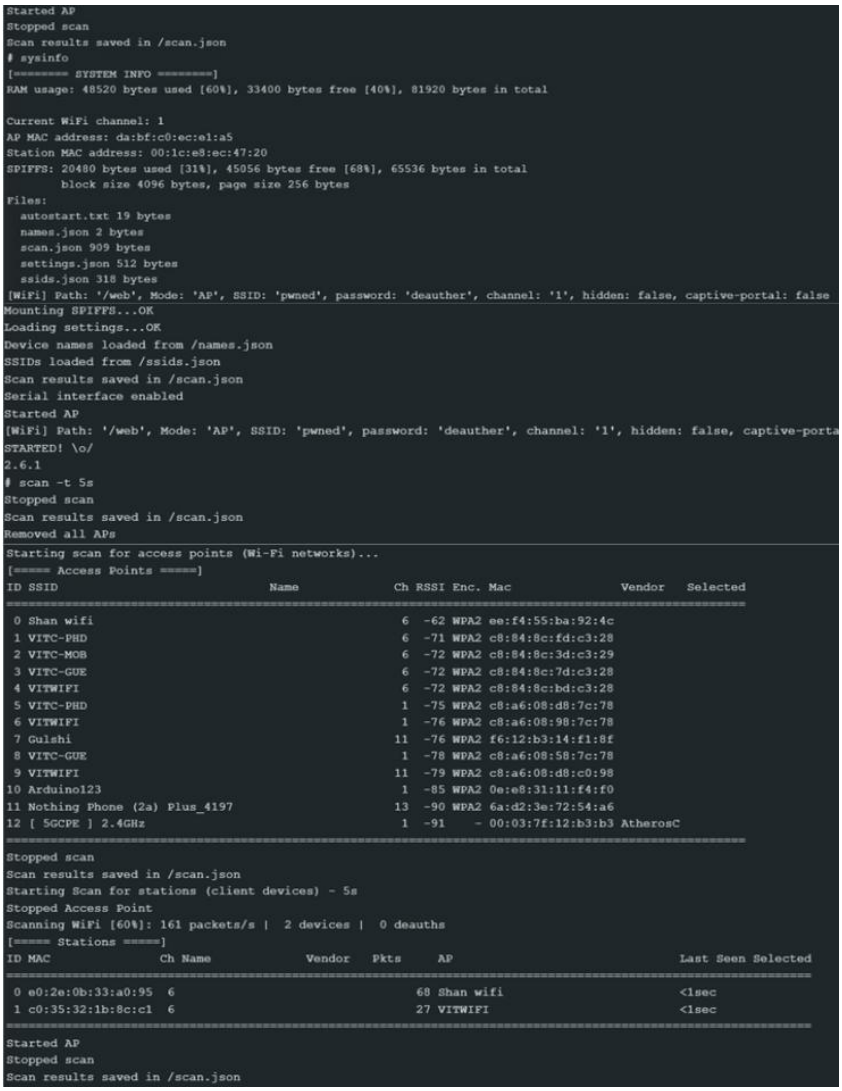
4. Scanning for Client Devices (Stations)

Following the AP scan, the device scans for client stations associated with the detected networks:

- MAC Address: Unique identifier for each station (device)
- Associated AP: The network to which the station is currently connected
- Packets Captured: Number of packets observed during the scan window
- Last Seen: Time since the device was last detected
- Selection Status: Marks the station for potential targeting

Two client devices were identified—each associated with different networks ("Shan wifi" and "VITWIFI")—with active traffic captured during the scan duration.

These functionalities demonstrate the ESP8266's capabilities in Wi-Fi auditing, including identifying access points, mapping network topology, and preparing for deauthentication or beacon spoofing attacks—commonly used in penetration testing and educational demonstrations.



