UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Magistrale in Informatica

IoT Security

IoT Security Networking

A simple IoT Security project on secure and not secure communication tasks.

Candidati:

Antonio Garofalo

Fulvio Serao

Professori:

Prof. Christian Esposito

ANNO ACCADEMICO 2024/2025

Indice

Capitolo 1

Introduzione

1.1 Panoramica del Progetto

Il progetto *IoT Security Networking* rappresenta un'implementazione di un sistema di comunicazione **client-server** basato su due dispositivi **ESP32**. Questa comunicazione può avvenire tramite protocollo **HTTP** o **HTTPS**, a scelta dell'utente. Gli **ESP32**, noti per le loro capacità di rete e di elaborazione a basso costo, rappresentano una soluzione ideale per l'implementazione di reti sicure nell'ambito dell'**Internet of Things (IoT)**. In un mondo in cui i dispositivi IoT sono sempre più diffusi e interconnessi, la necessità di garantire la **sicurezza** della comunicazione tra di essi è fondamentale per proteggere **dati sensibili** e garantire l'integrità dei sistemi.

1.2 Obiettivi del Progetto

Il progetto ha come obiettivo la creazione di un sistema di comunicazione client-server che consenta di esplorare le vulnerabilità della sicurezza nella trasmissione dei dati in reti IoT. In particolare, il sistema mira a:

- **Instaurare un servizio lato server** per la gestione della comunicazione con il client ESP32, consentendo sia connessioni HTTP sia HTTPS.
- **Intercettare il traffico di rete** generato tra i due dispositivi tramite diverse modalità (che verranno illustrate nei capitoli successivi) per analizzare le vulnerabilità e la sicurezza dei dati trasmessi.
- **Simulare un attacco di replay** per dimostrare la differente risposta del sistema a seconda del protocollo utilizzato. Un *replay attack*, o *attacco di ripetizione*, è un tentativo di intercettare e ritrasmettere pacchetti di dati allo scopo di ingannare il sistema o modificare lo stato della comunicazione.
- **Dimostrare come il server sia in grado di proteggere** la comunicazione da attacchi (ad es., replay attacks) quando si utilizza il protocollo HTTPS, ma non riesca a farlo quando la connessione avviene tramite HTTP. Questo risultato evidenzia i benefici dell'HTTPS, che protegge l'integrità e la riservatezza dei dati trasmessi attraverso crittografia e certificati digitali, rendendo impraticabile la manipolazione e sabotaggio dei pacchetti.

1.3 Struttura della Documentazione

La documentazione è organizzata come segue:

- **Architettura Progettuale**: descrizione dell'architettura del sistema, che comprende le componenti client, server e le tecnologie di test utilizzate. In questa sezione verranno illustrati i ruoli di ciascun componente nella gestione della comunicazione, nell'analisi del traffico e nei test di connessione.
- **Tecnologie Adoperate**: presentazione delle principali tecnologie utilizzate nel progetto, che includono:
  - **Espressif ESP32**: il dispositivo hardware utilizzato per implementare le comunicazioni IoT.
  - **Arduino e ESP-IDF**: le piattaforme di sviluppo per programmare e gestire il firmware dell'ESP32.
  - **PlatformIO**: un *IDE* per lo sviluppo di applicazioni in ambito IoT.
  - **Firmware Marauder**: un firmware specializzato che consente la rilevazione e il monitoraggio delle reti.
  - **Strumenti e Tecniche di Sniffing**: vengono utilizzati strumenti di analisi del traffico di rete e tecniche di sniffing per intercettare e analizzare i pacchetti scambiati tra client e server.
- **Implementazione**: sezione pratica che fornisce le istruzioni per la configurazione e l'utilizzo del sistema.
- **Conclusioni e Sviluppi Futuri**: riflessioni sui risultati ottenuti, i limiti dell'implementazione e discussione delle possibili estensioni e miglioramenti del progetto, come l'integrazione di ulteriori protocolli di sicurezza o miglioramenti nella rilevazione di attacchi.

1.4 Contesto Applicativo

L'**Internet of Things (IoT)** rappresenta una delle frontiere più promettenti della tecnologia moderna, con applicazioni che spaziano dalla **domotica**, al **monitoraggio industriale**, fino alla **sanità** e ai **trasporti intelligenti**. Tuttavia, la diffusione capillare di dispositivi connessi pone importanti sfide dal punto di vista della **sicurezza**, poiché ogni dispositivo rappresenta un potenziale punto di accesso per attacchi informatici. In tale contesto, la protezione delle comunicazioni diventa essenziale.

Questo progetto, quindi, si inserisce in un panorama in cui garantire la sicurezza delle reti IoT non è solo consigliabile, ma è un requisito imprescindibile per applicazioni critiche. La dimostrazione pratica di come una comunicazione **HTTPS** possa prevenire varie tipologie di attacchi è particolarmente rilevante per sviluppatori, ingegneri e ricercatori che lavorano su progetti IoT, fornendo un caso d'uso reale dell'importanza di scegliere protocolli di comunicazione sicuri.

Capitolo 2

Architettura Progettuale

2.1 Panoramica dell'Architettura

L'architettura del sistema è progettata per implementare un ambiente di comunicazione sicuro in ambito **IoT**, utilizzando due dispositivi **ESP32** configurati come **client** e **server**. La comunicazione tra i dispositivi avviene tramite il protocollo **HTTP** o **HTTPS**. Una componente cruciale di questo progetto è l'impiego di tecniche di **sniffing** per monitorare e analizzare i pacchetti di rete, al fine di valutare le vulnerabilità dei protocolli utilizzati.

L'architettura del sistema comprende le seguenti componenti principali:

- **Client ESP32**: dispositivo ESP32 configurato per inviare richieste al server, con supporto per le comunicazioni HTTP e HTTPS.
- **Server ESP32**: dispositivo ESP32 configurato per ricevere e processare le richieste dal client, con meccanismi di **logging** per la rilevazione di anomalie (ad es., problemi di handshake).
- **Network Testing**: un sistema per **intercettare** e **monitorare** i pacchetti scambiati tra client e server, consentendo l'analisi delle vulnerabilità dei protocolli.

2.2 Client ESP32

Il **client ESP32** è responsabile dell'invio di richieste al server, simulando una comunicazione tipica tra dispositivi IoT. La configurazione del client consente di scegliere tra protocolli HTTP e HTTPS, offrendo flessibilità per analizzare la sicurezza in diversi scenari. Le principali funzionalità del client includono:

- **Configurazione del Protocollo**: Il client è programmato per supportare sia comunicazioni HTTP che HTTPS, con una selezione programmabile all'inizio della sessione di comunicazione.
- **Generazione di Richieste**: Il client invia richieste periodiche al server, includendo dati simulati per rappresentare informazioni sensibili tipiche delle applicazioni IoT, come un *segreto* condiviso per l'autenticazione (ad es., una stringa identificativa `"iotsecurity"`).
- **Memorizzazione dei Certificati**: Il client conserva in memoria il **certificato della CA** e altri certificati necessari per stabilire una connessione HTTPS sicura con il server. Sono inoltre memorizzati il certificato del client e la relativa chiave privata, per supportare eventuali autenticazioni mutue client-server.
- **Sistema di Logging**: Il client registra le risposte ricevute dal server e segnala **anomalie di comunicazione** tramite un modulo di logging dedicato per HTTP/HTTPS.

