

**ATTIVITA' PROGETTUALE IN SICUREZZA DELL'INFORMAZIONE M:  
ESTRAZIONE E ANALISI STATICA DEL FIRMWARE IN DISPOSITIVI IoT**

Candidato: Luca Bongiovanni  
n. Matricola: 0000980192

# SOMMARIO

---

Sommario .....	2
1 Introduzione .....	4
1.1 Cos'è il Firmware .....	4
1.2 La sicurezza del Firmware .....	4
1.3 L'analisi del Firmware.....	4
1.3.1 Analisi Statica.....	5
1.3.2 Analisi Dinamica .....	5
2 Stato dell'Arte .....	6
3 Descrizione del progetto .....	7
3.1 La raccolta di informazioni .....	7
3.2 L'acquisizione del Firmware .....	7
3.2.1 Metodi software.....	8
3.2.2 Metodi hardware .....	8
3.2.2.1 Universal Asynchronous Receiver-Transmitter (UART) .....	8
3.2.2.2 JTAG .....	9
3.2.2.3 Estrazione diretta dalla memoria.....	10
3.3 L'analisi statica.....	12
3.3.1 L'ispezione del firmware.....	12
3.3.2 La cifratura del firmware .....	14
3.3.2.1 Identificare la cifratura .....	14
3.3.2.2 Distinguere la cifratura dalla compressione.....	15
3.3.2.3 I diversi scenari di cifratura .....	16
3.3.3 I firmware semplici.....	18
3.3.4 I firmware complessi .....	19
4 Implementazione .....	21
4.1 Alice Gate 2 Plus .....	21
4.1.1 Raccolta delle informazioni.....	21
4.1.2 Individuazione dell'interfaccia UART.....	22
4.1.2.1 Le misurazioni col multimetro .....	23
4.1.2.2 La verifica con l'oscilloscopio .....	24

4.1.3	Estrazione del firmware .....	24
4.1.3.1	Connettersi all'interfaccia UART .....	24
4.1.3.2	Analizzare e intercettare la sequenza di boot .....	25
4.1.3.3	Estrazione tramite il bootloader.....	27
4.1.4	Analisi del firmware.....	30
4.2	Fastweb Home Access Gateway .....	35
4.2.1	Raccolta delle informazioni.....	35
4.2.2	Analisi del bootloader.....	36
4.2.3	Identificazione dell'interfaccia JTAG.....	37
4.2.4	Connessione all'interfaccia JTAG.....	38
5	Le contromisure da adottare.....	39
5.1	Proteggersi dall'estrazione del firmware .....	39
5.1.1	Aggiornamenti remoti sicuri.....	39
5.1.2	Disabilitare le interfacce di debug.....	40
5.2	Secure boot.....	40
5.3	Cifratura del firmware .....	41
6	Conclusioni .....	43

# 1 INTRODUZIONE

---

## 1.1 COS'È IL FIRMWARE

Il firmware è un programma memorizzato sul dispositivo per il quale è stato specificatamente creato, in modo che possa svolgere le sue attività. Sono firmware sia software per dispositivi semplici, dove vi è un unico programma che si interfaccia direttamente con l'hardware sottostante, sia per dispositivi più complessi costituiti da un vero e proprio sistema operativo e filesystem. È quindi evidente che i firmware si trovino in una vasta gamma di dispositivi elettronici: da dispositivi semplici, come per esempio una sveglia, a dispositivi più complessi, come per esempio un router. In questa attività progettuale si è concentrata l'attenzione sui dispositivi relativi all'ambito dell'*Internet of Things* (IoT).

## 1.2 LA SICUREZZA DEL FIRMWARE

Con l'avanzare della tecnologia il numero di dispositivi "intelligenti" che ci circonda è senza dubbio cresciuto esponenzialmente [1] e con esso anche l'importanza dei ruoli che questi dispositivi ricoprono. Tali dispositivi, infatti, possono contenere o condividere informazioni confidenziali (ad esempio il video di una smart camera), interagire con sistemi di sicurezza "hardware" (ad esempio un campanello smart), e trovarsi spesso in posizioni privilegiate per un potenziale attacco (poiché connessi ad una rete locale).

Consapevoli di questo, negli ultimi anni si è incominciato uno studio su come migliorare la sicurezza dei firmware arrivando infine a delineare delle linee guida standard per lo sviluppo e testing di dispositivi IoT.

## 1.3 L'ANALISI DEL FIRMWARE

Il processo con cui si verifica la presenza di eventuali vulnerabilità all'interno del firmware è detto analisi del firmware, tale analisi può essere statica o dinamica. Queste due tipologie di analisi condividono come punto di partenza l'ottenimento del firmware (che può avvenire in diverse maniere) per poi svolgere l'analisi a due livelli differenti ed infine raggiungere l'obiettivo condiviso di individuare possibili vulnerabilità. Le due tipologie di analisi forniscono quindi due "viste" dello stesso firmware, e ciò molto spesso porta ad effettuare un'analisi ibrida in cui le informazioni, anche apparentemente inutili, ricavate da un'analisi vengono poi utilizzate come punto di partenza nell'altra.

Ai fini dell'analisi è importante distinguere i firmware in due tipologie: d'ora in avanti chiameremo *firmware complessi* quei firmware che contengono un sistema operativo e un filesystem, mentre *firmware semplici* quelli composti da un unico file binario eseguibile.

### **1.3.1 Analisi Statica**

L'analisi statica consiste nell'analizzare il comportamento del firmware ispezionando i codici sorgenti e gli eseguibili in esso contenuti. Nel caso di firmware semplici si tratterà quindi di analizzare il codice binario in essi contenuto tramite l'ausilio di disassemblatori e decompilatori, mentre nel caso di firmware complessi si tratterà di esplorare la struttura del filesystem ed analizzare i programmi che contiene. Se da un lato questo processo risulta facilmente automatizzabile, dall'altro non è in grado di individuare tutte le vulnerabilità scoperte da un'analisi dinamica.

### **1.3.2 Analisi Dinamica**

L'analisi dinamica consiste nel testare il firmware mentre è operativo, testandone le funzionalità e la loro sicurezza. Ciò è possibile farlo emulando il firmware grazie a degli appositi strumenti per poi analizzare come esso si comporta se sollecitato in diversi modi. Pur essendo questo tipo di analisi più complicato da predisporre e da automatizzare permette però di testare direttamente varie vulnerabilità che invece in fase statica sarebbero solo ipotizzabili.

## 2 STATO DELL'ARTE

---

Una volta compresa l'importanza della sicurezza dei vari firmware, specialmente nel campo IoT, si è reso necessario capire quindi quali fossero le strategie da seguire in fase di progettazione e di testing per evitare di inserire involontariamente delle vulnerabilità. Un forte contributo è stato dato dalla fondazione *Open Web Application Security Project* (OWASP) [2], un'organizzazione no profit con l'obiettivo di migliorare la sicurezza del software.

Tra i progetti dell'OWASP è presente il progetto relativo all'ambito IoT [3] che è stato concepito al fine di aiutare produttori, sviluppatori e consumatori ad avere una maggiore consapevolezza dei problemi di sicurezza relativi all'ambito IoT. In particolare, l'*OWASP Firmware Security Testing Methodology* [4] stila un procedimento, composto di nove passaggi, atto ad eseguire correttamente un test della sicurezza di un firmware. Questi nove passaggi comprendono l'iniziale ottenimento del firmware e la sua successiva analisi, sia statica che dinamica.

Inoltre, l'OWASP mette a disposizione attraverso il progetto *IoTGoat* [5] un firmware deliberatamente insicuro al fine di educare gli sviluppatori nel testare le vulnerabilità più comuni nei dispositivi IoT. Questo viene realizzato mediante una serie di sfide alla ricerca di vulnerabilità con livelli di difficoltà diversi. Questi livelli di difficoltà rispecchiano i dieci livelli della *OWASP IoT Top 10* [6] dove sono elencate le vulnerabilità più comuni riscontrabili all'interno dei dispositivi IoT.

## 3 DESCRIZIONE DEL PROGETTO

---

In questa attività progettuale si concentrerà l'attenzione sulle prime fasi della metodologia fornita dall'OWASP, in particolare, a partire dalla raccolta di informazioni relative ai dispositivi target fino ad arrivare alla fase di analisi dei contenuti del filesystem.

### 3.1 LA RACCOLTA DI INFORMAZIONI

L'obiettivo della prima fase è quello di raccogliere più informazioni possibili sul dispositivo scelto come target. Tra le informazioni che si tenta di recuperare vi sono:

- Informazioni sul produttore (sito web, ecc.)
- Architettura della CPU
- Sistema operativo
- Schemi hardware e datasheet
- Changelogs
- Penetration test eseguiti precedentemente

Oltre a questo, se si possiede il dispositivo target lo si può aprire per tentare di recuperare informazioni analizzando i circuiti stampati. Questo non è sempre possibile a causa delle dimensioni e della complessità di certi dispositivi ed anche a causa di possibili contromisure messe in atto dai produttori per ostacolare l'apertura (uso di viti speciali, uso della colla al posto di viti, interruttori anti-manomissione, potting<sup>1</sup> dei componenti, ecc.).

L'ottenimento di queste informazioni è importante in quanto potrebbe fare la differenza nelle fasi successive.

### 3.2 L'ACQUISIZIONE DEL FIRMWARE

La fase di acquisizione del firmware è, evidentemente, una fase cruciale dato che senza il firmware stesso non si può intraprendere nessun tipo di analisi. L'acquisizione può risultare molto semplice, per esempio nel caso in cui il produttore rilasci un aggiornamento del firmware su una pagina web, oppure molto complessa richiedendo una copia del dispositivo target e l'uso di determinate attrezzature.

L'acquisizione del firmware non è una scienza esatta, per cui bisognerà adattarsi alle caratteristiche del dispositivo target e selezionare la metodologia più adatta. I vari metodi possono essere divisi nelle due macro-tipologie software e hardware.

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Potting\\_\(electronics\)](https://en.wikipedia.org/wiki/Potting_(electronics))

### 3.2.1 Metodi software

I metodi software di acquisizione del firmware non richiedono necessariamente un accesso fisico al dispositivo e spesso si basano sull'intercettare gli aggiornamenti *Over The Air* (OTA) effettuati da un dispositivo. Dei possibili esempi sono:

- Controllare se nel sito web del produttore gli aggiornamenti del firmware sono disponibili pubblicamente
- Analizzare il traffico di rete del dispositivo alla ricerca di un possibile indirizzo per il download del firmware
- Se la connessione utilizzata per il download fa uso di TLS tentare un attacco man-in-the-middle usando certificati auto-firmati per decrittare il traffico

Questi metodi però non sempre sono attuabili, infatti, spesso i produttori invece di rilasciare l'intero aggiornamento forniscono solo un *diff file* contenente solo i file modificati, rendendo perciò l'immagine del firmware incompleta. Un altro motivo per cui questi metodi non siano sempre attuabili è dovuto al fatto che spesso i produttori impacchettano e/o cifrano il firmware e le relative chiavi vengono salvate in maniera sicura all'interno dei dispositivi.

### 3.2.2 Metodi hardware

I metodi hardware di acquisizione richiedono un accesso fisico al dispositivo e particolari attrezzature che dovranno essergli collegate per effettuare l'estrazione.

Spesso il metodo più semplice per ottenere l'accesso al firmware di un certo dispositivo è attraverso le interfacce di debug utilizzate in precedenza durante la fase di sviluppo.

#### 3.2.2.1 Universal Asynchronous Receiver-Transmitter (UART)

Si è dimostrato che l'uso dell'interfaccia UART nell'oltre il 45% dei casi è sufficiente a realizzare un'estrazione del firmware [7]. Infatti, attraverso questo tipo di comunicazione seriale si ottiene spesso una shell con i permessi di amministratore che permette di eseguire diversi comandi per effettuare un'estrazione dell'intero filesystem, impacchettando e/o comprimendo tutti i file in un archivio oppure estraendo i vari blocchi di memoria in sequenza.

Come già detto in precedenza, non esiste un unico modo per effettuare queste estrazioni, perciò si possono seguire delle linee guida generali:

1. Identificare l'interfaccia UART grazie alle informazioni raccolte precedentemente oppure, generalmente compiendo molti tentativi, mediante un'ispezione visiva o con l'aiuto di attrezzature quali multimetro e/o oscilloscopio (o in un alternativa logic analyzer).

Trattandosi di una comunicazione seriale asincrona, l'interfaccia UART è composta principalmente dai pin di trasmissione e ricezione (TX e RX) e opzionalmente vi possono essere affiancati i pin di alimentazione (non sempre disponibili in quanto possono essere "recuperati" in altri punti della scheda)



2. Se collegandosi all'interfaccia si ottiene una shell non protetta, allora si può procedere ad estrarre il firmware scaricando i file su un computer connesso alla stessa rete utilizzando comandi come *netcat* o simili
3. Se invece la shell richiede l'inserimento di credenziali si può innanzi tutto provare ad inserire le coppie username/password più comuni. Se ciò non bastasse, si procede tentando di interrompere il processo di boot per entrare nella shell del bootloader. Se non si riesce ad interrompere il processo di boot si può tentare di "disturbare" l'interfaccia della memoria flash (per esempio collegando a massa il pin data o clock) durante il processo di caricamento del kernel in modo da costringere il bootloader a caricare la propria shell
4. Estrarre il firmware dalla shell del bootloader utilizzando i comandi disponibili che permettono la visualizzazione di aree della memoria.