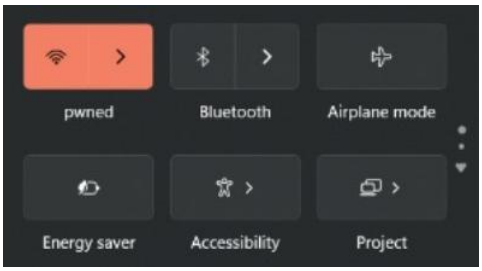
```
Started AP
Stopped scan
Scan results saved in /scan.json
# sysinfo
[======== SYSTEM INFO ========]
RAM usage: 48520 bytes used [60%], 33400 bytes free [40%], 81920 bytes in total

Current WiFi channel: 1
AP MAC address: da:bf:c0:ec:e1:a5
Station MAC address: 00:1c:e8:ec:47:20
SPIFFS: 20480 bytes used [31%], 45056 bytes free [68%], 65536 bytes in total
       block size 4096 bytes, page size 256 bytes
Files:
  autostart.txt 19 bytes
  names.json 2 bytes
  scan.json 909 bytes
  settings.json 512 bytes
  ssids.json 318 bytes
[WiFi] Path: '/web', Mode: 'AP', SSID: 'pwned', password: 'deauther', channel: '1', hidden: false, captive-portal: false
Mounting SPIFFS...OK
Loading settings...OK
Device names loaded from /names.json
SSIDs loaded from /ssids.json
Scan results saved in /scan.json
Serial interface enabled
Started AP
[WiFi] Path: '/web', Mode: 'AP', SSID: 'pwned', password: 'deauther', channel: '1', hidden: false, captive-porta
STARTED! \o/
2.6.1
# scan -t 5s
Stopped scan
Scan results saved in /scan.json
Removed all APs
Starting scan for access points (Wi-Fi networks)...
[===== Access Points =====]
ID SSID                                Name             Ch RSSI Enc. Mac               Vendor   Selected
=============================================================================================
 0 Shan wifi                                             6  -62 WPA2 ee:f4:55:ba:92:4c
 1 VITC-PHD                                              6  -71 WPA2 c8:84:8c:fd:c3:28
 2 VITC-MOB                                              6  -72 WPA2 c8:84:8c:3d:c3:29
 3 VITC-GUE                                              6  -72 WPA2 c8:84:8c:7d:c3:28
 4 VITWIFI                                               6  -72 WPA2 c8:84:8c:bd:c3:28
 5 VITC-PHD                                              1  -75 WPA2 c8:a6:08:d8:7c:78
 6 VITWIFI                                               1  -76 WPA2 c8:a6:08:98:7c:78
 7 Gulshi                                               11  -76 WPA2 f6:12:b3:14:f1:8f
 8 VITC-GUE                                              1  -78 WPA2 c8:a6:08:58:7c:78
 9 VITWIFI                                              11  -79 WPA2 c8:a6:08:d8:c0:98
10 Arduino123                                           1  -85 WPA2 0e:e8:31:11:f4:f0
11 Nothing Phone (2a) Plus_4197                        13  -90 WPA2 6a:d2:3e:72:54:a6
12 [ 5GCPE ] 2.4GHz                                     1  -91    - 00:03:7f:12:b3:b3 AtherosC
=============================================================================================
Stopped scan
Scan results saved in /scan.json
Starting Scan for stations (client devices) - 5s
Stopped Access Point
Scanning WiFi [60%]: 161 packets/s |  2 devices |  0 deauths
[===== Stations =====]
ID MAC               Ch Name             Vendor   Pkts     AP                                  Last Seen Selected
=============================================================================================
 0 e0:2e:0b:33:a0:95  6                              68 Shan wifi                           <1sec
 1 c0:35:32:1b:8c:c1  6                              27 VITWIFI                             <1sec
=============================================================================================
Started AP
Stopped scan
Scan results saved in /scan.json
```

Web Interface

To access the web interface, your Deauther must be running, and you must be connected to its WiFi network **pwnd** using the password **deauther**.

Then open your browser and visit 192.168.4.1. Make sure you're not connected to a VPN or anything else that could get in the way. You must temporarily disable the mobile connection on some phones to make it work.