Questa configurazione del client consente di testare diversi scenari di attacco e di monitorare la risposta del sistema in presenza di interferenze o tentativi di compromissione del traffico.

2.3 Server ESP32

Il **server ESP32** rappresenta il nodo ricevente della comunicazione, il cui compito è rispondere alle richieste del client e garantire l'integrità della connessione. Il server è configurato per gestire sia richieste HTTP che HTTPS, con meccanismi di **sicurezza integrati** per proteggere le comunicazioni. Le principali funzionalità del server includono:

- **Elaborazione delle Richieste**: Il server riceve le richieste dal client (ad esempio, richieste di accesso a servizi) e risponde con i dati richiesti, emulando una tipica comunicazione client-server.
- **Controllo dell'Integrità**: Quando viene utilizzato HTTPS, il server verifica i certificati per **autenticare e cifrare** la connessione, garantendo che i servizi siano erogati solo in modo sicuro e protetto.
- **Sistema di Logging**: Il server registra le richieste ricevute e annota eventuali anomalie o richieste sospette, come fallimenti nel **handshake SSL/TLS**, soprattutto in scenari non sicuri con HTTP.

Il server è progettato per analizzare le richieste lecite e sospette, generando log dettagliati che consentono una successiva valutazione dei tentativi di intrusione.

2.4 Network Testing

Per valutare l'efficacia della comunicazione sicura in ambito IoT, vengono impiegati strumenti e tecniche avanzate di **sniffing** e **analisi di rete**. Questi test sono eseguiti tramite software di analisi e simulazione di attacco (ad esempio, **Wireshark**, **tcpreplay** e **dsniff**), seguendo una metodologia strutturata:

- **Intercettazione dei Pacchetti**: Viene intercettato il traffico di rete nell'area di interesse, con l'applicazione di **filtri specifici** (ad es., IP di destinazione, solo traffico HTTP, uso di HTTPS).
- **Analisi del Traffico**: I pacchetti intercettati vengono analizzati in dettaglio. I file di output (`.pcap`) vengono esaminati per estrarre i pacchetti rilevanti, seguire i flussi TCP e, dove possibile, ricostruire le richieste.
- **Simulazione di Attacchi**: Viene simulato un attacco **Man In The Middle (MITM)**, che può essere di tipo passivo o attivo (ad es., **DDoS/DoS**) per verificare la capacità del sistema di mitigare questi attacchi.

Questi strumenti consentono di simulare scenari di **penetration testing** su dispositivi IoT, come una sandbox, per verificare la robustezza delle difese di sicurezza implementate.

Capitolo 3

Tecnologie Adoperate

3.1 Espressif ESP32

L'*Espressif ESP32*[1] è il microcontrollore scelto per la comunicazione IoT in questo progetto, noto per le sue capacità di connettività integrate (Wi-Fi e Bluetooth) e la sua versatilità. Progettato da Espressif Systems, l'ESP32 è dotato di un processore dual-core a 32 bit basato su architettura Xtensa, che opera fino a 240 MHz, offrendo elevate prestazioni con un consumo energetico ridotto.

Le principali caratteristiche dell'ESP32 includono:

- **Connettività Wi-Fi e Bluetooth**: L'ESP32 supporta il Wi-Fi 802.11 b/g/n e Bluetooth 4.2/Bluetooth Low Energy (BLE), consentendo comunicazioni flessibili e robuste in applicazioni IoT.
- **Basso Consumo Energetico**: Grazie a modalità di risparmio energetico avanzate, l'ESP32 è ideale per applicazioni che richiedono operatività continua e basso consumo, caratteristica fondamentale per i dispositivi IoT alimentati a batteria.
- **Supporto per Sicurezza Avanzata**: L'ESP32 include un coprocessore di crittografia hardware, che consente l'implementazione di SSL/TLS, supportando così HTTPS per garantire comunicazioni sicure.

- **Versatilità e Supporto per protocolli built-in**: Oltre a HTTP e HTTPS, l'ESP32 supporta altri protocolli di rete (MQTT, WebSocket) che lo rendono idoneo per un'ampia gamma di applicazioni IoT.

L'ESP32 è utilizzato sia come **client** che come **server** in questo progetto. In particolare grazie al framework **Arduino**, semplice e versatile, unito alle funzioni built-in del framework **ESP-IDF**, è stato possibile creare una connessione HTTPS anche se non supportata nativamente.

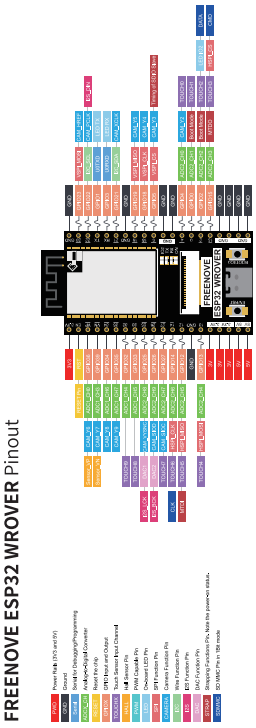


Figura 3.1: ESP32 Pinout

3.2 Developer Frameworks

Per programmare e gestire il firmware dell'ESP32, sono state utilizzate due piattaforme di sviluppo: *Arduino* e *ESP-IDF (Espressif IoT Development Framework)*.

3.2.1 Arduino

La piattaforma *Arduino*[2] fornisce un ambiente di sviluppo semplice e intuitivo, ideale per lo sviluppo rapido di applicazioni IoT non avanzate. Con Arduino, è possibile configurare l'ESP32 per inviare e ricevere dati su protocolli come HTTP in modo relativamente semplice ma senza un supporto nativo per la connessione HTTPS (è necessario utilizzare ESP-IDF). I principali vantaggi dell'uso di Arduino includono:

- **Compatibilità con Librerie IoT**: Arduino offre librerie specifiche per protocolli di comunicazione (seppur non complete) semplificando l'implementazione della comunicazione HTTP e HTTPS (HTTPS solo lato client).
- **Semplicità e Rapidità di Sviluppo**: Grazie alla vasta disponibilità di librerie, Arduino permette una programmazione rapida, riducendo i tempi di configurazione dei protocolli di rete.
- **Ampia Comunità e Supporto**: Arduino gode di una vasta comunità di sviluppatori, che forniscono documentazione, tutorial e soluzioni per problemi comuni, rendendo il processo di sviluppo più agevole.

