PENTEST DE UN DISPOSITIVO IOT: EXPLOTACIÓN DE VULNERABILIDADES DE UNA BOMBILLA INTELIGENTE

PENTEST OF AN IOT DEVICE: EXPLOITING VULNERABILITIES OF A SMART BULB

TRABAJO DE FIN DE MÁSTER EN IOT
DEPARTAMENTO DE INFORMÁTICA

AUTOR
ESTEBAN BONILLA RODRÍGUEZ

DIRECTOR
JOAQUÍN RECAS PIORNO
GUILLERMO BOTELLA JUAN

CONVOCATORIA: SEPTIEMBRE 2021
CALIFICACIÓN: 7,5

MÁSTER EN IOT
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID
29 DE SEPTIEMBRE DE 2021

DEDICATORIA

A todos aquellos que ponen su esfuerzo
en mejorar la seguridad informática, tarea
tan compleja como importante y que,
con ello, contribuyen a hacer de éste un
mundo más seguro y amable.

AGRADECIMIENTOS

A Ana por su cariño y apoyo constante sin el que no habría llegado hasta aquí. A Joaquín por su brillante dirección, por guiarme con su conocimiento e indicarme siempre el camino. A Raquel por su generosa disponibilidad. A José por su desinteresada ayuda. A mi familia por animarme y creer siempre en mí.

RESUMEN

Pentest de un dispositivo IoT: Explotación de vulnerabilidades de una bombilla inteligente

El objetivo de este trabajo fin de Máster es realizar una auditoría de seguridad o test de penetración (*penetration testing - pentesting)* o sobre un dispositivo IoT: una bombilla inteligente (*SmartBulb*). Si bien una auditoría de seguridad engloba múltiples aspectos, el presente trabajo se ha enfocado en el ámbito de las comunicaciones utilizadas (*BLE – Bluetooth Low Energy*). Para ello se han analizado dos casos diferentes: uno en el que el dispositivo sigue los estándares BLE y se exponen sus servicios de forma sencilla, fácilmente explotable, y otro también basado en dichos estándares, pero con mecanismos adicionales que añaden seguridad al sistema y que demuestran qué elementos considerar para poder aumentar la seguridad en este ámbito.

Palabras clave

IoT, Pentesting, Seguridad, BLE, SmartBulb.

ABSTRACT

Pentest of an IoT device: Exploiting vulnerabilities of a smart bulb

The objective of this Master's thesis is to perform a security audit or penetration testing (*pentesting*) on an IoT device: a SmartBulb. Although a security audit has multiple aspects, this work is focused on the area of communications used by the device (*Bluetooth Low Energy - BLE*). For this, two different cases have been analyzed: one in which the device follows the BLE standards for the exposure of its services in a simple way, easily exploitable, and another also based on these standards, but with other additional mechanisms that add security to the system, which demonstrates what type of elements can be considered to add security to the system in this context.

Keywords

IoT, Pentesting, Security, BLE, SmartBulb.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

El término Internet de las cosas (IoT – Internet of Things) fue acuñado por Kevin Ashton ya en 1999 [1], pero sigue muy vigente en los actuales sistemas de información. Describió un mundo donde todas las 'cosas' tienen una identidad digital por sí mismas y se organizan y administran por medio de computadoras – en lo que se pueden denominar 'Sistemas IoT'.

Desde otra perspectiva, el IoT es un entorno donde estas 'cosas' tienen la capacidad de detectar, controlar, enviar y transferir datos a otros elementos y donde los datos recopilados serán visibles a través de diferentes aplicaciones.

Por tanto, a la hora de diseñar y desarrollar un sistema IoT se han de tener en cuenta múltiples elementos, incluyendo: qué dispositivos y sensores hardware utilizar (y con qué firmware), qué protocolos de comunicación usar entre los diferentes componentes o qué procesamiento (middleware) y datos se van a exponer a más alto nivel (aplicaciones).

En la siguiente imagen, se puede ver un ejemplo de arquitectura de un sistema IoT, a alto nivel:

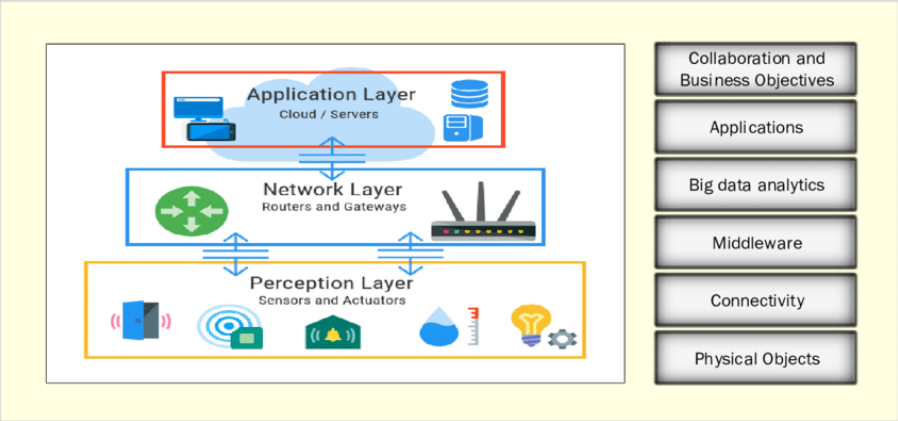


Ilustración 1: IoT - ejemplo de Arquitectura [2]

En esta arquitectura se observan tres capas diferenciadas:

- capa de percepción: que incluye los dispositivos físicos como son los sensores y actuadores.
- capa de red: en la que se puede incluir tanto la comunicación entre dispositivos, así como la comunicación con las aplicaciones de la capa superior (*middleware*, capa de comunicación entre los elementos hardware y las aplicaciones software).
- capa de aplicación: que en un entorno IoT pueden ser de diferente naturaleza según la necesidad que cubran (e.g. Smart City, Smart Home, Smart Lightning… etc.). Esta capa se puede apoyar o no en componentes de '*Big Data Analytics*'.

Los criterios de selección de unos u otros componentes en estas capas variará dependiendo del caso de uso, pero hay un aspecto que siempre es importante considerar, y en el que se centra este trabajo: la **seguridad**.

En sistemas complejos, con múltiples capas y componentes, como en el caso de los sistemas IoT, todos sus componentes pueden ser una fuente de riesgos de seguridad, siendo por tanto necesario hacer un análisis global del mismo (auditoría de seguridad o *'pentesting'*) para poder estimar con cierto grado de confianza si un sistema es seguro o no.

Existen organismos internacionales, como OWASP [3] - fundación internacional independiente sin ánimo de lucro, formada por diferentes empresas, organizaciones y comunidades académicas dedicada a la seguridad informática -, que se dedican a investigar sobre las diferentes vulnerabilidades en sistemas software. Además, proveen una serie de indicaciones e incluso herramientas para poder estudiar y mitigar dichas vulnerabilidades. OWASP en concreto facilita una lista de '*top10*' vulnerabilidades en IoT que sirve sin duda como muy buena referencia a la hora de realizar una auditoría de seguridad en un sistema IoT.

Las vulnerabilidades que aplican en cada momento son cambiantes (nuevos ataques y mitigaciones surgen a menudo), pero en general, y a alto nivel se pueden dividir según la naturaleza del componente analizado, incluyendo:

- Hardware (*chipset*, E/S…).
- Software (*firmware*, *web app, APIs*…).

- Comunicaciones (protocolos: *Wifi, BLE, Sigfox*…).

Por tanto, con el objetivo de identificar vulnerabilidades de un sistema o elemento concreto del mismo se puede realizar un **pentesting** (*'penetration testing')*. Un pentesting es un ataque a un sistema informático con la intención de encontrar sus debilidades de seguridad y todo lo que podría tener acceso a él, su funcionalidad y datos.

Un pentesting servirá tanto para ver si un sistema tiene vulnerabilidades de seguridad explotables, así como para identificar posibles soluciones para mejorar la seguridad del mismo.

1.2 Objetivos

El objetivo de este trabajo fin de Máster es realizar un pentesting o auditoría de seguridad sobre un dispositivo IoT – en este caso para una Bombilla inteligente (SmartBulb), de la cual se han considerado 2 modelos diferentes.

Ilustración 2: SmartBulb

Una bombilla 'inteligente' es simplemente una bombilla que tiene conectividad con una aplicación móvil, con la que se pueden controlar sus funcionalidades como por ejemplo el encendido/ apagado de la misma, o el cambio de color de su luz.

En una auditoría de seguridad completa habría que analizar todos los componentes del sistema, incluyendo tanto el hardware como el firmware de la

bombilla, lo cual es una tarea compleja, normalmente realizada por un equipo multidisciplinar. Este trabajo se ha centrado sobre todo en el ámbito de las comunicaciones utilizadas por la bombilla (BLE o Bluetooth Low Energy) así como en una parte del código de la aplicación móvil que controla dicha bombilla.

Los objetivos de este *pentesting* son, por un lado, saber si el dispositivo es explotable o tiene alguna vulnerabilidad, y, por otro lado, conociendo dicha vulnerabilidad, poder identificar posibles soluciones o mitigaciones para evitarla, y poder contar con mecanismos para desarrollar sistemas más seguros en el ámbito estudiado.

1.3 Plan de trabajo

El *pentesting* realizado en este trabajo, ha sido realizado basándose en el manual '*IoT Exploitation Lab*' de la empresa ***Attify*** [4]. Este manual facilita en gran medida la realización de *pentesting* en diferentes ámbitos al indicar cuáles son los pasos –normalmente indicados a alto nivel - que hay que realizar para completarlo.

Ilustración 3: logo de Attify

Aparte del manual, en este 'kit' de *pentesting* se incluyen los dispositivos IoT sobre los que se va a realizar el *pentesting* (y otro tipo de dispositivos hardware si fueran necesarios).

En este caso se ha contado con 2 bombillas inteligentes con agujeros de seguridad conocidos y explicados en el manual, pero que como se ha visto durante las pruebas han sido corregidos en una de ellas (aunque en la otra no).

Los pasos a seguir, descritos en el manual para realizar este pentesting, se pueden resumir en:

1. Capturar el tráfico BLE entre la bombilla inteligente y la aplicación móvil al realizar una interacción con la bombilla desde la propia aplicación (e.g. cambio de color de la luz).
2. Analizar las trazas de comunicación, para descubrir qué características y servicios son escritos para realizar dicha interacción. Además, se busca si existe alguna lógica en la construcción de estos 'comandos de control' capturados en los que, por ejemplo, se cambia el color de la luz de la bombilla.
3. Realizar un *reply-attack* basado en la información anterior. Un *reply-attack* consiste en reenviar la información capturada, siguiendo el mismo formato, al dispositivo y comprobar si así se puede tomar el control del mismo.

Capítulo 2 - Estado de la cuestión

2.1.1 Auditoria de Seguridad

En sistemas complejos como los sistemas IoT, todos sus componentes pueden ser una fuente de riesgos de seguridad, siendo por tanto necesario hacer un análisis global del mismo - o auditoría de seguridad - para poder estimar con cierto grado de confianza si un sistema es seguro o no.

Una forma muy extendida de entender la seguridad en un sistema de información se refiere a los siguientes tres aspectos – también denominados 'triada de seguridad' [5] – que ha de seguir un sistema para considerarse seguro:

Ilustración 4: Triada de Seguridad

- Confidencialidad: solo aquellos que están autorizados tienen acceso a elementos concretos y a aquellos que no están autorizados se les impide el acceso de manera activa.
- Integridad: asegurarse que los datos no hayan sido manipulados y, por lo tanto, sean confiables. Que los datos sean: correctos, auténticos y confiables.
- Disponibilidad: en cuanto a que las redes, los sistemas y las aplicaciones estén siempre en funcionamiento. Garantizar que los usuarios autorizados tengan acceso seguro a los recursos cuando los necesiten.