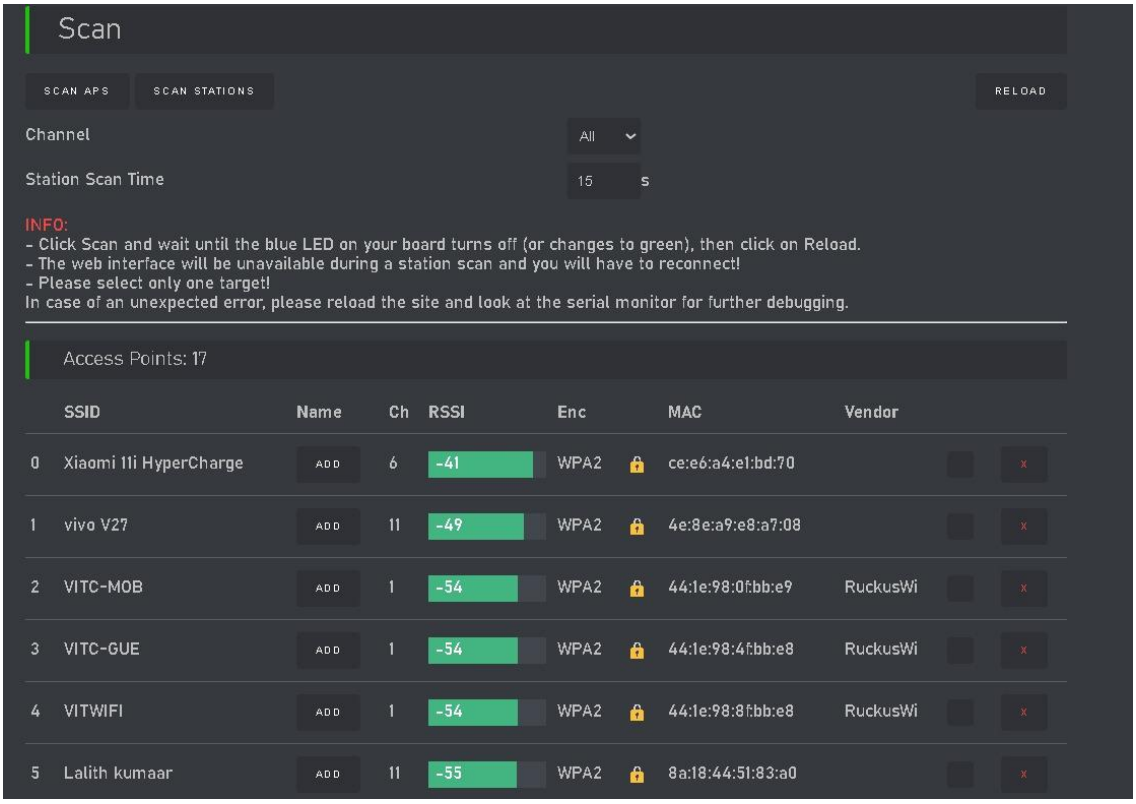
Scan Page

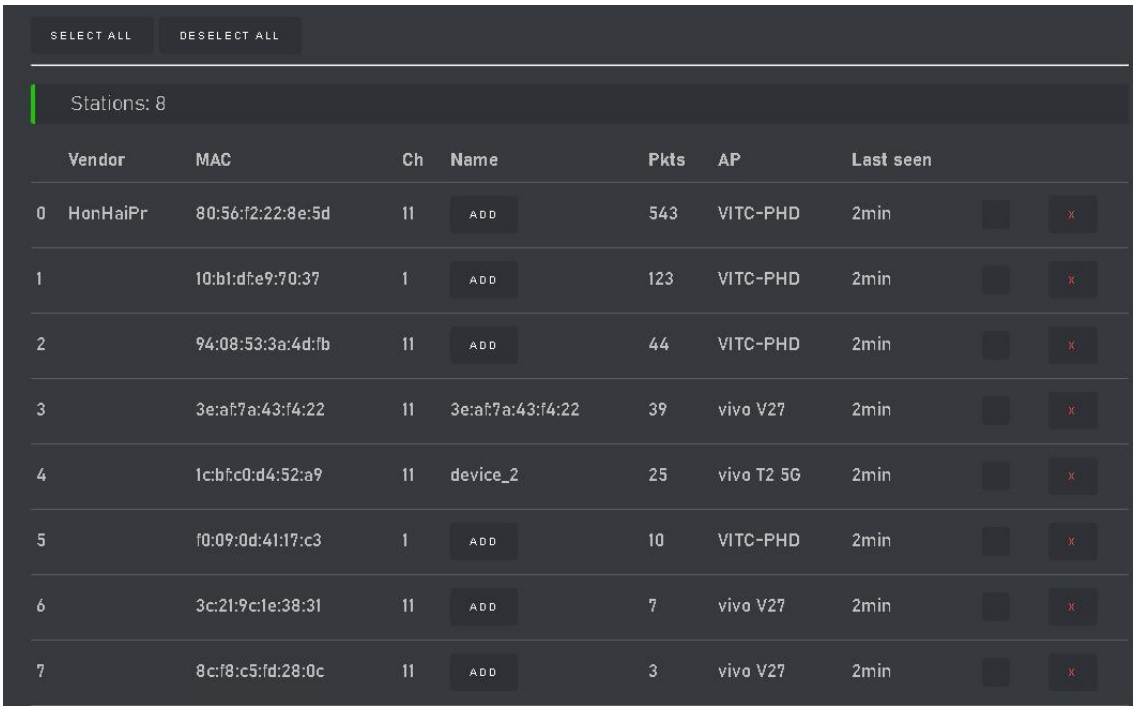
In the scan interface, you can enumerate nearby Wi-Fi access points and client stations. If the access point list is empty, click “Scan APs” to initiate a new search. The scan typically completes within 2–5 seconds; on some boards, an on-board LED will illuminate during the scan and extinguish when it finishes. At that point, click “Reload” to display the updated results.

Once the access point list appears, select the network(s) you wish to test—ideally only your own, as targeting others’ networks is strictly prohibited. Although the interface allows multiple selections, we recommend choosing a single target to ensure optimal stability and performance.

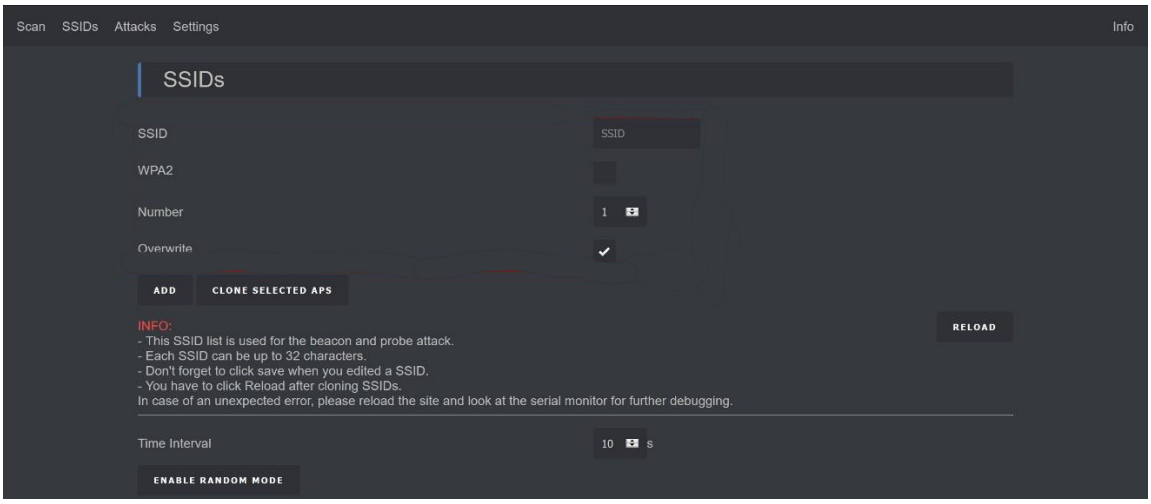
To target a specific client rather than the entire network, use “Scan Stations” to enumerate associated devices. Please note that while a station scan is in progress, the web interface will be temporarily unavailable; wait for the scan to complete before refreshing or reconnecting.