3.2.2 ESP-IDF (Espressif IoT Development Framework)

L'*ESP-IDF*[3] è il framework ufficiale di Espressif per lo sviluppo avanzato su ESP32, che consente un controllo più approfondito dell'hardware e l'accesso a funzionalità avanzate. A differenza di Arduino, ESP-IDF richiede un ambiente di sviluppo più complesso, ma offre vantaggi significativi:

- **Controllo Completo della Configurazione di Rete**: Con ESP-IDF è possibile personalizzare le configurazioni di rete e di sicurezza, gestendo parametri come i certificati SSL, la configurazione della rete e l'autenticazione del client.
- **Accesso alle Funzioni Avanzate dell'ESP32**: ESP-IDF permette di configurare funzioni avanzate, come la gestione della crittografia hardware, l'autenticazione SSL/TLS e la gestione della memoria non volatile.
- **Ottimizzazione del Consumo Energetico**: Consente una gestione fine delle modalità di risparmio energetico, ottimizzando l'uso della batteria nei dispositivi IoT.

ESP-IDF è stato utilizzato per implementare funzioni di sicurezza avanzate, come la mutua autenticazione tra client e server, per ottimizzare l'uso della memoria e della CPU dell'ESP32 e per la compatibilità di librerie di comunicazione HTTPS avanzate (esp32_https).

3.3 PlatformIO

PlatformIO[4] è una moderna piattaforma di sviluppo integrata (IDE) e un ecosistema open-source per lo sviluppo di software embedded. PlatformIO offre un ambiente di sviluppo potente e flessibile, integrabile con Visual Studio Code. Questa piattaforma è particolarmente utile per lo sviluppo su microcontrollori grazie alle sue numerose funzionalità, tra cui gestione automatica delle librerie, supporto per debugger integrati, configurazione personalizzabile degli ambienti di sviluppo e un robusto sistema di build.

Le principali caratteristiche di PlatformIO che hanno supportato lo sviluppo del progetto includono:

- **Gestione delle Librerie**: PlatformIO consente la gestione automatica delle librerie, con un accesso diretto a una vasta libreria di pacchetti e moduli. Questa funzione ha permesso di importare facilmente le librerie necessarie per la configurazione della comunicazione e la gestione della sicurezza.
- **Supporto Multi-Framework**: PlatformIO supporta più framework e piattaforme hardware, incluso ESP-IDF (Espressif IoT Development Framework), utilizzato in questo progetto per configurare le funzionalità avanzate dell'ESP32.
- **Sistemi di Build e Integrazione Continua**: PlatformIO include un sistema di build automatizzato che gestisce dipendenze, compilazione e upload del firmware. Questo sistema ha semplificato lo sviluppo e il test, garantendo un flusso di lavoro continuo ed efficiente.
- **Debugging Avanzato**: La piattaforma offre strumenti di debugging avanzati, essenziali per monitorare le variabili, rilevare errori e ottimizzare il codice durante lo sviluppo. Nel contesto di questo progetto, il debugging ha facilitato l'identificazione di eventuali problemi di comunicazione o di sicurezza.

3.4 Firmware Marauder

Il *Firmware Marauder*[5] è un firmware open-source sviluppato per ESP32, mirato alla scansione, rilevazione e monitoraggio delle reti Wi-Fi e dei dispositivi connessi (oltre anche ad moduli legati al Bluetooth LE). Creato da "justcallmekoko", questo firmware fornisce funzionalità avanzate per il testing di sicurezza e il monitoraggio delle reti wireless, rendendolo uno strumento efficace per valutare vulnerabilità e mappare l'ambiente di rete.

Le sue principali funzionalità includono:

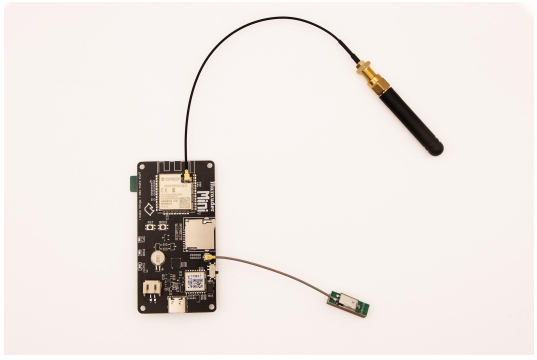
- **Full Active Commandline**: Marauder può, con connessione seriale attiva, permettere l'accesso ad una comoda console per avviare sniffing/attacchi non supportati tramite GUI come per esempio "sniffesp".
- **Scansione delle Reti Wi-Fi**: Marauder può eseguire una scansione dettagliata delle reti circostanti, rilevando SSID, BSSID, livelli di potenza del segnale e canali. Questa funzione è utile per l'identificazione di reti non autorizzate o potenziali vulnerabilità nell'ambiente di rete.
- **Rilevamento di Dispositivi**: Oltre alla semplice rilevazione delle reti, Marauder è in grado di identificare dispositivi connessi, mostrando informazioni come MAC address e tipi di dispositivo, rendendo possibile la costruzione di una mappa della rete.
- **Packet Injection**: Marauder può essere configurato per inviare pacchetti personalizzati all'interno della rete, permettendo di testare la risposta dei dispositivi a traffico anomalo o potenzialmente malevolo.
- **Monitoraggio Passivo**: Oltre alle funzioni di attacco, Marauder può essere utilizzato in modalità di monitoraggio passivo, intercettando e registrando il traffico di rete per un'analisi dettagliata senza interferire attivamente con le comunicazioni.

Nel progetto, Marauder è utilizzato per monitorare e raccogliere dati sul traffico tra client e server ESP32 successivamente analizzato tramite strumenti appositi (i.e Wireshark).

3.4.1 ESP32 Marauder Mini

L'*ESP32 Marauder Mini* rappresenta una versione compatta del firmware Marauder, focalizzata sul monitoraggio passivo e l'analisi preliminare della rete. Sebbene supporti solo alcune delle funzionalità avanzate di Marauder, questa versione permette una rapida raccolta di informazioni preliminari necessarie ad una prima analisi.

(a) Esterno



(b) Interno

Figura 3.2: ESP32 Marauder - Mini

3.4.2 Flipper Zero

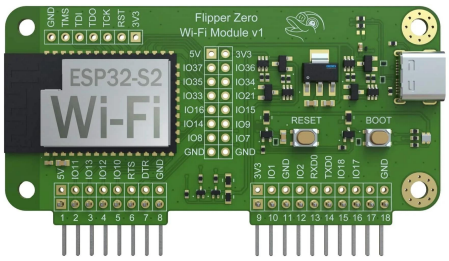
Il *Flipper Zero*[6] è un dispositivo versatile per test di sicurezza e analisi di reti, comunemente utilizzato per hacking etico e valutazione della sicurezza di ambienti IoT. Nel contesto di questo progetto, il firmware *ESP32 Marauder* è installato sulla Flipper Zero Dev Board, estendendo le capacità di monitoraggio e test di rete del dispositivo. Questa combinazione permette di sfruttare appieno le funzioni avanzate di Marauder insieme alla portabilità e alle funzionalità aggiuntive della Flipper Zero.

