ESP32 WiFi Auditor

Guide de Fonctionnement Complet

FIRMWARE v2.2

Framework de Test de Securite WiFi
Architecture ESP32 + Dashboard Next.js
Communication HTTP en temps reel

Compatible ESP32 / ESP32-S2 / ESP32-S3 / ESP32-C3
Arduino Core 2.x et 3.x

00 Sommaire

01 Architecture du Systeme

02 Communication ESP32 ↔ Dashboard

03 Modules du Firmware ESP32

04 Machine a Etats (State Machine)

05 Flux d'Attaque Evil Twin

06 Captive Portal & Verification

07 Capture PCAP & Trames 802.11

08 API de Communication

09 Guide d'Installation

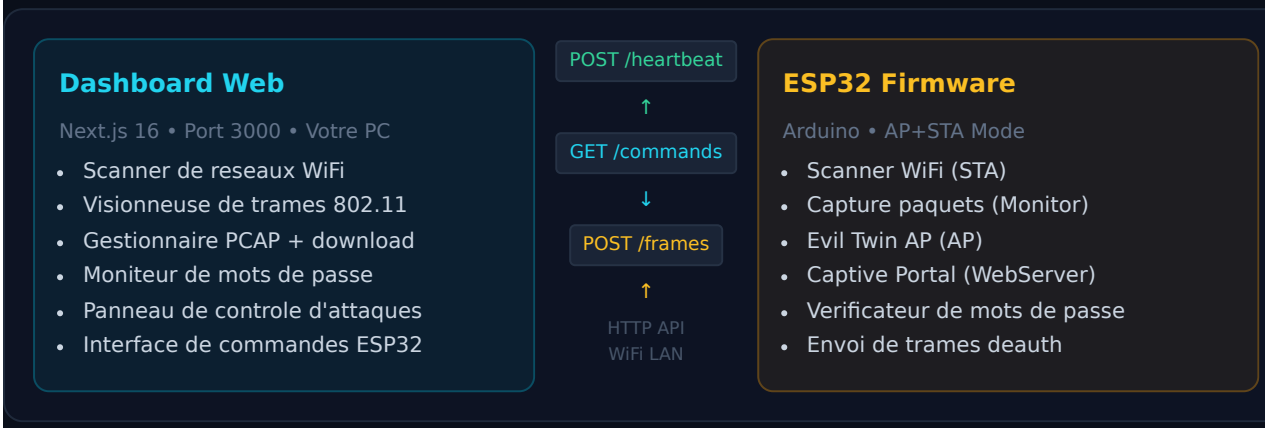
10 Commandes Serie & Depannage

Avertissement Legal

Ce projet est destine exclusivement a des fins educatives et de test de securite autorise. L'envoi de trames deauth, la creation de faux points d'acces et la capture de mots de passe sans autorisation sont illegaux dans la plupart des pays. Utilisez ce systeme uniquement sur des reseaux dont vous etes proprietaire ou pour lesquels vous avez une autorisation ecrite.

01 Architecture du Systeme

Le systeme est compose de deux parties qui communiquent via le reseau WiFi local : le Dashboard web (Next.js) et le firmware ESP32 (Arduino).



Principe de fonctionnement

L'ESP32 fonctionne en mode **AP+STA** : il cree un faux point d'acces (Evil Twin) tout en restant connecte au reseau WiFi local pour communiquer avec le Dashboard. Le Dashboard envoie des commandes via une file d'attente HTTP. L'ESP32 interroge cette file toutes les **5 secondes**. Les resultats sont envoyes au Dashboard via des requetes POST HTTP.

Fichiers du firmware

Fichier	Role	Taille
`esp32_wifi_auditor.ino`	Fichier principal - setup/loop/commandes	9.5 KB
`wifi_scanner.h`	Scan des reseaux WiFi	3.8 KB
`packet_capture.h`	Capture promiscuous + PCAP + deauth	6.2 KB
`captive_portal.h`	Portail captif + capture mots de passe	7.0 KB
`password_verifier.h`	Verification contre le vrai reseau + NAT	4.8 KB
`api_client.h`	Communication HTTP avec le Dashboard	4.5 KB

02 Communication ESP32 ↔ Dashboard

La communication repose sur un modele pull : l'ESP32 interroge le Dashboard pour obtenir des commandes, puis envoie les resultats.