The scan results table presents several key columns to help you choose the optimal target: SSID displays the network name, while Channel indicates the 2.4 GHz frequency in use (1–13); selecting an AP on a less crowded channel can improve reliability. RSSI shows signal strength in dBm (e.g. –41 dBm) with a colored bar—values nearer to 0 dBm (green) denote a strong, stable link, whereas readings below –70 dBm (red) may lead to packet loss. Enc. identifies the encryption protocol (e.g. WPA2), and MAC lists the BSSID, ensuring you target the correct hardware even if multiple APs share an SSID. The Vendor column derives the manufacturer from the MAC’s OUI, offering hints about device capabilities, and the checkboxes under Selection Controls let you mark (or unmark) exactly which AP(s) to include in your attack, though we recommend selecting only one for best performance.

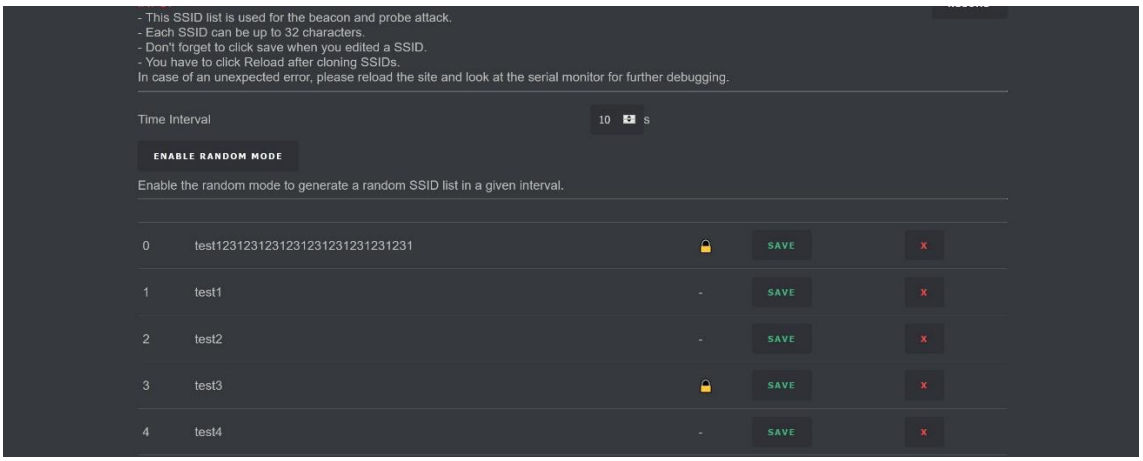

Scan
SCAN APS
SCAN STATIONS
RELOAD
Channel
All
Station Scan Time
15
s
INFO:
- Click Scan and wait until the blue LED on your board turns off (or changes to green), then click on Reload.
- The web interface will be unavailable during a station scan and you will have to reconnect!
- Please select only one target!
In case of an unexpected error, please reload the site and look at the serial monitor for further debugging.
Access Points: 17
SSID Name Ch RSSI Enc MAC Vendor
0 Xiaomi 11i HyperCharge ADD 6 -41 WPA2 ce:e6:a4:e1:bd:70
1 vivo V27 ADD 11 -49 WPA2 4e:8e:a9:e8:a7:08
2 VITC-MOB ADD 1 -54 WPA2 44:1e:98:0f:bb:e9 RuckusWi
3 VITC-GUE ADD 1 -54 WPA2 44:1e:98:4f:bb:e8 RuckusWi
4 VITWIFI ADD 1 -54 WPA2 44:1e:98:8f:bb:e8 RuckusWi
5 Lalith kumaar ADD 11 -55 WPA2 8a:18:44:51:83:a0


SELECT ALL
DESELECT ALL
Stations: 8
Vendor MAC Ch Name Pkts AP Last seen
0 HonHaiPr 80:56:f2:22:8e:5d 11 ADD 543 VITC-PHD 2min
1 10:b1:df:e9:70:37 1 ADD 123 VITC-PHD 2min
2 94:08:53:3a:4d:fb 11 ADD 44 VITC-PHD 2min
3 3e:af:7a:43:f4:22 11 3e:af:7a:43:f4:22 39 vivo V27 2min
4 1c:bf:c0:d4:52:a9 11 device_2 25 vivo T2 5G 2min
5 f0:09:0d:41:17:c3 1 ADD 10 VITC-PHD 2min
6 3c:21:9c:1e:38:31 11 ADD 7 vivo V27 2min
7 8c:f8:c5:fd:28:0c 11 ADD 3 vivo V27 2min

SSID Page



This is where you can add, edit and remove SSIDs. An SSID (Service Set Identifier) is the name of a WiFi network. They are used in beacon and probe attacks.