Le principali funzionalità della Flipper Zero con firmware ESP32 Marauder includono:

- **Funzioni di Penetration Testing**: Grazie al firmware ESP32 Marauder, il dispositivo dispone di strumenti di hacking etico per simulare tentativi di intrusione, deautenticazione e altri test di sicurezza. Ciò consente di valutare la robustezza delle reti Wi-Fi e identificare eventuali vulnerabilità.
- **Intercettazione e Scansione delle Reti Wi-Fi**: La Flipper Zero con firmware Marauder può effettuare scansioni Wi-Fi dettagliate, intercettando pacchetti di traffico specifici e raccogliendo dati su SSID, BSSID, livelli di potenza del segnale e canali. Questa capacità di monitoraggio migliora l'efficacia dell'analisi di rete in ambienti IoT.
- **Supporto Multifunzione**: Oltre alla scansione Wi-Fi, la Flipper Zero supporta altri protocolli come NFC, Bluetooth e Sub-GHz, consentendo di ampliare l'analisi e il monitoraggio su diversi dispositivi e protocolli IoT. Questa versatilità lo rende un dispositivo completo per l'analisi di sicurezza in contesti diversificati.

L'integrazione del firmware ESP32 Marauder sulla Flipper Zero Dev Board unisce le capacità avanzate di Marauder con le funzionalità versatili della Flipper Zero, trasformandola in uno strumento potente per l'analisi di rete e i test di sicurezza in ambienti IoT. Questa configurazione permette di simulare numerosi scenari di sicurezza, offrendo dati preziosi per valutare e rafforzare le difese della rete.

(a) Flipper Zero Tool



(b) Dev Board

Figura 3.3: Flipper Zero

3.5 Network Testing

Per valutare la sicurezza della comunicazione tra client e server, vengono utilizzati strumenti di sniffing e tecniche di analisi del traffico. Questi metodi consentono di intercettare e monitorare i pacchetti, verificando la vulnerabilità del traffico HTTP rispetto a quello HTTPS e simulando potenziali attacchi passivi o attivi (i.e replay attack).

3.5.1 Cos'è un Replay Attack?

Un **replay attack**, o *attacco di ripetizione*, è un attacco informatico in cui un attore malevolo intercetta e ritrasmette pacchetti di dati validi, ingannando il sistema di destinazione e facendogli credere che i dati siano stati inviati nuovamente in modo legittimo. Questo attacco sfrutta la mancanza di meccanismi per verificare l'unicità dei pacchetti, permettendo all'attaccante di replicare una comunicazione legittima e manipolare lo stato del sistema.

In pratica, un replay attack intercetta pacchetti autentici e li ritrasmette in seguito, eseguendo azioni non autorizzate come ripetere trasferimenti di fondi o attivare comandi remoti. Poiché i pacchetti replicati sembrano legittimi, il sistema non può distinguerli senza misure di sicurezza specifiche.

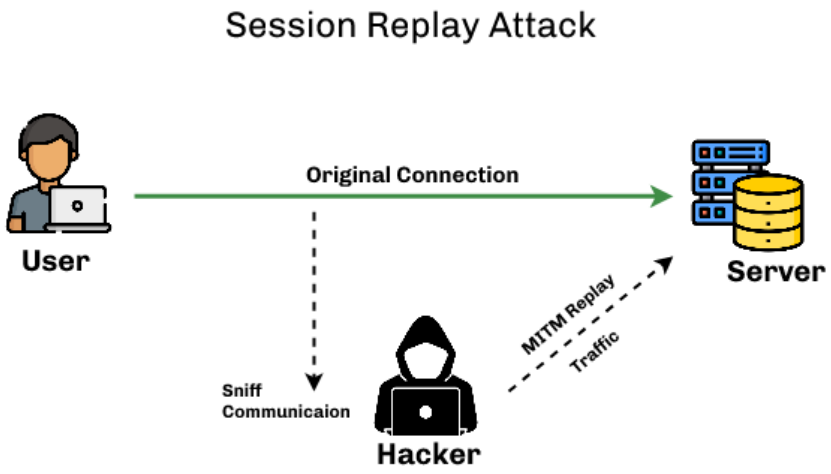


Figura 3.4: Session Repaly Attack

Perché un Replay Attack è Dannoso per i Dispositivi IoT

Un replay attack è particolarmente dannoso per i dispositivi IoT per diverse ragioni:

- **Mancanza di Risorse per la Sicurezza**: Molti dispositivi IoT hanno capacità di calcolo, memoria e consumo energetico limitati. Per questo motivo, spesso non sono dotati di protocolli di sicurezza avanzati (come TLS o HTTPS) e di misure anti-ritrasmissione, che richiederebbero risorse aggiuntive.

- **Interazione Automatica con l'Ambiente**: I dispositivi IoT sono spesso configurati per eseguire azioni basate su comandi ricevuti senza intervento umano. Se un attaccante riesce a intercettare e ritrasmettere pacchetti legittimi, può indurre il dispositivo a ripetere azioni critiche o indesiderate. Ad esempio, potrebbe riattivare una serratura smart, spegnere un sensore di sicurezza o riavviare un sistema critico.

- **Trasmissione di Dati Sensibili**: Molti dispositivi IoT trasmettono dati sensibili, come misurazioni, dati personali o informazioni mediche, attraverso reti potenzialmente insicure. Un replay attack può compromettere la riservatezza e l'integrità di questi dati, portando alla loro esposizione o manipolazione.

- **Impatto su Sistemi Critici**: In ambiti industriali o sanitari, dove i dispositivi IoT sono parte di infrastrutture critiche, un replay attack potrebbe avere gravi conseguenze sulla sicurezza e sull'efficacia del sistema. Ritrasmettere comandi obsoleti o non autorizzati potrebbe alterare la produzione, interrompere la fornitura di servizi essenziali o mettere in pericolo i pazienti.

- **Difficoltà di Rilevazione**: Poiché i pacchetti ritrasmessi in un replay attack sono copie esatte dei pacchetti legittimi, molti dispositivi IoT non sono in grado di rilevare che si tratta di pacchetti duplicati. Questo rende l'attacco difficile da individuare e da prevenire senza misure di sicurezza avanzate, come timestamp o numeri di sequenza unici.

Mitigazione dei Replay Attack

Per proteggere una comunicazione da attacchi di replay, molti protocolli di sicurezza come **HTTPS** e **TLS** adottano tecniche di autenticazione e cifratura che rendono ogni pacchetto univoco e impossibile da replicare con successo. Tra le tecniche comunemente utilizzate per mitigare questi attacchi ci sono:

- **Timestamp**: Ogni pacchetto viene contrassegnato con un timestamp che indica l'ora precisa di invio. Questo permette al destinatario di verificare che il pacchetto sia stato ricevuto entro un certo intervallo di tempo, scartando eventuali pacchetti ritrasmessi successivamente.

- **Numeri di sequenza univoci**: Ogni pacchetto include un numero di sequenza unico che viene incrementato ad ogni nuova richiesta. Se il sistema riceve un pacchetto con un numero di sequenza già visto, lo scarta automaticamente, prevenendo la possibilità di ritrasmissione.

- **Token di sessione**: Nelle connessioni autenticati, un token di sessione unico viene assegnato a ciascuna sessione. Questo token è valido solo per quella sessione specifica, impedendo l'uso di pacchetti intercettati per future comunicazioni.

Queste tecniche sono integrate in HTTPS, che garantisce una comunicazione sicura tramite crittografia e autenticazione, rendendo difficile per un attaccante intercettare e ritrasmettere pacchetti in modo efficace.

