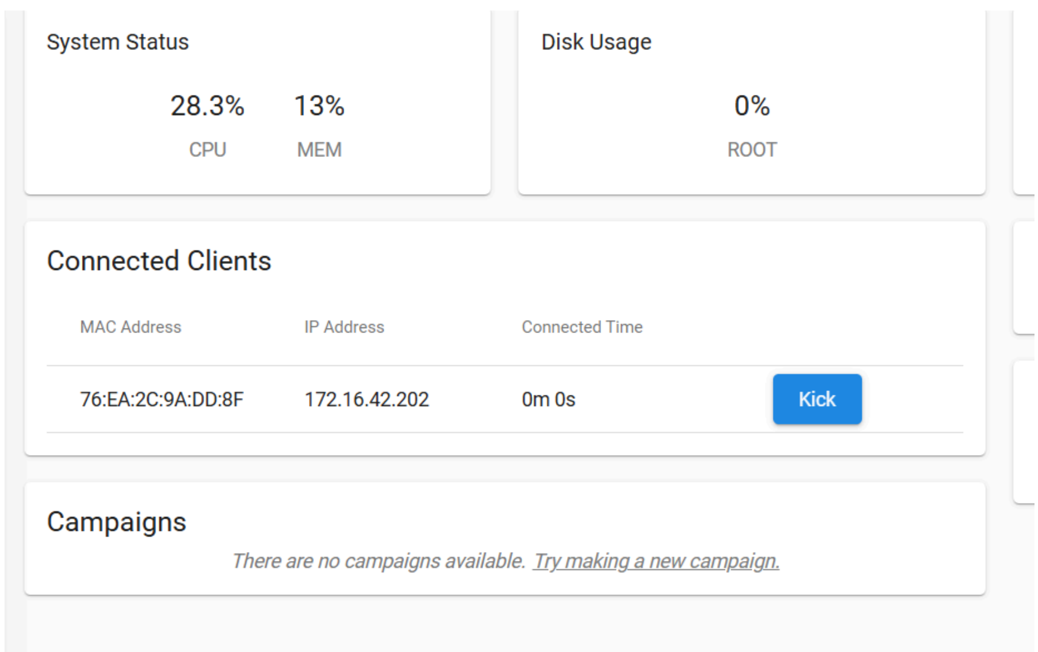
Layla Nassar, Thomas Clarke, Brady Chan
Section #: 003
April 2, 2025
LTN22001, TJC20007, BIC22003
10.13.6.145, 10.13.6.44, 10.13.6.29

Lab 5 Report – WiFi Attacks With Pineapples

Question #1:

Interface name: wlan2

Dashboard:



The interface name used in this lab was wlan2, which is the default wireless interface on the Pineapple device for connecting to other networks. After setting up the Pineapple Mark VII, we accessed the Dashboard via `http://172.16.42.1:1471`. The Dashboard provided a comprehensive overview of the device's current system status.

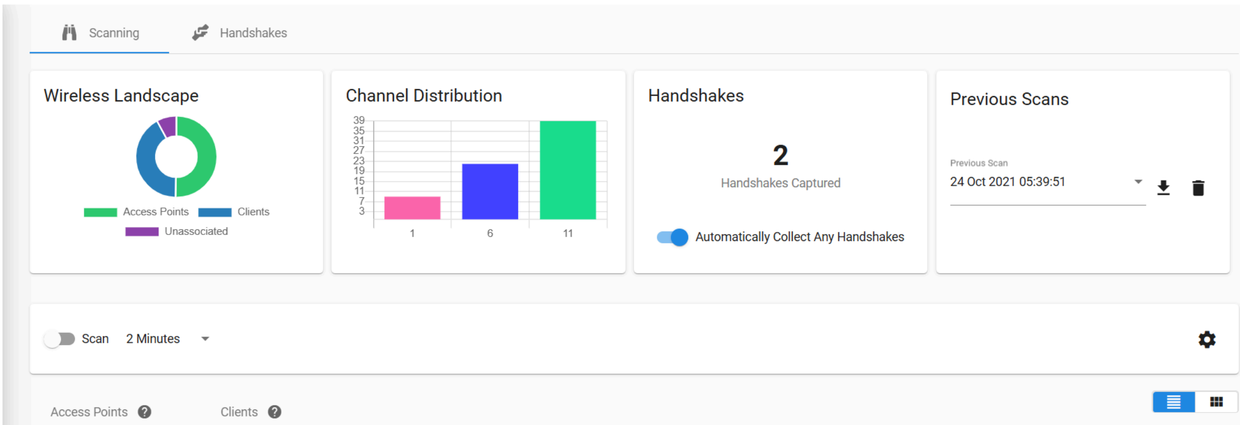
At the top, we could see system resource usage like CPU, RAM, and storage. Below that, the SSIDs Collected section indicated that one SSID had been detected at the time, and the Connected Clients section displayed that one client had successfully connected to the Pineapple's open network. These panels gave us a real-time snapshot of wireless activity and device interactions. The Notifications area alerted us to any updates or changes, and the Campaigns section was available to display the status of any running engagements.