### 3.2.2.2 JTAG

L'interfaccia JTAG viene utilizzata per semplificare i test una volta conclusa la fase di sviluppo: permette di impostare e controllare il valore di ciascun bit di ogni circuito integrato, queste due abilità sono dette rispettivamente controllabilità e osservabilità. Come risultato si ha che l'interfaccia JTAG permette di leggere e scrivere il contenuto della memoria flash e di eseguire una "in-circuit emulation"<sup>2</sup>. Spesso questa interfaccia viene lasciata erroneamente sbloccata anche quando ormai il dispositivo è in vendita sul mercato, ciò permette di effettuare l'estrazione dell'intero firmware e attacchi di firmware injection. Le linee guida generali per eseguire l'estrazione del firmware mediante JTAG sono:

1. Identificare l'interfaccia grazie alle informazioni raccolte precedentemente oppure visivamente scegliendo dei pin candidati e testandoli con l'ausilio di un multimetro. L'interfaccia JTAG non ha un connettore standard ufficiale per cui può avere i pin in varie disposizioni e, sebbene i pin necessari siano solamente 4/5 (uno è opzionale) si possono avere connettori con un numero di pin che va da 8 a 20<sup>3</sup>. Come regola generale, due righe di quattro o più pin sono candidate ad essere un'interfaccia JTAG
2. Se ciò non fosse sufficiente si potrebbe reperire il datasheet del microcontrollore utilizzato per cercare di identificare i pin seguendo le tracce sulla scheda. Risulta però spesso infattibile poiché spesso i *System-on-a-chip* (SoC) hanno i pin al di sotto della loro superficie
3. Se non fosse disponibile alcun datasheet si possono utilizzare particolari schede, come la JTAGulator [8], che, collegate ai pin candidati, eseguono una serie di test per identificare automaticamente i pin dell'interfaccia
4. Una volta identificati tutti i pin si può procedere a collegare il programmatore che farà da tramite tra la scheda ed un computer per effettuare l'estrazione del firmware se non è abilitata nessun tipo di protezione

---

<sup>2</sup> [https://en.wikipedia.org/wiki/In-circuit\\_emulation](https://en.wikipedia.org/wiki/In-circuit_emulation)

<sup>3</sup> Lista dei pin layout più diffusi: <http://www.jtagtest.com/pinouts/>

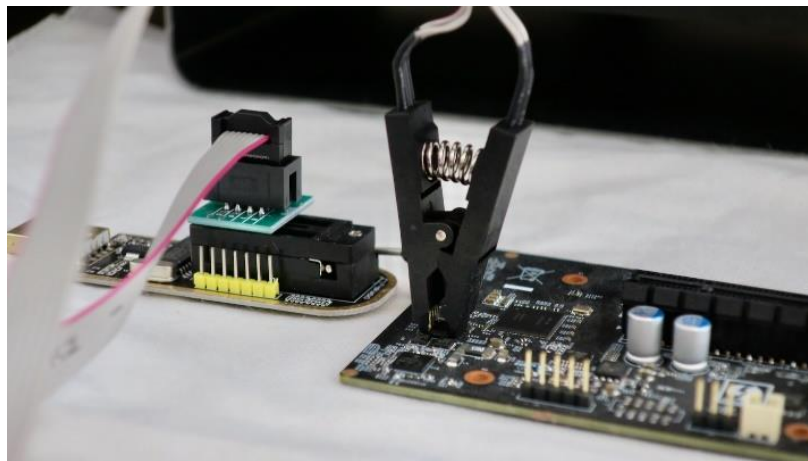
Lo stesso procedimento può essere effettuato anche per altre interfacce di debug proprietarie come, ad esempio, l'interfaccia *Serial Wire Debug* (SWD) utilizzata nei processori ARM.

Come si può notare, a causa delle possibili configurazioni differenti dei pin e all'uso di debugger differenti in base al tipo di architettura, l'estrazione attraverso l'interfaccia JTAG richiede un maggiore sforzo rispetto all'interfaccia UART.

### **3.2.2.3 Estrazione diretta dalla memoria**

Se le interfacce precedentemente descritte non fossero presenti o risultassero bloccate, un ulteriore modo di estrarre il firmware è farlo direttamente dalla memoria flash. Per farlo sono necessari connettori per collegarsi direttamente alla memoria e dispositivi di programmazione in grado di comunicare con essa mediante interfacce come la *Serial Peripheral Interface* (SPI) e l'*Inter-Integrated Circuit* (I<sup>2</sup>C) (Figura 3.1). In generale bisogna:

1. Identificare il chip della memoria flash (leggendo il nome del chip) e recuperare il rispettivo datasheet
2. Identificare i pin con il datasheet precedentemente recuperato o con l'ausilio di un oscilloscopio
3. Utilizzare l'adattatore e il programmatore più adatto per interfacciarsi con la memoria
4. Se l'estrazione direttamente dalla scheda non è possibile è necessario dissaldare la memoria ed effettuare l'estrazione separatamente



*Figura 3.1 - Connettore a clip per connettersi alla memoria collegato al programmatore per leggerne il contenuto*

L'estrazione con la memoria ancora saldata sulla scheda non è sempre possibile per varie motivazioni:

- Il chip della memoria potrebbe risultare troppo complesso per potersi collegare direttamente con un connettore
- Dovendo alimentare la memoria per estrarre i dati, si potrebbe di conseguenza alimentare anche la CPU che interferirebbe con la lettura. In questo caso se si vuole evitare la dissaldatura si può verificare se è possibile interrompere il segnale di clock della CPU per poi ripristinarlo una volta finita l'estrazione oppure tentare di resettare la CPU costantemente mediante il rispettivo pin.

È bene notare inoltre che dissaldando la memoria si rischia di danneggiarla irrimediabilmente a causa delle alte temperature della pistola ad aria calda e che una volta dissaldata sarà quasi impossibile risaldarla al circuito originale.

### 3.3 L'ANALISI STATICA

Una volta entrati in possesso del firmware con una delle tecniche descritte in precedenza si può intraprendere l'analisi statica in cerca di vulnerabilità nel software e informazioni confidenziali. Alcuni esempi di informazioni che si possono trovare all'interno del firmware sono:

- Chiavi private RSA e i rispettivi certificati auto-firmati
- Hash di password scritte direttamente all'interno dei programmi (spesso molto semplici da attaccare)
- Possibili backdoors, come per esempio il file `authorized_keys` che elenca le chiavi SSH che sono autorizzate a connettersi remotamente al dispositivo
- Vulnerabilità nella logica nei programmi dei servizi offerti dal dispositivo o nelle librerie utilizzate (le quali potrebbero essere non aggiornate)

Seppur soggetti ad analisi diverse, sia i firmware semplici che quelli complessi condividono, all'inizio dell'analisi, l'ispezione del firmware e il controllo sulla presenza o meno di cifratura.

#### 3.3.1 L'ispezione del firmware

Elencheremo di seguito una serie di strumenti ampiamente utilizzati (eseguibili da linea di comando) come supporto per l'ispezione del firmware. [9]

- `hexdump`  
Permette di mostrare il contenuto di un file in esadecimale, decimale, ottale, o ASCII. Risulta essere molto utile per verificare se esistono porzioni del file riconducibili ad una struttura familiare
- `strings`  
Mostra tutte le sequenze di caratteri stampabili di un file. Restrungendo il campo solo ai caratteri stampabili, permette un'immediata visualizzazione di informazioni potenzialmente importanti quali: credenziali, indirizzi web, ecc.  
Particolarmente utile l'opzione `-n` che permette di restringere ulteriormente il campo di ricerca impostando la lunghezza minima delle stringhe.
- `file`  
Permette di determinare il tipo di file eseguendo tre tipi di test: *filesystem tests*, *magic tests*, e *language tests*. In particolare, i *magic tests* sono basati sull'uso dei cosiddetti *magic numbers*<sup>4</sup>, ovvero, delle costanti numeriche utilizzate per identificare il formato di un file o di un protocollo.

---

<sup>4</sup> [https://en.wikipedia.org/wiki/List\\_of\\_file\\_signatures](https://en.wikipedia.org/wiki/List_of_file_signatures)

- `binwalk`

Binwalk [10], sfruttando gli stessi principi con cui `file` determina la tipologia di un file, fornisce uno strumento completo di ispezione dell'intero firmware, evidenziando la posizione esatta nel firmware di: stringhe, certificati, chiavi private, filesystem e molto altro. In particolare, con l'opzione `-e` permette l'estrazione automatica dei filesystem identificati nei firmware complessi, rendendo la procedura iniziale semplice ed immediata. Mentre, se si sospetta che il firmware sia semplice, con l'opzione `-A` si possono identificare i codici operandi di vari set di istruzioni.

Di contro, è noto che, a causa della sua natura, binwalk produca spesso falsi positivi; perciò, occorre che l'analista verifichi che la struttura prodotta da binwalk rispecchi la vera struttura del firmware.

- `dd`

Permette di convertire o copiare un file o sue porzioni e può risultare molto utile per ovviare al problema dei falsi positivi di `binwalk`. Spesso, infatti, vengono usati in coppia in modo che `dd` estragga porzioni del firmware da far esaminare successivamente a binwalk affinché si riducano la produzione di falsi positivi.

- Visualizzatore binario (`binvis.io`)

Offre la possibilità di visualizzare un intero file binario come un'immagine, evidenziando con diversi colori le tipologie di sezioni da cui è composto un file. Il motivo dell'esistenza di uno strumento del genere risiede nel fatto che, seppur esistano strumenti che permettono la visualizzazione di porzioni di binario (`hexdump`), nessuno di questi permette una visualizzazione immediata dell'intera struttura del file. Solitamente, nell'immagine finale che rappresenta il firmware, i byte nulli (`0x00`) vengono visualizzati con il colore nero, i byte del tipo `0xFF` con il colore bianco, e infine gli altri colori vengono utilizzati per evidenziare porzioni di file interessanti come stringhe di testo.

Esistono vari strumenti di questa tipologia ma in particolare segnaliamo `binvis.io` [11] che, grazie alla sua natura browser-based, permette oltre alla visualizzazione standard anche un'interazione diretta, offrendo la possibilità di selezionare e visualizzare il contenuto delle sezioni di interesse. Inoltre, oltre alla visualizzazione basata sulla tipologia di byte, permette anche di visualizzare l'entropia del firmware colorando diversamente l'immagine in base al valore dell'entropia. Come vedremo tra breve, questa funzione risulta essenziale al fine di determinare la presenza o meno di cifratura del firmware.

### 3.3.2 La cifratura del firmware

Se l'ispezione del firmware non restituisce nessun dato significativo, è necessario indagare se esso sia o meno cifrato, poiché i produttori potrebbero averlo cifrato per evitare che delle eventuali vulnerabilità siano scoperte attraverso una sua analisi.

#### 3.3.2.1 Identificare la cifratura

Un metodo che si può inizialmente attuare per identificare la cifratura comprende l'uso degli strumenti descritti precedentemente (v. L'ispezione del firmware 3.3.1) per controllare se vi siano o meno stringhe leggibili in chiaro all'interno del firmware. Tuttavia, questo metodo non è sufficiente per poter trarre conclusioni sulla natura del firmware: per questo si ricorre al calcolo dell'entropia. L'entropia può infatti confermare o meno le ipotesi fatte precedentemente: banalmente, un valore di entropia alto significa che molto probabilmente siamo in presenza di cifratura (Figura 3.2), un valore basso (o comunque non alto e variabile) ci indica che probabilmente non siamo in presenza di cifratura (Figura 3.4). Bisogna precisare, inoltre, che non necessariamente l'intero firmware debba essere cifrato, potrebbe essere che solo alcune sezioni ritenute più importanti siano state cifrate (Figura 3.3).

Come anticipato in precedenza, uno degli strumenti utilizzabili per il calcolo dell'entropia è *binvis.io* che, visualizzando con diverse gradazioni di colore i diversi valori di entropia, permette un'analisi immediata del livello di entropia. Nel caso in cui il firmware da analizzare fosse troppo pesante per essere caricato online si può utilizzare il già sopra citato *binwalk* che, avviato con l'opzione `-E`, conduce un'analisi dell'entropia del firmware generando il relativo grafico. In questi grafici sull'asse delle ordinate vi è l'entropia mentre sull'asse delle ascisse la posizione all'interno del firmware.

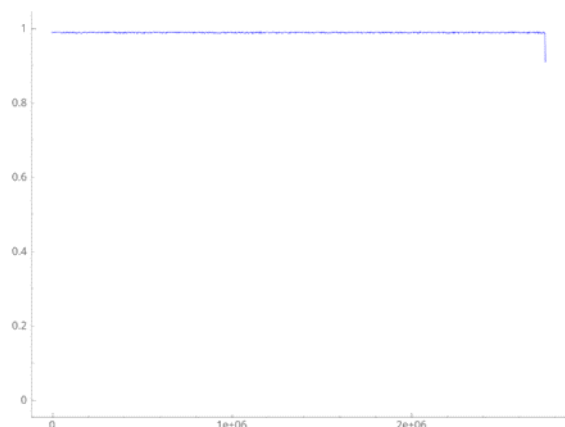


Figura 3.2 - Grafico dell'entropia di un firmware cifrato

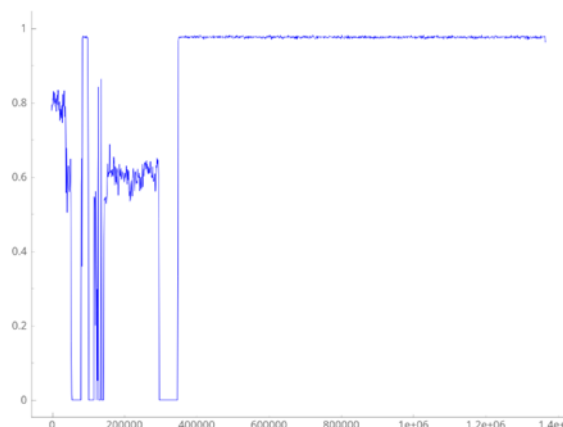


Figura 3.3 - Grafico dell'entropia di un firmware parzialmente cifrato



Figura 3.4 - Grafico dell'entropia di un firmware non cifrato

### 3.3.2.2 Distinguere la cifratura dalla compressione

Spesso a causa delle ridotte dimensioni delle memorie nei dispositivi IoT si fa ricorso ad algoritmi di compressione. Di questo bisogna tenerne conto durante l'identificazione della cifratura in quanto anche la compressione contribuisce ad un aumento dell'entropia.

Come metodo base per riuscire a distinguere un caso di cifratura da uno di compressione permane l'analisi del grafico dell'entropia: una cifratura presenterà una linea retta pressoché pari a 1.0 come livello di entropia (Figura 3.5), mentre la compressione presenterà qualche variazione in negativo (Figura 3.6). Questo metodo di verifica, però, non è sempre valido dato che con alcuni algoritmi di compressione risulta impossibile notare le differenze dal grafico dell'entropia (Figura 3.7). In questo caso si può ricorrere ad alcuni test statistici [12] come il test chi quadrato e il metodo Monte Carlo (con l'aiuto di strumenti per effettuare il calcolo) per riuscire a distinguere una cifratura da una compressione.