1. ESP32 envoie un heartbeat toutes les 30s → `POST /api/esp32/heartbeat`

↓

2. Dashboard enregistre le statut et retourne les commandes en attente

↓

3. ESP32 interroge les commandes toutes les 5s → `GET /api/esp32/commands`

↓

4. ESP32 execute la commande (scan, capture, attaque...)

↓

5. ESP32 envoie les resultats → `POST /api/networks`, `/frames`, `/passwords`

↓

6. ESP32 rapporte le resultat → `PATCH /api/esp32/commands`

Heartbeat (toutes les 30 secondes)

L'ESP32 envoie son statut complet. Si aucun heartbeat pendant 60s, le Dashboard marque l'ESP32 **hors ligne**.

```
{
  "deviceId": "AA:BB:CC:DD:EE:FF",    // Adresse MAC
  "ip": "192.168.1.42",
  "freeHeap": 145632,
  "uptime": 3600,
  "wifiMode": "AP_STA",
  "apSsid": "MonWiFi",
  "apClients": 2,
  "currentChannel": 6,
  "isCapturing": true,
  "isAttackRunning": true,
  "attackType": "evil_twin",
  "firmwareVersion": "2.2"
}
```

Intervales de communication

Action	Intervalle	Direction
Heartbeat	30 secondes	ESP32 → Dashboard
Poll de commandes	5 secondes	ESP32 → Dashboard
Stats de capture	5 secondes	ESP32 → Dashboard
Stats de trames	10 secondes	ESP32 → Dashboard
Trames individuelles	Temps reel	ESP32 → Dashboard

03 Modules du Firmware ESP32

Le firmware est divise en 5 modules independants, chacun dans un fichier .h.

`wifi_scanner.h`

Scanner WiFi

Scanne les reseaux WiFi. Collecte SSID, BSSID, canal, RSSI et type de chiffrement (Open, WEP, WPA, WPA2, WPA3). Utilise `WiFi.scanNetworks()`.

`packet_capture.h`

Capture de Paquets

Mode promiscuous pour capturer les trames 802.11. Accumule au format PCAP. Envoie les metadonnees en temps reel. Envoie des trames deauth via `esp_wifi_80211_tx()`.

`captive_portal.h`

Portail Captif

Cree un faux AP avec page de connexion. Intercepte les DNS, redirige vers le portail, capture les mots de passe. Compatible Android, iOS, Windows.

`password_verifier.h`

Verificateur de Mots de Passe

Teste les mots de passe par connexion reelle au reseau cible. Si correct, configure un bridge NAT AP+STA pour donner au client l'acces Internet.

`api_client.h`

Client API

Classe `APIClient` pour toutes les communications HTTP : heartbeat, reseaux, captures, mots de passe, activites, trames, resultats de commandes.

Types de trames 802.11 detectees

Type	Code	Description
Beacon	0x0080	Trames de diffusion du point d'acces
Probe Request	0x0040	Demande de recherche de clients
Probe Response	0x0050	Reponse du point d'acces
Authentication	0x00B0	Demande d'authentification
Deauthentication	0x00C0	Deconnexion forcee (deauth)
Data	0x0008	Donnees utilisateur
QoS Data	0x0088	Donnees avec qualite de service

04 Machine a Etats

L'ESP32 passe par differents etats en fonction des commandes recues et des evenements.

IDLE - En attente de commandes

↓ commande `scan`

SCANNING - Scan des reseaux en cours

↓ scan termine

IDLE - Retour a l'attente

↓ commande `capture_start`

CAPTURING - Mode promiscuous actif, capture PCAP

↓ commande `evil_twin`

EVIL_TWIN - Faux AP actif + DNS intercept

↓ client se connecte au portail

CAPTIVE_PORTAL - Portail actif, capture mot de passe

↓ mot de passe correct

CONNECTED - Bridge NAT actif, client a Internet

Indicateur LED par etat

Etat	LED	Comportement
IDLE	Off	LED eteinte
SCANNING / CAPTURING	Clignotement rapide	200ms on/off
EVIL_TWIN / CAPTIVE_PORTAL	Clignotement lent	1000ms on/off
CONNECTED	Allumee fixe	LED constamment allumee

Boucle principale (loop) - 8 taches cooperatives

1. **DNS + HTTP** - Traite les requetes du portail captif
2. **Capture** - Flush periodique des stats de capture (5s)
3. **Verification** - Teste les mots de passe captures
4. **Heartbeat** - Envoi du statut toutes les 30s
5. **Poll** - Interrogation des commandes toutes les 5s
6. **Frame stats** - Rapport des statistiques de trames (10s)
7. **LED** - Mise a jour de l'indicateur visuel
8. **Serial** - Traitement des commandes serie manuelles

05 Flux d'Attaque Evil Twin

L'attaque Evil Twin cree un faux point d'acces avec le meme SSID que le reseau cible.