Wi-Fi Console:

In the Wi-Fi Administration Console, we saw similar summary metrics: 1 SSID collected, 1 client connected, and 1 handshake captured. This validated that our PineAP services were functioning correctly. This console also provides access to PineAP options, such as controlling how the Pineapple impersonates access points or scans for devices. Having the `wlan2` interface active ensured that the Pineapple could communicate with nearby wireless environments while being controlled through the management network.

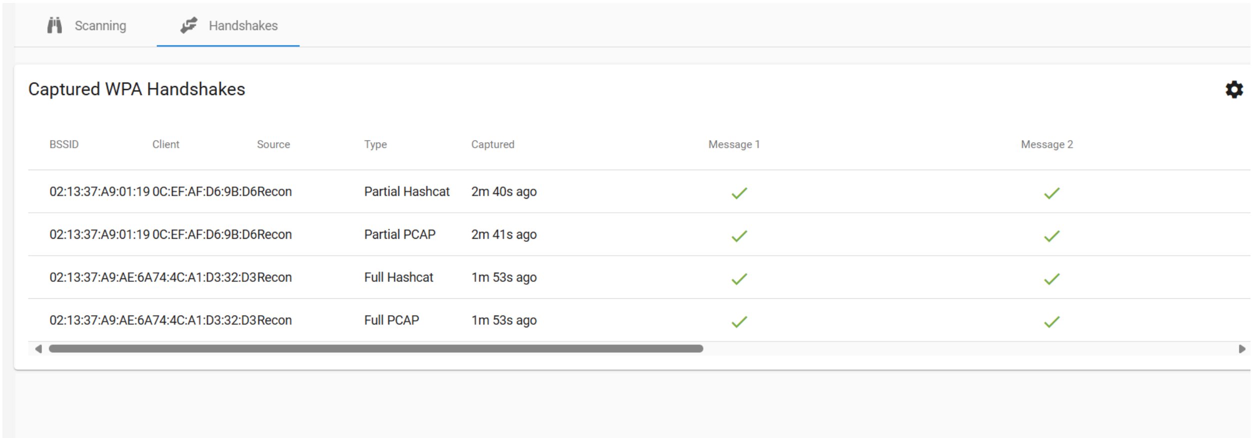
Question #2:

Scanning Tab



Within the **Recon tab**, we explored both the **Scanning** and **Handshakes** sub-tabs. The Scanning tab allowed us to observe nearby access points and any clients connected to them. The scan revealed several SSIDs, including `TP-Link_XXXX`, `CSE3140`, and other residential or enterprise networks. Each entry included its SSID, MAC address, the vendor (OUI), number of connected clients, and the type of security in use. For example, the `CSE3140` network showed **0 clients connected** at that moment.