Teniendo en cuenta los aspectos que definen la seguridad en un sistema de información, el siguiente paso es identificar cuáles son los posibles riesgos que afectan a

dicha seguridad. Para conocer cuáles son estos riesgos, existen múltiples empresas y organizaciones que analizan y evalúan constantemente los riesgos de seguridad actuales, e incluso facilitan herramientas para su gestión y dan indicaciones sobre posibles mitigaciones.

Entre estas empresas y organizaciones encontramos, por ejemplo:

- NIST *National Vulnerability Database* (NVD) [U.S.] [6]: Repositorio del gobierno de EE. UU. de datos de gestión de vulnerabilidades basados en estándares.
- OWASP [3]: entidad independiente para el análisis de vulnerabilidades software - especialmente interesante para este estudio por su apartado dedicado a IoT – por su importancia, se detalla a continuación el apartado 2.1.1.1 .
- Empresas de seguridad: *IoT Inspector* (interesante por su capacidad de realizar auditorías de seguridad de manera automatizada), *Attify* (entre otras, facilita información en forma de cursos, libros, manuales… etc. para realizar *pentesting*) [7]

A través de la información facilitada por estas organizaciones, se pueden identificar posibles riesgos referentes a varios ámbitos de la seguridad. El presente trabajo fin de Máster se apoya especialmente en una herramienta de *pentesting* facilitada por la empresa *Attify* – descrita en más detalle en el apartado 2.1.2 .

2.1.1.1 OWASP IoT

OWASP [3] es una fundación internacional sin ánimo de lucro, formada por diferentes empresas, organizaciones y comunidades académicas, que se dedican a investigar sobre las diferentes vulnerabilidades de los sistemas software. Además, provee una serie de indicaciones y herramientas para poder estudiar y mitigar dichas vulnerabilidades.

En principio esta organización surgió centrada en el desarrollo de software para aplicaciones web (OWASP de hecho quiere decir '*Open Web Application Security Project'*) pero actualmente cubre otros ámbitos como el IoT.

Una de las contribuciones fundamentales de OWASP es su lista de *'top10'* vulnerabilidades identificadas en un determinado ámbito, que evoluciona con el

tiempo. Aparte de listar estas vulnerabilidades también da indicaciones sobre qué actuaciones se pueden llevar a cabo para mitigarlas.

En el *top10* de vulnerabilidades IoT – actualizada a 2018 encontramos:

# Top10	Vulnerabilidad	Descripción
1	Contraseñas débiles, adivinables o '*harcodeadas*'	Uso de credenciales que se pueden adivinar fácilmente por fuerza bruta, disponibles públicamente o inalterables, inclusión de puertas traseras en el firmware o software de cliente que permiten el acceso no autorizado a los sistemas implementados.
2	Servicios de red inseguros	Servicios de red innecesarios o inseguros que se ejecutan en el propio dispositivo, especialmente aquellos expuestos a Internet, que comprometen la confidencialidad, integridad / autenticidad o disponibilidad de información o permiten el control remoto no autorizado.
3	Interfaces del ecosistema inseguras	Interfaces web, *backend API*, *cloud* o móviles inseguras en el ecosistema en torno al dispositivo, que permiten comprometer el dispositivo o sus componentes. Los problemas comunes incluyen la falta de autenticación / autorización, un cifrado débil o inexistente y la falta de filtrado en la entrada/salida.
4	Falta de mecanismo de actualización seguro	Falta de capacidad para actualizar de forma segura el dispositivo. Esto incluye la falta de validación del *firmware* en el dispositivo, la falta de entrega segura (sin cifrar en tránsito), la falta de mecanismos *anti-rollback* o la falta de notificaciones de cambios de seguridad debido a actualizaciones.

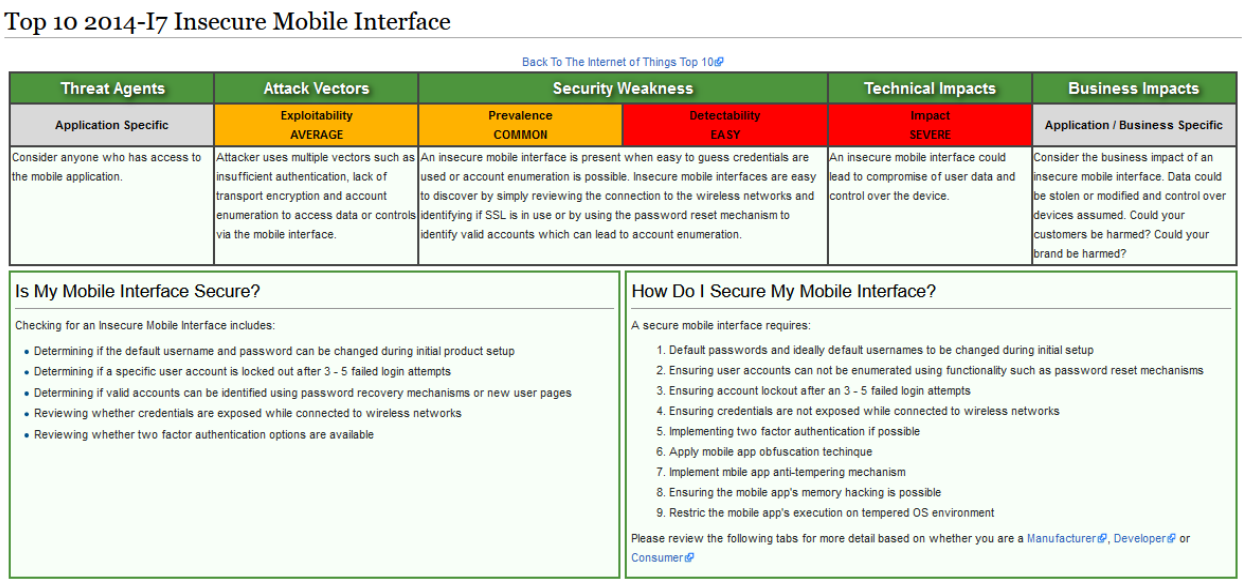
5	Uso de componentes inseguros u obsoletos	Uso de librerías / componentes de software obsoletos o inseguros que podrían permitir que el dispositivo se vea comprometido. Esto incluye la personalización insegura de las plataformas del sistema operativo y el uso de software o componentes de hardware de terceros de una cadena de suministro comprometida.
6	Protección de privacidad insuficiente	La información personal del usuario almacenada en el dispositivo o en el ecosistema que se usa de manera insegura, inapropiada o sin permiso.
7	Transferencia y almacenamiento de datos inseguro	Falta de cifrado o control de acceso de datos confidenciales en cualquier lugar del ecosistema, incluso en reposo, en tránsito o durante el procesamiento.
8	Falta de gestión de dispositivos	Falta de soporte de seguridad en los dispositivos desplegados en producción, incluida la gestión de activos, la gestión de actualizaciones, el desmantelamiento seguro, la supervisión de sistemas y las capacidades de respuesta.
9	Configuración por defecto insegura	Los dispositivos o sistemas que se envían con configuraciones predeterminadas inseguras o carecen de la capacidad de hacer que el sistema sea más seguro al impedir que los operadores modifiquen las configuraciones.
10	Falta de endurecimiento (*hardening*) físico	Falta de medidas de endurecimiento (*hardening*) físico, lo que permite a los atacantes potenciales obtener información confidencial que puede ayudar en un futuro ataque remoto o tomar el control local del dispositivo.

Tabla 2-1. OWASP top10 IoT [8]

Con respecto a este 'top10', es importante remarcar que este tipo de información es muy útil como guía a la hora de realizar un *pentest*. Nos indicará los puntos que, de partida, habría que analizar en un sistema, si bien, debido a su amplio espectro de cobertura este tipo de *pentesting* completos se suelen hacer por equipos multidisciplinares.

Este trabajo se centra en realizar un *pentesting* para identificar vulnerabilidades incluidas en el punto #3 de este 'top10' que se refiere a 'Interfaces del ecosistema inseguras'. En concreto esto sucede por el uso que se hace de las comunicaciones entre la aplicación móvil de gestión de las luces y la bombilla inteligente.

Para conocer más detalle sobre cómo se describe esta vulnerabilidad en OWASP y especialmente qué medidas se recomienda tomar al respecto, podemos referirnos a la propia página web (*wiki*) de OWASP [9].



Top 10 2014-I7 Insecure Mobile Interface

Back To The Internet of Things Top 10

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability AVERAGE	Prevalence COMMON	Detectability EASY	Impact SEVERE	Application / Business Specific
Consider anyone who has access to the mobile application.	Attacker uses multiple vectors such as insufficient authentication, lack of transport encryption and account enumeration to access data or controls via the mobile interface.	An insecure mobile interface is present when easy to guess credentials are used or account enumeration is possible. Insecure mobile interfaces are easy to discover by simply reviewing the connection to the wireless networks and identifying if SSL is in use or by using the password reset mechanism to identify valid accounts which can lead to account enumeration.		An insecure mobile interface could lead to compromise of user data and control over the device.	Consider the business impact of an insecure mobile interface. Data could be stolen or modified and control over devices assumed. Could your customers be harmed? Could your brand be harmed?

Is My Mobile Interface Secure?

Checking for an Insecure Mobile Interface includes:

- Determining if the default username and password can be changed during initial product setup
- Determining if a specific user account is locked out after 3 - 5 failed login attempts
- Determining if valid accounts can be identified using password recovery mechanisms or new user pages
- Reviewing whether credentials are exposed while connected to wireless networks
- Reviewing whether two factor authentication options are available

How Do I Secure My Mobile Interface?

A secure mobile interface requires:

1. Default passwords and ideally default usernames to be changed during initial setup
2. Ensuring user accounts can not be enumerated using functionality such as password reset mechanisms
3. Ensuring account lockout after an 3 - 5 failed login attempts
4. Ensuring credentials are not exposed while connected to wireless networks
5. Implementing two factor authentication if possible
6. Apply mobile app obfuscation techinque
7. Implement mbile app anti-tempering mechanism
8. Ensuring the mobile app's memory hacking is possible
9. Restric the mobile app's execution on tempered OS environment

Please review the following tabs for more detail based on whether you are a Manufacturer, Developer or Consumer

Ilustración 5: OWASP Mobile Interface vulnerability [9]

En concreto la vulnerabilidad #3 del 'top10/2018' corresponde a 3 vulnerabilidades que en el 'top10/2014' se dividían en *web, cloud y mobile* - cabe indicar que las descripciones detalladas solo están disponibles para la versión 2014, y para el 2018 se ha hecho un mapeo con referencia a las descritas en la versión 2014. Si se echa un vistazo a la vulnerabilidad de 'Insecure Mobile Interface' se ve que las

propuestas de mejora del sistema incluyen: añadir mecanismos de Autenticación segura e incluir técnicas de ofuscación o *anti-tampering* (para impedir su modificación) para la aplicación móvil.

2.1.2 'IoT Exploitation Lab' de Attify

Este ejercicio de *pentesting* ha usado como base el manual '*IoT Exploitation Lab*' [4] de la empresa *Attify*. Este manual facilita en gran medida la realización de un *pentesting* al indicar cuáles son los pasos que hay que replicar para conseguirlo. Cabe indicar que los pasos indicados en este manual son dados normalmente a alto nivel y que no siempre se adaptan al 100% al caso real, ya que como se verá con uno de los ejemplos analizados, los sistemas y firmware de los dispositivos estudiados evolucionan.

En términos prácticos, el manual 'IoT Exploitation Lab' es es un manual en el que se muestran, con ejemplos prácticos, las vulnerabilidades presentes en dispositivos comerciales y que sirve como base para realizar *pentesting* sobre ellos. Está basado en un curso 'Offensive IoT Exploitation' [4] impartido por la misma empresa, y se compone de 157 páginas con 'guías' con diferentes casos de *pentesting* para diferentes dispositivos (e.g. *SmartPlug, SmartLock, SmartBulb*…) según diferentes ámbitos de cobertura.