Attack Page

The Attack page of the Deauther web UI allows you to inject three types of 802.11 management frames—Deauthentication, Beacon, and Probe—against selected targets. Each row in the attack table displays:

- Attack Type: the frame-injection mode (Deauth, Beacon, or Probe)
- Targets: number of selected access points or stations
- Pkts/s: current packet-per-second rate (sent/acknowledged)
- START / STOP button: to begin or halt the attack

- Deauthentication (Deauth)

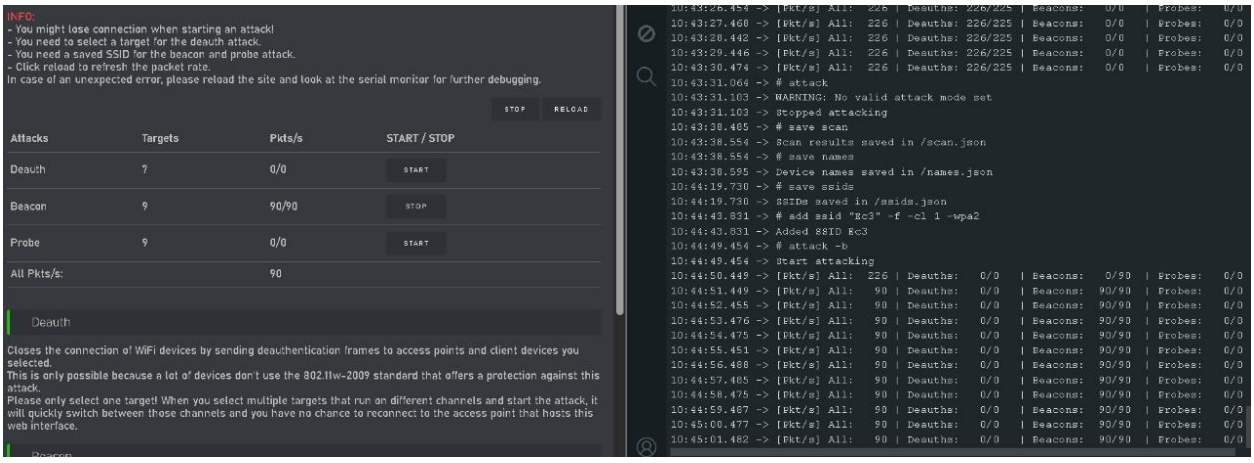
Sends forged deauthentication frames to your chosen AP(s) or client station(s), compelling devices to disconnect. This is only effective on networks without Management Frame Protection (802.11w). Selecting a single target is recommended to avoid rapid channel switching and maximize stability.

- Beacon Flooding (Beacon)

Broadcasts fake or cloned beacon frames for each SSID in your saved list. By overwhelming the air with spurious AP announcements, clients may become confused or inadvertently connect to a rogue network. The UI displays the number of beacons sent versus the total SSIDs selected.

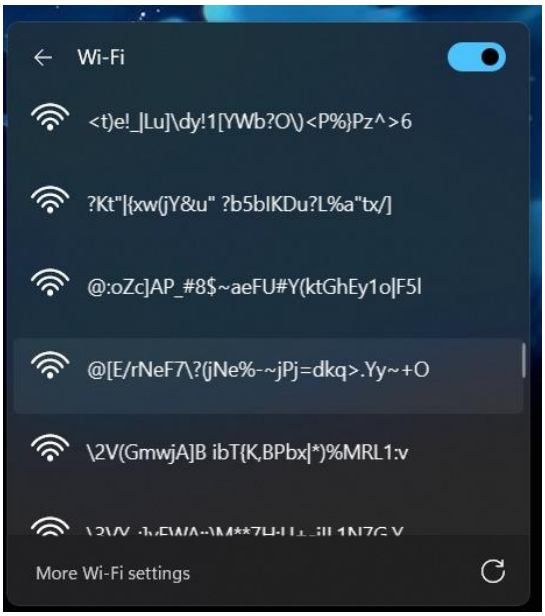
- Probe Request (Probe)

Injects active probe requests to solicit responses from hidden or passive SSIDs. This can reveal concealed network names and provoke client devices to reveal their preferred SSIDs.



- Demonstration of Beacon Flooding

When the beacon-flood mode is active, the ESP8266 continuously transmits forged 802.11 Beacon frames advertising dozens of non-existent networks. The attached screenshot shows the result in your Wi-Fi scanner or client device's AP list: a long column of "ghost" SSIDs—each with its own BSSID, channel, and (in this case) default WPA2 lock icon—flooding every available channel.



- Demonstration of Deauthentication Attack

In our trial, Device A was associated with the network through Device B acting as the access point. Upon initiating the deauthentication attack and directing forged deauth frames at Device B, we observed that Device A's Wi-Fi link was repeatedly severed. Each burst of deauth packets caused Device A to lose association, confirming the effectiveness of the ESP8266 Deauther in forcing client disconnections.

