Binghamton University, Watson School of Engineering

Stage 3:
Detection against 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
November 11th, 2024

Contents

1. Setting up the Environment for Detection 2
1.1 Recap of the Evil Twin Attack and Prevention 2
1.2 Detection Objective 2
1.3 Host Machine Setup (Attacker and Victim).....3
2. Theoretical Detection Techniques 4
2.1 Signal Strength Analysis 4
2.2 SSID and BSSID Verification 4
2.3 Wireless Intrusion Detection Systems (WIDS) 4
2.4 RF Fingerprinting 5
2.5 802.1X Authentication and RADIUS Servers 5
2.6 Certificate-Based Authentication 5
2.7 Time-of-Flight and Round-Trip Time Measurements 5
2.8 Machine Learning Approaches for Rogue AP Detection 6
3. Evil Twin Attack Detection Implementation (Practical).....7
3.1 Overview of Practical Detection Methods 7
3.2 Recap of Evil Twin Attack 7
3.3 Implementation of Detection Scripts/Tools 8
3.4 Detection Performance Analysis Using Confusion Matrix 14
4. Conclusion and Future Work 18
5. Bibliography 19

1. Setting up the Environment for Detection

1.1 Recap of the Evil Twin Attack and Prevention

In the **first stage** of this project, I performed an Evil Twin attack to highlight vulnerabilities in wireless networks. Using two USB Wi-Fi adapters with the Atheros AR9271 chipset on a Kali Linux machine, I created a rogue access point named "FreeWiFi" that mimicked a legitimate network. By employing tools like airmon-ng, aireplay-ng, and airbase-ng, I was able to perform de-authentication attacks and establish the rogue AP. Additionally, I hosted a captive portal using Python to capture sensitive user credentials—such as usernames, passwords, and email addresses—from a Windows victim machine that connected to this rogue AP.

In the **second stage**, I focused on prevention techniques to mitigate the risk of such attacks. I described theoretical measures to prevent evil twin attacks, such as hardening wireless configurations, using strong authentication and encryption protocols (like WPA3 and 802.1X authentication), and educating users about the dangers of connecting to unknown networks. On the practical side, I developed a Python script named prevent_evil_twin.py, which monitors the connected SSID on the Windows victim machine and automatically disconnects from any untrusted networks.

Despite these preventive efforts, I recognized that the possibility of an Evil Twin attack still exists, especially if attackers exploit new vulnerabilities or if users bypass security protocols. This realization led me to the third stage of the project: developing effective detection mechanisms to identify and respond to such threats proactively.

1.2 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.

1.3 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 script developed in Python.
    - Network monitoring tools like Wireshark for traffic analysis.
  - **Network Configuration:**
    - Initially connected to a legitimate SSID (e.g., "Ron").
    - Equipped with preventive measures from Stage 2, including the **prevent_evil_twin.py** script.

Special Note - 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:

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.

2. 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.

2.1 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.

2.2 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.

2.3 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.

2.4 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.

2.5 802.1X 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.

2.6 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.

2.7 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.

2.8 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 3, 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 2.

3. Evil Twin Attack Detection Implementation (Practical)

3.1 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:

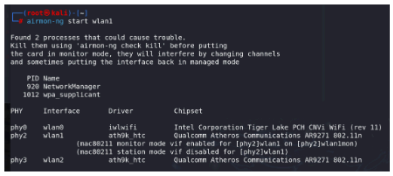
- Detection of de-authentication Frames using Wireshark on Kali Linux (Victim)
- Detection of Evil Twin Attack Using a Python Script on Windows

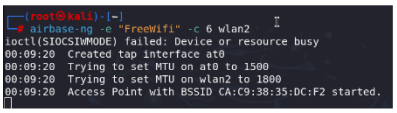
These methods aim to detect the Evil Twin attack from both the victim's perspective on a Windows machine and through network analysis on Kali Linux.

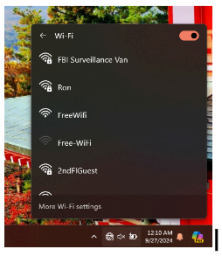
3.2 Recap of Evil Twin Attack

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.