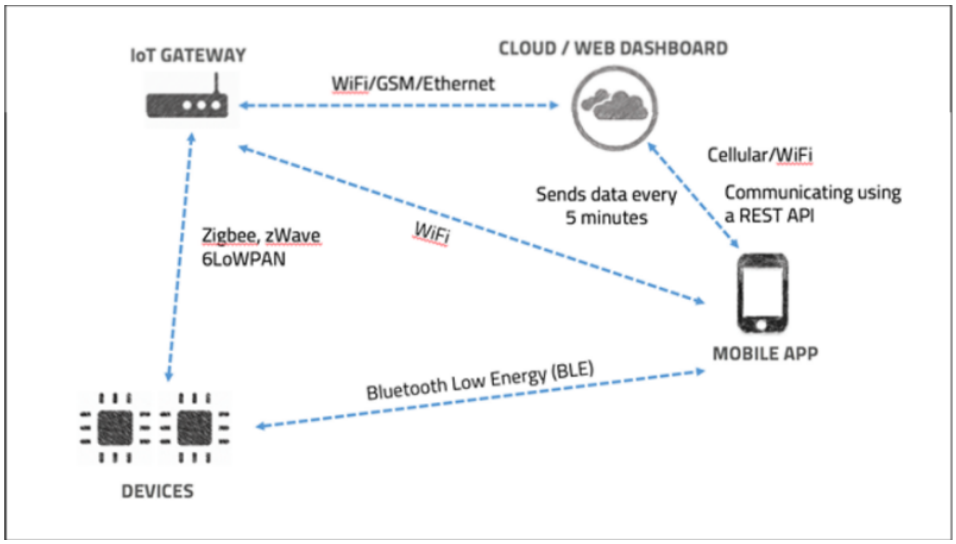


Ilustración 6: pentesting de Attify – diferentes componentes y comunicaciones [4]

En esta figura se muestran los componentes que se pueden analizar siguiendo los diferentes casos de uso del manual 'IoT Exploitation Lab' – en el presente trabajo el caso de uso es de un dispositivo que se comunica con la aplicación móvil a través de una conexión BLE.

Además de este manual, Attify incluye en su 'kit de *pentesting*' tanto los dispositivos en sí (en este caso, la bombilla inteligente, de la que se han usado 2 versiones diferentes), así como una serie de componentes hardware que pueden facilitar el *pentesting*.

En este caso, el componente adicional más interesante habría sido un sniffer hardware para comunicaciones BLE: 'BLE ubertooth one', aunque cabe reseñar que finalmente no se ha necesitado utilizar este componente hardware para capturar y analizar las trazas BLE, sino que se ha hecho mediante software: capturando el log 'bluetooth hci snooplog' que se genera en el móvil al usar la app asociada.

Ubertooth One

Advanced BLE Packet capture tool with capabilities of following the channel hopping pattern.

(comes only with the complete version of IoT Exploitation Learning Kit)

Smart BLE Light Bulb

Real World BLE target device. We will capture the traffic between the Smart BLE Light Bulb and mobile device, and use that information to take control of the light bulb.

Ilustración 7: componentes del kit para el pentesting de la SmartBulb

Los diferentes puntos a analizar para realizar un *pentesting* completo, según la metodología seguida por Attify incluyen:

1. 'Attack Surface Mapping': Diagrama de arquitectura de alto nivel de toda la solución que especifica los diferentes componentes en uso y los medios de comunicación que utilizan los distintos componentes, para identificar posibles puntos de ataque.

   En el caso estudiado tenemos el siguiente diagrama:

Attack Surface Mapping	Componente	Descripción
	Bombilla Inteligente	Bombilla que es posible gestionar desde una aplicación móvil: encendido/apagado o cambio de color de la luz. Se comunica a través de BlueTooth Low Energy con la aplicación móvil.

		Se cuenta con 2 versiones diferentes.
	Aplicación Móvil	Utilizada para la gestión de la bombilla inteligente. Permite encender/apagar la bombilla o realizar cambios de color. No requiere usuario registrado ni autenticación a la hora de emparejar la bombilla. Se comunica a través de BLE con la bombilla inteligente. Son necesarias 2 apps (2 versiones de bombilla) – *HappyLightning y HaoDeng* – ambas disponibles a través de Google Play para plataforma Android.
	Usuarios aplicación móvil	No necesarios para registrarse en la app.
	Sniffer Hardware para comunicaciones BLE	Componente Hardware para interceptar comunicaciones BLE. NO usado en este ejercicio.

Tabla 2-2. Attack Surface Mapping para SmartBulb

2. 'Firmware Security Analysis': El *firmware* de un dispositivo IoT contiene una gran cantidad de información sobre el dispositivo, incluida información sobre cómo funciona el dispositivo, configuraciones sensibles, claves API y más.
   Para obtener el *firmware*, en muchos casos se puede descargar de la web o incluso interceptar durante una actualización. Teniendo acceso al *firmware* se puede conocer toda la lógica sobre la funcionalidad del dispositivo, así como identificar posibles agujeros de seguridad.
3. 'Attacking Web, Mobile and cloud assets': Las aplicaciones móviles, con acceso a la red o con recursos en la nube también pueden ser un punto de entrada a un sistema IoT.
   Cabe destacar que éste es uno de los puntos de análisis en el *pentesting*, en particular en lo referente a la aplicación móvil y a su interacción con el dispositivo.

4. 'Hardware Exploitation': El acceso físico a los dispositivos (las 'cosas') en un sistema IoT puede dar acceso a sus conectores, facilitando vulnerabilidades como, por ejemplo: uso de UART para obtener un *shell root* (en muchos casos, el proveedor los mantiene accesibles en las versiones desplegadas en producción), lectura y escritura de contenido desde el chip/bus SPI (*Serial Peripheral Interface* - en algunos casos, el *firmware* se almacena en chips externos al *SoC*), depuración y explotación a través de JTAG. Además, este acceso físico puede dar información sobre el tipo de chips y componentes usados.

   En este caso, no se ha realizado este tipo de explotación, ya que la bombilla inteligente no se puede abrir fácilmente sin romperla/inutilizarla.
5. 'Radio based exploitation': análisis de la comunicación entre dos componentes. Bien entre dos dispositivos o entre un dispositivo y un *endpoint*. En IoT, ejemplos de protocolos muy usados son: BLE, ZigBee, LoraWan, Sigfox o WiFi.

   Este será el punto de partida para el presente *pentesting*.

Como ya se ha indicado, en este caso el trabajo se ha centrado sobre todo en este último punto, en concreto en las comunicaciones mediante el protocolo BLE. Primero interceptando estas comunicaciones (para ello hay que conocer bien el estándar usado – explicado en el siguiente apartado 2.1.3) pero también haciendo un análisis de la aplicación móvil, para identificar los 'comandos de control' que se mandan desde la app hasta el dispositivo y que se pueden encontrar en dichas comunicaciones. Estos comandos, que estarán incluidos en el *payload* de los paquetes BLE, pueden estar construidos siguiendo diferentes lógicas más o menos complejas y, de hecho, pueden estar cifrados (en el apartado 2.1.4 se explican los mecanismos de cifrado más habituales)

2.1.3 Protocolo BLE y características GATT

Bluetooth Low Energy (BLE) [10] [11] [12] es una tecnología de red de área personal (del inglés: PAN, *Personal Area Network*) inalámbrica, diseñada y

comercializada por Bluetooth Special Interest Group (Bluetooth SIG) especialmente interesante para su uso en aplicaciones IoT.

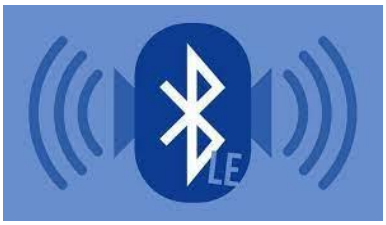


Ilustración 8: BLE [12]

Comparado con Bluetooth clásico, Bluetooth Low Energy está diseñado para proporcionar un bajo consumo de energía, manteniendo un rango de alcance de comunicación similar (o incluso mayor, bajando la velocidad de transmisión).

Specifications	Classic Bluetooth	Bluetooth Low Energy (BLE)
Range	100 m	Greater than 100 m
Data Rate	1-3 Mbps	1 Mbps
Application Throughput	0.7 -2.1 Mbps	0.27 Mbps
Frequency	2.4 GHz	2.4 GHz
Security	56/128-bit	128-bit AES with Counter Mode CBC-MAC
Robustness	Adaptive fast frequency hopping, FEC, fast ASK	24-bit CRC, 32-bit Message Integrity Check
Latency	100 ms	6 ms
Time Lag	100 ms	3 ms
Voice Capable	Yes	No
Network Topology	Star	Star
Power Consumption	1 W	0.01 to 0.5 W
Peak Current Consumption	less than 30 mA	less than 15 mA

Ilustración 9: Bluetooth Clásico vs BLE [13]

En esta imagen se pueden ver los principales parámetros para comparar Bluetooth tradicional con Bluetooth Low Energy – destacando para el caso IoT el consumo y el rango.

Bluetooth tradicional (BR/EDR): está orientado a conexión, y aunque tiene modos de bajo consumo (corriente máxima 30 mA), no se ha logrado optimizar suficiente para poder utilizarlos con pilas de poca capacidad (e.g. pilas tipo 'botón') o con tecnologías de tipo '*energy harvesting*' – en muchos casos, necesarias en IoT. Por otra parte, no está diseñado como protocolo de propósito general, en cuanto a que la capa de aplicación está muy integrada en la pila de protocolos.

Por otro lado, Bluetooth LE (BLE) está diseñado de forma independiente a Bluetooth tradicional (en principio son protocolos incompatibles, aunque existen modelos híbridos) y está optimizado para bajo consumo, ya que, entre otras cosas, utiliza paquetes pequeños y permite apagar la radio entre transmisiones (consumo en modo reposo: 5 µA, máximo < 15 mA). Está pensado para corto alcance, pero bajando mucho la tasa de transmisión puede llegar a distancias largas (incluso cercanas a 1km – disponible desde Bluetooth5).

BLE está pensado para aplicaciones en las que se transmitan pocos datos y la interfaz del modelo es simple (basada en GATT, como se explica más adelante).

El modelo de BLE (independiente a OSI) muestra tres capas diferenciadas: Perfiles (aplicaciones), Host (con protocolos GATT y GAP) y Controller (con la capa física y de enlace):

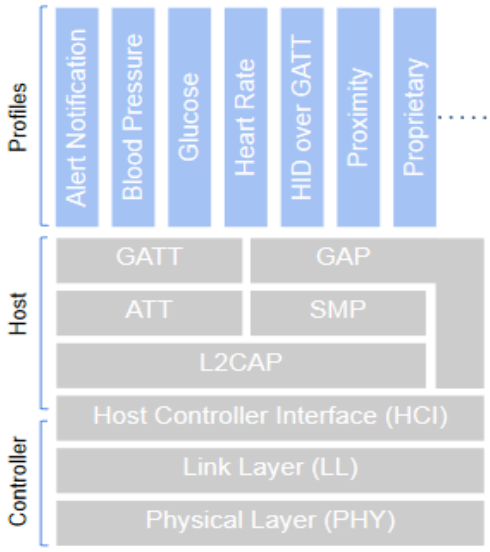


Ilustración 10: arquitectura BLE con 3 capas

En la capa de Perfiles/Aplicaciones, se pueden definir muchos perfiles diferentes para dispositivos de bajo consumo. Un perfil es una especificación que define cómo funciona un dispositivo para una aplicación determinada (el modelo soporta perfiles/aplicaciones tanto estándares como propietarias).

Los perfiles definen cómo van a comunicarse entre sí los dispositivos, utilizando los roles, modos, procedimientos y seguridad disponibles en el protocolo GAP (descrito en más detalle en el apartado 2.1.3.1)

Por otra parte, el protocolo GATT establece los modelos y procedimientos para el intercambio de atributos.