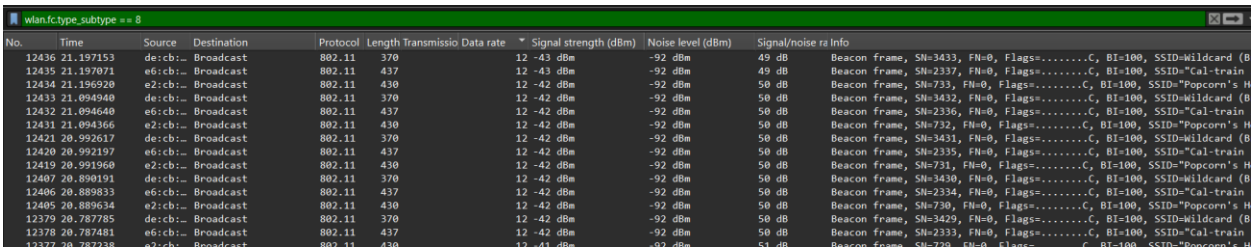
- Demonstration of Probe Attack

In the probe-request test, we configured the ESP8266 to flood the network with active probe requests targeting our chosen SSID. As soon as the attack began, the access point (Device B) replied to each probe with a probe-response frame containing its SSID and other network information. Meanwhile, any client devices (Device A) that had previously connected began issuing their own probe requests—revealing their preferred networks. This two-way exchange confirmed that by injecting crafted probe frames, an attacker can not only discover hidden or passive SSIDs but also map which networks clients are actively seeking. The ESP8266's serial output logged both sent probe requests and received probe responses, demonstrating the effectiveness of the probe attack in uncovering network topology and client behavior.

Wireshark Capture

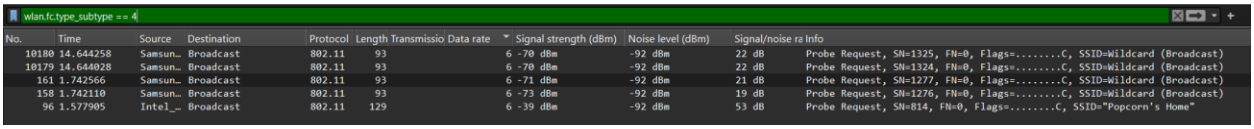
- Beacon Frames

The Wireshark capture, filtered with `wlan.fc.type_subtype == 8`, shows a continuous stream of beacon frames sent to the broadcast address from varying spoofed MAC addresses. The consistent signal strength (~–42 dBm) and high signal-to-noise ratio (~50 dB) indicate strong transmission from a nearby device. Incrementing sequence numbers and varying frame lengths (370–437 bytes) confirm active beacon flooding, with the ESP8266 generating fake SSIDs through crafted 802.11 management frames.



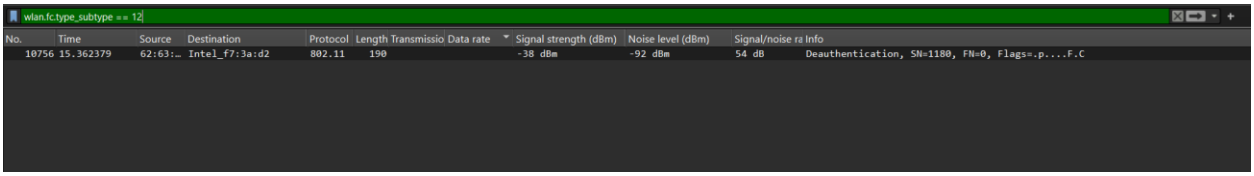
- Probe Request Frames

The capture filtered using `wlan.fc.type_subtype == 4` reveals a high frequency of probe request frames broadcast from randomized spoofed MAC addresses. These frames are typically sent to the broadcast address, querying for SSIDs either explicitly or as wildcard requests. The consistent signal strength and signal-to-noise ratio indicate that these frames originate from the nearby ESP8266. The volume and diversity of the requests confirm that the probe flooding function is working, effectively simulating multiple devices actively searching for networks.



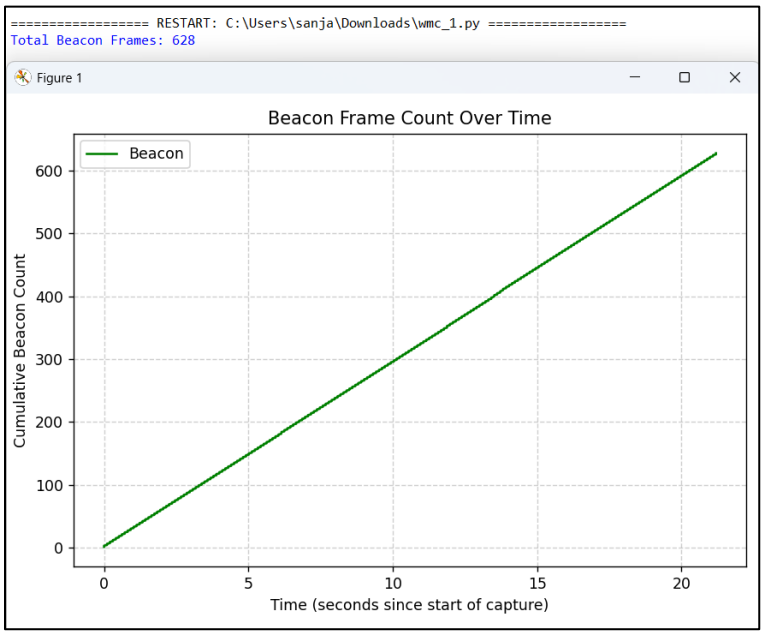
- Deauthentication Frames

Using the filter `wlan.fc.type_subtype == 12`, Wireshark displays numerous deauthentication frames sent from spoofed AP MAC addresses to specific client MAC addresses or broadcast. These frames include standard reason codes (e.g., “Class 3 frame received from nonassociated STA”), which are used to forcibly disconnect devices from access points. The rapid and repeated appearance of these frames, along with stable signal strength values, confirms that the ESP8266 is performing continuous deauthentication attacks to disrupt nearby Wi-Fi connections.