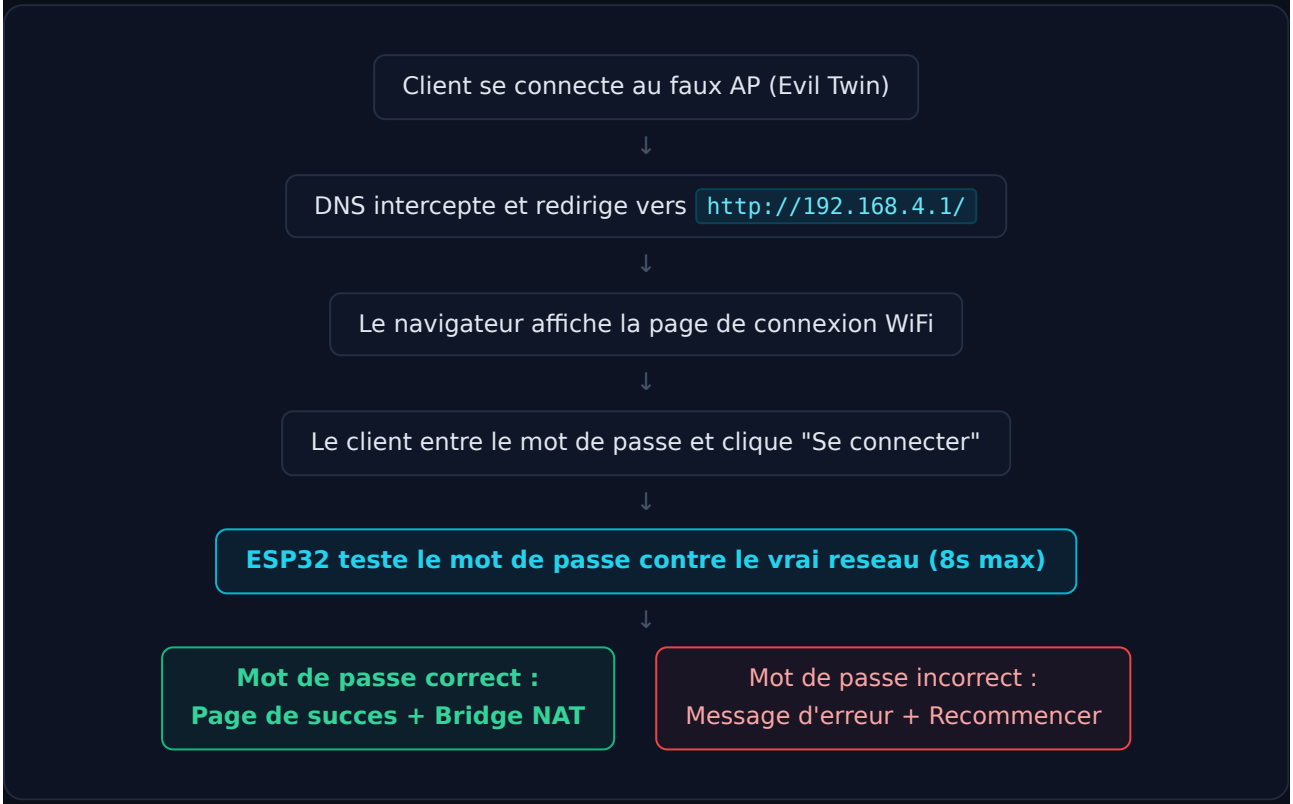
1. **Scan des reseaux**
   L'utilisateur clique "Scan". L'ESP32 detecte les reseaux et envoie la liste au Dashboard.
2. **Selection de la cible**
   L'utilisateur selectionne un reseau (SSID, BSSID, canal) et lance l'attaque Evil Twin.
3. **Creation du faux AP**
   L'ESP32 cree un AP avec le meme SSID : `WiFi.softAP(ssid, "", channel)`. DNS intercepte tout.
4. **Envoi de trames Deauth**
   Optionnel : trames deauth broadcast via `esp_wifi_80211_tx()` pour deconnecter les clients du vrai AP.
5. **Client se connecte au faux AP**
   Le client est redirige vers le portail captif. Son navigateur affiche une page de connexion.
6. **Capture du mot de passe**
   Le client entre son mot de passe. L'ESP32 le teste immediatement contre le vrai reseau.