GATT, en sus capas inferiores se basa en el Protocolo de atributos (ATT). Cada atributo en el protocolo ATT se define de la siguiente manera:

- se exponen a través de un '*Handle*' (ID) en el servidor.
- se identifica por un identificador único universal (UUID), que representa su 'tipo' (estándar/registrado o no).
- tienen un campo 'valor' con el valor en sí del atributo

- tienen un campo de 'permisos' que, entre otras cosas, indican si un valor puede ser leído/escrito o el nivel de seguridad requerido para leer/escribir el atributo (autenticación o cifrado).
- pueden contener o no descriptores: Los descriptores son atributos definidos que describen el valor de una característica y aportan información adicional (como, por ejemplo, si la característica es 'notificada' periódicamente o no).

Por su parte, GATT define tanto la estructura jerárquica para la representación de aplicaciones (por medio de atributos del protocolo ATT) como la interfaz para intercambio de información (roles y operaciones de lectura/ escritura/ anuncios/ indicaciones).

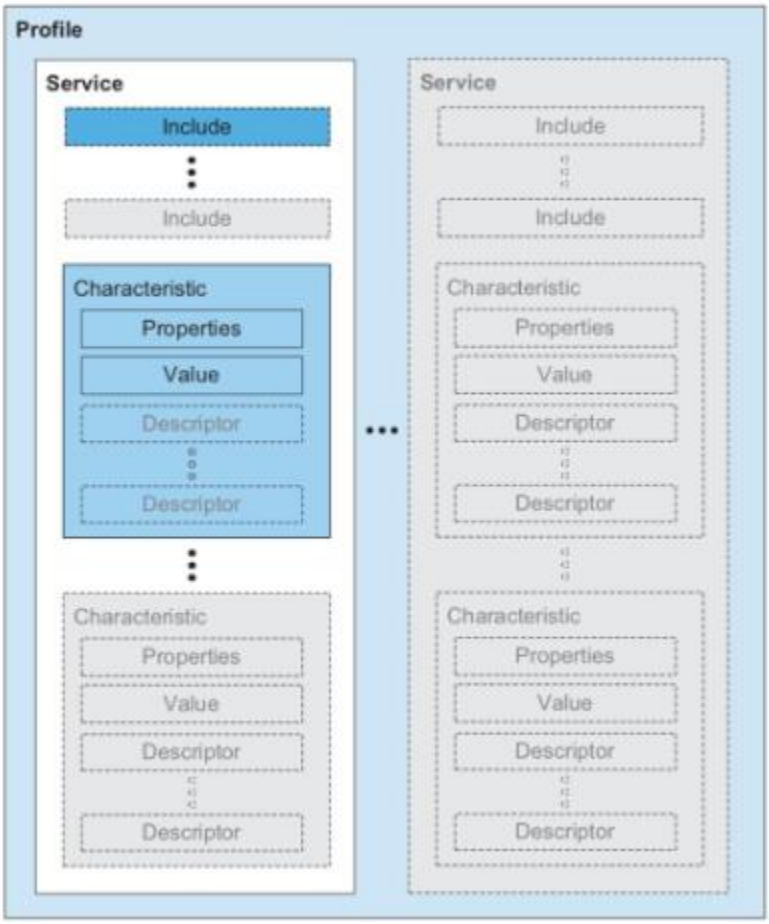


Ilustración 11: representación jerárquica en GATT

La estructura jerárquica incluye:

- Perfiles – representan las aplicaciones, incluye uno o más servicios.
- Servicios – conjunto de características. Pueden ser primarios o secundarios (incluidos en los primarios).
- Características – un valor, puede tener descriptores asociados.

Los roles definidos en GATT para definir la interfaz de intercambio de datos son:

- Cliente: envía comandos (operaciones de lectura/ escritura/ suscripción) y peticiones al servidor.
- Servidor: recibe las peticiones y comandos del cliente y envía respuestas al cliente.

En términos prácticos, cuando se interceptan las trazas de comunicación BLE, se puede ver en ellas que un Cliente (aplicación móvil) manda comandos (normalmente de lectura/escritura) a un Servidor (bombilla inteligente), para un determinado 'Handle' (que no es más que un atributo que representa una característica del servicio expuesto) con un determinado valor en el payload de la traza. Para ello será interesante averiguar tanto los identificadores del cliente/ servidor (normalmente su dirección mac), así como el conjunto de 'características/ servicios GATT' que expone el servidor (o dispositivo).

2.1.3.1 Seguridad en BLE: GAP & SMP

En BLE, el protocolo GAP (Generic Access Profile) define principalmente cómo un dispositivo puede descubrir y conectarse/ emparejarse a otro dispositivo. Por otra parte, y muy relevante en este caso de estudio, el protocolo GAP también se utiliza para fijar los niveles de seguridad que se pueden utilizar en la conexión BLE.

Por una parte, define los roles que pueden tener los dispositivos:

- *Broadcaster*: un dispositivo que difunde información (sin interacción).
- *Observer*: simplemente escucha paquetes de broadcast.

- *Peripheral*: difunde información y permite conexión (en este caso es el dispositivo IoT – la bombilla inteligente). Manda paquetes de '*advertising'* para aceptar conexiones de dispositivos tipo Central.
- *Central*: inicia conexiones con un *peripheral* (en este caso es la aplicación móvil de gestión de la bombilla inteligente)

Estos perfiles, junto con los modos y procedimientos aplicables definidos en el protocolo definen las interacciones posibles entre dispositivos (por ejemplo, si un dispositivo se puede conectar o no a otro).

Por otra parte, y como parte más relevante a este ejercicio de *pentesting*, cabe destacar que utilizando el protocolo GAP sobre SMP se pueden definir opciones de seguridad, que si son incluidas pueden cubrir los siguientes aspectos relacionados con la seguridad:

- Autenticación: se refiere a tener la seguridad de que el interlocutor es quien dice ser. Se consigue si durante la fase de emparejamiento se utilizan las opciones '*PassKey'*, '*OOB'* ('*out of band'*) o '*Numeric'* (pero no con '*Just Works'*). Como se verá esto también depende de las opciones de entrada/salida disponibles en el dispositivo.
- Integridad: se refiere a que un tercero no pueda modificar los datos transferidos. Se consigue mediante la firma de los mensajes, que en BLE se denomina MAC (Message Authentication Code).
- Confidencialidad: se refiere a que un tercero no pueda leer los datos transferidos. Esto se consigue mediante encriptación AES-CCM.
- Privacidad: se refiere a que un tercero no pueda identificar un nodo unívocamente. Se consigue mediante el uso de direcciones aleatorias.

Los diferentes niveles de seguridad soportados en BLE se definen según el tipo de información intercambiada durante la fase 1 del emparejamiento entre dos dispositivos:

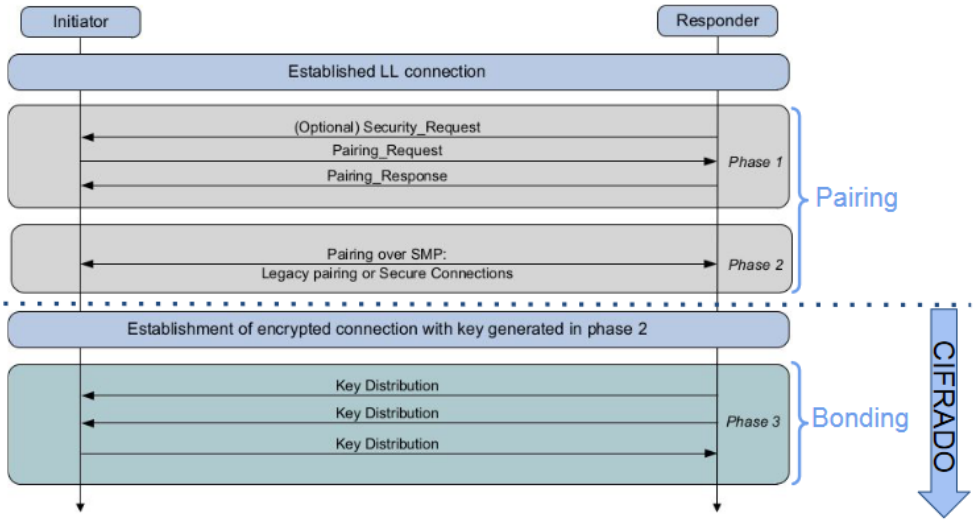


Ilustración 12: pairing y bonding en BLE

En esta primera fase del emparejamiento se definen:

- las capacidades I/O del dispositivo cliente – esto servirá para conocer el tipo de opciones de emparejamiento se pueden usar entre cliente/servidor
- el nivel de seguridad requerido: posibilidad o no de '*bonding*', seguridad frente a ataques '*MITM*' ('*man-in-the-middle*'), conexión segura/legacy… etc

Según esta información, la versión de Bluetooth soportada y si se ha requerido una conexión segura, en la fase 2 del emparejamiento se pueden definir los siguientes tipos de clave:

- En conexiones *Legacy* (BT4.0 o BT4.1) se genera una STK (*short term key*) – clave simétrica. Fácil de comprometer si se escucha esta conexión inicial
- En conexiones *Seguras* (BT4.2 y posteriores) utiliza el mecanismo de *Diffie-Hellman* – par clave publica/privada – para generar una LTK (*long term key*). Más segura y recomendada.

Cabe destacar que, en este caso, el dispositivo IoT (bombilla inteligente) no dispone de opciones de entrada/ salida para poder usar durante el emparejamiento. Esto limita las posibilidades de Autenticación.

Por otro lado, como se verá más adelante, y según las trazas que se logran interceptar, no se observa ningún nivel de cifrado en la conexión. Se considera que ésta es una decisión de diseño en la que posiblemente se pretenda mantener la compatibilidad con las versiones más antiguas de BLE.

2.1.4 Métodos de cifrado en comunicaciones - criptografía

Como se pondrá de manifiesto en uno de los casos de uso estudiados, independientemente de si la comunicación entre dispositivos está cifrada o no, una vez que estas trazas hayan podido ser interceptadas y expuestas, el cifrado del *payload* en los mensajes intercambiados tiene un papel fundamental en la seguridad del sistema.

En el caso estudiado, en el *payload* de las comunicaciones interceptadas se encuentra el valor de los 'comandos de control' enviados entre la aplicación móvil y el dispositivo que controlan la funcionalidad de la misma (encendido/ apagado/ cambio de color de luz).

En general, el cifrado [14] de un mensaje tiene como objetivo hacer ininteligible la información a todas las personas que no sean destinatarios del mensaje. Para conseguir dicho cifrado se aplican algoritmos criptográficos a los mensajes, para los cuales es necesaria una o varias claves. Aplicando la clave al mensaje original se obtiene un mensaje cifrado, que solo podrá ser descifrado conociendo dicha clave (o par de claves en el caso de cifrado asimétrico, como se verá más adelante).

Como aspectos interesantes a considerar en referencia al uso que se hace de las claves durante el desarrollo de un sistema IoT, se pueden destacar:

- Tamaño de la clave: en general, cuanto mayor sea, mayor será la seguridad, pero también los recursos necesarios para el cifrado/descifrado. Hay que tener en cuenta que un sistema IoT normalmente los recursos del sistema son limitados, tanto en

espacio como en procesamiento. Los tamaños habituales de las claves son (128-512-1024-2048bytes)

- Clave fija o variable: si bien es recomendable cambiar regularmente el valor de la clave, en muchos sistemas IoT se utiliza una clave fija. En muchas ocasiones esta elección puede ser debida a los ya citados recursos limitados en sistemas IoT, pero muchas otras veces puede ser debido a simplificar el sistema.
- Clave simétrica o asimétrica. Dependiendo de si se utiliza la misma clave para encriptar/desencriptar (simétrica) o un par de claves publica/privada (asimétrica).