Beacon Frame Analysis using Python and Scapy

A custom Python script was developed to analyze beacon frames from a captured .pcapng file using the scapy library. The script reads the packet capture and filters for IEEE 802.11 management frames with subtype 8, which correspond to beacon frames. The timestamp of each beacon frame is recorded relative to the start of the capture session.
A helper function then calculates the cumulative count of beacon frames over time. This is achieved by incrementally counting each beacon frame and mapping it against its timestamp, resulting in two arrays: time (x-axis) and frame count (y-axis).
The output is a line plot generated using matplotlib, displaying the cumulative number of beacon frames as a function of time. The graph typically appears as a straight, upward-sloping line, indicating a consistent and high-frequency transmission of beacon frames over the capture period. This linear trend confirms the effectiveness of the beacon flooding attack, where spoofed access points are being broadcast repeatedly at regular intervals.



---

5. Challenges and Future Scope

- Current Challenges
  - Difficulty in distinguishing legitimate deauthentication requests from attacks.
  - Hardware limitations of low-cost IDS implementations.

- Possible Improvements and Future Research
  - Implementing machine learning algorithms for more accurate attack detection.
  - Exploring WPA3 security enhancements.

---

6. Conclusion

- Summary of Key Findings

  This project successfully demonstrated the vulnerabilities of Wi-Fi networks through the implementation of beacon flooding, probe request spoofing, and deauthentication attacks using the ESP8266 microcontroller. The system was able to generate and broadcast fake management frames, disrupting normal wireless communication. Wireshark analysis confirmed the transmission of spoofed frames, with a high frequency and consistent signal levels, indicating effective operation. Python-based visualization of beacon frames further supported the findings by showing a steady and linear increase in frame count over time, aligning with expected behavior during a flooding attack.

- How the Objectives Were Met

  - Configured the ESP8266 with the Deauther firmware and accessed it via a web interface (192.168.4.1).
  - Used scanning and target selection to launch Beacon and Deauth attacks on specific networks and devices.
  - Captured network traffic using Wireshark in monitor mode and filtered relevant 802.11 frames.
  - Used Python scripts to visualize frame counts and timestamps.
  - Validated the presence and effect of each type of frame, fulfilling the detection and analysis objective.

- Challenges faced in Implementation and how did you overcome it.

  Challenge: Could not connect the OLED display and push buttons to the ESP8266 due to hardware compatibility and wiring issues.
  Solution: Decided to proceed with the web interface for interaction and visualization. The OLED and buttons were excluded from the final version to avoid delays and complexity in implementation.

  Challenge: Limited debugging capability on ESP8266 for testing real-time frame generation.
  Solution: Used serial monitor outputs and Wireshark side-by-side to correlate expected behavior with actual network traffic.

  Challenge: The project functioned correctly on the ESP8266 v1.0 (NodeMCU mini), but encountered consistent failures on the ESP8266 v3 (full-sized NodeMCU).
  Solution: Despite testing, the v3 board consistently failed to support stable operation with the Deauther firmware. The final implementation used the v1.0 board, which performed reliably.

7. References *(Use IEEE format)*

[1] "Deauther Project Official Website," [Online]. Available: https://deauther.com/

[2] Y. Kristiyanto and Ernastuti, “Analysis of Deauthentication Attack on IEEE 802.11 Connectivity Based on IoT Technology Using External Penetration Test,” *CommIT (Communication & Information Technology) Journal*, vol. 14, no. 1, pp. 45–51, 2020. doi: [Insert DOI if available].

[3] R. V. Reddy, P. Patel, Y. Sanjana, A. J. Reddy, and M. Tejashwini, “Design and Implementation of Attack Flow Model Using ESP8266: Wireless Networks,” *International Journal of Engineering Research & Technology (IJERT)*, vol. 13, no. 03, Mar. 2024. ISSN: 2278-0181.

[4] L. Saranya, R. V. Reddy, A. B. Reddy, B. S. Dinesh, and M. Muneeruddin, “Detect Wi-Fi De-Authentication Attacks Using ESP8266,” *International Journal of Engineering Research & Technology (IJERT)*, vol. 13, no. 03, Mar. 2024. ISSN: 2278-0181.

---

8. Appendices

A.1 Complete Python Code for Beacon Packets Detection and Plotting:

```
from scapy.all import *

import matplotlib.pyplot as plt


pcap_file = "C:/Users/sanja/Downloads/Monitormode_packet_capture.pcapng"

packets = rdpcap(pcap_file)

beacon_times = []

start_time = packets[0].time if packets else 0

for pkt in packets:

    if pkt.haslayer(Dot11):

        if pkt.type == 0 and pkt.subtype == 8:  # Beacon frame

            t = pkt.time - start_time

            beacon_times.append(t)
```

```
def cumulative_count(times):
    times.sort()
    x = []
    y = []
    count = 0
    for t in times:
        count += 1
        x.append(t)
        y.append(count)
    return x, y


bx, by = cumulative_count(beacon_times)
print(f"Total Beacon Frames: {len(beacon_times)}")
plt.figure(figsize=(8, 5))
plt.plot(bx, by, label="Beacon", color="green", marker='o' if len(bx) <= 10 else
None)
plt.title("Beacon Frame Count Over Time")
plt.xlabel("Time (seconds since start of capture)")
plt.ylabel("Cumulative Beacon Count")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()
```