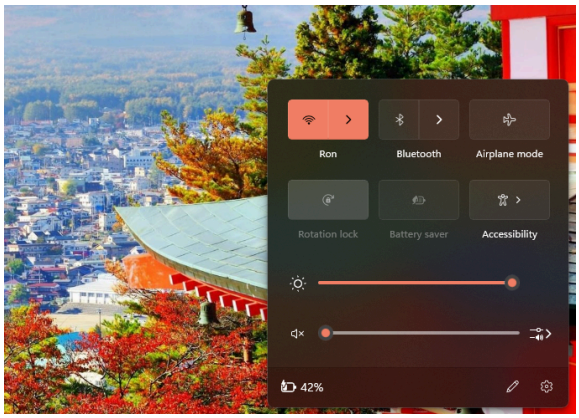




3.3 Implementation of Detection Scripts/Tools

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

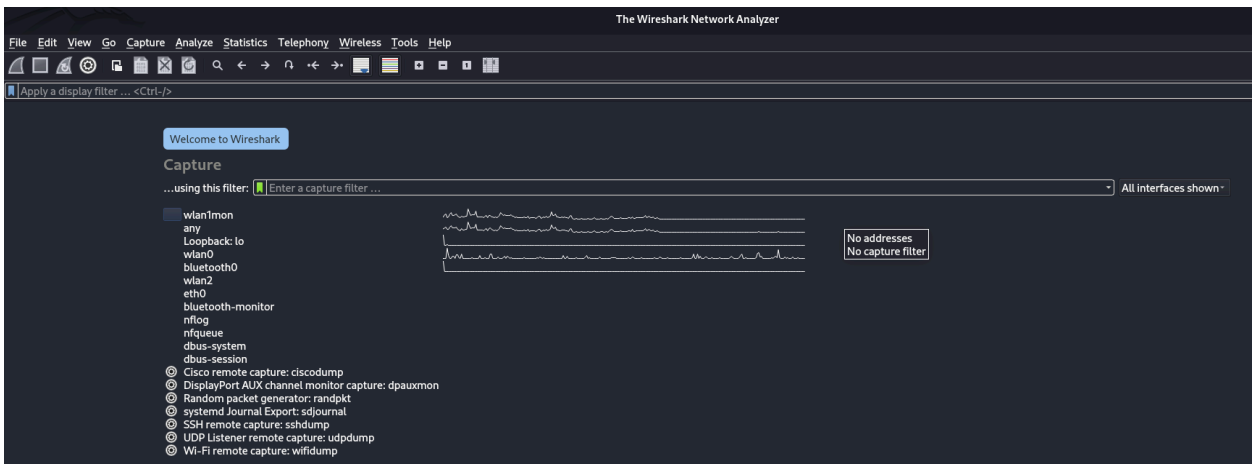
Wireshark is a powerful network protocol analyzer that can capture and display network packets in real time. In this implementation, I used Wireshark on a Kali Linux machine to detect de-authentication frames, which are indicative of an Evil Twin attack. The victim machine, running Windows, was initially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