3.5.2 Firmware Marauder

In questo progetto è stato utilizzato il **Firmware Marauder** per condurre un'analisi approfondita del traffico di rete. Grazie all'*accesso alla commandline* tramite la **Flipper Zero Dev Board** e il supporto dell'**ESP32 Marauder Mini** in modalità *promiscua*, è stato possibile eseguire operazioni di **sniffing** sulla rete.

Queste configurazioni hanno permesso di:

- intercettare e monitorare il **traffico di rete** in tempo reale,
- catturare i pacchetti trasmessi,
- salvare i dati raccolti in formato `.pcap` per successive analisi con strumenti dedicati.

L'utilizzo combinato della Flipper Zero (con Dev Board) e del Firmware Marauder ha reso possibile la raccolta dettagliata di informazioni sulla rete, fornendo un ambiente versatile per l'**analisi di sicurezza** e il **monitoraggio passivo** delle comunicazioni IoT.

3.5.3 Proxy Server

Un ulteriore tecnica consiste nell'utilizzare un **proxy server** per monitorare il traffico tra client e server. In particolare, un **terzo ESP32** (o qualunque strumento IoT) è configurato come proxy server, posizionandosi tra la comunicazione *client-server* per fungere da intermediario.

In modalità **promiscua**, il proxy ESP32 è in grado di:

- intercettare tutto il **traffico in chiaro** trasmesso tra client e server,
- analizzare i pacchetti ricevuti e trasmessi,
- registrare e raccogliere informazioni utili su ogni comunicazione per successive analisi dettagliate.

A causa delle limitazioni di memoria dell'ESP32, i pacchetti sono raccolti in **piccoli buffer** e successivamente **ritrasmessi** dopo un breve *delay*. Questa tecnica di buffering permette di gestire in modo efficiente lo spazio di memoria limitato dell'ESP32, garantendo comunque la cattura e l'elaborazione del traffico di rete.

3.5.4 Reindirizzamento del Traffico

ARP Poisoning

La tecnica di **ARP Poisoning**, nota anche come *arpspoof*, è utilizzata per reindirizzare il traffico di rete verso una macchina controllata dall'attaccante, posizionata come **Man-in-the-Middle (MITM)** tra il client e il server. Attraverso l'ARP Poisoning, l'attaccante invia risposte ARP false per ingannare i dispositivi in rete, facendogli credere che il suo indirizzo MAC corrisponda all'indirizzo IP del gateway o del server.

Una volta stabilita questa posizione, l'attaccante può:

- reindirizzare tutto il **traffico di rete** verso la propria macchina, dove può essere analizzato;
- utilizzare strumenti di **sniffing** come *tcpdump*, *Wireshark*[7], o il tool **dsniff** su *Kali Linux* per visualizzare i pacchetti in transito e analizzare il contenuto del traffico intercettato.

L'uso di **dsniff**, disponibile su *Kali Linux*, facilita la raccolta e l'analisi di informazioni sensibili intercettate nelle comunicazioni, consentendo all'attaccante di monitorare traffico non crittografato e potenzialmente captare dati come credenziali, sessioni e altre informazioni trasmesse in chiaro.

AP Clone e Deauth Spam

Un'altra tecnica simile è l'attacco di **AP Clone** combinato con la **deautenticazione forzata**, eseguibile tramite dispositivi come il **Flipper Zero** o l'**ESP32 Marauder Mini**. In questo caso, l'attaccante invia pacchetti di deautenticazione al dispositivo target, costringendolo a disconnettersi dalla rete originale e a riconnettersi a una rete fasulla (clonata) controllata dall'attaccante.

Una volta che il dispositivo si connette alla rete fasulla, il traffico può essere **reindirizzato** e analizzato tramite strumenti di sniffing come *tcpdump*, *Wireshark*, o *dsniff*. Questa tecnica consente di monitorare il traffico in modo simile all'ARP Poisoning, ma con il vantaggio di forzare la connessione a una rete gestita direttamente dall'attaccante.

Capitolo 4

Implementazione

4.1 Generazione di Certificati e Segreti

Per garantire la sicurezza della comunicazione tra le due ESP32, viene utilizzata una serie di certificati generati tramite **OpenSSL**. Due script, `generate_certs.bash` e `generate_secret.bash`, automatizzano il processo di creazione dei certificati e del **segreto condiviso**.

In particolare:

- **generate_certs.bash**: Questo script genera i certificati necessari per il protocollo HTTPS utilizzando **OpenSSL**. I certificati creati includono:
  - **CA (Certificate Authority)**: il certificato della CA utilizzato per firmare e verificare gli altri certificati.
  - **Certificato Server**: specifico per l'ESP32 configurato come server, utilizzato per autenticare la connessione e crittografare i dati.
  - **Certificato Client**: utilizzato dall'ESP32 configurato come client per autenticarsi presso il server.

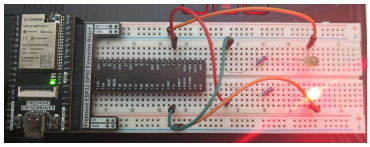
  Sia il certificato del server che quello del client sono accompagnati dalle relative chiavi private in formato `.pem`, necessarie per la crittografia e l'autenticazione reciproca.

- **generate_secret.bash**: Questo script crea un file denominato `secret.txt`, contenente un **segreto condiviso** tra il client e il server. Il segreto è utilizzato per **autenticare le richieste** e validare l'accesso ai servizi offerti dal server, garantendo un ulteriore livello di sicurezza.

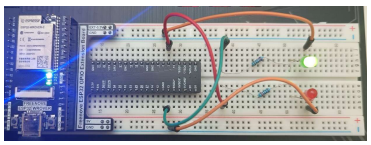
L'insieme di questi certificati viene salvato nel file system **SPIFFS** dell'ESP32 ed è essenziale per la **crittografia** e l'**autenticazione reciproca** tra client e server. L'utilizzo di questi certificati e di un segreto condiviso fornisce una protezione robusta contro tentativi di accesso non autorizzato e garantisce che i dati trasmessi tra client e server rimangano sicuri.

4.2 Comunicazione tra Client e Server

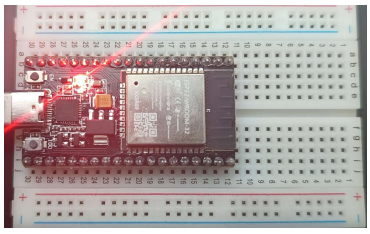
Dopo aver effettuato l'upload del file system **SPIFFS**, contenente i certificati e il file `secret.txt`, il **firmware** viene flashato su entrambe le **ESP32**. Una volta avviate, le due ESP32 si mettono in comunicazione tramite **HTTP** o **HTTPS**, a seconda del protocollo selezionato dall'utente al primo avvio. Il **client ESP32** tenta di connettersi al **server** ogni **10 secondi** per inviare il **segreto** e richiedere l'accesso al **servizio**.