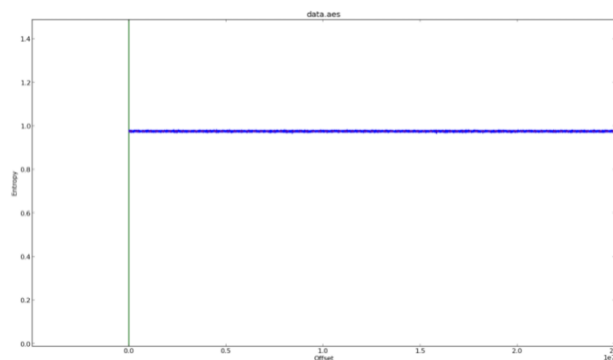


Figura 3.5 - Grafico dell'entropia di un file cifrato con AES

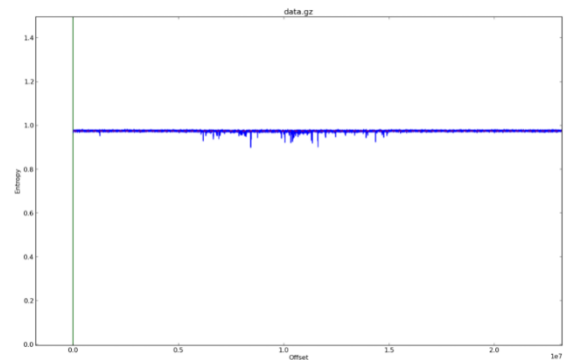


Figura 3.6 - Grafico dell'entropia di un file compresso con gzip

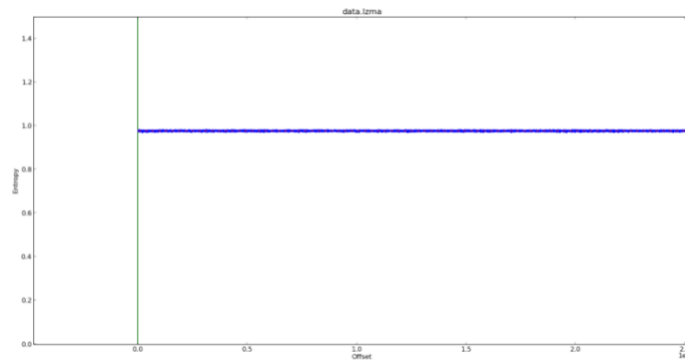


Figura 3.7 - Grafico dell'entropia di un file compresso con LZMA

### 3.3.2.3 I diversi scenari di cifratura

Non sempre un firmware è cifrato già dalla sua prima versione e ciò può volgere a nostro vantaggio. Possiamo distinguere tre possibili scenari di rilascio del firmware [13]:

- Scenario 1

La cifratura del firmware viene introdotta solo da una certa versione, per cui è necessaria una versione intermedia (non cifrata) contenete la funzione di decifrazione per decifrare le versioni successive.

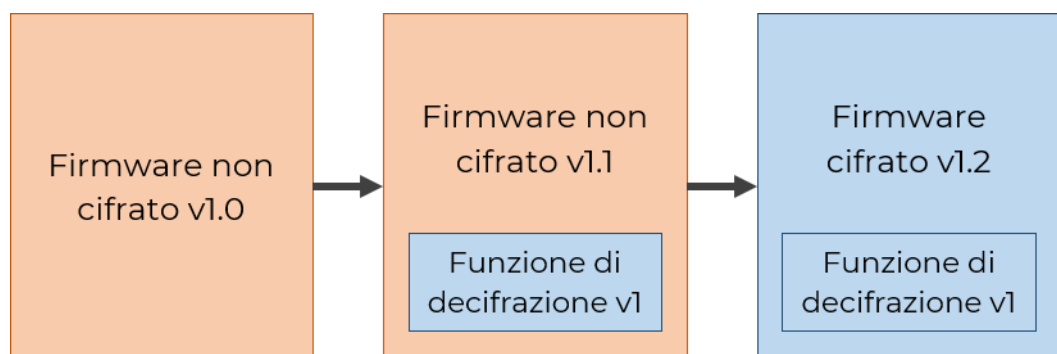


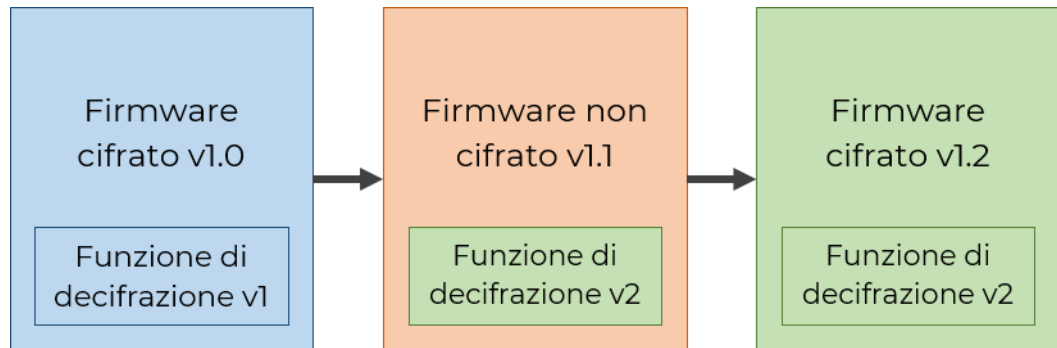
Figura 3.8 - Primo scenario di cifratura

In questo scenario se si riesce ad entrare in possesso della versione 1.1 si può tentare di estrarre la funzione per decifrare il firmware [13].



- Scenario 2

La cifratura è già presente dalla prima versione ma da una determinata versione viene cambiata la funzione di decifrazione e questo cambiamento viene attuato con una versione intermedia non cifrata.

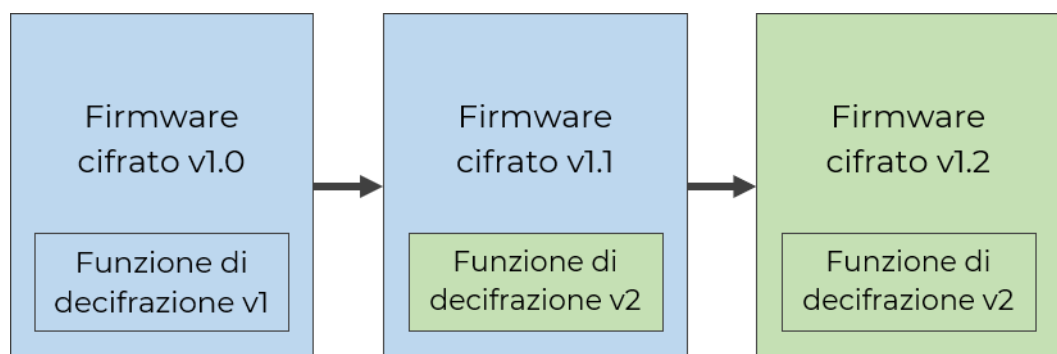


*Figura 3.9 - Secondo scenario di cifratura*

Dal punto di vista dell'analista la situazione è pressoché la stessa dello scenario 1: se si riesce ad entrare in possesso della versione intermedia non cifrata si può entrare in possesso della funzione di decifrazione.

- Scenario 3

Simile al secondo scenario solo che in questo caso è cifrata anche la versione intermedia contenente la nuova funzione di decifrazione.



*Figura 3.10 - Terzo scenario di cifratura*

In questo scenario, a differenza degli altri, non c'è modo di ottenere la funzione di decifrazione dai firmware per questo dispositivo. Tuttavia, la stessa funzione di decifrazione potrebbe essere utilizzata in altri dispositivi dello stesso produttore che potrebbero invece trovarsi in uno degli altri scenari.

### 3.3.3 I firmware semplici

Se durante la fase d'ispezione del firmware si è riconosciuta la struttura di un eseguibile con un set di istruzioni di una determinata architettura, allora si ricade nel caso di un firmware semplice. In questo caso l'analisi mira a risalire al codice corrispondente all'eseguibile in modo da poter ricercare la presenza di vulnerabilità, come quelle già citate elencate dall'OWASP [6].

Per procedere in questa analisi è, ovviamente, fondamentale l'uso di strumenti che traducano l'eseguibile in codice: i disassemblatori e i decompilatori. I disassemblatori si occupano di tradurre il linguaggio macchina in linguaggio assembler, contrapponendosi quindi alla funzione dell'assemblatore. Sebbene sia possibile ottenere una corrispondenza diretta fra i due linguaggi, spesso ci possono essere delle differenze che dipendono dall'espressività dell'assemblatore, il quale potrebbe tradurre diversamente la stessa istruzione. I decompilatori invece svolgono la funzione inversa del compilatore, ovvero, si occupano di tradurre i file eseguibili nel codice sorgente che li ha generati. Purtroppo, ciò non è sempre possibile a causa di varie azioni che possono essere state intraprese in fase di compilazione:

- Offuscamento del codice<sup>5</sup>: il compilatore produce un codice molto difficile da decompilare o che se decompilato produce un codice indeducibile.
- Ottimizzazione del compilatore<sup>6</sup>: il compilatore attua una serie di ottimizzazioni del codice non permettendo più quindi una corrispondenza biunivoca tra linguaggio macchina e codice che l'ha generato.
- Stripped binary: aggiungendo un'opzione è possibile richiedere al compilatore che non vengano mantenuti i simboli di debug. Ciò significa che nei file eseguibili risultanti (detti "stripped binaries") mancheranno riferimenti come, per esempio, i nomi di variabili e funzioni, rendendo molto più difficile la ritraduzione dell'eseguibile in codice.

La maggior parte degli strumenti che vengono utilizzati offrono un'interfaccia grafica che permette di analizzare sia l'eseguibile disassemblato che decompilato e rende disponibile una visualizzazione a grafo delle funzioni. Permettono, inoltre, di commentare o rinominare elementi (variabili, funzioni, ecc.) agevolando il processo di analisi. I più famosi sono IDA [14], Ghidra [15] e radare2 [16] con la relativa interfaccia grafica Cutter [17].

In aggiunta, alcuni strumenti per sopperire al problema indecidibile degli stripped binaries utilizzano approcci euristici e basati su firma per rilevare la presenza di librerie conosciute all'interno dell'eseguibile. Alcuni esempi sono: IDA F.L.I.R.T. [18], che utilizza sequenze di byte per creare le firme delle funzioni, Ghidra FunctionID [19], e il disassemblatore di JEB [20] che crea le firme delle funzioni applicando una funzione di hash al codice assembler.

---

<sup>5</sup> [https://en.wikipedia.org/wiki/Obfuscation\\_\(software\)](https://en.wikipedia.org/wiki/Obfuscation_(software))

<sup>6</sup> [https://en.wikipedia.org/wiki/Optimizing\\_compiler](https://en.wikipedia.org/wiki/Optimizing_compiler)

### 3.3.4 I firmware complessi

I firmware complessi sono facilmente riconoscibili poiché tipicamente sono composti dai seguenti elementi [9]:

- Bootloader
- Kernel
- Filesystems
- File binari
- File di risorse o di supporto
- Web-server/web-interface

Il processo che si compie in questa fase è quello di analizzare il contenuto dei file presenti nel filesystem alla ricerca di possibili vulnerabilità. Essendo i filesystem, normalmente, compressi, per analizzarli è necessario prima estrarli; ciò può essere fatto automaticamente (per esempio con `binwalk`) oppure manualmente utilizzando il giusto estrattore in base al tipo di filesystem. Come elencato nella Firmware Security Testing Methodology dell'OWASP [4], le vulnerabilità più comuni da ricercare all'interno del filesystem sono:

- Demoni di rete insicuri come telnetd (a volte rinominati per mascherarli)
- Credenziali cablate (username, password, chiavi API, chiavi SSH)
- Dettagli sulle API cablate e sui servizi
- Funzionalità di aggiornamento e di avvio (possono essere utilizzate come punto d'accesso)
- Analizzare i codici sorgenti dei programmi (per una possibile remote code execution)
- Analizzare i codici compilati (con le tecniche descritte per i firmware semplici)

Questa fase può essere velocizzata grazie all'esistenza di strumenti che automatizzano l'analisi. Un esempio è `firmwalker` [21] che, passandogli come parametro il percorso del filesystem estratto, crea un report in formato testuale contenente le informazioni estratte analizzando:

- I file `etc/shadow` e `etc/passwd`
- File correlati ad SSL (`.pem`, `.crt`, etc.)
- File di configurazione
- File di scripting
- Cercando parole chiavi come `admin`, `password`, `remote`, etc.
- Server web comuni
- URL, indirizzi e-mail e indirizzi IP
- E molto altro

Un altro strumento che può automatizzare questa fase di analisi è il Firmware Analysis Comparison Toolkit (FACT) [22]. FACT risulta più complesso rispetto ad un analizzatore semplice come `firmwalker` in quanto avvia un server web sulla macchina che lo esegue per

esporre un'interfaccia web con supporto a delle API REST. Permette di importare direttamente l'intero firmware che verrà aggiunto ad una coda dei firmware sotto analisi, la quale verrà eseguita in background. Una volta terminata l'analisi produrrà una pagina web relativa al firmware (di cui sarà possibile generare un report in PDF) contenente varie informazioni, tra cui:

- Identificazione dei vari componenti (sistema operativo, architettura della CPU, ecc.)
- Estrazione del firmware
- Rilevamento dei certificati e chiavi private
- Rilevamento delle vulnerabilità collegandole alle relative Common Weakness Enumeration (CWE)
- e molto altro...

Un'altra caratteristica di FACT è quella di permettere l'estensione delle proprie funzionalità tramite dei plug-in, in particolare ne sono stati definiti tre tipi:

- unpacker: si occupa di estrarre il filesystem
- analysis: si occupa di analizzare i firmware estratti dall'unpacker
- compare: si occupa di comparare due firmware e mostrare le differenze, utilizzato per esempio per scoprire le novità introdotte da un aggiornamento dello stesso firmware

Questa complessità richiede però che vengano soddisfatti alcuni requisiti riguardo alle prestazioni che dovrà avere la macchina che compie l'analisi, rendendolo non sempre utilizzabile.

## 4 IMPLEMENTAZIONE

In questo capitolo illustreremo come mettere in pratica un'analisi statica seguendo i passaggi descritti precedentemente. I dispositivi che si sono scelti di analizzare sono due modem adsl:

- Alice Gate 2 Plus
- Fastweb Home Access Gateway

### 4.1 ALICE GATE 2 PLUS



Figura 4.1 - Alice Gate 2 Plus

#### 4.1.1 Raccolta delle informazioni

Il modem in questione è stato prodotto da Industrie Dial Face e distribuito da Telecom Italia intorno la fine degli anni 2000. Trattandosi quindi di un dispositivo oramai datato, dopo un'approfondita ricerca, non è stato possibile ottenere informazioni né dal sito del distributore, né da quello del produttore (poiché l'azienda è stata chiusa nel 2012) e nemmeno da altri siti non ufficiali. Per cui l'unico modo per ottenere più informazioni sul dispositivo è aprirlo.