Handshakes Tab:



BSSID	Client	Source	Type	Captured	Message 1	Message 2
02:13:37:A9:01:19	0C:EF:AF:D6:9B:D6	Recon	Partial Hashcat	2m 40s ago	✓	✓
02:13:37:A9:01:19	0C:EF:AF:D6:9B:D6	Recon	Partial PCAP	2m 41s ago	✓	✓
02:13:37:A9:AE:6A	74:4C:A1:D3:32:D3	Recon	Full Hashcat	1m 53s ago	✓	✓
02:13:37:A9:AE:6A	74:4C:A1:D3:32:D3	Recon	Full PCAP	1m 53s ago	✓	✓

The **Handshakes** tab initially showed just one handshake captured, but later this number increased to **eight handshakes**, confirming that the Pineapple was actively intercepting Wi-Fi traffic during recon scans. Each handshake record detailed the SSID, the client MAC address, access point MAC address, timestamp, and whether the handshake was valid. These handshakes are useful for offline password cracking, which highlights the potential severity of this type of surveillance.

Question #3:

Access Points | Clients

Search: him

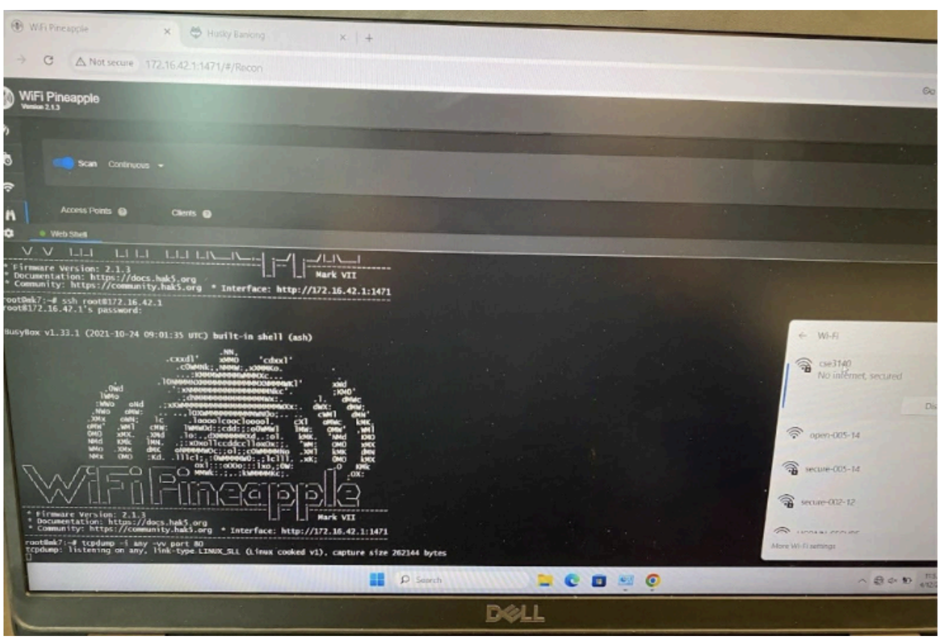
Items per page: 10 | 1 – 1 of 1

	SSID	MAC	OUI	Clients	Security	MFP	WPS	Channel	Signal	First Seen	Last Seen
−	Himanshu's iPhone	2E:7D:8C:09:AC:B2	Unknown Vendor	1	WPA2 (PSK)	No	No	6		3m 29s ago	0m 19s ago
	Client	DA:AA:3C:71:7A:9B	Unknown Vendor							3m 57s ago	0m 13s ago

To test mobile hotspot detection, we set up a personal hotspot and renamed it according to lab specifications. The hotspot successfully appeared in the scanning results, listed by its SSID and MAC address. The Pineapple also displayed **1 client connected** to the hotspot, although we later accumulated more (around 8), and by expanding the entry, we were able to view the MAC address of the client device. This confirmed that the Pineapple could monitor clients associated with nearby wireless networks.

Question #4:

Screenshots:



The security column listed the hotspot as using WPA2(PSK), a common personal-level encryption protocol. Although no new handshake was captured during the initial scan, it is likely that the client hadn't disconnected and reconnected during the capture window. This demonstrates the importance of timing when attempting to capture handshake packets. Continuous scanning could improve the chance of capturing handshakes in real time.