(a) Server (fail state)



(b) Server (success state)

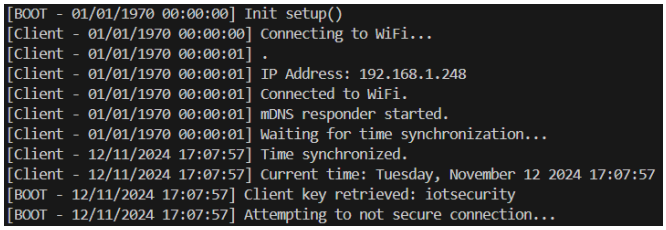


(c) Client

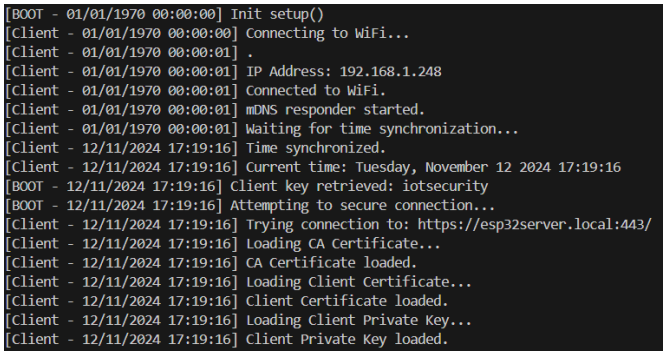
Figura 4.1: System

4.2.1 Client ESP32

Il **client** è programmato per inviare richieste ogni **10 secondi**, includendo nel payload il **segreto condiviso**. All'inizio della comunicazione, l'utente seleziona se il client dovrà utilizzare il protocollo **HTTP** o **HTTPS** per le connessioni successive. In **HTTPS**, il client stabilisce una connessione **crittografata e autenticata** con il server utilizzando i certificati generati, mentre in **HTTP** la connessione avviene senza *crittografia*.



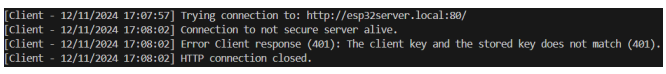
(a) Init Client HTTP



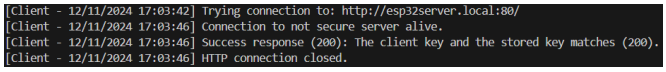
(b) Init Client HTTPS

Figura 4.2: Init Client

Il client riceve l'eventuale messaggio di **success** o **fail** e lo mostra tramite il logging, chiudendo l'eventuale connessione (se essa non finalizzata dal server:



(a) Fail



(b) Success

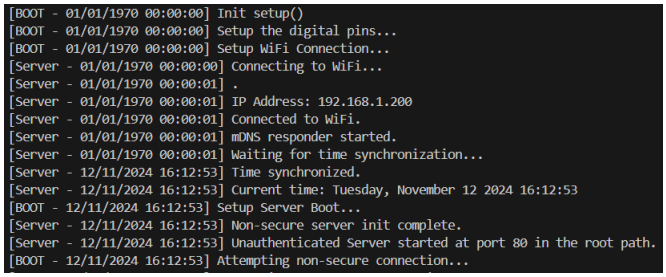
Figura 4.3: Client Response Handler

4.2.2 Server ESP32

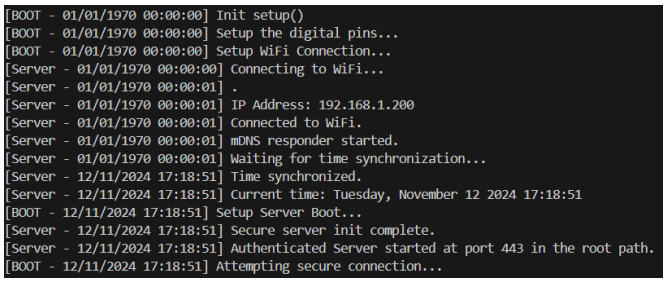
Il **server ESP32** riceve e processa le richieste del **client**, validando l'handshake **SSL/TLS** (in **HTTPS**) o accettando la richiesta direttamente (in **HTTP**). Dopo aver accettato la connessione, il server verifica che il **payload** della richiesta contenga il **segreto corretto**.

- **Accesso Consentito (Segreto Corretto)**: Se il segreto inviato dal client coincide con quello presente nel file `secret.txt`, il server concede l'accesso al servizio, attivando un LED verde.
- **Accesso Negato (Segreto Errato)**: Se il segreto è incorretto o la richiesta è sospetta, il server nega l'accesso al servizio, accendendo un LED rosso.

Il serverviene inizializzato in modalità **HTTP** o **HTTPS**, l'inizializzazione si differenzia dal caricare o no il **certificato SSL** del server (generato opportunamente).



(a) Init Server HTTP



(b) Init Server HTTPS

Figura 4.4: Init Server

Il server catturerà tutte le informazioni utili dalla richiesta cliente (se essa supera l'eventuale handshake TCP), le informazioni vengono catturate e poi si passa alla validazione:

```
[Metadata - 12/11/2024 16:13:18] ----- Request Metadata -----
[Metadata - 12/11/2024 16:13:18] Client IP: 192.168.1.248
[Metadata - 12/11/2024 16:13:18] Resource: /
[Metadata - 12/11/2024 16:13:18] Method: POST
[Metadata - 12/11/2024 16:13:18] ---
[Metadata - 12/11/2024 16:13:18] Headers:
[Metadata - 12/11/2024 16:13:18] --Host: esp32server.local
[Metadata - 12/11/2024 16:13:18] --User-Agent: Espressif ESP32/1.0
[Metadata - 12/11/2024 16:13:18] --Connection: keep-alive
[Metadata - 12/11/2024 16:13:18] --Accept-Encoding: identity;q=1,chunked;q=0.1,*;q=0
[Metadata - 12/11/2024 16:13:18] --Content-Type: text/plain
[Metadata - 12/11/2024 16:13:18] --Content-Length: 11
[Metadata - 12/11/2024 16:13:18] ---
[Metadata - 12/11/2024 16:13:18] Content-Length: 11
[Metadata - 12/11/2024 16:13:18] Body Content: iotsecurity
[Metadata - 12/11/2024 16:13:18] ----------------------------
```

Figura 4.5: Metadata Capture

In caso di server HTTPS, l'handshake fallimentare sarà opportunamente mostrato nel loggind di errore:

```
[HTTPS:I:     46634] New connection. Socket FID=49
[HTTPS:E:     47395] SSL_accept failed. Aborting handshake. FID=49
[Server - 12/11/2024 17:19:33] Aborting handshake, SSL_accept failed.
[HTTPS:I:     47397] Connection closed. Socket FID=49
[HTTPS:D:     47410] Free headers
```

Figura 4.6: Handshake Fail

Infine ecco il comportamento del server in caso di successo o fallimento:

```
[Server - 12/11/2024 16:26:28] Client key does not match.
```

(a) Fail

```
[Server - 12/11/2024 16:28:11] Client key matches.
```

(b) Success

Figura 4.7: Server Response Handler

La richiesta viene gestita opportunamente e la connessione chiusa subito dopo tramite la clausola *"close"*, evitando (se non necessaria) la clausola *"keep-alive"*.