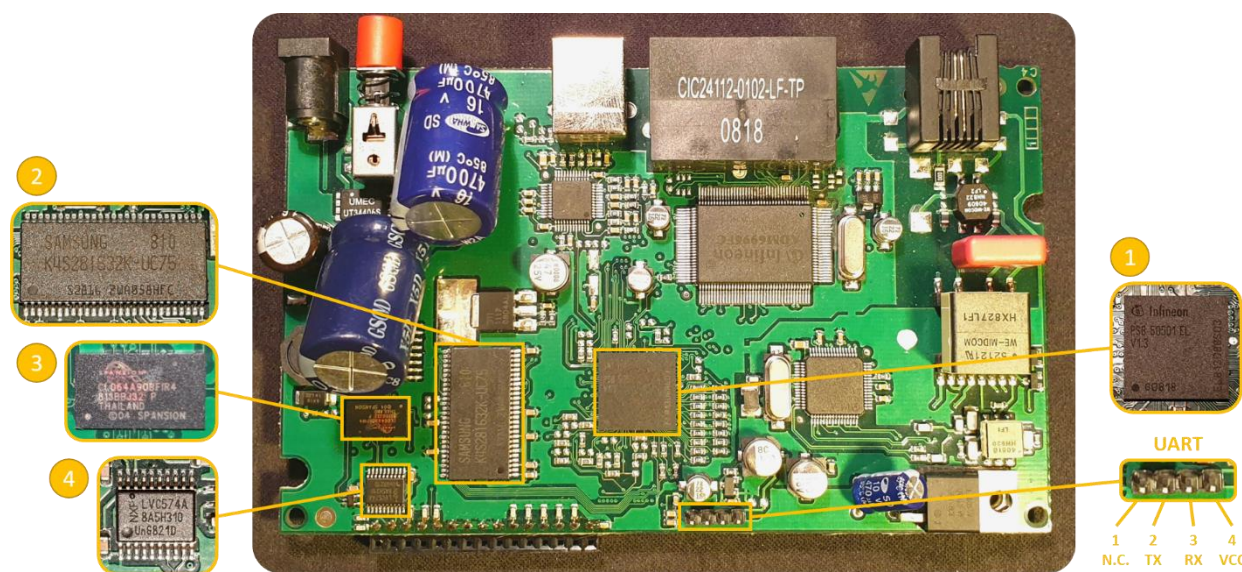


Figura 4.2 - Scheda principale dell'Alice Gate 2 Plus con evidenziate le componenti e l'interfaccia UART

Analizzando i circuiti integrati presenti sulla scheda principale<sup>7</sup> (Figura 4.2) è possibile identificare:

1. Il System-on-a-chip (SoC) è l'Infineon PSB 50501 che ha un'architettura MIPS 4KEc RISC a 32-bit ed una frequenza di funzionamento di 235 MHz
2. La RAM ha una capacità di 128Mb Samsung K4S281632K-UC75 8M x 16Bit x 4 Banchi SDRAM con una frequenza massima di 133MHz
3. La memoria Flash è una SPANSION S29GL064A da 8Mb
4. L'integrato LVC574A (contenente dei flip-flop d) può essere utile come punto di riferimento per la tensione di alimentazione ( $V_{CC}$ ) dato che supporta una tensione compresa nell'intervallo 1.65-3.6 V<sup>8</sup>

Alla luce delle informazioni raccolte, l'unico modo rimasto per acquisire il firmware tramite un metodo software sarebbe intercettare un aggiornamento. Purtroppo, nel nostro caso questo non è possibile poiché, trattandosi di un modem, gli aggiornamenti avvengono tramite l'interfaccia WAN che risulta molto difficile da analizzare. Si potrebbe cercare di risolvere questo problema collegando il modem ad una LAN preesistente (cambiando opportunamente le tabelle di routing) per poi analizzare il traffico della porta ethernet (nel caso in cui il protocollo sia PPPoE). Tuttavia, ciò non basterebbe poiché molto probabilmente le connessioni che venivano utilizzate per gli aggiornamenti non saranno più disponibili trattandosi di un dispositivo datato.

L'unico modo per entrare in possesso del firmware, perciò, rimane effettuare uno dei metodi hardware. Partiamo dall'analizzare se è fattibile l'estrazione dall'interfaccia UART che è il metodo più semplice.

#### 4.1.2 Individuazione dell'interfaccia UART

Per individuare l'interfaccia UART sulla scheda si può:

- Utilizzare le informazioni raccolte nella fase precedente
- Cercare se siano presenti delle etichette stampate sulla scheda che identifichino l'interfaccia
- Trovare dei possibili pin candidati sulla scheda (di solito nel caso della UART gruppi di 3 o 4 pin). Una volta localizzati si possono identificare i singoli pin nei modi seguenti:
  - Utilizzando un multimetro per misurare le resistenze e le tensioni dei vari pin per identificare quelli di alimentazione e quelli per la comunicazione seriale
  - Seguendo le tracce del circuito stampato (spesso infattibile a causa della miniaturizzazione di piste e componenti e al fatto che spesso i pin dei SoC sono collocati sotto di loro, e quindi irraggiungibili)
  - Usando i dispositivi di identificazione come JTAGulator [8]

---

<sup>7</sup> Nel dispositivo è presente anche una scheda con dei led che è stata omessa poiché non rilevante

<sup>8</sup> La tensione tipica per l'interfaccia UART è di 3.3 V

- Utilizzando un oscilloscopio (o anche un semplice logic analyzer) per identificare il pin di trasmissione che presumibilmente trasmetterà informazioni (quindi la sua tensione varierà) durante la fase di boot

Nel nostro caso la fase di raccolta delle informazioni non ci ha fornito alcuna informazione sulla collocazione dell'interfaccia UART e non è nemmeno presente alcuna indicazione sulla scheda. Tuttavia, è possibile individuare nella parte inferiore della scheda quattro fori placcati ben allineati che sono dei buoni candidati.

D'ora in poi indicheremo i quattro pin numerandoli come mostrato nella Figura 4.2 ed indicheremo la massa con GND e la tensione di alimentazione (dell'interfaccia) con  $V_{CC}$ . GND può essere trovata molto semplicemente in vari punti del circuito, mentre per  $V_{CC}$  prenderemo come riferimento il pin di alimentazione dell'integrato LVC574A.

#### 4.1.2.1 Le misurazioni col multimetro

Effettuiamo ora due misurazioni con il multimetro: a dispositivo spento misuriamo la resistenza fra ciascun pin e GND e fra ciascun pin e  $V_{CC}$ , successivamente alimentiamo il dispositivo e misuriamo la differenza di potenziale fra i vari pin e GND.

Pin	$R_{GND} (\Omega)$	$R_{V_{CC}} (\Omega)$	$V (V)$	Note
1	$\infty$	$\infty$	0	N.C.
2	1.39K	1.52K	3.36	Sospetto TX
3	1.37K	1.48K	3.33	
4	104.5	0	3.37	$V_{CC}$

Tabella 1 - Misurazioni di resistenze e tensioni per ciascun pin

Dalle misurazioni effettuate (Tabella 1) apprendiamo che:

- Il pin 1 presenta una resistenza molto grande per entrambi i riferimenti e una tensione nulla per cui lo considereremo non connesso (N.C.)
- Il pin 4 presenta una piccola resistenza rispetto GND e nulla rispetto a  $V_{CC}$  e una tensione pari a 3.37 V, probabilmente questo pin è  $V_{CC}$
- Il pin 2 e 3 presentano entrambi una resistenza in entrambe le misurazioni e una tensione simile a quella di alimentazione. Questi dati non sono sufficienti per poter avanzare ipotesi, ma notiamo che durante l'avvio del dispositivo la tensione sul pin 2 oscilla lievemente; quelle oscillazioni potrebbero rappresentare i dati inviati attraverso la seriale durante il processo di boot, perciò, sospettiamo che tale pin possa essere quello di trasmissione (TX).

#### 4.1.2.2 La verifica con l'oscilloscopio

Per eliminare ogni dubbio sulla posizione del pin di trasmissione, procediamo con una verifica dei segnali presenti sui pin 2 e 3 durante il processo di boot grazie all'uso di un oscilloscopio. Per quanto riguarda il pin 3 non viene riscontrata alcuna variazione del segnale, mentre per il pin 2 viene visualizzata un'onda quadra che rappresenta i dati trasmessi (Figura 4.3).

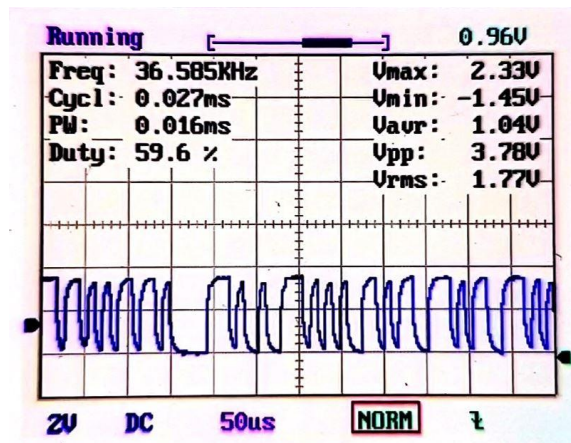


Figura 4.3 - Segnale sul pin 2 durante il boot visualizzato sull'oscilloscopio

Abbiamo confermato che il pin 2 è quello che trasmette (TX) e per esclusione il pin 3 è quello che riceve (RX).

Pin	UART
1	N.C.
2	TX
3	RX
4	V <sub>CC</sub>

Tabella 2 - Pinout finale dell'interfaccia UART

#### 4.1.3 Estrazione del firmware

Una volta individuata l'interfaccia UART ci si può connettere ad essa per verificare se uno dei metodi per estrarre il firmware descritti precedentemente (v. 3.2.2.1) sia possibile da attuare.

##### 4.1.3.1 Connettersi all'interfaccia UART

Per connetterci all'interfaccia utilizziamo il Raspberry Pi Zero W<sup>9</sup>, un single-board computer (SBC) dalle ridotte dimensioni provvisto dell'interfaccia seriale e di un modulo wifi.

Per connettersi sono sufficienti tre cavi: due per i pin di trasmissione e ricezione che andranno collegati in maniera opposta (il pin di trasmissione di uno andrà collegato al pin di ricezione dell'altro) e uno per collegare assieme le masse dei due dispositivi in modo da porli

<sup>9</sup> <https://www.raspberrypi.org/products/raspberry-pi-zero-w/>



allo stesso potenziale. L'alimentazione non è necessaria e non è necessario regolare alcuna tensione poiché i livelli logici del raspberry sono di 3.3 V come nel caso di questo dispositivo.

Una volta connessi al raspberry tramite ssh e configurata l'interfaccia UART<sup>10</sup> è possibile connettersi mediante vari emulatori di terminale, nel nostro caso abbiamo utilizzato *picocom* [23].

Prima di avviare la connessione è necessario identificare quale sia la velocità di trasmissione utilizzata dal SoC, per farlo si può eseguire uno di questi metodi:

- Utilizzare le informazioni raccolte in precedenza
- Misurare la lunghezza di un bit o di un byte attraverso l'uso di un oscilloscopio
- Provare una serie di velocità più comuni<sup>11</sup> finché non si ottengono caratteri ASCII stampabili

Nel nostro caso non abbiamo alcuna informazione sulla velocità di trasmissione e la strumentazione con la quale è stato misurato il segnale TX in precedenza non è abbastanza precisa per poter calcolare la velocità. Per cui opteremo per la terza opzione.

Per connettersi è sufficiente eseguire il seguente comando:

```
sudo picocom /dev/ttyS0 -b 115200
```

dove è stato specificato il dispositivo a cui connettersi e la velocità di trasmissione da utilizzare. Individuiamo che la velocità di trasmissione è di 115200 bit/s poiché è l'unica che restituisce un output leggibile.

#### **4.1.3.2 Analizzare e intercettare la sequenza di boot**

Una volta avviato, il dispositivo scrive sull'interfaccia seriale l'intero processo di boot da cui è possibile ricavare una serie di informazioni quali: informazioni sui vari componenti, informazioni sul tipo di sistema operativo e filesystem, suddivisioni in partizioni della memoria, ecc.

Come è possibile notare in Figura 4.4, apprendiamo che il bootloader installato è U-Boot<sup>12</sup> (in rosso), uno tra i più utilizzati nei dispositivi embedded, e ci viene confermato quanto raccolto nelle fasi precedenti (v. 4.1.1) ovvero che il SoC ha un'architettura MIPS e una frequenza di 235 MHz (in arancione) e che modello e dimensioni della memoria flash corrispondono (in blu). In Figura 4.5 si può notare una sequenza di autoboot nella quale sceglie l'immagine attiva e veniamo a conoscenza che l'immagine in questione ha come kernel Linux.

---

<sup>10</sup> <https://www.raspberrypi.org/documentation/configuration/uart.md>

<sup>11</sup> [https://en.wikipedia.org/wiki/Serial\\_port#Speed](https://en.wikipedia.org/wiki/Serial_port#Speed)

<sup>12</sup> <https://github.com/u-boot/u-boot>

```

Version 1.0.1
Read EEPROM
Jump to Flash
Head : Amazon Version 1.0.0
DRAM: 16 MB

Head : relocate_code start
Head: relocate code finish.
Image Name: u-boot image
Image Type: MIPS Linux Firmware (lzma compressed)
Data Size: 40672 Bytes = 39.7 kB
Load Address: 80100000
Entry Point: 80100000
Disabling all the interrupts
Uncompressing UBoot Image ...
Uncompression completed successfully with destLen 124144.
Head: Jumping to u-boot in the ram at 0x80100000
Infineon Amazon
U-Boot 2.0.16-16 (Jun 19 2008 - 17:51:23)
In env_init : env_ptr = 0xb37c0000
For environment CRC32 is OK
Board: AMAZON Yangtse Version, Chip V1.3, CPU Speed 235 MHz
IDF LED fix ... done
DRAM: 16 MB
USB support: 48Mhz clock enabled

relocate_code start
relocate code finish

Entering flash_init()
detected SPANSION S29GL064A

IDF-FLASH fix...done
Flash: 8 MB
env_relocate[228] malloced ENV at 80aa0008

```

Figura 4.4 - Prima parte del processo di boot da cui è possibile ricavare informazioni sul bootloader (in rosso), CPU (in arancione), e memoria flash (in blu)

```

Initialize devices...
In: serial
Out: serial
Err: serial
Net: AMAZON Switch

Type "run flash_nfs" to mount root filesystem over NFS

Press enter key to stop autoboot: 0
getting active image...
active image seems to be 1
## Booting image at b33b0094 ...
Image Name: openrg_AGIB_1.2.1
Created: 2008-03-26 17:48:43 UTC
Image Type: MIPS Linux Kernel Image (lzma compressed)
Data Size: 2626643 Bytes = 2.5 MB
Load Address: 80002000
Entry Point: 80003d50
Verifying Checksum ... OK
Uncompressing Kernel Image ... Uncompression time : 1076650/117500000
Uncompression length is 4083712
OK

Starting kernel ...

Yangtse Version

```

Figura 4.5 - Seconda parte del processo di boot da cui si ricava il kernel utilizzato

Una volta terminato il boot non viene presentato alcun prompt di accesso e il dispositivo non risponde ad alcun comando. L'unica alternativa che abbiamo è di interrompere il processo di boot prima che venga selezionata l'immagine in modo da entrare nel bootloader. Fortunatamente si ha abbastanza tempo (2s) per poter interrompere il processo di boot premendo il tasto invio ed entrare così nel bootloader:

```

Press enter key to stop autoboot: 0
AMAZON-DIALFACE #

```

A questo punto digitando il comando `help` ci verrà fornita la lista dei comandi possibili all'interno di U-Boot poiché alcuni comandi potrebbero essere stati rimossi.

