Name: Justin Ngo
PSU ID: jvn5439
Professor: Mark Mahon
Class: CompEn 462
Date: 05.07.2025
Project : Wireless MAC Address Spoofing Detection System Using ESP32

Abstract:

The purpose of this project is to use an ESP32 microcontroller to create a device that passively detects wireless connections to an access point and logs the MAC addresses and RSSI of the various connections that are made. The device operates in two modes: learning and monitoring. When the device detects that there are no known connections in the known_connections_file, it will enter learning mode and begin logging the MAC addresses and corresponding RSSI of the connections that are made for the next 2 minutes and 30 seconds. Assuming that these connections are "normal" connections, the device will save these MAC addresses and their corresponding RSSI values to the known_connections_file and use that as a sort of whitelist of valid MAC addresses. After the learning period, the device will enter monitoring mode and try to determine if the MAC address of incoming traffic is a spoofed MAC address or not by comparing the RSSI of any new connections to the RSSI of other connections that were made recently.

Outline:

The main idea behind this project was just to experiment with the ESP32 microcontroller and see if I could get it to detect wireless connections made to the access point in my apartment living room.

Tools and Libraries:

1. Arduino Nano ESP32 Microcontroller: this was the main hardware part used in the project. I selected it primarily because it was easy to get and I have a little experience with using other Arduino microcontrollers.

2. Arduino IDE: all the code was done in the Arduino IDE. I selected this IDE because I have prior experience with it from another class and remember it being fairly intuitive to use

3. SPIFFS Library: the SPIFFS library is what allowed the ESP32 to create a JSON file on its onboard flash memory. I found out about it while looking for a way to save the packet information to a file for the device to reference later.

4. Wifi Library: the Wifi library allowed the ESP32 to connect to the same wifi network as my computer. I'm not sure that it was entirely necessary to go through with adding the ESP32 to the network but did so anyway to ensure that it was properly scanning the network for connections.

5. ESP32 Wifi Library: I used this library to set the ESP32 in promiscuous mode. This library is core to the device's functionality as it's what allows the ESP32 to detect the MAC addresses and RSSI of the connections that are made to the access point.

6. ArduinoJson Library: I used this to maintain the JSON file which stored the known connections that the device learned during its learning phase.

Code Walkthrough:

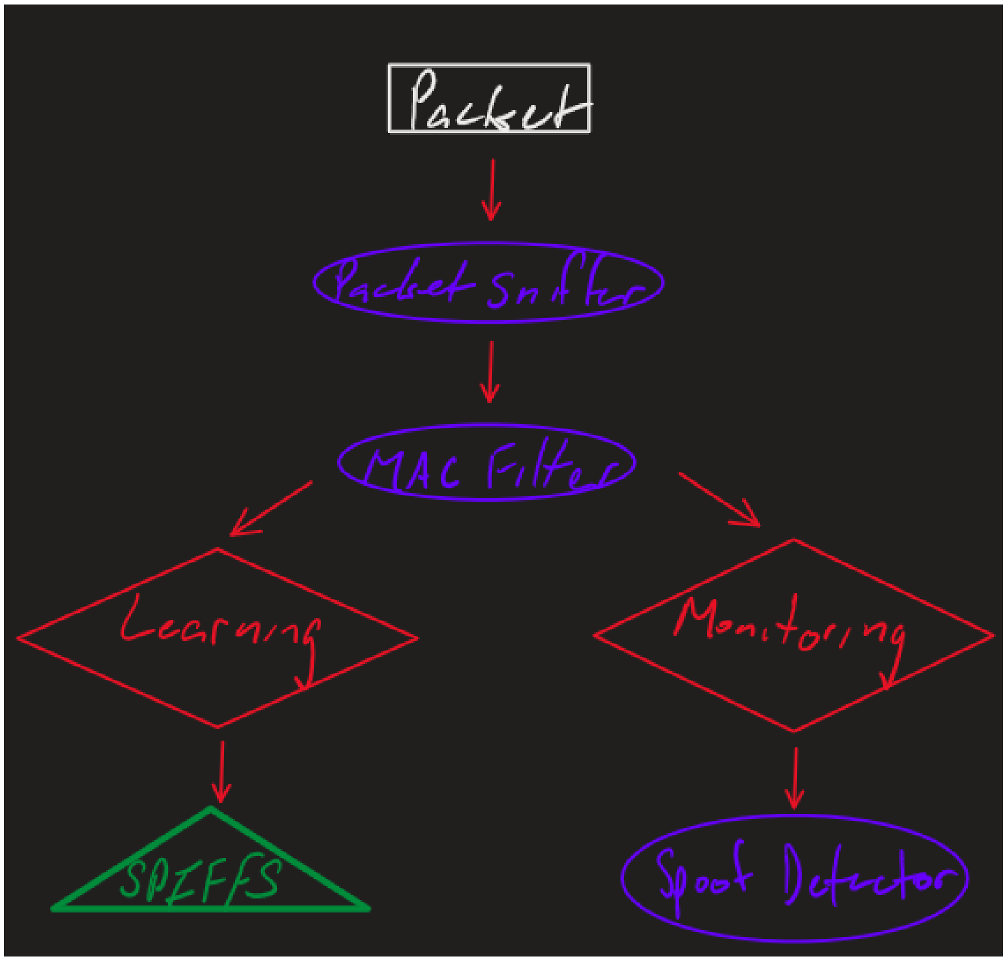

Figure 1: System Diagram

The figure above shows a very general overview of how the system works. The ESP32 detects packets in promiscuous mode which are then processed by the Packet Sniffer function. The MAC Filter removes packets to or from the other access points on the network and also ignores packets sent to FF:FF:FF:FF:FF:FF, which is the broadcast address to prevent the device from flooding the storage with unnecessary packets. Depending on what mode the device is currently operating in, the device will either save the packet information to the known_connections_file or compare the RSSI of the incoming packet to the RSSI of the other recently received packets to try and determine if it's coming from a spoofed MAC address.

```
// Ignored MACs
const DeviceInfo ignoredMACs[] = {
    { { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }, 0, 0, 0 }  // Broadcast address
};
// Function to check if a MAC address should be ignored
bool isIgnoredMAC(const uint8_t *mac) {
    // Check against ignored MACs list
    for (const auto &ignored : ignoredMACs) {
    if (memcmp(mac, ignored.mac, 6) == 0) {
        return true;
    }
    }

    // Make sure not to ignore the Host AP's MAC address
    if (mac[0] == 0x84 && mac[1] == 0x23 && mac[2] == 0x88 && mac[3] == 0x7B &&
   mac[4] == 0x90 && mac[5] == 0xA0) {
    return false;
    }
    //Ignore the MAC addresses of other APs on network
    else if (mac[0] == 0x84 && mac[1] == 0x23 && mac[2] == 0x88) {
    return true;
    }
    return false;
}

// Packet sniffer function
void sniffer(void *buf, wifi_promiscuous_pkt_type_t type) {
  wifi_promiscuous_pkt_t *packet = (wifi_promiscuous_pkt_t *)buf;
  uint8_t *srcMAC = &packet->payload[10];
  int rssi = packet->rx_ctrl.rssi;

  // Skip ignored MACs
  if (isIgnoredMAC(srcMAC)) return;

  updateObservedDevices(srcMAC, rssi);
  if (learningMode) {
    // Print during active window
      const char *pktType = (type == WIFI_PKT_MGMT) ? "MGMT" : "DATA";
      Serial.printf("%s - MAC: %s RSSI: %d dBm\n", pktType, macToString(srcMAC).
   c_str(), rssi);
  } else {
    checkForSpoofing(srcMAC, rssi);
  }
}
```

Listing 1: Packet Sniffer and MAC Filter

The packet sniffer is one of the main functions, and is responsible for detecting the packets that are sent to the access point. After detecting a packet, the function checks if the MAC address is in the ignored MACs list by calling isIgnoredMAC. If it is, the function will ignore the packet and move on to the next one. If it isn't, the function will call the updateObservedDevices function to update the list of recently observed devices. If the device is in learning mode, the function will also print the MAC address and RSSI of the packet to the serial monitor for the user to see. If the device is in monitoring mode, the function will call checkForSpoofing to try and determine if the MAC address is being spoofed.

```
// Check for MAC spoofing
void checkForSpoofing(const uint8_t *mac, int rssi) {
  unsigned long currentTime = millis();

  // Check if MAC matches AP's MAC
  if (memcmp(mac, AP_BSSID, 6) == 0) {
    Serial.println("\nALERT: Device using AP's MAC address detected!");
    return;
  }

  // Check against known devices
  for (const auto &known : knownDevices) {
    if (memcmp(mac, known.mac, 6) == 0) {
      if (abs(rssi - known.avgRSSI) > RSSI_VARIATION_THRESHOLD) {
        Serial.printf("\nALERT: Known device %s shows abnormal RSSI change!
  Current RSSI: %d dBm, Known Avg RSSI: %d dBm", macToString(mac).c_str(), rssi,
  known.avgRSSI);
        break; // Stop after the first match
      }
      return;
    }
  }

  // Check for MAC randomization
  for (auto &observed : observedDevices) {
    if (abs(rssi - observed.avgRSSI) < RANDOM_MAC_RSSI && memcmp(mac, observed.mac
  , 6) != 0 && (currentTime - observed.lastSeen) < TIME_WINDOW) {
      if (currentTime - observed.lastAlertTime > TIME_WINDOW) { // Cooldown check
        Serial.printf("\nALERT: Potential MAC randomization detected!");
        Serial.printf("\nCurrent: %s, Previous: %s\n", macToString(mac).c_str(),
  macToString(observed.mac).c_str());
        observed.lastAlertTime = currentTime; // Update last alert time
        break; // Stop after the first match
      }
    }
  }
}
```

Listing 2: Spoof Detection

The checkForSpoofing function is the other primary function and is what runs the main logic to determine if the MAC address is being spoofed or not. It checks for 3 main criteria:

1. If the MAC address matches the MAC address of the access point
2. If the RSSI from a known device falls outside of the variation threshold (set to 20 dBm)
3. If the RSSI from an unknown device compared to another recently observed device is within the random MAC RSSI threshold (set to 10 dBm)

If any of these criteria are met, the function will print an alert to the serial monitor. The logic behind the third criteria is that if the RSSI of an unknown device is within a 10 dBm range of another recently observed device, it could be a sign that the device is using a randomized MAC address.