4.3 Test di Sniffing e Sicurezza

Per verificare la protezione offerta dal protocollo **HTTPS** rispetto a **HTTP**, sono state utilizzate tre tecniche di sniffing e attacco per intercettare e replicare i pacchetti della comunicazione.

4.3.1 ESP32 Marauder come Sniffer per il Replay Attack

L'**ESP32 Marauder**, configurato con un firmware specializzato, è stato utilizzato come **sniffer** per intercettare i pacchetti scambiati tra client e server. Grazie alla sua modalità **commandline**, Marauder ha permesso di sfruttare comandi avanzati per effettuare scansioni mirate e catturare pacchetti di interesse.

Sono state utilizzate due modalità di scansione:

- **sniffesp**: Una modalità di scansione che utilizza un filtro specifico per riconoscere dispositivi basati su **ESP32** (Espressif) tramite il loro **MAC address**. Questo scan ha consentito di intercettare pacchetti provenienti esclusivamente dai dispositivi coinvolti nella comunicazione client-server, migliorando la precisione dell'analisi.
- **sniffraw**: Una modalità di scansione più "grezza", che cattura tutto il traffico in transito indipendentemente dall'origine o dalla destinazione. Questo approccio ha permesso di avere una visione completa del traffico nella rete, utile per identificare potenziali interferenze o pacchetti sospetti.

Una volta intercettati, i pacchetti sono stati salvati in formato standard `.pcap`, ideale per l'analisi successiva tramite strumenti come `Wireshark`. Il processo è stato suddiviso in diverse fasi chiave:

- **Intercettazione del Traffico**: Utilizzando Marauder, il traffico di rete tra client e server è stato monitorato in modalità promiscua, consentendo di catturare sia pacchetti HTTP che HTTPS.
- **Salvataggio dei Pacchetti**: I pacchetti intercettati sono stati memorizzati in file `.pcap` per una successiva analisi dettagliata.
- **Replay dei Pacchetti HTTP**: Con l'utilizzo di strumenti come `tcpreplay`, i pacchetti HTTP catturati sono stati **ritrasmessi al server** per testare la risposta del sistema.

4.3.2 Sniffing con Proxy Server su ESP32

La tecnica del **proxy server** è implementata utilizzando un'**ESP32** configurata per agire come intermediario tra il client e il server. Questa configurazione consente al proxy di monitorare e intercettare il traffico in transito, analizzando ogni pacchetto inviato o ricevuto. L'ESP32, grazie alla sua modalità **promiscua**, è in grado di acquisire i dati di rete senza interrompere il normale flusso della comunicazione.
Il processo di configurazione e funzionamento si articola nei seguenti passaggi:

- **Intercettazione del Traffico**: L'ESP32 configurata come proxy server cattura i pacchetti scambiati tra il client e il server. Essendo posizionata direttamente nel percorso di comunicazione, l'ESP32 può osservare tutte le richieste e le risposte, analizzando in particolare i messaggi HTTP.

- **Analisi del Traffico in Chiaro**: Per le comunicazioni in HTTP, il traffico intercettato viene elaborato e decodificato, consentendo di leggere il **payload** delle richieste e altre informazioni come intestazioni, URL e dati trasmessi. Questa operazione è particolarmente utile per visualizzare dati sensibili come il **segreto condiviso**.

- **Bufferizzazione e Ritrasmissione**: A causa delle limitazioni di memoria dell'ESP32, i pacchetti intercettati vengono temporaneamente memorizzati in piccoli buffer e successivamente ritrasmessi al server dopo un breve *delay*. Questa operazione consente di simulare un comportamento di replay, in cui il proxy replica una richiesta già intercettata per verificare il comportamento del server.

L'ESP32 configurata come proxy server è stata programmata per distinguere tra richieste HTTP e HTTPS:

- **Traffico HTTP**: Il proxy può leggere e analizzare il contenuto delle richieste, essendo il traffico trasmesso in chiaro. Questa configurazione permette di catturare dati come il **payload** e replicare facilmente i pacchetti verso il server.

- **Traffico HTTPS**: Sebbene il proxy possa intercettare i pacchetti HTTPS, il loro contenuto rimane **crittografato**. In questa modalità, l'ESP32 agisce esclusivamente come uno sniffer passivo, catturando i dati per un'eventuale analisi dei metadati (come indirizzi IP, porte e dimensioni dei pacchetti).

4.3.3 Reindirizzamento del Traffico

Il **reindirizzamento del traffico** rappresenta una tecnica fondamentale per posizionare un attaccante come **Man-in-the-Middle (MITM)** e intercettare le comunicazioni tra client e server.

ARP Poisoning

La tecnica di **ARP Poisoning**, o *arp spoofing*, sfrutta il protocollo **Address Resolution Protocol (ARP)** per manipolare le tabelle ARP dei dispositivi in una rete locale. Attraverso l'invio di pacchetti ARP falsificati, l'attaccante induce i dispositivi a credere che il proprio indirizzo MAC corrisponda all'indirizzo IP del gateway o del server. Questo permette all'attaccante di reindirizzare il traffico di rete verso la propria macchina, agendo come intermediario.

Passaggi della Tecnica:

- **Falsificazione ARP**: L'attaccante invia risposte ARP false a uno o più dispositivi, alterando la mappatura **IP-MAC** nelle tabelle ARP.
- **Intercettazione del Traffico**: Una volta stabilita la posizione MITM, tutto il traffico di rete viene reindirizzato attraverso la macchina dell'attaccante, che può analizzarlo o modificarlo.
- **Strumenti Utilizzati**: Per l'esecuzione dell'attacco sono stati impiegati strumenti come `arpspoof` della suite **dsniff** e analizzatori di rete come `tcpdump` e `Wireshark`.

L'utilizzo di `dsniff` si è rivelato particolarmente efficace per semplificare il processo di raccolta e analisi dei pacchetti, evidenziando la vulnerabilità delle comunicazioni non crittografate in ambienti **IoT**.

AP Clone e Deauth Spam

L'attacco di **AP Clone** combinato con pacchetti di **deautenticazione forzata**, eseguibile tramite dispositivi come il **Flipper Zero** o l'**ESP32 Marauder Mini**. Questa metodologia consente di creare una rete Wi-Fi fasulla, inducendo i dispositivi target a disconnettersi dall'access point legittimo e a connettersi all'AP clonato.

Fasi dell'Attacco:

- **Scansione dell'Access Point**: L'attaccante identifica le reti disponibili, acquisendo informazioni su **SSID**, **BSSID** e canali utilizzati.
- **Creazione dell'AP Clone**: Un access point fasullo viene configurato con lo stesso **SSID** e canale dell'AP originale, inducendo il dispositivo target a connettersi alla rete clonata.
- **Deautenticazione Forzata**: Vengono inviati pacchetti di **deautenticazione** al dispositivo target, costringendolo a disconnettersi dalla rete originale.
- **Reindirizzamento del Traffico**: Una volta connesso, il traffico del dispositivo target passa attraverso l'AP clonato, dove può essere intercettato e analizzato.