Uno dei comandi disponibili è `printenv` che ci consente di visualizzare i vari parametri di configurazione utilizzati da U-Boot (Figura 4.6).

```
AMAZON-DIALFACE # printenv
bootcmd=jboot
bootdelay=2
baudrate=115200
preboot=echo;echo Type "run flash_nfs" to mount root f
mem=16M
ipaddr=192.168.1.1
serverip=192.168.1.10
ethaddr=08:01:11:11:11:11
netdev=eth1
baudrate=115200
rootpath=/opt/nfs
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=$(ser
ramargs=setenv bootargs root=/dev/ram rw
addip=setenv bootargs $(bootargs) ip=$(ipaddr):$(serve
addmisc=setenv bootargs $(bootargs) console=ttyS0,$(ba
flash_nfs=run nfsargs addip addmisc;bootm $(kernel_add
ramdisk_addr=B0100000
flash_self=run ramargs addip addmisc;bootm $(kernel_ad
bootfile=/tftpboot/AMAZON/uImage
net_nfs=tftp 80500000 $(bootfile);run nfsargs addip ad
u-boot=/tftpboot/AMAZON/u-boot.bin
load=tftp 80500000 $(u-boot)
update=protect off 1:0-2;era 1:0-2;cp.b 80500000 B0000
flashargs=setenv bootargs root=/dev/mtdblock2
flash_flash=run flashargs addip addmisc;bootm $(kerne
update_uboot=tftpboot 80400000 u-boot.img;upgrade uboo
update_kernel=tftpboot 80400000 uImage;upgrade kernel
update_openrg=tftpboot 80500000 openrg.img; jprogram 0
update_factory=tftpboot 80500000 rg_factory; jfactory
update_all=run update_openrg; run update_factory; run
reset_uboot_config=erase 0xB37C0000 0xB37DFFFF 0
part0_begin=0xB3000000
part1_begin=0xB3020000
```

Figura 4.6 - Porzione delle variabili d'ambiente utilizzate per configurare il bootloader

#### 4.1.3.3 Estrazione tramite il bootloader

Tra i comandi disponibili nel bootloader quello di nostro maggiore interesse è `md` che permette di visualizzare un indirizzo specifico della memoria RAM:

```
AMAZON-DIALFACE # help md
md [.b, .w, .l] address [# of objects]
- memory display
```

Essendo noi interessati al contenuto della memoria flash, è necessario sapere quali indirizzi della RAM corrispondono alla memoria flash. Ciò è possibile ottenerlo dal comando `flinfo` (Figura 4.7) che ci informa che la memoria inizia all'indirizzo `0xB3000000` e termina all'indirizzo `0xB37FFFFFFF` (corrispondenti a 8388608 bytes, ovvero 8 Mb proprio come la dimensione della memoria)

AMAZON-DIALFACE # flinfo

Bank # 1: AMD S29GL064A (64 Mbit)  
Size: 8 MB in 135 Sectors

Sector Start Addresses:

B3000000	B3002000	B3004000	B3006000	B3008000
B300A000	B300C000	B300E000	B3010000	B3020000
B3030000	B3040000	B3050000	B3060000	B3070000
B3080000	B3090000	B30A0000	B30B0000	B30C0000
B30D0000	B30E0000	B30F0000	B3100000	B3110000
B3120000	B3130000	B3140000	B3150000	B3160000
B3170000	B3180000	B3190000	B31A0000	B31B0000
B31C0000	B31D0000	B31E0000	B31F0000	B3200000
B3210000	B3220000	B3230000	B3240000	B3250000
B3260000	B3270000	B3280000	B3290000	B32A0000
B32B0000	B32C0000	B32D0000	B32E0000	B32F0000
B3300000	B3310000	B3320000	B3330000	B3340000
B3350000	B3360000	B3370000	B3380000	B3390000
B33A0000	B33B0000	B33C0000	B33D0000	B33E0000
B33F0000	B3400000	B3410000	B3420000	B3430000
B3440000	B3450000	B3460000	B3470000	B3480000
B3490000	B34A0000	B34B0000	B34C0000	B34D0000
B34E0000	B34F0000	B3500000	B3510000	B3520000
B3530000	B3540000	B3550000	B3560000	B3570000
B3580000	B3590000	B35A0000	B35B0000	B35C0000
B35D0000	B35E0000	B35F0000	B3600000	B3610000
B3620000	B3630000	B3640000	B3650000	B3660000
B3670000	B3680000	B3690000	B36A0000	B36B0000
B36C0000	B36D0000	B36E0000	B36F0000	B3700000
B3710000	B3720000	B3730000	B3740000	B3750000
B3760000	B3770000	B3780000	B3790000	B37A0000
B37B0000	B37C0000	B37D0000	B37E0000	B37F0000

Figura 4.7 - Elenco dei settori che corrispondono alla memoria flash

Per effettuare l'estrazione della memoria sarà quindi sufficiente eseguire il seguente comando dove indichiamo l'indirizzo di partenza e il numero di blocchi che vogliamo stampare:

```
AMAZON-DIALFACE # md.b B3000000 800000
```

b3000000:	10 00 00 07 00 00 00 00 00 00 00 00 00 00 00 00	.....
b3000010:	68 8c 68 8c 00 00 00 00 00 00 00 00 00 00 00 00	h.h.....
b3000020:	40 80 90 00 40 80 98 00 40 1a 60 00 24 1b ff fe	@...e...`.\$...
b3000030:	03 5b d0 24 40 9a 60 00 40 80 68 00 40 80 48 00	.[.\$@.`.e.h.@.H.
b3000040:	40 80 58 00 24 08 00 02 40 88 80 00 04 11 00 02	@.X.\$...@.....
b3000050:	00 00 00 00 00 00 4c 3c 03 e0 e0 21 8f e9 00 00	.....L<...!....
b3000060:	03 89 e0 20 8f 99 00 94 00 00 00 00 03 20 f8 09	... ..
b3000070:	00 00 00 00 8f 99 00 f4 00 00 00 00 03 20 f8 09	.....
b3000080:	00 00 00 00 24 08 00 03 40 88 80 00 8f 99 00 98	...\$.@.....
b3000090:	03 20 f8 09 00 00 00 00 3c 08 9f c0 35 08 10 00	. ....<...5...
b30000a0:	25 1d 00 00 8f 99 01 04 03 20 00 08 00 00 00 00	%.....
b30000b0:	8f 99 00 44 03 20 f8 09 00 00 00 00 03 e0 00 08	...D. ....
b30000c0:	00 80 c8 21 03 20 08 00 00 00 00 00 00 80 40 21	...!.....@!
b30000d0:	00 a0 48 21 3c 01 00 02 01 01 50 20 8d 0b 00 00	..H!<....P ....
b30000e0:	ad 2b 00 00 25 08 00 04 01 48 08 2a 10 20 ff fb	.+...%...H.*. ..
b30000f0:	25 29 00 04 00 00 00 00 03 e0 00 08 00 00 00 00	%).....
b3000100:	00 80 e8 21 03 80 70 21 3c 01 b3 00 03 81 e0 22	...!.p!<...."
b3000110:	03 86 e0 20 03 8e 70 22 3c 08 b3 00 21 0a 4d e0	... .p"<...!.M.
b3000120:	00 c0 48 21 8d 0b 00 00 ad 2b 00 00 25 08 00 04	..H!.....+.%...
b3000130:	01 48 08 2a 10 20 ff fb 25 29 00 04 20 c8 01 54	.H.*. ...%)... ..T
b3000140:	01 00 00 08 00 00 00 00 b3 00 4d c0 b3 00 4d f0	.....M...M.
b3000150:	00 00 00 4c 8d 0b ff fc 23 8c 00 08 24 0a 00 02	...L...#...\$...
b3000160:	8d 89 00 00 11 20 00 02 01 2e 48 20 ad 89 00 00	.....H ....
b3000170:	21 4a 00 01 01 4b 08 2a 14 20 ff f9 21 8c 00 04	!J...K.*. ...!
b3000180:	8d 09 ff f4 8d 0a ff f8 01 2e 48 20 01 4e 50 20	.....H .NP
b3000190:	21 29 ff fc 21 29 00 04 01 2a 08 2a 54 20 ff fd	!)..!)...*.T ..
b30001a0:	ad 20 00 00 00 a0 20 21 8f 99 00 24 03 20 00 08	.... !...\$. ..
b30001b0:	00 c0 28 21 10 00 ff ff 10 00 ff ff 00 00 00 00	..(!.....
b30001c0:	3c 08 b0 10 35 08 53 00 3c 09 13 00 35 29 00 31	<...5.S.<...5).1
b30001d0:	ad 09 00 20 3c 09 14 00 35 29 00 31 ad 09 00 24	... <...5).1...\$
b30001e0:	3c 09 18 00 35 29 00 30 ad 09 00 28 3c 09 1c 00	<...5).0...(<...
b30001f0:	35 29 00 60 ad 09 00 2c 3c 09 00 01 35 29 d7 ff	5).`....,<...5)..
b3000200:	ad 09 00 60 ad 09 00 64 3c 08 bf 80 24 09 00 02	...`....d<...\$...

Figura 4.8 - Inizio dell'estrazione della memoria flash: nella colonna di sinistra è specificato l'indirizzo, al centro il contenuto in esadecimale, a destra la rappresentazione in ASCII

Per salvare in un file queste informazioni inviate sulla seriale è sufficiente avviare la comunicazione impostando con l'opzione `-g` un file di log:

```
sudo picocom /dev/ttyS0 -b 115200 -g dump.txt
```

A questo punto il file `dump.txt` conterrà tutto quello che è stato inviato e ricevuto sull'interfaccia seriale. Rimuoviamo le varie informazioni che sono state stampate prima del contenuto della memoria (processo di boot, prompt dei comandi, ecc.). Oltre al contenuto della memoria saranno però ancora presenti i vari indirizzi e la rappresentazione in carattere ASCII. Per eliminare questi dati non necessari ed ottenere così il vero firmware è sufficiente lanciare il seguente comando:

```
egrep -o '([a-f|0-9]{2} ){15}[a-f|0-9]{2}' dump.txt | xxd -r -p > alice.bin
```

Tale comando, tramite una regex, seleziona esattamente gli esadecimali del contenuto della memoria che poi successivamente vengono riconvertiti da rappresentazione testuale a binario.

#### 4.1.4 Analisi del firmware

Utilizziamo gli strumenti descritti nel paragrafo 3.3.1 per analizzare il firmware. Per il comando `file` il firmware risulta essere dei dati generici, mentre il comando `strings` visualizza molte stringhe chiaramente leggibili all'interno del firmware. Sembrano essere le stesse informazioni che vengono scritte sull'interfaccia seriale durante il processo di boot (Figura 4.9) e gli stessi parametri di configurazione visualizzabili nel bootloader (Figura 4.10 e Figura 4.6).

```
bongio PC-LUCA ../firmware dump strings -n 10 alice.bin
U-Boot 2.0.16-16 (Jun 19 2008 - 17:51:45)
*** failed ***
Head : Amazon Version 1.0.0
Invalid OS
Unknown OS
Unknown Architecture
Invalid CPU
MIPS 64 Bit
SPARC 64 Bit
Unknown Image
Invalid Image
Standalone Program
Kernel Image
RAMDisk Image
Multi-File Image
gzip compressed
uncompressed
bzip2 compressed
lzma compressed
unknown compression
%s %s %s (%s)
Image Name:  %.*s
Image Type:
Data Size:   %d Bytes =
Load Address: %08x
Entry Point: %08x
Contents:
Image %d: %8ld Bytes =
Head : relocate_code start
Head: relocate code finish.
Bad Magic Number at address 0x%08lx
Bad Header Checksum
Disabling all the interrupts
Uncompressing UBoot Image ...
```

Figura 4.9 - Risultato del comando `strings` sul firmware. L'opzione `-n 10` limita il più possibile la stampa di stringhe prive di significato

```
rg_factory
(rg_factory
(panic_timeout(0))
(contact(rg_support@juno.com))
(network
(rg_mac(00:17:37:75:68:6E))
(codepage())
(session_lifetime(900000))
(manufacturer
(hardware
(version(S2PF-6))
(serial_num(N0708227HMM))
(vendor_name(Industrie Dial Face))
(vendor_oui(001737))
(boot_rom_version(7.8))
(model_number(Alice Gate 2 Plus))
(sys_object_id(1.3.6.1.4.1.10799.250.10))
(description(Alice Gate 2 Plus Basic))
(product_class(R2P))
bootcmd=jboot
bootdelay=2
baudrate=115200
preboot=echo;echo Type "run flash_nfs" to mount root filesystem
ipaddr=192.168.1.1
serverip=192.168.1.10
ethaddr=08:01:11:11:11:11
netdev=eth1
baudrate=115200
rootpath=/opt/nfs
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=$(serverip):$(
ramargs=setenv bootargs root=/dev/ram rw
addip=setenv bootargs $(bootargs) ip=$(ipaddr):$(serverip):$(ga
addmisc=setenv bootargs $(bootargs) console=ttyS0,$(baudrate) e
flash_nfs=run nfsargs addip addmisc;bootm $(kernel_addr)
ramdisk_addr=B0100000
flash_self=run ramargs addip addmisc;bootm $(kernel_addr) $(ram
```

Figura 4.10 - Continuazione dell'output di Figura 4.9 contenente vari parametri di configurazione, gli stessi della Figura 4.6

Da ciò si deduce che per lo meno una parte del firmware non è cifrata. Proseguiamo l'analisi invocando il comando `binwalk` sul firmware.