2.1.4.1 Clave simétrica

En este tipo de algoritmos se utiliza la misma clave tanto en el emisor como en el receptor durante el encriptado/desencriptado.

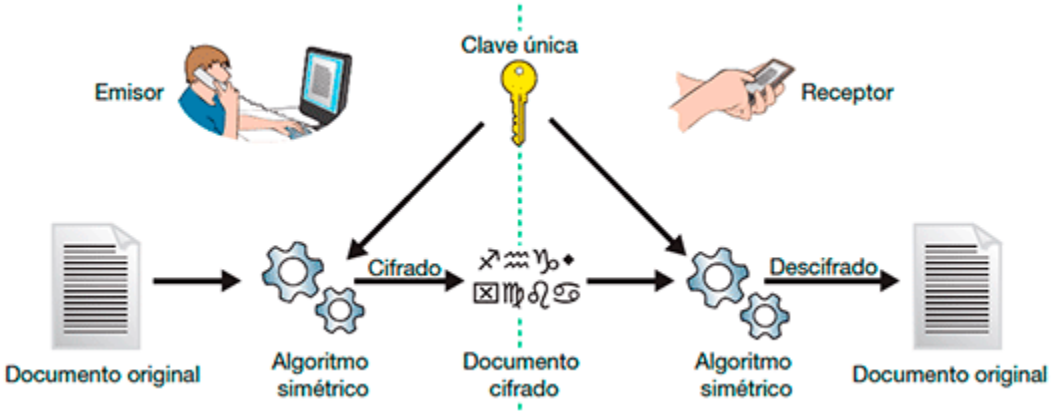


Ilustración 13: cifrado por clave simétrica

Estos algoritmos son bastante eficientes (tardan poco en realizar los procesos de encriptar y desencriptar) y, por lo tanto, muy convenientes para sistemas IoT.

Entre los más utilizados encontramos: DES, 3DES, AES, IDEA y Blowfish.

La clave del algoritmo al ser la misma para los dos procesos, la tienen que conocer tanto el emisor como el receptor, por lo tanto, en el caso de que esta cambie para los diferentes dispositivos del sistema tiene problemas relacionados con la circulación y el almacenaje de dichas claves.

2.1.4.2 Clave asimétrica

Introducida por los criptógrafos Diffie y Hellman, este algoritmo de cifrado de la criptografía asimétrica utiliza dos claves matemáticamente relacionadas de manera que lo que ciframos con una (clave pública) sólo puede descifrarse con la segunda (clave privada). De esta manera el emisor no necesita conocer y proteger una clave propia ya que es el receptor el que tiene el par de claves.

Ilustración 14: cifrado por clave asimétrica

Algunos algoritmos y técnicas de clave asimétrica son: RSA, DSA y ElGamal

Estos algoritmos son menos eficientes (desde el punto de vista de cómputo) que los de clave simétrica ya que las claves han de ser largas para mantener su independencia matemática entre ellas [14] y se ha de gestionar especialmente el almacenaje de las claves privadas. Es por esto que este sistema es menos usado en sistemas IoT en los que prime la sencillez y/o eficiencia.

Capítulo 3 - Casos de uso

Para realizar el *pentesting* sobre una bombilla inteligente se considera el siguiente escenario de partida:

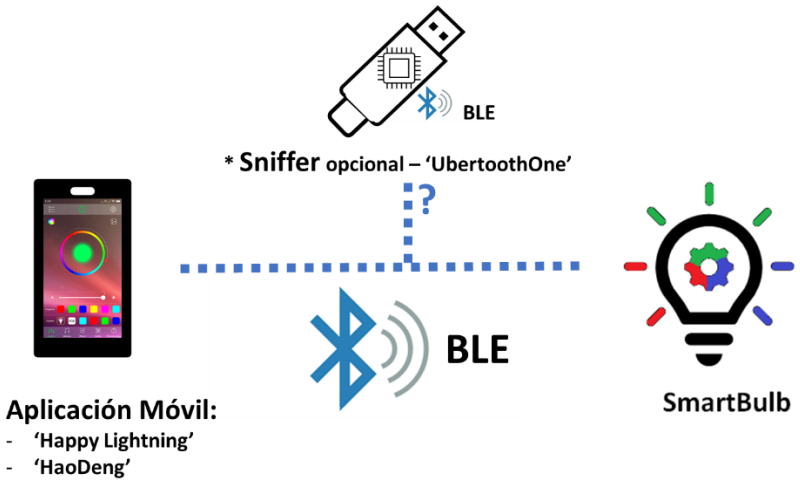


Ilustración 15: escenario para pentesting de una SmartBulb

Como se ve en la figura los principales elementos de este escenario son:

- Bombilla Inteligente – SmartBulb: este es el dispositivo IoT sobre el que se va a realizar el *pentesting*. Se trata de una bombilla conectada a través de Bluetooth Low Energy (BLE) a una aplicación móvil, desde la que se pueden configurar varias funcionalidades (encendido/ apagado y cambio de color de la luz).
- Aplicación Móvil: desde la que se puede controlar la bombilla. Se vale de la comunicación BLE para conectarse a la bombilla. Después de un emparejamiento inicial que no requiere autenticación (puede emparejarse con cualquier bombilla compatible) puede mandarle 'comandos de control' que controlan el encendido/apagado, así como el cambio de color.
- Protocolo de comunicación BLE: protocolo que usan los dos elementos anteriores para conectarse entre ellos. Como está explicado en el apartado 2.1.3, éste es de hecho un protocolo muy usado en escenarios IoT.

- *Sniffer* BLE – 'Ubertooth One': dispositivo hardware opcional que sirve para interceptar las comunicaciones BLE. En este caso no se ha usado porque se han podido capturar las trazas BLE mediante el log generado en el teléfono móvil (tras configurarlo para ello, como se detalla más adelante).

En este escenario de partida, se conoce que la bombilla inteligente tiene un agujero de seguridad explotable – que es el que se pondrá de manifiesto durante la realización del pentesting – que es debido fundamentalmente a que las comunicaciones mediante BLE no están cifradas y a que los 'comandos de control' encontrados en el payload de las trazas BLE capturadas siguen una lógica sencilla fácil de replicar y de explotar a través de un ataque tipo '*reply attack*'.

Cabe destacar que, si bien este era el punto de partida, en el ejercicio se ha contado de hecho con 2 versiones diferentes de la bombilla inteligente:

1. una de ellas con el agujero de seguridad mencionado, descrito en el caso de uso 3.1
2. otra en la que este agujero de seguridad ha sido corregido, introduciendo elementos que veremos en el apartado 3.2

3.1 – SmartBulb 'básica'

En este primer caso se analiza la bombilla 'básica', con el agujero de seguridad conocido.

En este caso, la bombilla sigue los estándares definidos en el protocolo BLE para la exposición de sus servicios (por medio de características/ atributos) y para la interacción con éstos (aceptando comandos de lectura/ escritura a los '*handles*' expuestos).

La comunicación BLE no está cifrada y el formato de los 'comandos de control' mandados en el payload sigue una lógica sencilla, fácil de replicar. Es por tanto un sistema relativamente fácil de vulnerar mediante un *reply-attack*.

3.1.1 Paso 1: Descubrimiento del dispositivo y de sus servicios

En este primer paso del pentesting se realiza el descubrimiento del dispositivo y de sus servicios.

Como veremos, el ID del dispositivo es su dirección mac y los servicios que ofrece se identifican a través de las características GATT que expone.

3.1.1.1 Herramientas para el descubrimiento

Como paso previo al descubrimiento de las características GATT, es importante realizar el descubrimiento del ID del dispositivo (su dirección mac) para poder usarlo en la herramienta '*gatttool*' [15] para poder ver sus servicios expuestos, así como para poder identificar posteriormente las trazas de comunicación capturadas.

El descubrimiento del dispositivo se realiza con la herramienta '*hcitool*' [16].

```
masteriot@ubuntu:~$ sudo hcitool lescan
LE Scan ...
FF:FF:20:2F:0B:A5 Triones-FFFF202F0BA5
FF:FF:20:2F:0B:A5 (unknown)
38:A9:26:56:5E:56 (unknown)
7E:70:F0:3B:8D:37 (unknown)
44:5C:E9:7E:6F:15 (unknown)
49:F1:B6:D8:77:DA (unknown)
8C:79:F5:D6:B2:F9 (unknown)
4D:22:CF:B5:4E:E9 (unknown)
40:24:96:AE:01:02 (unknown)
40:24:96:AE:01:02 (unknown)
54:83:3E:21:13:D9 (unknown)
66:2B:12:9F:36:63 (unknown)
4F:AB:B7:3A:B0:F0 (unknown)
8C:79:F5:D6:B2:F9 [TV] Samsung Q60 Series (55)
5D:9F:4B:DD:8B:FB (unknown)
7E:70:F0:3B:8D:37 (unknown)
49:F1:B6:D8:77:DA (unknown)
```

Ilustración 16: HCI tool lescan

Esta herramienta, como su nombre indica, se comunica a través de la interfaz HCI (ver Ilustración 10: arquitectura BLE con 3 capas) con los dispositivos BLE que estén en modo '*advertising*' (ver apartado 2.1.3.1) y muestra sus identificadores (direcciones mac).

En el caso estudiado la mac obtenida, asociada a la bombilla inteligente (llamada 'Triones') es FF:FF:20:2F:0B:A5 – esta es la dirección mac que se usará para conectarse al dispositivo y descubrir sus servicios, y más adelante en las trazas de comunicación para identificar los mensajes intercambiados por el dispositivo.

A continuación, para descubrir las características GATT(ATT) se usa la herramienta '*gatttool*' [15]. Esta herramienta permite descubrir los servicios y características de un dispositivo BLE. Para hacer esto, primero hay que conectarse a la mac identificada:

```
masteriot@ubuntu:~$ sudo gatttool -I -b FF:FF:20:2F:0B:A5
[sudo] password for masteriot:
[FF:FF:20:2F:0B:A5][LE]> connect
Attempting to connect to FF:FF:20:2F:0B:A5
Connection successful
[FF:FF:20:2F:0B:A5][LE]>
```

Ilustración 17: gatttool - connect

y después se pueden solicitar las características y servicios primarios que expone el dispositivo:

```
[FF:FF:20:2F:0B:A5][LE]> primary
attr handle: 0x0001, end grp handle: 0x0004 uuid: 0000ffd0-0000-1000-8000-00805f9b34fb
attr handle: 0x0005, end grp handle: 0x0007 uuid: 0000ffd5-0000-1000-8000-00805f9b34fb
```

Ilustración 18: gatttool - primary

Al igual que en el caso de la dirección mac, conocer las características y servicios primarios del dispositivo da una información valiosa a la hora de analizar las comunicaciones entre el dispositivo y la app de gestión.

3.1.2 Paso 2: Captura y análisis de tráfico BLE

La funcionalidad probada para realizar una captura del tráfico BLE generado entre la *app* móvil y la bombilla inteligente es un cambio de color de la bombilla. Para

ello, en la *app* móvil – app '*HappyLighting*' [17] previamente emparejada con el dispositivo – simplemente se selecciona un tono de 'color' de luz:

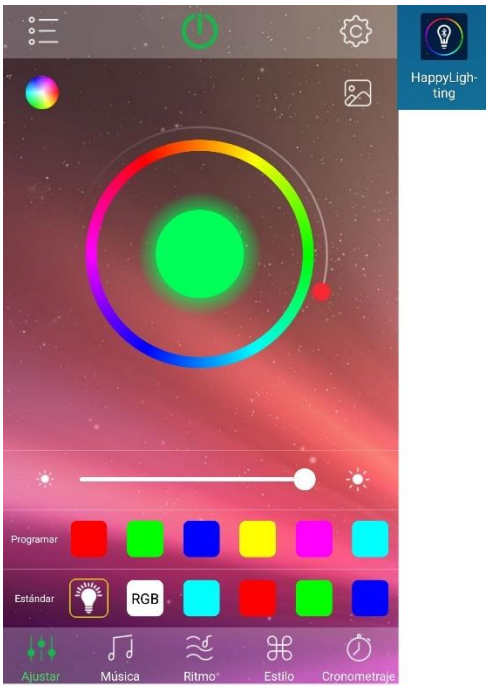


Ilustración 19: app móvil 'HappyLighting' para gestionar la SmartBulb

Este cambio de color genera un comando de 'cambio de color' que es enviado a través de BLE a la bombilla inteligente, que cambia al nuevo color. Para probar este caso, se realizan varios cambios de color en la app y a continuación se capturan las trazas BLE generadas.