IPs sending HTTP traffic:

IPs sending HTTP traffic:

- **10.13.6.145**

This IP corresponds to one of the client laptops used during testing, accessing the lab's sample HTTP site (`10.13.4.80`). Since the network was open and unencrypted, all HTTP headers, browser info, and requested URLs were fully visible. This shows how easy it is for a third party to eavesdrop on unsecured networks.

Question #5:

We repeated the tcpdump command after switching to the secure wireless network we had configured. Interestingly, despite being on a secure WPA2 network, the screenshot of the terminal showed that some HTTP traffic was still visible, including URLs, browser headers, and even image filenames (e.g., .jpg, .gif).

What traffic do you see now?

Some readable HTTP traffic, including GET requests and user-agent information, was still visible in plaintext.

```
--------------------------------------------------------------------------------
root@mk7:~# tcpdump -i interface-name -vv port
tcpdump: interface-name: No such device exists
(SIOCGIFHWADDR: No such device)
root@mk7:~# tcpdump -i wlan2 -vv port 80
tcpdump: listening on wlan2, link-type EN10MB (Ethernet), capture size 262144 bytes
05:22:39.397701 IP (tos 0x0, ttl 127, id 52225, offset 0, flags [DF], proto TCP (6), length 52)
    10.13.8.74.49195 > 10.13.4.80.80: Flags [S], cksum 0x111a (correct), seq 3062941238, win 65535, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
05:22:39.402334 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    10.13.4.80.80 > 10.13.8.74.49195: Flags [S.], cksum 0x018d (correct), seq 2246807200, ack 3062941239, win 64240, options [mss 1460,nop,nop,sackOK,nop,ws
cale 7], length 0
05:22:39.403378 IP (tos 0x0, ttl 127, id 52226, offset 0, flags [DF], proto TCP (6), length 52)
    10.13.8.74.49196 > 10.13.4.80.80: Flags [S], cksum 0x111e (correct), seq 819743718, win 65535, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
05:22:39.404672 IP (tos 0x0, ttl 127, id 52227, offset 0, flags [DF], proto TCP (6), length 40)
    10.13.8.74.49195 > 10.13.4.80.80: Flags [.], cksum 0x3c51 (correct), seq 1, ack 1, win 255, length 0
05:22:39.405416 IP (tos 0x0, ttl 127, id 52228, offset 0, flags [DF], proto TCP (6), length 465)
    10.13.8.74.49195 > 10.13.4.80.80: Flags [P.], cksum 0x3907 (correct), seq 1:426, ack 1, win 255, length 425: HTTP, length: 425
        GET / HTTP/1.1
        Host: 10.13.4.80
        Connection: keep-alive
        Upgrade-Insecure-Requests: 1
        User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
        Accept-Encoding: gzip, deflate
        Accept-Language: en-US,en;q=0.9

05:22:39.405657 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    10.13.4.80.80 > 10.13.8.74.49196: Flags [S.], cksum 0x5414 (correct), seq 2124301162, ack 819743719, win 64240, options [mss 1460,nop,nop,sackOK,nop,wsc
ale 7], length 0
05:22:39.406665 IP (tos 0x0, ttl 64, id 7818, offset 0, flags [DF], proto TCP (6), length 40)
    10.13.4.80.80 > 10.13.8.74.49195: Flags [.], cksum 0x39b4 (correct), seq 1, ack 426, win 499, length 0
05:22:39.412150 IP (tos 0x0, ttl 64, id 7819, offset 0, flags [DF], proto TCP (6), length 905)
    10.13.4.80.80 > 10.13.8.74.49195: Flags [P.], cksum 0x0e0b (correct), seq 1:866, ack 426, win 501, length 865: HTTP, length: 865
        HTTP/1.1 200 OK
        Date: Wed, 02 Apr 2025 16:41:07 GMT
        Server: Apache/2.4.41 (Ubuntu)
        Vary: Accept-Encoding
        Content-Encoding: gzip
        Keep-Alive: timeout=5
        Connection: Keep-Alive
        Transfer-Encoding: chunked
```

How does it differ from the traffic you saw in the previous question?