DECIMAL	HEXADECIMAL	DESCRIPTION
16368	0x3FF0	U-Boot version string, "U-Boot 2.0.16-16 (Jun 19 2008 - 17:51:45)"
17632	0x44E0	CRC32 polynomial table, big endian
19936	0x4DE0	uImage header, header size: 64 bytes, header CRC: 0x25BFFA15, created: 2008-06-19 15:51:45, image size: 40672 bytes, Data Address: 0x80100000, Entry Point: 0x80100000, data CRC: 0x65DE756C, OS: Linux, CPU: MIPS, image type: Firmware Image, compression type: lzma, image name: "u-boot image" ❶
20000	0x4E20	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 124144 bytes13
1220	0x20094	uImage header, header size: 64 bytes, header CRC: 0xE90E64B, created: 2008-03-28 11:28:55, image size: 262539 bytes, Data Address: 0x80002000, Entry Point: 0x80003D50, data CRC: 0xE43A00E6, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "openrg_AGIB_1.2.1-t" ❷
131284	0x200D4	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 4083712 bytes
3866772	0x3B0094	uImage header, header size: 64 bytes, header CRC: 0x58A60386, created: 2008-03-26 17:48:43, image size: 262643 bytes, Data Address: 0x80002000, Entry Point: 0x80003D50, data CRC: 0x51D75F52, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "openrg_AGIB_1.2.1" ❸
3866836	0x3B00D4	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 4083712 bytes
7733400	0x760098	Zlib compressed data, default compression

Figura 4.11 - Risultato del comando `binwalk` sul firmware. È possibile notare l'immagine del bootloader (in rosso), l'immagine principale del firmware e quella di backup (in arancione e blu) e una porzione di dati compressi (in verde)

Seppur ancora da verificare, l'output di `binwalk` (Figura 4.11) ci illustra una certa struttura del firmware: oltre alla stringa della versione di U-Boot e il checksum CRC32 si possono notare tre header di immagini di boot seguite ognuna da una porzione di dati compressa con LZMA. La prima immagine (etichettata "u-boot image") è l'immagine del bootloader stesso, le altre due immagini che hanno due etichette molto simili ("openrg\_AGIB\_1.2.1" <sup>13</sup>) potrebbero essere una l'immagine che viene effettivamente avviata, l'altra un'immagine di backup in caso di qualche errore sulla prima. Infine, è presente una porzione di dati generici compressi con Zlib.

A questo punto è molto improbabile che il firmware sia cifrato poiché solitamente o è cifrato l'intero firmware o comunque una buona parte di esso, mentre nel nostro caso `binwalk` è stato in grado di reperire molte informazioni. Dal grafico dell'entropia si può infatti notare i picchi in corrispondenza solamente delle porzioni di dati compressi (Figura 4.12).

<sup>13</sup> OpenRG è un cosiddetto "middleware" creato appositamente per gateway domestici contenente i driver, il sistema operativo, i protocolli e le varie applicazioni necessarie.



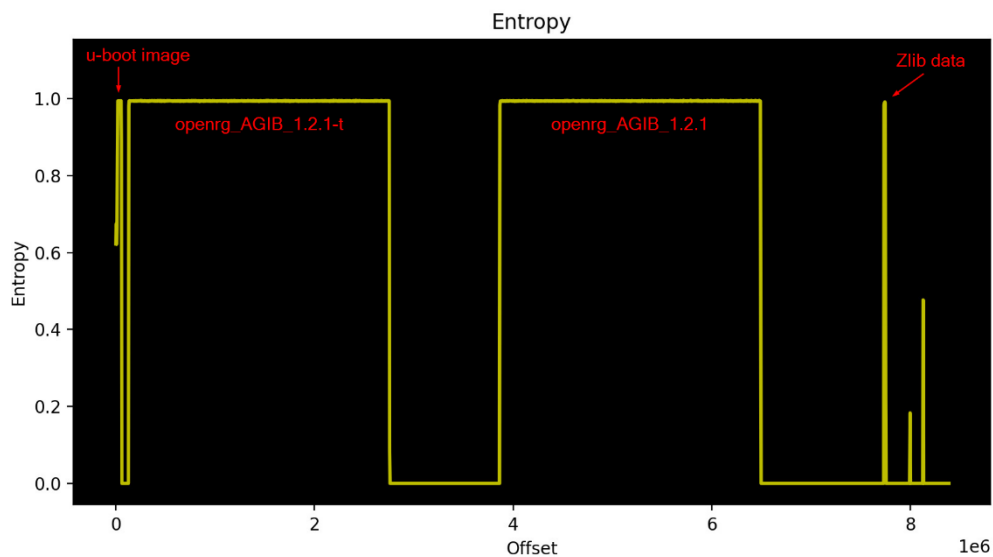


Figura 4.12 - Grafico dell'entropia del firmware. È possibile notare i picchi di entropia in corrispondenza delle porzioni di firmware compresse

Per una visione più chiara dell'intero firmware utilizziamo [binvis.io](https://binvis.io).

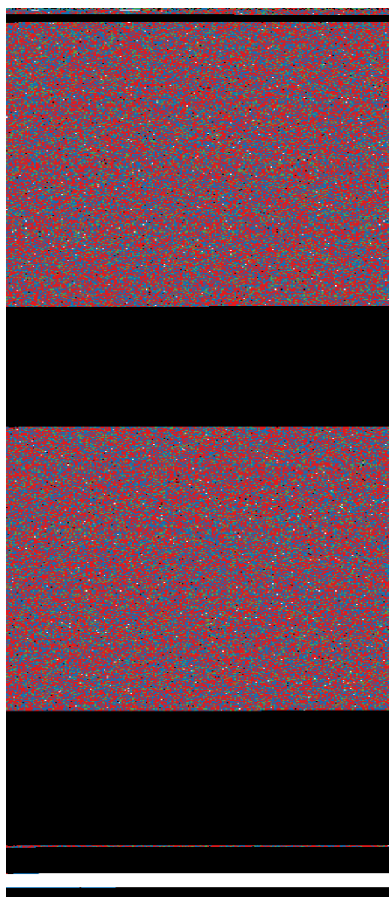


Figura 4.13 - Rappresentazione grafica di binvis.io del firmware

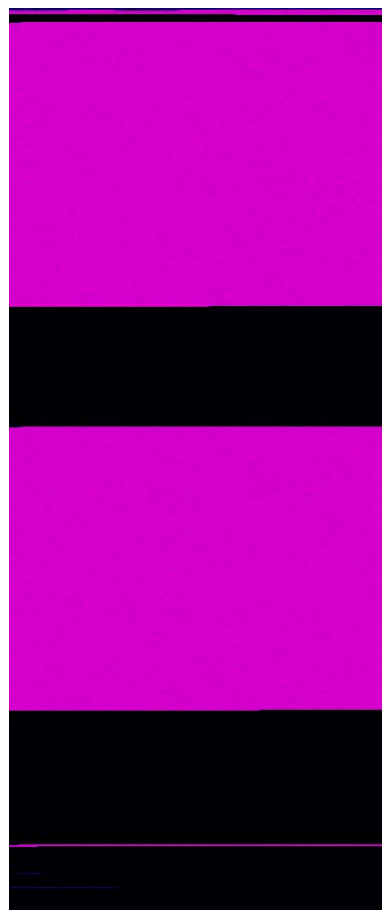


Figura 4.14 - Rappresentazione grafica dell'entropia di binvis.io del firmware. Ad un colore più "acceso" corrisponde un valore di entropia più alto



La Figura 4.14 rappresenta l'entropia del firmware (più il colore è "acceso" più l'entropia è alta) mostrandoci esattamente la stessa struttura di Figura 4.12. Dalla Figura 4.13 invece, tramite l'esplorazione mediante pagina web, notiamo oltre alle già note immagini di avvio e alla partizione di dati compressi anche una serie di stringhe presenti nella porzione finale del firmware. Tali stringhe non sono altro che le stesse stringhe visualizzate in precedenza (Figura 4.9 e Figura 4.10) e che è possibile visualizzarle all'interno del firmware con il seguente comando:

```
hexdump -C -s 0x7a0090 alice.bin
```

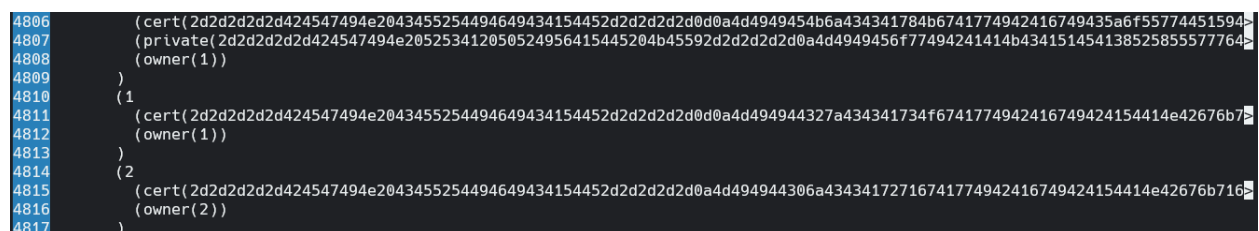
Concentriamo ora la nostra attenzione sulla porzione di dati compressi con Zlib. Tramite `binvis.io` capiamo che tale porzione di dati inizia all'offset `0x760000`. È possibile isolare tale porzione dal resto del firmware estrapolandola grazie al comando `dd` e specificando l'offset e la dimensione:

```
dd if=alice.bin of=760000.bin bs=17019 skip=7733248 count=1 iflag=skip_bytes
```

Procediamo estraendo grazie a `binwalk` le informazioni dal file appena creato:

```
binwalk -e 760000.bin
```

All'interno della cartella appena creata da `binwalk` in seguito alla decompressione si trova un file di testo che pare essere un altro file di configurazione del modem. Con un qualsiasi editor di testo controlliamo il contenuto del file e troviamo alcune informazioni interessanti: sono presenti quelli che potrebbero essere dei certificati ed una chiave privata (Figura 4.15), e delle password (Figura 4.18).



```
4806 (cert(2d2d2d2d424547494e2043455254494649434154452d2d2d2d0a4d4949454b6a434341784b6741774942416749435a6f55774451594b
4807 (private(2d2d2d2d424547494e205253412050524956415445204b45592d2d2d2d0a4d4949456f77494241414b434151454138525855577764
4808 (owner(1))
4809 )
4810 (1
4811 (cert(2d2d2d2d424547494e2043455254494649434154452d2d2d2d0a4d494944327a434341734f6741774942416749424154414e42676b71
4812 (owner(1))
4813 )
4814 (2
4815 (cert(2d2d2d2d424547494e2043455254494649434154452d2d2d2d0a4d494944306a43434172716741774942416749424154414e42676b716
4816 (owner(2))
4817 )
```

Figura 4.15 - Certificati e chiave privata presenti all'interno del file di configurazione

I certificati e la chiave sembrano essere scritti in esadecimale per cui è possibile riconvertirli da testo a file con il seguente comando:

```
echo '<stringa certificato>' | xxd -r -p > cert.pem
```

```

bongio PC-LUCA ../_760000.bin.extracted openssl x509 -in cert1.pem -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 26245 (0x6685)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C = IT, O = I.T. Telecom, OU = Servizi di certificazione, CN = I.T. Telecom Private CA
    Validity
      Not Before: Jun 26 10:36:42 2007 GMT
      Not After : Jun 25 10:36:35 2008 GMT
    Subject: C = IT, ST = ITALY, L = ROMA, O = TELECOM ITALIA SPA, OU = IT TELECOM, CN = homenet.telecomitalia.it, emailAddress = or.ap3@telecomitalia.it
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public-Key: (2048 bit)
      Modulus:
        00:f1:15:d4:5b:07:43:0f:9c:b1:e2:2a:b7:6d:c7:
        d5:cf:98:01:f3:03:ee:74:4e:97:ec:dc:60:37:ef:
        f4:75:29:b9:fd:f1:ad:7a:87:93:40:03:bd:b7:da:
        a1:66:6a:a8:7b:fd:70:f5:2c:11:4b:37:40:cd:2d:
        93:f7:e2:b4:49:b5:e8:bb:58:e5:d8:df:31:58:24:

```

Figura 4.16 - Visualizzazione dei dati di uno dei certificati presenti all'interno del file di configurazione

```

bongio PC-LUCA ../_760000.bin.extracted cat private_key.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA8RXUWdDD5yx4iq3bcfVz5gB8wPudE6X7NgN+/0dSm5/fGt
eoeTQA09t9qhZmqoe/1w9SwRSzdAzS2T9+K0SbXou1jL2N8xwCTHymQ0g0+shPyt
2NeegBgLiThQkmJ/NKWiCeyoDJumvSiloaAexmaE0Id/KfgJDt0hv34lB8Ly4Lt20
2Ak6QTW5PH1vvWwySE+8fd5KB1MrQIQdchYq8L5o0pI5f4iIRBnRIyLejcSYzOKi
W5b4gPwhw+RWkGEL1xgLiW6dVlVl+fr7kvqlLjN+mteztz+rRpoumM7NqFwosT
an1NfUa18uakhQzyZx0WtwH/xWYWsUUtCyo8wIDAQABAoIBABB4Gkqho14kqS43
FtLEDQIiczejtB1z5QMERvprRPrf/9muMdl/FQqLUkrB82USbA2rwnAmenWe3HI
xEvxy7khWSW70/BLxtwroNUPosICHXn0PPgM0sKrYc9RKvLWL9C85kh9ptt6SUQ6
sewxYluJXKNXw2B6TR/vLd1KZhvi1q00XMr002ck40G/Amk4JDYgmTRbRaFhcjMZ
4Qd6rF0QldZnWF10QUh50nipBssxoyUserwznz7B5HCFhs1NAjCNSBarIsJFiteL
+/wU2mksk9khntMKzAT8Qu2YAX/kGpqrZBKCMUjLIgyks5BKQ7RHVrJkjmA7U3/
GhXFpckCgYEA/1FzvL9LnL11EUXIbfI/vRhhWtKw766QL5ywWqDdMw5w0VAzReZS
gxKbmoB1p0uLC0e0IG6edD9ck+LSix0WOL267sTaKKuUeHoE9dCLyM5Qc6JbpymF
BTUBLQfjidvrqefIdNA0diWsSrZMyh5XzD3/mdi0sWwMq+8UZRDuL8CgYEA8bql
qTPPKGjNukhYtShhDvXnNw0b8S150pPLA0ySeN3LvJk4boxHgRAmgCSgUdv3qSZu
TSprlh/SQzVuqsorUX6axjLP9PVM9TbWfcQ3BoPLXv4IDsBLNA9Gtjz22ofV16RN
BLR4TxroFZpQmpjM3IM5CaXrTRbG9PN1oWacyM0CgYACzjHCn8qGgH0quzcCrLK
QM/rEU5JHGbP1CvJbdDG3PD5jJTCJdayVw0bN1VAtLMxS1dubUyPTP7C6VYVvtL5
/93xRmBeqk0L8qhQm4D13Q0SnsS8bWdGn55MwFGBokBKYQ4qr7aGPhXcMAk13JqH
Po9x6hMo2I8mstEKWzTfvQKBgQCS01VIHJ1vVqdNz8KJRlCEaRD9Zz6sKYNN7bL
bLHVagD9UjsXNSl0fC6uSnnoz74sx6DmcYkIz+NE4SghjBHS5QIAcHC5QUPC+99a
345iFJxHc0my8tTGP4+JYyv6Wz4T68Lj28EEKcTIFmFShimgJIn2uja0Ndk6CaXZ
c9FY6QKBgBTeeCMPknEX2YPLyT5fMbc5U44G8ZkWUbdBxq0xdxeKLNJ0gmtRG0g
Gf2giJ+Kt99TqmAOSH134NQY+mjddgUpDxdH3tHKFwR6UpeJJen4uMaWmktPddY
qiJ8lupzIrYLx8U67GYL9ZE6/qLiC/zax7XCuHnC+3YQLAcuwrI
-----END RSA PRIVATE KEY-----