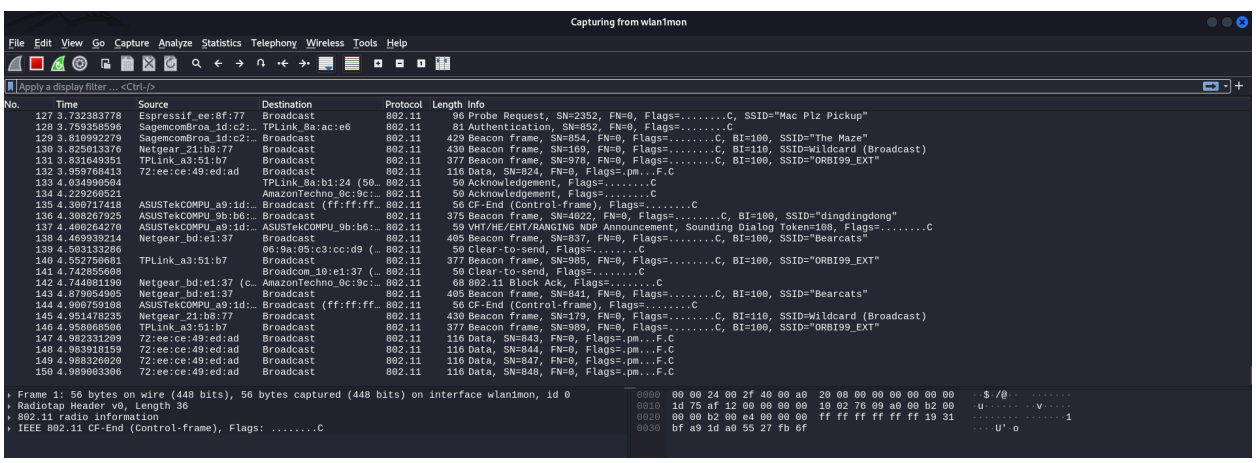


On the Kali Linux machine, I had two USB Wi-Fi adapters: wlan1 and wlan2. I enabled monitor mode on wlan1 by executing “airmon-ng start wlan1”, allowing the adapter to intercept wireless packets necessary for detection. Using wlan1mon, I performed an airodump with airodump-ng wlan1mon to identify the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

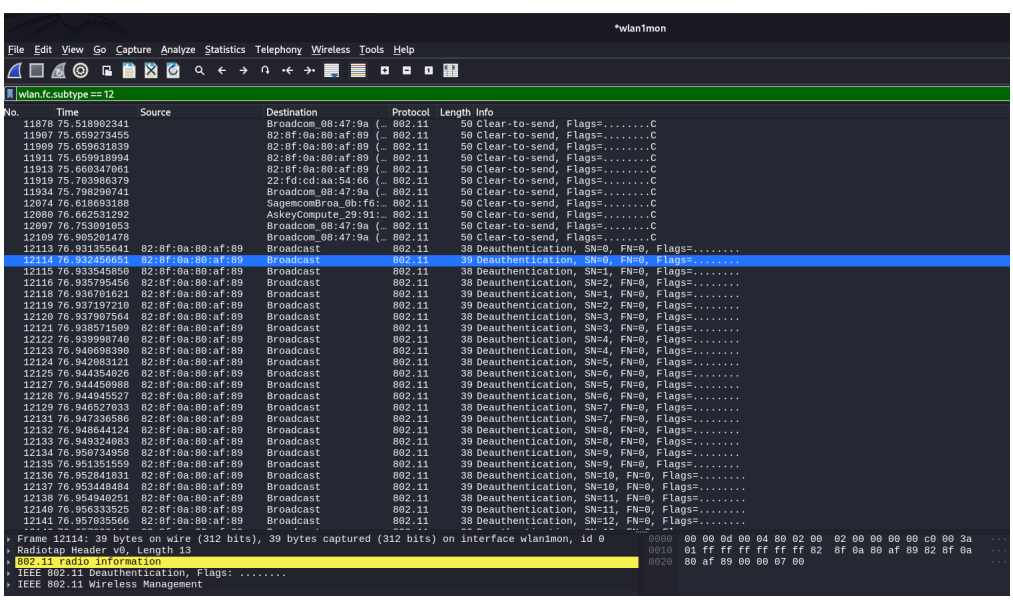
Since I was using the same Kali Linux machine for both attacking and detecting, I opened Wireshark and selected wlan1mon as my wireless interface to start capturing packets. While Wireshark was running, I executed the aireplay-ng command to perform the de-authentication attack on the "Ron" network:

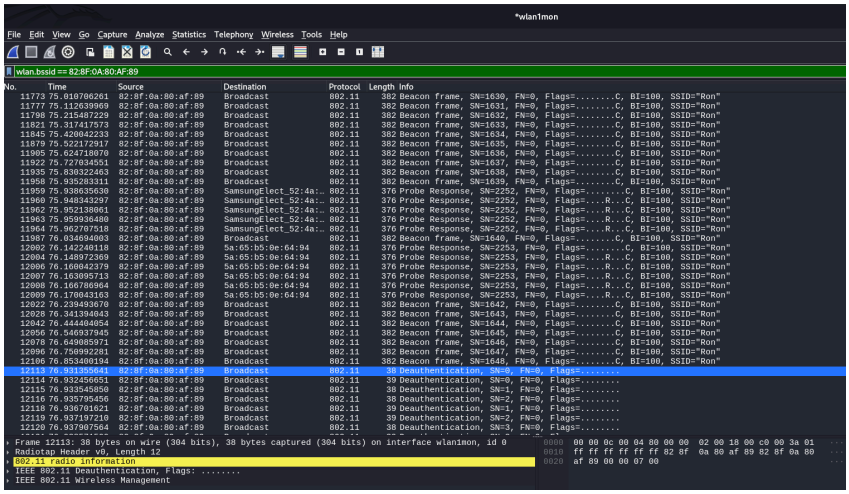




```
┌──(root㉿kali)-[~]
└─# aireplay-ng --deauth 20 -a 82:8F:0A:80:AF:89 wlan1mon
21:47:48  Waiting for beacon frame (BSSID: 82:8F:0A:80:AF:89) on channel 6
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
21:47:48  Sending DeAuth (code 7) to broadcast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

3.3.2 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)
```