Como se ha mencionado anteriormente, la captura del tráfico generado entre la app y la bombilla inteligente se puede realizar con un *sniffer hardware (BLE ubertooth one)* incluido en el kit de *pentesting* de Attify. Sin embargo, en este caso se ha considerado más directo realizar la captura del tráfico analizando el log *hci* que se genera en el dispositivo móvil.

Para poder capturar este log en un dispositivo Android hay que realizar una serie de pasos:

- Habilitar opciones de desarrollador en el móvil, para poder habilitar la opción de 'log hci'.

- Habilitar la opción de *'Bluetooth HCI snooplog'* de las herramientas de desarrollador de Android.
- Conectar (en este caso mediate *adb,* conociendo la IP del teléfono) al teléfono móvil y obtener el fichero btsnoop_hci.log como se muestra a continuación:

```
masteriot@ubuntu:~/hci_log$ adb connect 192.168.0.10
already connected to 192.168.0.10:5555
masteriot@ubuntu:~/hci_log$ adb root
adbd cannot run as root in production builds
masteriot@ubuntu:~/hci_log$ adb pull /sdcard/btsnoop_hci.log
/sdcard/btsnoop_hci.log: 1 file pulled. 0.6 MB/s (47473 bytes in 0.076s)
masteriot@ubuntu:~/hci_log$
```

Ilustración 20: descarga de btsnoop_hci.log

Una vez obtenidas las trazas de comunicación, se pueden analizar con la herramienta Wireshark [18]:

Ilustración 21: wireshark

Wireshark es una herramienta de uso muy extendido para monitorizar trazas de diferentes protocolos de red. Permite analizar las trazas hasta el nivel más bajo y soporta un gran número de protocolos de red. Dichas trazas pueden ser por tanto analizadas según la estructura del protocolo utilizado.

Dependiendo de las interfaces disponibles en el sistema es capaz de capturar trazas de Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay o FDDI.

Además, permite filtrar las trazas para enfocarse solo en las que se requieran – por ejemplo, en este caso se ha utilizado el siguiente filtro:

- *'btl2cap.cid==0x004'* → *para ver solo paquetes BLE*

3.1.2.1 Trazas a analizar

El caso propuesto para el estudio del comportamiento de la bombilla inteligente es realizar varios cambios de color, para analizar a continuación las diferentes trazas que se producen:

- Primer cambio de color. Se obtiene una traza como la siguiente:

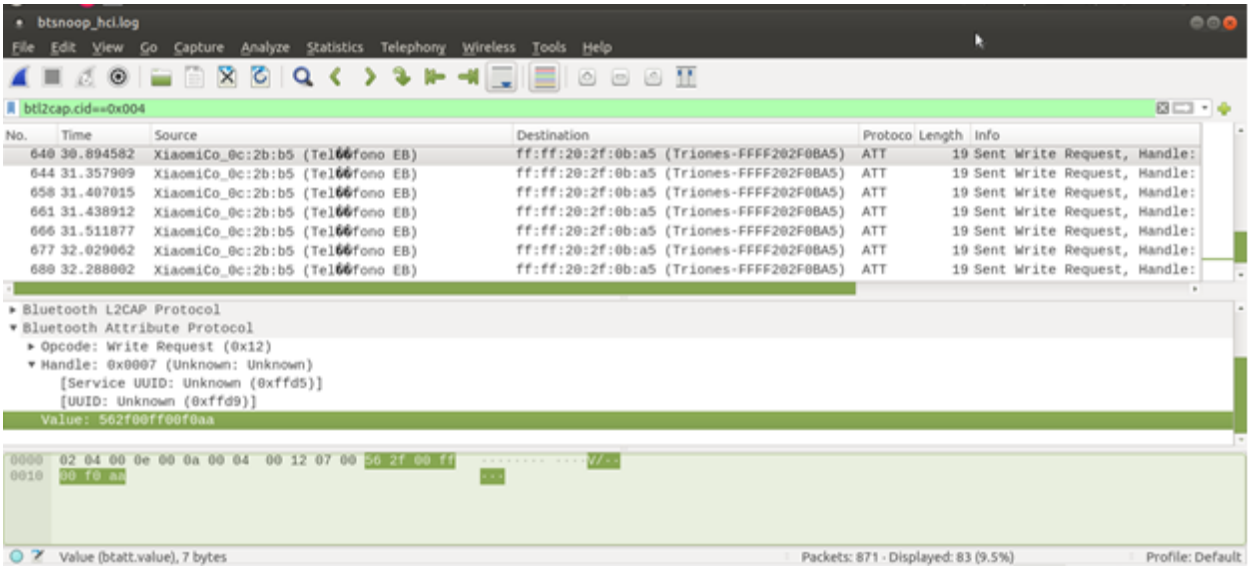


Ilustración 22: traza para primer cambio de color

Como se ha explicado anteriormente, se han de buscar paquetes con destino a la mac identificada en pasos anteriores y que además escriben a uno de los servicios primarios. En este primer caso se escribe el valor 56**2F00FF**00F0AA al servicio expuesto en el handle 0x0007

- Segundo cambio de color. Se obtiene una traza como la siguiente:

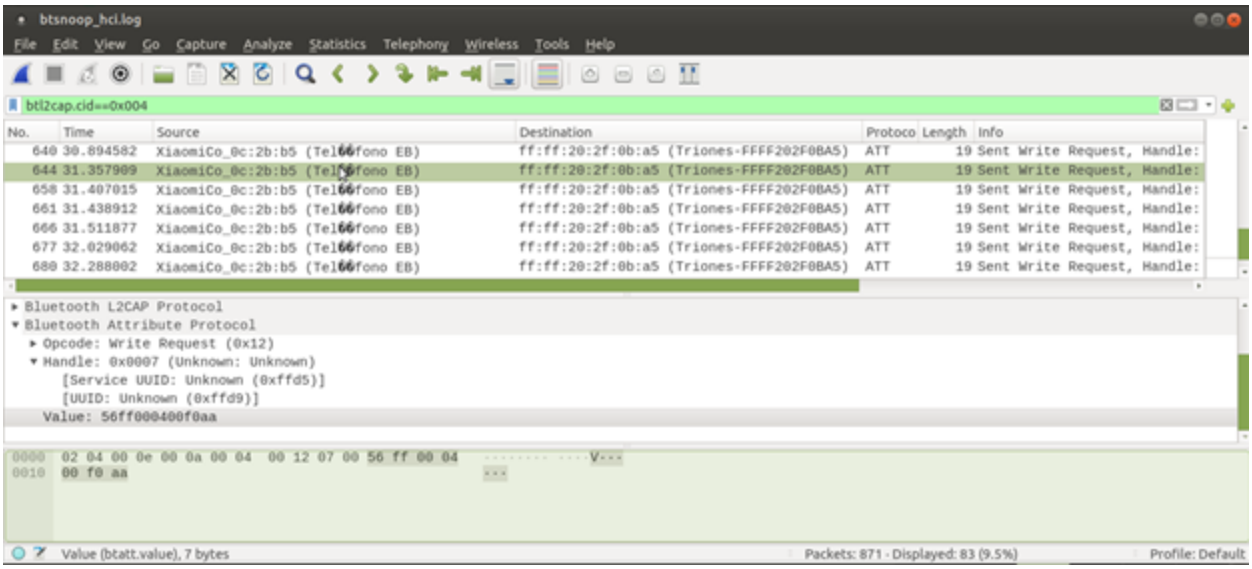


Ilustración 23: traza para segundo cambio de color

En este segundo caso se escribe el valor 56**FF0004**00F0AA al handle 0x0007

Tras este análisis – y según lo indicado en el manual de pentesting - parece que los cambios de color en la bombilla inteligente simplemente se ejecutan mandando un comando de escritura al handle 0x0007 con el siguiente formato: 56**XXYYZZ**00F0AA, donde los bytes XX, YY y ZZ parecen representar los valores de intensidad de R-red, G-green, B-blue.

Para comprobar que efectivamente esto es así, se realizará un 'reply-attack' probando diferentes combinaciones.

3.1.3 Paso 3: Reply-attack

Tras identificar el formato de los comandos de 'cambio de color' en las trazas capturadas y los handlers de escritura para los correspondientes servicios, el *reply-attack* se realiza simplemente escribiendo un comando construido de manera similar (debidamente modificado según el caso de uso que se quiera replicar) al handler correspondiente (0x0007).

Para ello, se utiliza el comando 'char-write-req' de la herramienta 'gatttool', para escribir diferentes combinaciones e ir viendo el resultado, como se muestra en la siguiente figura:

```
masteriot@ubuntu:~$ sudo gatttool -I -b FF:FF:20:2F:0B:A5
[FF:FF:20:2F:0B:A5][LE]> connect
Attempting to connect to FF:FF:20:2F:0B:A5
Connection successful
[FF:FF:20:2F:0B:A5][LE]> char-write-req 0x0007 5600000000f0aa
Characteristic value was written successfully
[FF:FF:20:2F:0B:A5][LE]> char-write-req 0x0007 56ff000000f0aa
Characteristic value was written successfully
[FF:FF:20:2F:0B:A5][LE]> char-write-req 0x0007 5600ff0000f0aa
Characteristic value was written successfully
[FF:FF:20:2F:0B:A5][LE]> char-write-req 0x0007 560000ff00f0aa
Characteristic value was written successfully
```

Ilustración 24: reply-attack 'char-write-req'

Los resultados en cada uno de los casos probados se muestran en la siguiente tabla:

# Reply-attack	Comando	Resultado
Reply-attack 1	56**000000**00f0aa	
Reply-attack 2	56**ff0000**00f0aa	
Reply-attack 3	56**00ff00**00f0aa	

Reply-attack 4	56**0000ff**00f0aa	

Tabla 3-1. Reply-attack - resultados

Con estos resultados, se puede confirmar que el comportamiento del sistema es de hecho el que se presuponía tras este primer análisis de las trazas.

3.1.4 Resultados / Análisis del sistema 'básico'

Como se ha visto en los apartados anteriores, esta versión de la bombilla inteligente presenta un claro agujero de seguridad que, como se ha explicado anteriormente, es debido fundamentalmente a que:

1. La conexión entre los dispositivos no añade ningún elemento de seguridad durante el emparejamiento y no fuerza a que este emparejamiento sea exclusivo → esto implica que cualquiera puede conectarse al dispositivo sin ningún tipo de mecanismo de control.
2. las comunicaciones mediante BLE no están cifradas → es posible ver las trazas interceptadas con algún dispositivo o mecanismo que nos permita 'escuchar' comunicaciones BLE
3. los comandos de control mandados no están cifrados → una vez interceptadas las trazas es sencillo ver su valor
4. los 'comandos de control' encontrados en las trazas BLE capturadas siguen una lógica sencilla fácil de replicar y de explotar a través de un ataque tipo 'reply attack'. → es posible construir otros comandos para tomar el control de la bombilla tras un análisis sencillo.

Cabe destacar que esta vulnerabilidad coincide de hecho con la vulnerabilidad identificada por OWASP (ver apartado 2.1.1.1) en el #3 de su Top10 de vulnerabilidades IoT conocidas, relativa a Interfaces del ecosistema inseguras'.