```

Figura 4.17 - Chiave privata presente all'interno del file di configurazione

Come si può notare dalla Figura 4.18, le password presenti nel file non sono però in chiaro.

```

bongio PC-LUCA ../_760000.bin.extracted grep password 98
(password(&b7;`X&b3;&99;&18;T&e2;&b6;&b2;))
(password(&b7;X&5c;&b9;&a2;))
(password(&9f;+&b6;&a6;&df;c&9d;&be;&ba;u&a0;&05;8&ae;&28;F&07;t&0c;&ff;&0c;&fe;&cf;))
(conn_req_password())

```

Figura 4.18 - Password presenti nel file di configurazione

Dopo una veloce analisi attraverso vari strumenti presenti online arriviamo alla conclusione che l'entropia delle password è troppo bassa per trattarsi di password cifrate e non rispettano alcuno standard di codifica. L'unica opzione rimanente è che le password siano state offuscate<sup>14</sup> con un qualche algoritmo. Infatti, sul manuale dell'OpenRG<sup>15</sup> (paragrafo 18.3) viene spiegato che le informazioni segrete (come le password) vengono salvate sulla memoria flash e che per evitare che queste siano leggibili ma allo stesso tempo semplici da cifrare e decifrare viene utilizzata una funzione di offuscamento. Questa procedura, come

<sup>14</sup> [https://en.wikipedia.org/wiki/Obfuscation\\_\(software\)](https://en.wikipedia.org/wiki/Obfuscation_(software))

<sup>15</sup> <http://docshare04.docshare.tips/files/21115/211152344.pdf>

spiega il manuale, non è veramente sicura, infatti, un utente è riuscito a risalire e ad invertire l'algoritmo con cui le password sono state oscurate<sup>16</sup> permettendoci di leggere in chiaro le password presenti nel file.

<b>Username</b>	<b>Password offuscata</b>	<b>Password in chiaro</b>
N0708227HMW-001737@#drate#.#realm#	&b7;`X&b3;&99;&18;T&e2;&b6;&b2;	alicenewag
admin	&b7;X&5c;&b9;&a2;	admin
001737-IDF-AGBasAdv	&9f;+&b6;&a6;&df;c&9d;&be;&ba;u&a0;&05;8&ae;&28;F&07;t&0c;&ff;&0c;&fe;&cf;	I74fr6t2iordj76tIU78d930

*Tabella 3 - Credenziali presenti all'interno di un file di configurazione. Le password erano state offuscate tramite un apposito algoritmo*

A questo punto dell'analisi si procederebbe con l'estrazione del filesystem e la sua successiva esplorazione con il supporto dei vari strumenti ma si sono verificati degli errori che non permettono la corretta estrazione.

## 4.2 FASTWEB HOME ACCESS GATEWAY



*Figura 4.19 - Fastweb Home Access Gateway*

### 4.2.1 Raccolta delle informazioni

Questo modem è stato prodotto dalla Telsey e distribuito in Italia dalla Fastweb intorno la fine degli anni 2000. Trattandosi anche in questo caso di un dispositivo datato, non è ovviamente disponibile alcuna pagina web né tantomeno la possibilità di scaricare il firmware. In questo caso, però, è presente una pagina informativa sul sito di OpenWRT<sup>17</sup> (un famoso sistema per router open source).

<sup>16</sup> <http://www.zibri.org/2009/11/obfuscation-will-never-work.html>

<sup>17</sup> <https://oldwiki.archive.openwrt.org/toh/telsey/cpa-znte60t>

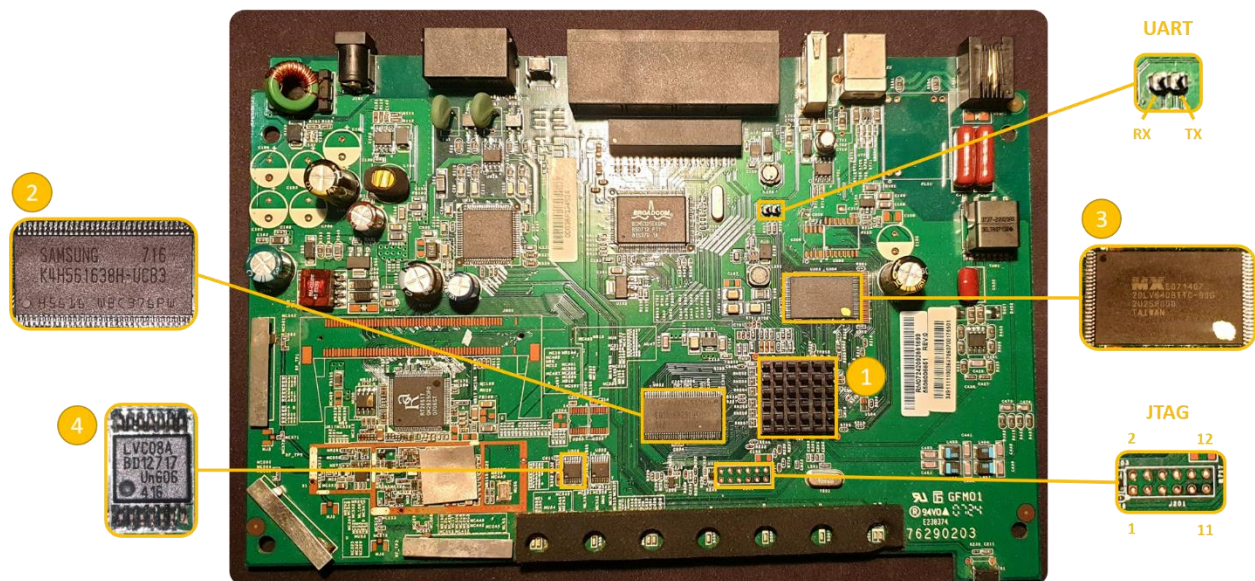


Figura 4.20 - Scheda principale del Fastweb Home Access Gateway con evidenziate le componenti e le interfacce

Dalla pagina e dalla scheda del dispositivo in Figura 4.20 è possibile ricavare informazioni sui circuiti integrati presenti:

1. Il SoC è un BCM6358 con una CPU dual core BMIPS che opera a 300MHz
2. La RAM è una SAMSUNG K4H561638H di dimensioni di 32 Mb
3. La memoria flash è una Macronix 29LV640D da 8 Mb
4. L'integrato LVC08A (contenente porte logiche and) può essere utile come punto di riferimento per la tensione di alimentazione ( $V_{CC}$ ) dato che supporta una tensione compresa nell'intervallo 1.65-3.6 V

Dalla pagina web veniamo anche a conoscenza della collocazione dell'interfaccia UART e dell'interfaccia JTAG. Procediamo quindi a collegare il raspberry all'interfaccia seriale seguendo le indicazioni trovate nella pagina web.

#### 4.2.2 Analisi del bootloader

Durante il processo di boot non vengono scritte sull'interfaccia seriale informazioni rilevanti, se non che il bootloader utilizzato è Common Firmware Environment (CFE)<sup>18</sup>, e al suo termine non viene visualizzato alcun prompt di accesso. Procediamo con l'accesso al bootloader come nel caso precedente.

<sup>18</sup> [https://en.wikipedia.org/wiki/Common\\_Firmware\\_Environment](https://en.wikipedia.org/wiki/Common_Firmware_Environment)

```

CFE> help
Available commands:

w          Write the whole image start from beginning of the flash
e          Erase [n]vram or [a]ll flash except bootrom
r          Run program from flash image or from host depend on [f/h] flag
p          Print boot line and board parameter info
c          Change bootline parameters
f          Write image to the flash
m          test mac client
i          Erase persistent storage data
b          Change board parameters
d          Set default mac address
reset      Reset the board
flashimage Flashes a compressed image after the bootloader.
help       Obtain help for CFE commands

For more information about a command, enter 'help command-name'
*** command status = 0

```

Figura 4.21 - Lista dei comandi disponibili all'interno del bootloader CFE. La lista dei comandi non è completa

Purtroppo, come si può notare in Figura 4.21, in questo caso la lista dei comandi disponibili è molto ridotta e non è presente alcun comando per leggere la memoria. L'unica soluzione rimasta è controllare se l'interfaccia JTAG è rimasta abilitata dalla fase di testing del dispositivo.

### 4.2.3 Identificazione dell'interfaccia JTAG

In questo caso nella fase di raccolta delle informazioni abbiamo scoperto che in questa scheda è predisposta un'interfaccia JTAG ma non sappiamo né la sua collocazione né la disposizione di pin. È possibile, però, notare nelle vicinanze del SoC e della RAM dodici pin placcati disposti su due file che sono dei buoni candidati.

A questo punto si seguono gli stessi passaggi descritti nel paragrafo 4.1.2 per individuare la disposizione dei pin tra quelle più comuni. Similarmente al caso precedente, useremo come riferimento per  $V_{cc}$  l'alimentazione dell'integrato LVC08A.

Pin	RGND ( $\Omega$ )	RVcc ( $\Omega$ )	V (V)	Note	Pin	RGND ( $\Omega$ )	RVcc ( $\Omega$ )	V (V)	Note
1	1.78K	1.71K	3.28		2	0	74.6	0	GND
3	1.83K	1.76K	3.38		4	0	74.6	0	GND
5	1.85K	1.78K	0		6	0	74.6	0	GND
7	1.83K	1.76K	3.38		8	0	74.6	0	GND
9	1.82K	1.75K	2.90		10	0	74.6	0	GND
11	10K	9.98K	3.36		12	0	74.6	0	GND

Tabella 4 - Misurazioni di resistenze e tensioni per ciascun pin dell'interfaccia JTAG

Dalle misurazioni riportate nella Tabella 4 è possibile individuare i pin connessi a GND poiché presentano una resistenza nulla rispetto a GND ed una tensione nulla. Su tutti gli altri pin è impossibile fare ipotesi poiché non si hanno informazioni a sufficienza. Tuttavia, anche

con solo i pin GND identificati è possibile riconoscere uno standard comune, si tratta di una versione troncata del connettore MIPS EJTAG<sup>19</sup> come riporta un'altra pagina di OpenWRT<sup>20</sup>.

<i>Pin</i>	<i>JTAG</i>	<i>Pin</i>	<i>JTAG</i>
1	nTRST	2	GND
3	TDI	4	GND
5	TDO	6	GND
7	TMS	8	GND
9	TCK	10	GND
11	nSRST	12	GND

*Tabella 5 - Pinout finale dell'interfaccia JTAG*

#### 4.2.4 Connessione all'interfaccia JTAG

Per connettersi all'interfaccia è sempre possibile farlo tramite il raspberry e l'utilizzo del software UrJTAG<sup>21</sup>. Una volta avviato il software è necessario indicare a quali pin gpio siano collegati i pin dell'interfaccia JTAG e tramite il comando `detect` avviare il riconoscimento del SoC collegato. Purtroppo, non è stato possibile riconoscere alcun dispositivo (Figura 4.22): molto probabilmente l'interfaccia JTAG è stata disabilitata una volta superata la fase di testing ed entrata in quella di produzione.

```
pi@raspberrypi:~ $ sudo jtag
UrJTAG 0.10 #2007
Copyright (C) 2002, 2003 ETC s.r.o.
Copyright (C) 2007, 2008, 2009 Kolja Waschk and the respective authors

UrJTAG is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
There is absolutely no warranty for UrJTAG.

warning: UrJTAG may damage your hardware!
Type "quit" to exit, "help" for help.

jtag> cable gpio tdi=23 tdo=24 tck=25 tms=18
Initializing GPIO JTAG Chain
jtag> detect
error: not found: queue is empty
```

*Figura 4.22 - Avvio e configurazione del software UrJTAG. Non viene riconosciuto alcun dispositivo collegato all'interfaccia*

<sup>19</sup> <http://www.jtagtest.com/pinouts/ejtag>

<sup>20</sup> <https://oldwiki.archive.openwrt.org/doc/hardware/port.jtag>

<sup>21</sup> <http://urjtag.org/>



## 5 LE CONTROMISURE DA ADOTTARE

---

In questo capitolo analizzeremo quali sono le contromisure che dovrebbero essere normalmente adottate per evitare sia l'estrazione che l'analisi del firmware. Utilizzeremo come esempio come è stata implementata la sicurezza all'interno del SoC ESP32<sup>22</sup>, divenuto uno dei più utilizzati in ambito IoT grazie al suo costo ridotto. Verrà trattata solo superficialmente, per cui per un'analisi approfondita si rimanda alla tesi di András Sándor Gedeon [24].

### 5.1 PROTEGGERSI DALL'ESTRAZIONE DEL FIRMWARE

Per evitare l'analisi del firmware è innanzi tutto necessario evitare che tale firmware possa essere estratto con uno dei metodi che sono stati descritti precedentemente nel paragrafo 3.2.

#### 5.1.1 Aggiornamenti remoti sicuri

Gli aggiornamenti del firmware remoti (aggiornamenti OTA) permettono ad un dispositivo di collegarsi ad un server per controllare se sia presente un aggiornamento per il firmware e in tal caso di sostituire quello precedente. È essenziale, però, che in questa fase delicata solo i direttamente interessati (dispositivo e server di aggiornamento) possano leggere il contenuto dei dati in transito. Per fare ciò è sufficiente stabilire una connessione sicura come il *Transport Layer Security* (TLS) che ci garantisce confidenzialità, autenticità ed integrità, a patto che un certificato auto-firmato del server di aggiornamento sia presente all'interno del dispositivo (memorizzato in modo sicuro). Per riassumere, durante un aggiornamento avvengono i seguenti passaggi:

1. Il dispositivo si collega al server di aggiornamento
2. Compara il certificato del server con quello salvato all'interno del dispositivo
3. Ottiene attraverso la connessione sicura appena instaurata il numero dell'ultima versione disponibile
4. Controlla se tale numero è più recente della versione attualmente installata
5. Se sì, scarica l'aggiornamento

L'ESP32 implementa questo schema per gli aggiornamenti<sup>23</sup> aggiungendo, inoltre, la capacità di effettuare automaticamente un rollback del firmware nel caso in cui qualcosa vada storto durante l'aggiornamento mediante l'utilizzo di più partizioni all'interno del chip identificate come OTA Data Partition.