Structure d'une trame Deauth (26 octets)

```
uint8_t deauthFrame[26] = {
  0xC0, 0x00,                         // Frame Control: Deauth
  0x00, 0x00,                         // Duration
  0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, // Destination (broadcast)
  0x00,0x00,0x00,0x00,0x00,0x00, // Source (AP BSSID)
  0x00,0x00,0x00,0x00,0x00,0x00, // BSSID
  0x00, 0x00,                         // Sequence number
  0x07, 0x00                          // Reason: Class 3 frame
};
```

06 Captive Portal & Verification

Le portail captif intercepte les clients et capture les mots de passe en les verifiant en temps reel contre le vrai reseau.



Processus de verification du mot de passe

1. **Sauvegarde** le mode WiFi actuel (AP+STA)
2. **Deconnecte** du faux AP temporairement
3. Passe en mode **STA uniquement**
4. Tente `WiFi.begin(ssid, password)` sur le vrai reseau
5. Attend jusqu'a **8 secondes** pour le resultat
6. Si `WL_CONNECTED` : le mot de passe est correct !
7. Si `WL_CONNECT_FAILED` ou `WL_WRONG_PASSWORD` : incorrect
8. **Restaure** le mode AP+STA et le faux AP

Bridge NAT (apres succes)

Si le mot de passe est verifie, l'ESP32 configure un bridge : mode `WIFI_AP_STA` (connecte au vrai reseau + faux AP actif). Le client obtient l'acces Internet via le NAT de l'ESP32. IP du faux AP : `192.168.4.1`. L'ESP32 agit comme routeur entre le client et le vrai reseau.

Detection du portail captif par OS

OS	Endpoint	Redirection
Android	`/generate_204`	302 vers le portail
iOS	`/hotspot-detect.html`	302 vers le portail
Windows	`/ncsi.txt`	302 vers le portail
Windows	`/fwlink`	302 vers le portail

07 Capture PCAP & Trames 802.11

Le module de capture enregistre les trames WiFi brutes au format PCAP standard, analysable avec Wireshark.

Format PCAP IEEE 802.11

```
// En-tete global (24 octets, au debut du fichier)
magic_number = 0xa1b2c3d4   // Little-endian
version_major = 2
version_minor = 4
network = 105               // IEEE 802.11 link type
snaplen = 65535

// En-tete de paquet (16 octets, avant chaque trame)
ts_sec, ts_usec             // Timestamp
incl_len, orig_len          // Longueur capturee / originale
```

Processus de capture

1. `esp_wifi_set_channel()` - Definit le canal d'ecoute
2. `esp_wifi_set_promiscuous(true)` - Active le mode moniteur
3. `esp_wifi_set_promiscuous_rx_cb()` - Enregistre le callback
4. `esp_wifi_set_promiscuous_filter()` - Filtre MGMT + DATA + CTRL
5. Chaque trame recue declenche le callback `IRAM_ATTR`
6. Les donnees sont accumulees dans un buffer PCAP de 8 KB
7. Les metadonnees sont envoyees au Dashboard en temps reel

> **Optimisation :** Seules les trames non-beacon sont envoyees en temps reel. Les trames Beacon (10/sec/AP) sont filtrees : 1 sur 10 seulement. Toutes les trames deauth, auth, probe, data sont envoyees.

08 API de Communication

Tous les echanges entre l'ESP32 et le Dashboard passent par l'API HTTP.