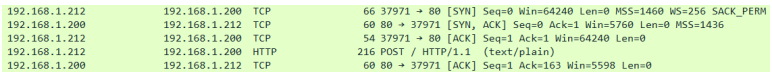
Questa tecnica si distingue dall'ARP Poisoning per la capacità di manipolare attivamente le connessioni dei dispositivi target, forzandoli a spostarsi su una rete controllata dall'attaccante. L'analisi del traffico reindirizzato evidenzia i rischi associati a reti non sicure e sottolinea l'importanza di adottare protocolli crittografati come **HTTPS**.

4.4 Risultati dei Test

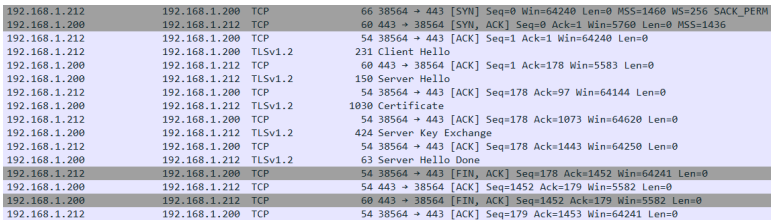
I test hanno confermato la vulnerabilità della comunicazione HTTP agli attacchi di intercettazione e replay, evidenziando al contempo la protezione fornita dal protocollo HTTPS. In particolare:

- Le connessioni **HTTP** sono risultate vulnerabili ai **tentativi di replay**. Ogni pacchetto HTTP intercettato e ritrasmesso è stato accettato come valido dal server, dimostrando come una connessione non sicura possa essere facilmente compromessa.
- Le connessioni **HTTPS**, al contrario, si sono dimostrate **immuni** a tali attacchi, grazie ai meccanismi di crittografia e autenticazione forniti dal protocollo **SSL/TLS**. Anche ritrasmettendo i pacchetti catturati, il server ha rifiutato le richieste, garantendo così la sicurezza della comunicazione.

I frame intercettati appaiono come seguono:



(a) HTTP Frame



(b) HTTPS Frame

Figura 4.8: Network Traffic Frames

In particolare per la connessione non sicura (HTTP):

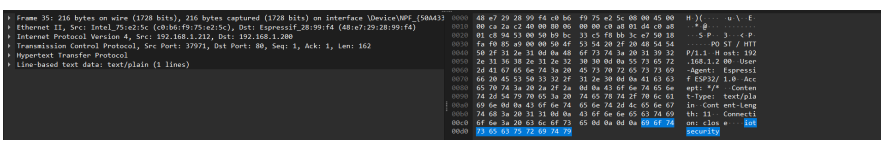


Figura 4.9: Frame Details

Capitolo 5

Conclusioni

In questo progetto è stato sviluppato e testato un sistema di comunicazione sicuro tra due dispositivi **ESP32**, con l'obiettivo di dimostrare l'importanza della scelta del **protocollo** per proteggere la trasmissione dei dati nelle reti **IoT**. Sono state eseguite diverse prove per valutare la vulnerabilità a specifici tipi di attacchi, come l'**attacco di replay**, dimostrando come questo sia realizzabile con il protocollo **HTTP** ma sostanzialmente infattibile con **HTTPS**.

L'implementazione del sistema ha consentito di:

- **Evidenziare i limiti di sicurezza** del protocollo HTTP in contesti IoT, specialmente per la protezione dei dati sensibili e l'integrità della comunicazione.
- **Dimostrare i vantaggi della crittografia in HTTPS**, che protegge le comunicazioni da attacchi di intercettazione e manipolazione dei pacchetti.
- **Sviluppare competenze pratiche nell'uso di strumenti di sniffing** per la cattura e l'analisi del traffico, mostrando come gli attacchi possano essere rilevati e mitigati con tecnologie avanzate.

I risultati ottenuti supportano la tesi che l'adozione di **protocolli sicuri** sia essenziale per applicazioni IoT che trattano **dati sensibili** e operano in ambienti potenzialmente vulnerabili a minacce esterne.

5.1 Sviluppi Futuri

Nonostante il progetto abbia raggiunto i suoi obiettivi principali, esistono diverse direzioni in cui potrebbe essere ampliato. Alcuni potenziali sviluppi includono:

- **Integrazione di Ulteriori Protocolli di Sicurezza**: Esplorare protocolli di sicurezza aggiuntivi, come *TLS* avanzato o *DTLS*, per migliorare la protezione in ambienti IoT e testare la resilienza contro attacchi più sofisticati. Inoltre, l'aggiunta di un sistema di *crittografia lightweight* o di opzioni sicure per la comunicazione, come il protocollo *MQTT*, potrebbe aumentare l'efficacia e la sicurezza.

- **Automazione dell'Analisi del Traffico**: Sviluppare script o tool che possano automaticamente rilevare comportamenti sospetti nel traffico di rete, migliorando la capacità di risposta a tentativi di attacco e riducendo il carico di monitoraggio manuale.

- **Implementazione di Tecniche di Intrusion Detection**: Integrare un sistema di rilevamento delle intrusioni (*IDS*) basato su **analisi comportamentale** o modelli di *machine learning* per identificare in tempo reale attività anomale o potenziali attacchi di replay.

- **Ottimizzazione del Consumo Energetico**: Considerare l'efficienza energetica del sistema, dato che i dispositivi IoT spesso operano con risorse limitate. Ottimizzare il consumo di energia durante la comunicazione sicura potrebbe migliorare l'adozione di questi protocolli anche su dispositivi alimentati a batteria.

Questi sviluppi futuri non solo migliorerebbero la sicurezza e l'efficienza di un generico sistema IoT, ma amplificherebbero anche la sua applicabilità a scenari più complessi e reali, offrendo una maggiore protezione per i **dati** e garantendo prestazioni affidabili.

Riferimenti

[1] Espressif Systems, *Espressif Systems: Wi-Fi and Bluetooth SoCs and Modules*, Manufacturer of ESP32 and other IoT-focused microcontrollers., 2008. indirizzo: `https://www.espressif.com/`.

[2] Massimo Banzi and David Cuartielles, *Arduino: An Open-Source Electronics Platform*, Documentation and resources for the Arduino platform., 2005. indirizzo: `https://www.arduino.cc`.

[3] Espressif Systems, *ESP-IDF: Espressif IoT Development Framework*, Official development framework for the ESP32 series., 2021. indirizzo: `https://docs.espressif.com/projects/esp-idf/en/latest/`.

[4] I. Kravets, *PlatformIO: Open-Source Ecosystem for IoT Development*, Documentation and resources for the PlatformIO ecosystem., 2014. indirizzo: `https://platformio.org`.

[5] justcallmekoko, *ESP32 Marauder Firmware*, Open-source firmware for Wi-Fi security analysis on ESP32., 2020. indirizzo: `https://github.com/justcallmekoko/ESP32Marauder`.

[6] Flipper Devices Inc., *Flipper Zero: Multitool Device for Hackers and Geeks*, Versatile open-source device for wireless communication analysis., 2020. indirizzo: `https://flipperzero.one/`.

[7] Gerald Combs, *Wireshark: Network Protocol Analyzer*, Open-source tool for capturing and analyzing network traffic., 1998. indirizzo: `https://www.wireshark.org/`.