Un altro servizio offerto dall'ESP32 è un meccanismo di anti-rollback che è in grado, grazie

---

<sup>22</sup> <https://www.espressif.com/en/products/socs/esp32>

<sup>23</sup> <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html>

all'uso di alcuni eFuse<sup>24</sup>, di evitare che si possa aggiornare un firmware ad una versione precedente (tipicamente meno sicura).

Sui dispositivi che utilizzano protocolli come il *Message Queuing Telemetry Transport* (MQTT) si può incrementare ulteriormente la sicurezza degli aggiornamenti utilizzando lo stesso protocollo MQTT per distribuire gli aggiornamenti<sup>25</sup>. Infatti, creare su questi dispositivi una connessione differente (per esempio HTTPS) da quella già utilizzata, offrirebbe ai possibili attaccanti un modo immediato per riconoscere la connessione sulla quale sta avvenendo un aggiornamento, e inoltre, eviterebbe di dover inserire all'interno del firmware le librerie necessarie per il nuovo protocollo risparmiando così memoria e risorse.

### 5.1.2 Disabilitare le interfacce di debug

Grazie all'utilizzo di aggiornamenti sicuri si può ostacolare l'estrazione del firmware remotamente, tuttavia potrebbe essere ancora possibile effettuarla fisicamente. Come abbiamo visto nel paragrafo 0, è relativamente facile estrarre un firmware dalle interfacce di debug. Per risolvere il problema basterebbe utilizzare i meccanismi offerti dai vari SoC per, una volta terminata la fase di testing, disabilitare le interfacce come UART e JTAG via software oppure in maniera permanente via "hardware" (eFuse). Se ciò non fosse possibile una soluzione appena sufficiente sarebbe rendere inaccessibili (o difficilmente raggiungibili) i pin delle interfacce e rimuovere i comandi dal bootloader che permettono l'estrazione della memoria (v. 4.2.2).

Nell'ESP32 se viene attivata la cifratura e/o il secure boot (descritti di seguito) all'avvio successivo viene di default disabilitata permanentemente l'interfaccia JTAG.

Tuttavia, rendere sicuri gli aggiornamenti e disabilitare le interfacce di debug non è sufficiente poiché esistono altri metodi per estrarre il firmware da cui è impossibile difendersi (v. 3.2.2.3).

## 5.2 SECURE BOOT

Il secure boot è progettato per proteggere un dispositivo dall'esecuzione di codice malevolo assicurandosi che solo software autenticato possa essere eseguito sul dispositivo. Per fare ciò è necessario che si crei una catena di fiducia (chain of trust) fra i vari componenti e stadi che compongono il processo di boot in modo che ognuno sia verificato da quello precedente. Alla base di questa catena è necessario che vi sia una radice immodificabile su cui si basa l'intera catena. Questa radice è solitamente rappresentata da una *Read-Only Memory* (ROM) o, come nel caso dell'ESP32 tramite l'uso di eFuse, da una memoria *One Time Programmable*

---

<sup>24</sup> Un microscopico fusibile elettronico utilizzato per implementare le memorie *One Time Programmable* (OTP) [https://en.wikipedia.org/wiki/Programmable\\_ROM](https://en.wikipedia.org/wiki/Programmable_ROM)

<sup>25</sup> Per esempio, il servizio AWS IoT offre questa possibilità: <https://docs.aws.amazon.com/freertos/latest/userguide/freertos-ota-dev.html>



(OTP) nella quale viene salvata una chiave. Tale chiave verificherà l'autenticità del bootloader che a sua volta verificherà l'autenticità del firmware.

Nella prima versione<sup>26</sup>, il secure boot dell'ESP32 si basava sulla memorizzazione di una chiave simmetrica (AES), generata da un *Random Number Generator* (RNG) via hardware e memorizzata in maniera sicura tramite gli eFuse, e di una impronta (digest) derivante dall'unione della suddetta chiave, un vettore di inizializzazione e il contenuto dell'immagine del bootloader. Tali eFuse, una volta generata e memorizzata la chiave, risultano inaccessibili via software. Purtroppo, non è lo stesso dal punto di vista hardware: utilizzando un voltage glitching (un tipo di fault injection) è stato infatti possibile estrarre la chiave di sicurezza permettendo di eseguire codice non verificato [25] [26]. Ciò è stato possibile a causa della natura simmetrica della chiave, per questo motivo è stata creata una nuova versione del secure boot<sup>27</sup> dove al posto di una cifratura simmetrica viene invece utilizzata una cifratura a chiavi asimmetriche, memorizzando la chiave pubblica all'interno del dispositivo. In questo modo, non è necessario che tale chiave resti segreta poiché serve esclusivamente a verificare la firma effettuata con la chiave privata (che invece dovrà essere mantenuta segreta e al di fuori del dispositivo).

### 5.3 CIFRATURA DEL FIRMWARE

La cifratura è un processo essenziale per evitare che il firmware possa essere analizzato. All'interno del dispositivo viene memorizzata la chiave che verrà utilizzata all'avvio per decifrare il contenuto del firmware.

Nel caso dell'ESP32<sup>28</sup> la chiave è del tipo AES-256 e può essere generata dall'RNG presente all'interno del SoC oppure può essere fornita esternamente. In tal caso è importante ricordare che è buona pratica non riutilizzare la stessa chiave per più dispositivi in modo che i dati cifrati non possano essere copiati da un dispositivo ad un altro. L'ESP32 offre inoltre due modalità di cifratura: la modalità sviluppo, che permette ancora di poter disabilitare la cifratura fino a tre volte e di effettuare la cifratura attraverso la UART, e la modalità pubblicazione, dove invece gli unici aggiornamenti che il firmware potrà ricevere dovranno essere effettuati esclusivamente attraverso gli aggiornamenti OTA. In entrambe le modalità viene disabilitato di default il debug attraverso l'interfaccia JTAG.

Come illustrato nel paragrafo 3.3.2.3, è necessario prestare attenzione al contesto nel quale viene applicata la cifratura. Per esempio, in un modello di router della DLINK è stato possibile decifrare un firmware cifrato grazie ad una versione precedente contenente la funzione di decifrazione<sup>29</sup>, esattamente lo scenario descritto dalla Figura 3.8.

Un altro episodio degno di nomina riguarda il firmware di un router della Netgear la cui cifratura non si è rivelata sufficientemente robusta. Questo perché la cifratura era basata

---

<sup>26</sup> <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/secure-boot-v1.html>

<sup>27</sup> <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/secure-boot-v2.html>

<sup>28</sup> <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/flash-encryption.html>

<sup>29</sup> <https://nstarke.github.io/0036-decrypting-dlink-proprietary-firmware-images.html>

principalmente sull'operazione di XOR che è vulnerabile ad attacchi statistici<sup>30</sup>. Netgear, in seguito, ha rilasciato un comunicato sulla sicurezza nel quale, utilizzando il Common Vulnerability Scoring System Vector (CVSS)<sup>31</sup>, afferma che tale vulnerabilità non ha avuto un impatto effettivo sull'usabilità e la sicurezza del dispositivo e che non è possibile cambiare il meccanismo di cifratura senza che questo venga identificato<sup>32</sup>. Questo perché ci si ricondurrebbe allo scenario descritto in Figura 3.10 dove, però, grazie alla conoscenza della prima versione della funzione di decifrazione sarebbe possibile scoprire la nuova versione della funzione rendendo così l'intero aggiornamento inutile.

---

<sup>30</sup> <https://www.pentestpartners.com/security-blog/breaking-bad-firmware-encryption-case-study-on-the-netgear-nighthawk-m1/>

<sup>31</sup> <https://www.first.org/cvss/>

<sup>32</sup> <https://kb.netgear.com/000061155/Security-Advisory-for-Firmware-Encryption-on-MR1100-PSV-2019-0055>

## 6 CONCLUSIONI

---

In questa attività progettuale si è descritto, nel modo più completo possibile, il processo col quale si conduce l'estrazione e la successiva analisi statica di un firmware. Si è appreso quali siano le reali difficoltà che un analista deve superare e quali siano gli strumenti che lo supportano durante il processo, ma si è anche analizzato come uno sviluppatore possa proteggersi da tali attacchi attuando proprio quelle pratiche.

Anche se può sembrare che i casi pratici presi in esame abbiano portato ad un fallimento (per motivi diversi), in realtà hanno dimostrato sia da un lato quanto sia semplice minacciare la sicurezza di un dispositivo (v. 4.1), utilizzando delle attrezzature piuttosto economiche, sia dall'altro quando sia altrettanto semplice proteggersi da tali attacchi (v. 4.2).

Si è mostrato come ormai la sicurezza sia già presente all'interno dei chip che vengono utilizzati nella gran parte dei dispositivi IoT lasciando allo sviluppatore solo il compito di utilizzare tale sicurezza. Bisogna comunque, però, prestare attenzione a come vengono implementati questi metodi di protezione, in quanto si è dimostrato più volte come una cattiva implementazione abbia portato a falle nella sicurezza.

## BIBLIOGRAFIA

---

- [1] «Espressif Achieves the 100-Million Target for IoT Chip Shipments,» Espressif, 2 Gennaio 2018. [Online]. Available: [https://www.espressif.com/en/news/Espressif\\_Achieves\\_the\\_Hundredmillion\\_Target\\_for\\_IoT\\_Chip\\_Shipments](https://www.espressif.com/en/news/Espressif_Achieves_the_Hundredmillion_Target_for_IoT_Chip_Shipments). [Consultato il giorno 27 Marzo 2021].
- [2] «Open Web Application Security Project (OWASP),» The OWASP Foundation, [Online]. Available: <https://owasp.org/>. [Consultato il giorno 2 Aprile 2021].
- [3] «OWASP Internet of Things,» The OWASP Foundation, [Online]. Available: <https://owasp.org/www-project-internet-of-things/>. [Consultato il giorno 2 Aprile 2021].
- [4] «OWASP Firmware Security Testing Methodology,» The OWASP Foundation, 15 Maggio 2020. [Online]. Available: <https://scriptingxss.gitbook.io/firmware-security-testing-methodology/>. [Consultato il giorno 30 Marzo 2021].
- [5] «IoTGoat,» OWASP, [Online]. Available: <https://github.com/OWASP/IoTGoat>. [Consultato il giorno 2 Aprile 2021].
- [6] «OWASP IoT Top 10 2018,» [Online]. Available: <https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf>. [Consultato il giorno 2 Aprile 2021].
- [7] S. Vasile, D. Oswald e T. Chothia, *Breaking all the Things - A Systematic Survey of Firmware Extraction Techniques for IoT*, University of Birming.
- [8] J. Grand, «JTAGulator,» Gran Idea Studio, [Online]. Available: <http://www.grandideastudio.com/jtagulator/>. [Consultato il giorno 26 Marzo 2021].
- [9] J. Zaddach e A. Costin, «Embedded Devices Security and Firmware Reverse Engineering,» *Black-Hat USA*, 2013.
- [10] R. Labs, «Binwalk,» [Online]. Available: <https://github.com/ReFirmLabs/binwalk>. [Consultato il giorno 27 Marzo 2021].
- [11] A. Cortesi, «binvis.io,» [Online]. Available: <http://binvis.io/#/>. [Consultato il giorno 28 Marzo 2021].
- [12] «Differentiate Encryption From Compression Using Math,» 12 Giugno 2013. [Online]. Available: <http://www.devtys0.com/2013/06/differentiate-encryption-from-compression-using-math/>. [Consultato il giorno 29 Marzo 2021].

- [13] V. Lee, «Dealing with encrypted router firmware,» 6 Febbraio 2020. [Online]. Available: <https://www.thezdi.com/blog/2020/2/6/mindshare-dealing-with-encrypted-router-firmware>. [Consultato il giorno 29 Marzo 2021].
- [14] Hex-Rays, «IDA homepage,» [Online]. Available: <https://www.hex-rays.com/products/ida/>. [Consultato il giorno 30 Marzo 2021].
- [15] N. R. Directorate, «Ghidra homepage,» [Online]. Available: <https://ghidra-sre.org/>. [Consultato il giorno 30 Marzo 2021].
- [16] «radare2 homepage,» [Online]. Available: <https://rada.re/n/radare2.html>. [Consultato il giorno 30 Marzo 2021].
- [17] «Cutter homepage,» [Online]. Available: <https://cutter.re/>. [Consultato il giorno 30 Marzo 2021].
- [18] H. Rays, «IDA F.L.I.R.T. Technology: In-Depth,» [Online]. Available: [https://www.hex-rays.com/products/ida/tech/flirt/in\\_depth/](https://www.hex-rays.com/products/ida/tech/flirt/in_depth/). [Consultato il giorno 30 Marzo 2021].
- [19] «Automate .fidb generation with headless Ghidra,» 20 Settembre 2019. [Online]. Available: <https://blog.threatrack.de/2019/09/20/ghidra-fid-generator/#function-id>. [Consultato il giorno 30 Marzo 2021].
- [20] J. Calvet, «JEB in Action - Native Signatures Generation,» 3 Maggio 2019. [Online]. Available: <https://www.pnfsoftware.com/blog/siglibgen-native-signatures-generation-for-jeb/>. [Consultato il giorno 30 Marzo 2021].
- [21] «firmwalker,» [Online]. Available: <https://github.com/craigz28/firmwalker>. [Consultato il giorno 30 Marzo 2021].
- [22] «The Firmware Analysis and Comparison Tool (FACT),» [Online]. Available: [https://github.com/fkie-cad/FACT\\_core](https://github.com/fkie-cad/FACT_core). [Consultato il giorno 30 Marzo 2021].
- [23] N. Patavalis, «picocom,» [Online]. Available: <https://github.com/npat-efault/picocom>. [Consultato il giorno 3 Aprile 2021].
- [24] A. S. Gedeon, L. Buttyán e D. F. Papp, «Secure boot and firmware update on a microcontroller-based embedded board,» 2020.
- [25] Espressif, «Espressif Security Advisory Concerning Fault Injection and Secure Boot (CVE-2019-15894),» 3 Settembre 2019. [Online]. Available: [https://www.espressif.com/en/news/Espressif\\_Security\\_Advisory\\_Concerning\\_Fault\\_Injection\\_and\\_Secure\\_Boot](https://www.espressif.com/en/news/Espressif_Security_Advisory_Concerning_Fault_Injection_and_Secure_Boot). [Consultato il giorno 6 Aprile 2021].

- [26] «Pwn the ESP32 Forever: Flash Encryption and Sec. Boot Keys Extraction,» 13 Novembre 2019. [Online]. Available: <https://limitedresults.com/2019/11/pwn-the-esp32-forever-flash-encryption-and-sec-boot-keys-extraction/>. [Consultato il giorno 6 Aprile 2021].