Endpoints ESP32 → Dashboard

Methode	Endpoint	Description
POST	/api/esp32/heartbeat	Heartbeat + statut complet
GET	/api/esp32/commands	Commandes en attente
PATCH	/api/esp32/commands	Resultat d'une commande
POST	/api/esp32/frames	Metadonnees de trames
POST	/api/networks	Reseaux decouverts
POST	/api/passwords	Mots de passe verifies
POST	/api/activity	Logs d'activite

Commandes Dashboard → ESP32

Commande	Parametres	Effet
scan	-	Lance un scan WiFi
capture_start	channel, captureId	Demarre la capture PCAP
capture_stop	-	Arrete la capture
evil_twin	ssid, bssid, channel	Lance l'Evil Twin + Portail
deauth	bssid, channel	Envoie des trames deauth
captive_portal	ssid, channel	Active le portail captif
stop_attack	-	Arrete l'attaque
verify_password	password, ssid	Teste un mot de passe
set_channel	channel	Change le canal de capture
restart	-	Redemarre l'ESP32

Exemple de commande

```
// Dashboard → ESP32
{ "command": "evil_twin",
  "params": { "ssid": "MonWiFi", "channel": 6 } }

// ESP32 → Dashboard (resultat)
{ "commandId": "abc123", "status": "executed" }
```

09 Guide d'Installation

Suivez ces etapes pour installer et configurer le systeme complet.

1. Installer le Dashboard (Next.js)

```
git clone <repo-url> my-project
cd my-project
bun install
bun run db:push
bun run dev
# Dashboard accessible sur http://localhost:3000
```

2. Installer Arduino IDE + ESP32

```
# Telecharger Arduino IDE 2.x depuis arduino.cc
# File > Preferences > Additional Board Manager URLs :
https://espressif.github.io/arduino-esp32/package_esp32_index.json
# Tools > Board > Boards Manager > "esp32" > Install
```

3. Configurer le firmware

```
# Creer un dossier esp32_wifi_auditor
# Placer le .ino et les 5 fichiers .h dedans
# Modifier l'IP du Dashboard :
const char* DASHBOARD_IP = "192.168.1.100";  // ← IP DE VOTRE PC
const int   DASHBOARD_PORT = 3000;
```

4. Configurer l'ESP32 dans Arduino IDE

Parametre	Valeur
Board	ESP32 Dev Module
Upload Speed	115200
Flash Frequency	80MHz
Flash Mode	QIO
Partition Scheme	Default 4MB with spiffs

Important

L'ESP32 et le PC doivent etre sur le **meme reseau WiFi local**. Verifiez que le Dashboard est accessible depuis un autre appareil : `http://IP_DU_PC:3000`

10 Commandes Serie & Depannage

Commandes Serie (115200 baud)

Commande	Description	Exemple
`scan`	Scanner les reseaux WiFi	`scan`
`capture <channel>`	Demarrer capture sur un canal	`capture 6`
`stop`	Arreter la capture	`stop`
`evil_twin <ssid> <ch>`	Lancer l'Evil Twin	`evil_twin MonWiFi 6`
`status`	Afficher le statut actuel	`status`

Depannage

esp_wifi_set_channel non declare

Ajoutez `#include <esp_wifi.h>` au debut du fichier .ino. Necessaire pour les fonctions ESP-IDF de bas niveau.

WL_WRONG_PASSWORD non declare

N'existe pas dans ESP32 Arduino Core 2.x. Le firmware v2.2 inclut un `#ifndef WL_WRONG_PASSWORD / #define WL_WRONG_PASSWORD 7` pour la compatibilite.

L'ESP32 n'envoie pas de heartbeat

Verifiez `DASHBOARD_IP` (IP de votre PC). Verifiez que le Dashboard est lance. Testez avec `curl http://IP:3000/api/status`.

Le deauth ne fonctionne pas

Certaines versions ESP-IDF requierent de desactiver PMF (Protected Management Frames) dans menuconfig. Sur les AP modernes avec PMF, le deauth peut etre ignore.

Le bridge NAT ne donne pas Internet

Le NAT depend de `IP_FORWARD=1` dans lwIP. Recompilez avec `make menuconfig` et activez IP forwarding.

Compatibilite des cartes

Carte	Scan	Capture	Evil Twin	Deauth
ESP32 (original)	Oui	Oui	Oui	Oui
ESP32-S2	Oui	Oui	Oui	Partiel
ESP32-S3	Oui	Oui	Oui	Partiel
ESP32-C3	Oui	Oui	Oui	Non