While both networks allowed us to see HTTP headers, the expectation was that the WPA2-secured network would block or encrypt traffic. However, **HTTP itself is unencrypted**, so the browser's communication with the web server over port 80 remained visible even when Wi-Fi was secured. The difference lies in **Wi-Fi-level encryption (WPA2)**, which protects the transmission medium, not the content of **unencrypted protocols** like HTTP. For full confidentiality, HTTPS should be used.

Question #6:

We scanned again while connected to our management SSID. The scan picked up a wide array of SSIDs, including:

- UCONN-SECURE
- UCONN-GUEST
- TP-Link_XXXX
- CSE3140
- Other residential and enterprise networks.

Each SSID had a unique or near-duplicate MAC address. We observed that UCONN-SECURE and UCONN-GUEST appeared multiple times with different MAC values, implying they were broadcast from the same access points, which were using multi-SSID capabilities.

The Security column showed:

- Open: No encryption at all.
- WPA2(PSK): Requires a pre-shared key, common for home networks like CSE3140.
- WPA2(802.1X Enterprise): Used by enterprise networks like UConn's, requiring individual authentication.

These security types demonstrate the wide range of protection strategies used across environments and how susceptible Open networks are to attacks like those simulated in this lab.

SSID	MAC	OUI	Clients	Security	MFP	WPS	Channel	Signal	First Seen	Last Seen
UCONN-GUEST	58:8B:1C:6C:9D:E2	Unknown Vendor	0	Open	No	No	6		0m 17s ago	0m 17s ago
UCONN-SECURE	58:8B:1C:6C:9D:E3	Unknown Vendor	0	WPA2 (802.1X Enterprise, 802.1X Enterprise FT)	Optional	No	6		0m 17s ago	0m 17s ago
UCONN-SECURE	C8:28:E5:BA:0E:E3	Unknown Vendor	0	WPA2 (802.1X Enterprise, 802.1X Enterprise FT)	Optional	No	1		0m 17s ago	0m 17s ago
UCONN-SECURE	C8:28:E5:BA:1B:E3	Unknown Vendor	0	WPA2 (802.1X Enterprise, 802.1X Enterprise FT)	Optional	No	6		0m 17s ago	0m 17s ago
open-008-09	00:13:37:A9:00:E3	Orient Power Home Network Ltd.	0	Open	No	No	11		0m 17s ago	0m 17s ago
secure-008-09	02:13:37:A9:00:E3	Unknown Vendor	0	WPA2 (PSK)	No	No	11		0m 17s ago	0m 17s ago
CSE3140	2C:33:11:D0:CD:E0	Cisco Systems Inc	0	WPA2 (PSK)	No	No	1		0m 29s ago	0m 18s ago
Survey	58:8B:1C:6C:9D:E0	Unknown Vendor	0	WPA3 (SAE)	Required	No	6		0m 18s ago	0m 18s ago
Survey	C8:28:E5:BA:0E:E0	Unknown Vendor	0	WPA3 (SAE)	Required	No	1		0m 18s	0m 18s ago

Question #7:

Link to video:
https://drive.google.com/file/d/1K7BRmG-kKP9ct_1X_-jTN_yX1pR7Kgzk/view?usp=sharing