Las propuestas de OWASP para mitigar estas vulnerabilidades, incluyen añadir mecanismos de Autenticación segura (mitigaría la vulnerabilidad #1), técnicas de ofuscación o *anti-tampering* (para impedir su modificación) en la aplicación móvil (mitigaría la vulnerabilidad #4).

Aparte de estas propuestas de mejora de OWASP, para poder mitigar las vulnerabilidades #2 y #3 habría que considerar cifrar tanto el protocolo (BLE lo soporta a través de GAP) así como el payload con los comandos de control.

Como se verá a continuación, algunas de estas medidas han sido de hecho incluidas en la segunda versión de SmartBulb estudiada.

3.2 SmartBulb 'más segura'

En este segundo caso se analiza la bombilla 'más segura, en la que se han tomado medidas para solucionar el agujero de seguridad conocido.

En este segundo caso, la bombilla también sigue los estándares definidos en el protocolo BLE para la exposición de sus servicios (por medio de características/ atributos) y para la interacción con éstos (aceptando comandos de lectura/ escritura a los 'handles' expuestos).

La comunicación BLE tampoco está cifrada pero sí parecen estarlo los 'comandos de control' incluidos en el payload, que en cualquier caso siguen una lógica más compleja, mucho más difícil de replicar y que requiere un estudio más detallado (mediante técnicas de ingeniería inversa) del firmware. Es por tanto un sistema más difícil de vulnerar mediante un reply-attack.

3.2.1 Paso 1: Descubrimiento del dispositivo y de sus servicios

Al igual que en el caso anterior, en este primer paso del pentesting se realiza el descubrimiento del dispositivo y de sus servicios.

El descubrimiento del ID del dispositivo (su dirección mac) se realiza con la herramienta 'hcitool lescan' [16]:

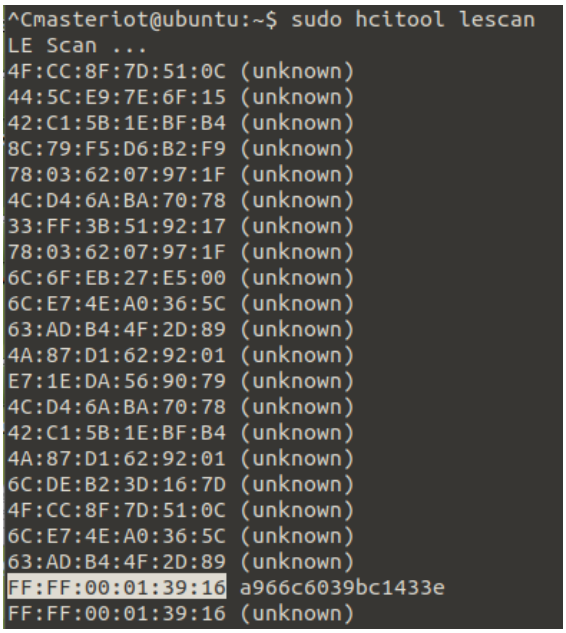


Ilustración 25: HCI tool - lescan

En este nuevo caso, la mac obtenida – asociada a la SmartBulb ('a966c6039bc1433e') - es FF:FF:00:01:39:16

A continuación, para descubrir las características GATT(ATT) se usa la herramienta '*gatttool*': primero hay que conectarse a la mac identificada y a continuación se solicitan sus características y servicios primarios:

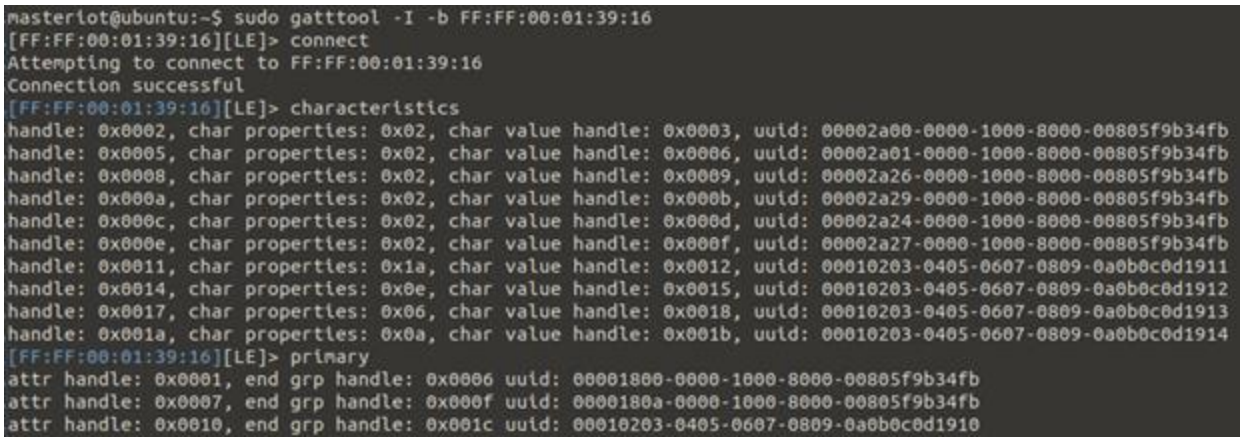


Ilustración 26: Gatt tool

En este punto, ya se puede ver que el conjunto de características y servicios expuestos por el dispositivo es diferente al caso anterior (ver 3.1.1), con lo cual parece que el firmware se comportará de manera diferente, lo que se confirmará analizando las trazas capturadas en el siguiente paso.

3.2.2 Paso 2: Captura y análisis de tráfico BLE

De nuevo, para realizar una captura del tráfico BLE generado entre la *app* móvil y la bombilla inteligente, se realizan una serie de cambios de color en la aplicación móvil asociada. En este caso, la app utilizada es 'Hao Deng' [19]:

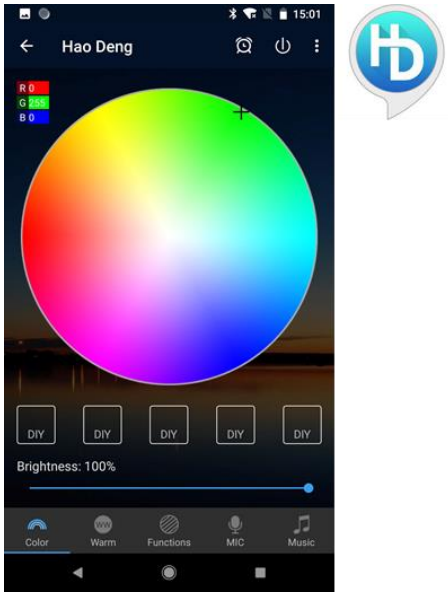


Ilustración 27: app móvil 'Hao Deng' para SmartBulb

De nuevo, la captura del tráfico generado entre la app y la bombilla inteligente se realiza capturando el log *hci* que se genera en el dispositivo móvil

Una vez obtenidas las trazas de comunicación, se analizan con la herramienta Wireshark ya mencionada anteriormente.

3.2.2.1 Trazas a analizar

Llegados a este punto, se puede ver una diferencia en el comportamiento de las comunicaciones respecto al caso de uso anterior.

Las trazas obtenidas son como la siguiente:

- Escritura a un handle diferente (0x0015) y el valor escrito tiene otro formato al esperado

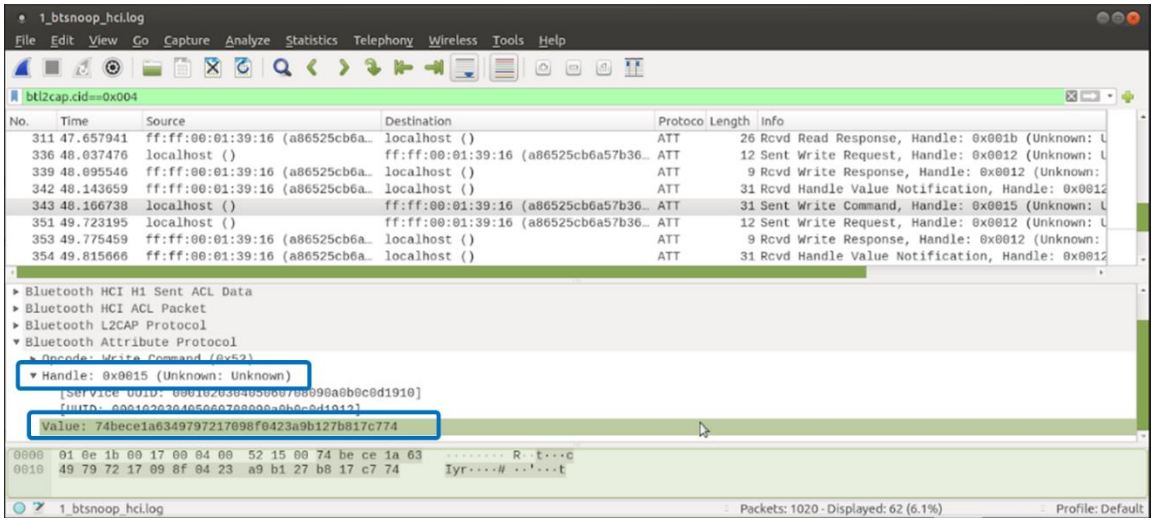


Ilustración 28: trazas de escritura al handle 0x0015

3.2.3 Paso 3: Reply-attack - fallido

Tras identificar la estructura de los comandos de 'cambio de color' en las trazas capturadas y los handlers de escritura correspondientes, se comprueba que como era de esperar, un reply-attack construido de manera similar al caso de uso anterior no tiene el efecto deseado.

```
[FF:FF:00:01:39:16][LE]> char-write-req 0x0007 56ff000000f0aa
[FF:FF:00:01:39:16][LE]>
```

Ilustración 29: reply-attack 'char-write-req'

Efectivamente, aunque se puede mandar la solicitud de escritura, no se recibe confirmación de que la característica haya sido escrita correctamente, y efectivamente no tiene ningún efecto en la bombilla.

Tampoco se deduce ninguna lógica 'sencilla' de replicar según el valor de los 'comandos de escritura' presentes en el payload de las trazas analizadas. Por tanto, parece que se ha añadido seguridad al firmware respecto a la versión anterior de bombilla.

3.2.4 Análisis del firmware

A continuación, se realiza un análisis más detallado del nuevo firmware utilizado, con el doble fin de:

- identificar el funcionamiento de nuevo sistema (en referencia a la construcción de 'comandos de control') para poder ver si es posible realizar un ataque al sistema,
- en cualquier caso, identificar cuáles son los elementos qué se han introducido para hacer más seguro el sistema

Para ello se han seguido los siguientes pasos:

- Análisis del código (tanto de la APK como de las librerías relacionadas)
- Instrumentación del código para ampliar el análisis

3.2.4.1 Análisis del Código – Reverse Engineering

El primer paso para poder realizar un análisis del código – o Reverse Engineering – es contar con el código en sí. Como es de esperar, el código de las aplicaciones comerciales para el control de dispositivos IoT no se difunde públicamente (sería un muy evidente agujero de seguridad), pero sí que es posible descargar la APK de la aplicación móvil y utilizar herramientas de 'reverse engineering' sobre ella.

Para ello se obtiene la versión oficial de la APK – las APK móviles se pueden descargar desde Google Play usando diferentes métodos, en este caso se ha usado un *plugin* disponible desde el navegador web [20].

Una vez descargada la APK, el código de la misma no es legible directamente (*bytecode*), sino que hay que utilizar alguna herramienta para poder 'descompilarla'. En este caso se ha utilizado JADX [21]

Ilustración 30: JADX - decompilador APK

JADX es un descompilador de APK que genera un código Java. También, realiza cierta desofuscación del código y presenta el resultado en una interfaz gráfica (GUI).