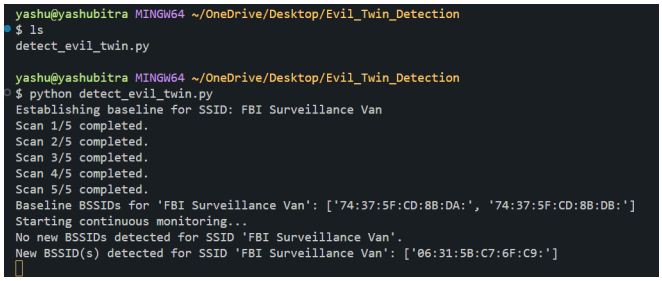
```python
def main():
    target_ssid = "FBI Surveillance Van"
    iface = get_wifi_interface()
    if iface is None:
        return

    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()
```

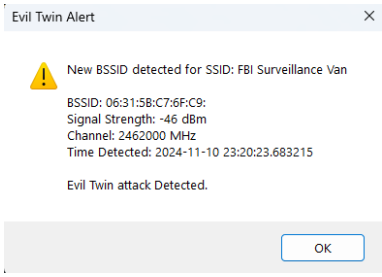
After running this script on the Windows machine, it establishes the baseline BSSIDs for "FBI Surveillance Van" by scanning the network environment multiple times. This ensures that all legitimate access points are recognized, which typically have one or at most a few BSSIDs. The script then enters a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

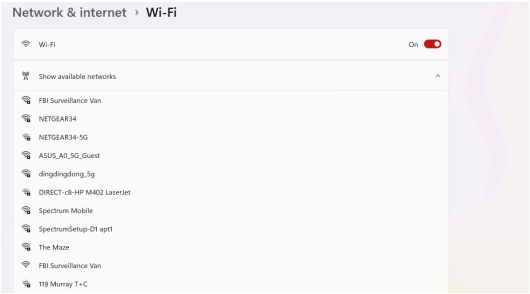
```
┌──(root㉿kali)-[~]
└─# airbase-ng -e "FBI Surveillance Van" -c 11 wlan2
23:20:17  Created tap interface at0
23:20:17  Trying to set MTU on at0 to 1500
23:20:17  Access Point with BSSID 06:31:5B:C7:6F:C9 started.
```

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.

3.4 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).

3.4.1 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.

3.4.2 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.

4. Conclusion and Future Work

This project stage 3 successfully explored theoretical detection methods and implemented two practical detection mechanisms against Evil Twin attacks on wireless networks. The first method utilized Wireshark on a Kali Linux machine to capture and analyze de-authentication frames, effectively identifying malicious attempts to disrupt legitimate connections. The second method employed a Python script on a Windows machine to monitor for new BSSIDs associated with a known SSID, providing automated and user-friendly alerts when a potential rogue access point was detected.

Both detection approaches demonstrated high effectiveness in controlled testing scenarios. Wireshark provided detailed packet-level insights, making it a powerful tool for network administrators, while the Python script offered continuous monitoring and real-time alerts suitable for broader user bases. However, each method has its limitations: Wireshark requires technical expertise for analysis, and the Python script depends on maintaining an accurate baseline of legitimate BSSIDs to minimize false positives.

Future Work should focus on enhancing these detection methods by integrating machine learning algorithms to improve accuracy and adaptability, automating baseline updates to accommodate legitimate network changes, and expanding alert mechanisms for more comprehensive incident response. Additionally, combining both detection techniques could provide a layered security approach, leveraging the strengths of each method to offer more robust protection against evolving wireless threats.

Overall, this project underscores the importance of proactive detection measures in complementing existing preventive strategies, thereby strengthening the overall security posture of wireless networks against sophisticated attacks like Evil Twins.

5. 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/