9A:67:41:DD:F1:70	EVILTWIN.22000	Evil WPA/2 Twin	Eviltwin Hashcat	4h 36m ago	?	?	?	?	?
9C:FC:E8:CF:55:12	EVILTWIN.PCAP	Evil WPA/2 Twin	Eviltwin PCAP	4h 18m ago	?	?	?	?	?
F2:CE:0E:CD:DD:71	EVILTWIN.22000	Evil WPA/2 Twin	Eviltwin Hashcat	0m 7s ago	?	?	?	?	?
70:D8:23:7D:31:24	EVILTWIN.PCAP	Evil WPA/2 Twin	Eviltwin PCAP	2h 32m ago	?	?	?	?	?
9C:FC:E8:CF:55:76	EVILTWIN.22000	Evil WPA/2 Twin	Eviltwin Hashcat	2h 14m ago	?	?	?	?	?
EA:86:0A:8D:6A:CB	14:AC:60:81:AB:D5	Recon	Partial PCAP	1h 4m ago	✓	✓	✓	✓	✓
BC:F4:D4:11:36:4B	EVILTWIN.22000	Evil WPA/2 Twin	Eviltwin Hashcat	2h 48m ago	?	?	?	?	?
42:8A:2B:0C:CC:35	30:03:C8:2C:E6:53	Recon	Partial Hashcat	3h 23m ago	✓	✓	✗	✓	✓
9C:FC:E8:D3:18:32	EVILTWIN.PCAP	Evil WPA/2 Twin	Eviltwin PCAP	2h 56m ago	?	?	?	?	?
02:13:37:A9:AE:88	30:03:C8:2C:E6:53	Recon	Full Hashcat	3h 33m ago	✓	✓	✓	✗	✓
02:13:37:A9:AE:88	84:FD:D1:5A:15:E0	Recon	Partial Hashcat	1h 10m ago	✓	✓	✓	✗	✓
2C:33:11:D0:CD:E0	0C:EF:AF:D7:2B:F1	Recon	Full Hashcat	1h 11m ago	✓	✓	✓	✓	✓
F2:CE:0E:CD:DD:71	EVILTWIN.PCAP	Evil WPA/2 Twin	Eviltwin PCAP	0m 7s ago	?	?	?	?	?
02:13:37:A9:AE:10	D0:C6:37:DB:51:9B	Recon	Partial PCAP	1h 9m ago	✗	✓	✓	✓	✓

Explain the steps you took and the impact(s) you observed. If the device joins successfully, you should notice a notification. Report your findings.

Steps and Observations:

To perform the Evil Twin attack, we used the Pineapple's **Evil WPA** feature under the **PineAP** tab. We configured it to impersonate our previously created personal hotspot by matching the same SSID and password. The options **"Enabled"** and **"Capture Handshakes"** were turned on, while **"Impersonate all networks"** remained off to target only our hotspot.

Once configured, we disconnected our laptop from Wi-Fi and attempted to reconnect to the personal hotspot. After a few tries, the laptop unknowingly joined the **fake Evil Twin network** instead of the original one. This was confirmed by a **notification** in the Pineapple interface, and multiple entries appeared in the **Handshake Capture Log**, as seen in the screenshot.

The table displays various **partial and full handshakes** captured, along with device MAC addresses and timestamps. This confirmed that multiple clients (including our test device) connected to the rogue network, and successful handshake data was collected for further analysis.

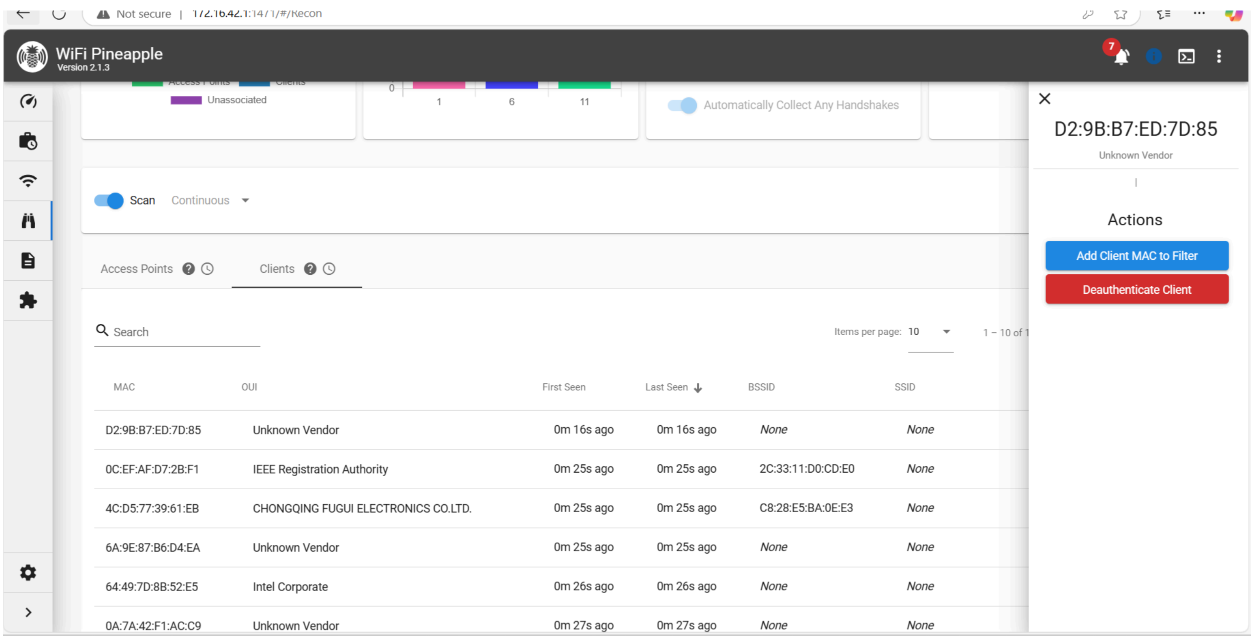
Traffic Observed:

After the device connected to the impersonated network, we ran:

```
tcpdump -i wlan2 -vv port 80
```

This command captured HTTP traffic sent through the rogue AP. We observed **GET requests**, **User-Agent headers**, and plaintext metadata for files like **images** and **web resources**. Although WPA2 was used for Wi-Fi security, the traffic remained readable because HTTP is unencrypted. This shows how an attacker can intercept sensitive data on a spoofed open or weakly encrypted network.

Question #8:



Explain the results of your deauthentication experiments. What did you observe?

For this part of the lab, we used the Recon tab to identify nearby clients connected to wireless networks. After scanning, we selected a specific client (MAC address: D2:9B:B7:ED:7D:85) from the client list and clicked the "Deauthenticate" button.

Upon sending the deauthentication signal, the client was immediately disconnected from its network. This was visually confirmed by observing a status change in the Recon dashboard and a drop in activity for that MAC address.

In some cases, the device attempted to automatically reconnect. If a spoofed Evil Twin access point was active, the client would occasionally connect to the rogue AP instead of the real one. This confirmed the effectiveness of the Evil Twin + Deauth attack combo, enabling an attacker to perform Man-in-the-Middle (MITM) interception.

Observation Summary:

- The target client was successfully disconnected.
- Reconnection behavior depended on signal strength and availability.
- The attack demonstrates how deauth packets can force clients onto malicious networks.

Question #9:

Steps:

1. Part I: After logging into Pineapple, use the web interface to configure DNS settings, modifying dnsmasq.

2. Use an SSH terminal to open dnsmasq configuration and add the entries bank.com and husky.com with address=(bank.com or husky.com)/<static IP, where "HuskyBanking" is hosted>
3. Restart dnsmasq to apply changes
4. Part II: After connecting to open network with laptop, go to bank.com/husky.com
5. It should redirect to the fake HuskyBanking website

(no screenshots because process isn't working :( )

Explanation:

To carry out the DNS hijack attack, we first logged into the Pineapple and accessed the DNS settings through the web interface. Then, using an SSH terminal, we opened the `dnsmasq` *configuration file, which controls how the Pineapple handles DNS requests. We added custom entries that redirected the domains* `bank.com` *and* `husky.com` *to a static IP address, which was where our fake HuskyBanking website was supposed to be hosted. This means that any device connected to the Pineapple's network would be tricked into going to our fake site instead of the real one. After adding these entries, we restarted the* `dnsmasq` *service to apply the changes. Finally, we connected a laptop to the open network we created and tried to visit* `bank.com` *and* `husky.com` *in a browser. While the redirect process was set up correctly, the fake website didn't fully load due to technical issues. However, the DNS hijack itself worked as intended, demonstrating how attackers can manipulate DNS responses to redirect users to malicious sites.*