A.2 Arduino/ESP8266 Code:

```
extern "C" {
  #include "user_interface.h"
}

#include "EEPROMHelper.h"
#include "src/ArduinoJson-v5.13.5/ArduinoJson.h"
#if ARDUINOJSON_VERSION_MAJOR != 5
#error Please upgrade/downgrade ArduinoJSON library to version 5!
#endif // if ARDUINOJSON_VERSION_MAJOR != 5

#include "oui.h"
#include "language.h"
#include "functions.h"
#include "settings.h"
#include "Names.h"
#include "SSIDs.h"
#include "Scan.h"
#include "Attack.h"
#include "CLI.h"
#include "DisplayUI.h"
#include "A_config.h"

#include "led.h"
```

```
// Run-Time Variables //

Names names;

SSIDs ssids;

Accesspoints accesspoints;

Stations     stations;

Scan   scan;

Attack attack;

CLI    cli;

DisplayUI displayUI;


simplebutton::Button* resetButton;

#include "wifi.h"

uint32_t autosaveTime = 0;

uint32_t currentTime  = 0;

bool booted = false;


void setup() {

    randomSeed(os_random());

    Serial.begin(115200);

    Serial.println();

    prnt(SETUP_MOUNT_SPIFFS);

    // bool spiffsError = !LittleFS.begin();

    LittleFS.begin();

    prntln(/*spiffsError ? SETUP_ERROR : */ SETUP_OK);
```

```
    // Start EEPROM
    EEPROMHelper::begin(EEPROM_SIZE);

#ifdef FORMAT_SPIFFS
    prnt(SETUP_FORMAT_SPIFFS);
    LittleFS.format();
    prntln(SETUP_OK);
#endif // ifdef FORMAT_SPIFFS

#ifdef FORMAT_EEPROM
    prnt(SETUP_FORMAT_EEPROM);
    EEPROMHelper::format(EEPROM_SIZE);
    prntln(SETUP_OK);
#endif // ifdef FORMAT_EEPROM

    // Format SPIFFS when in boot-loop
    if (/*spiffsError || */ !EEPROMHelper::checkBootNum(BOOT_COUNTER_ADDR)) {
        prnt(SETUP_FORMAT_SPIFFS);
        LittleFS.format();
        prntln(SETUP_OK);
        prnt(SETUP_FORMAT_EEPROM);
        EEPROMHelper::format(EEPROM_SIZE);
        prntln(SETUP_OK);

        EEPROMHelper::resetBootNum(BOOT_COUNTER_ADDR);
```

```
}

// get time
currentTime = millis();

// load settings
#ifndef RESET_SETTINGS
settings::load();
#else // ifndef RESET_SETTINGS
settings::reset();
settings::save();
#endif // ifndef RESET_SETTINGS

wifi::begin();
wifi_set_promiscuous_rx_cb([](uint8_t* buf, uint16_t len) {
    scan.sniffer(buf, len);
});

// start display
if (settings::getDisplaySettings().enabled) {
    displayUI.setup();
    displayUI.mode = DISPLAY_MODE::INTRO;
}

// load everything else
```

```
    names.load();
    ssids.load();
    cli.load();

    // create scan.json
    scan.setup();
    // dis/enable serial command interface
    if (settings::getCLISettings().enabled) {
        cli.enable();
    } else {
        prntln(SETUP_SERIAL_WARNING);
        Serial.flush();
        Serial.end();
    }
    // start access point/web interface
    if (settings::getWebSettings().enabled) wifi::startAP();
    // STARTED
    prntln(SETUP_STARTED);
    // version
    prntln(DEAUTHER_VERSION);
    // setup LED
    led::setup();
    // setup reset button
    resetButton = new ButtonPullup(RESET_BUTTON);
}
```

```
void loop() {
    currentTime = millis();

    led::update();   // update LED color
    wifi::update();  // manage access point
    attack.update(); // run attacks
    displayUI.update();
    cli.update();    // read and run serial input
    scan.update();   // run scan
    ssids.update();  // run random mode, if enabled

    // auto-save
    if (settings::getAutosaveSettings().enabled
        && (currentTime - autosaveTime > settings::getAutosaveSettings().time)) {
        autosaveTime = currentTime;
        names.save(false);
        ssids.save(false);
        settings::save(false);
    }

    if (!booted) {
        booted = true;
        EEPROMHelper::resetBootNum(BOOT_COUNTER_ADDR);
#ifdef HIGHLIGHT_LED
        displayUI.setupLED();
```

```
#endif // ifdef HIGHLIGHT_LED
    }

    resetButton->update();
    if (resetButton->holding(5000)) {
        led::setMode(LED_MODE::SCAN);
        DISPLAY_MODE _mode = displayUI.mode;
        displayUI.mode = DISPLAY_MODE::RESETTING;
        displayUI.update(true);

        settings::reset();
        settings::save(true);

        delay(2000);

        led::setMode(LED_MODE::IDLE);
        displayUI.mode = _mode;
    }
}
```

A.3 Wireshark Capture File

Monitormode_packet_capture.pcapng