Tras usar JADX sobre la APK se obtiene el siguiente código Java:

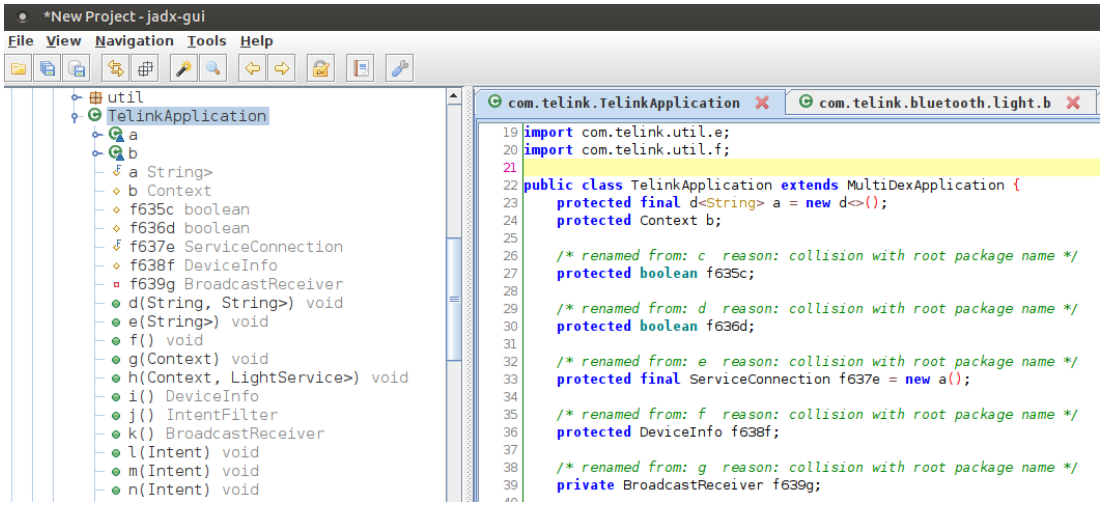


Ilustración 31: codigo decompilado de la APK

En este primer vistazo se puede ver que parte del código sigue parcialmente 'ofuscado' o no es directamente legible tras la descompilación (e.g. funciones con nombre 'a', 'b', 'f'… etc.).

Una buena manera de empezar el análisis del código, que además confirmará o no la validez una de las mitigaciones propuestas para hacer más seguro el sistema (i.e. encriptar el payload), es buscar métodos de encriptado/desencriptado.

Este tipo de funciones serán utilizadas previsiblemente en las lógicas de construcción de 'comandos de control' del dispositivo (encriptando el payload), lo cual necesitaríamos conocer si quisiéramos realizar un nuevo *reply-attack*.

En el código descompilado se encuentran efectivamente métodos que se refieren a AES y a una clave de 256bits:

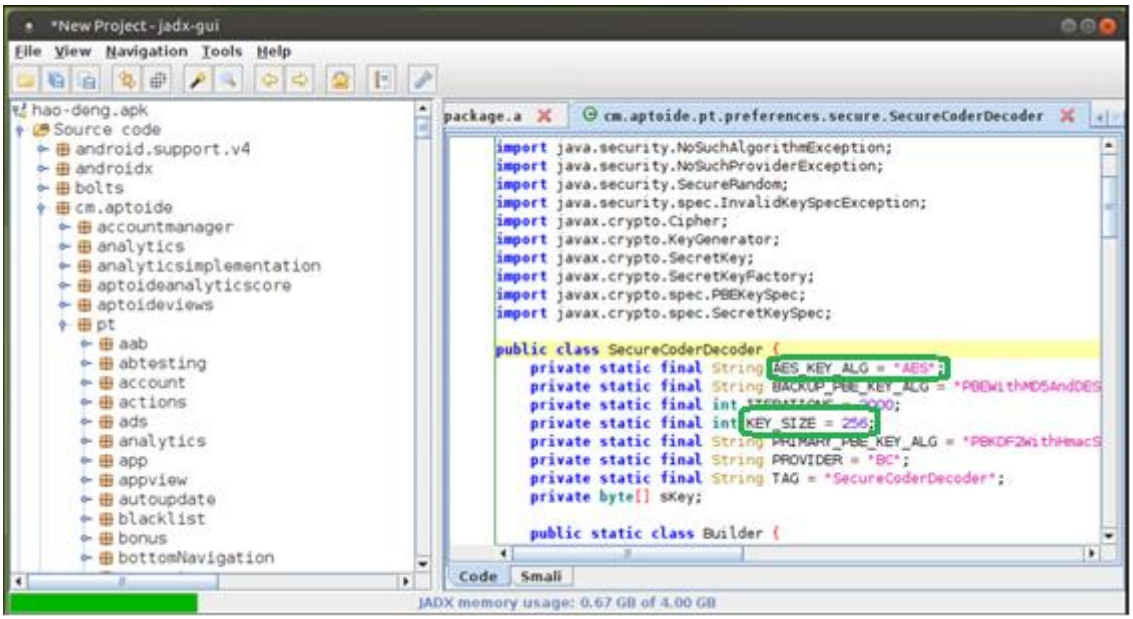


Ilustración 32: código con referencia a AES 256

Tal y como está definido [22], el sistema de encriptado AES, (*Advanced Encryption Standard*) es un sistema de cifrado por bloques muy común. Se trata de un algoritmo de criptografía simétrica con un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 o 256 bits. Todo esto parece indicar que efectivamente se está utilizando este sistema de encriptado basado en una clave de 256 bits

Además, también se encuentran llamadas a funciones encryptCmd/ decryptCmd, lo cual apunta a que los 'comandos de control' se mandan efectivamente encriptados en el payload:

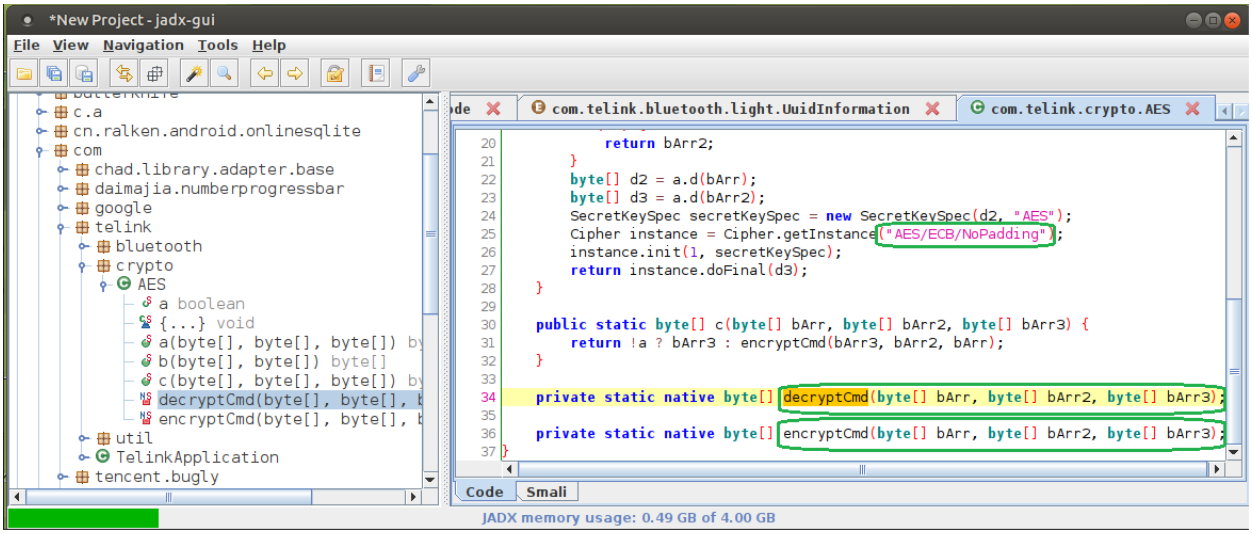


Ilustración 33: funciones decryptCmd/encryptCmd

Se hace también referencia a AES/ECB (Electronic Code Book) [23]. Es importante reseñar que este método de encriptado es susceptible a ataques de fuerza bruta, ya que la misma entrada siempre produce la misma salida, con lo cual, si se conoce un mensaje sin cifrar y el cifrado resultante (conociendo el método) se puede buscar la clave. En otros casos, como CBC (Cipher Block Chaining) [23] el mensaje de entrada se combina – mediante la operación lógica XOR - con mensajes anteriores antes de ser cifrado, por tanto, el mismo mensaje de entrada, en diferentes contextos, producirá diferente salida.

Esta 'aleatoriedad' en el cifrado es importante de conseguir, y dificultará mucho el descifrado. Por ejemplo, se puede añadir algún número aleatorio en el mensaje antes del cifrado.

Para poder conocer en más detalle el algoritmo de encriptación y, sobre todo, saber dónde se puede encontrar el valor de la clave o si se usa algún tipo de 'aleatoriedad', necesitamos ver en más detalle el código de las funciones decryptCmd/ encryptCmd.

Sin embargo, estas funciones no están implementadas en el código descompilado de la APK, sino que se implementan en una librería externa, de la cual sólo se puede encontrar un binario:

(la búsqueda se realiza con el comando *rgrep* [24]):

```
masteriot@ubuntu:~/Downloads$ cd com-zengge-telinkmeshlight1609736400.apk_FILES/masteriot@ubuntu:~/Downlo
ads/com-zengge-telinkmeshlight1609736400.apk_FILES$ rgrep decryptCmd
Binary file classes.dex matches
Binary file lib/x86_64/libTelinkCrypto.so matches
Binary file lib/armeabi/libTelinkCrypto.so matches
Binary file lib/mips/libTelinkCrypto.so matches
Binary file lib/x86/libTelinkCrypto.so matches
Binary file lib/mips64/libTelinkCrypto.so matches
Binary file lib/armeabi-v7a/libTelinkCrypto.so matches
Binary file lib/arm64-v8a/libTelinkCrypto.so matches
masteriot@ubuntu:~/Downloads/com-zengge-telinkmeshlight1609736400.apk_FILES$
```

Ilustración 34: rgrep decryptCmd

Aunque se encuentra una referencia a estas librerías en el código descompilado con JADX, no se puede ver su código ya que es un binario:

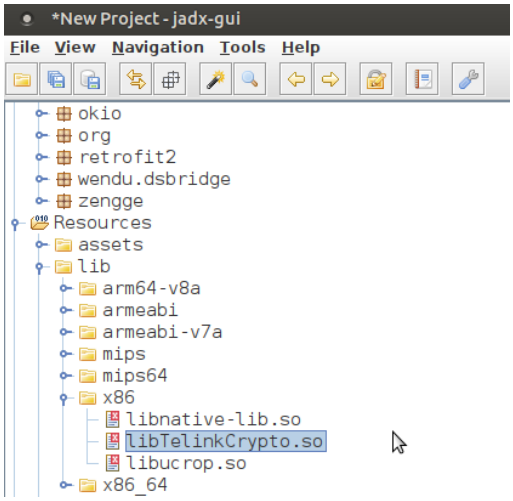


Ilustración 35: libreria criptográfica - sin código descompilado

De nuevo, para poder hacer el análisis del código de esta librería se necesita una herramienta para descompilar el binario a un código legible:

Para este propósito se usa la herramienta 'binary ninja' [25]:

Ilustración 36: binary ninja

'Binary-Ninja' es una herramienta de ingeniería inversa, que es capaz de descompilar ficheros binarios de múltiples plataformas (incluidas las basadas en x86 o ARM) y produce una salida en un lenguaje intermedio (IL – Intermediate Language) parecido al código máquina, que puede ser visualizado a través de una interfaz gráfica (GUI)

En el caso estudiado, el código obtenido tras descompilar la función de decryptCmd (de manera similar para encryptCmd) con binary-ninja no da ninguna indicación adicional respecto al valor de la clave usada o al método de construcción de dicha clave por lo tanto se han de buscar otros métodos para continuar con el análisis.

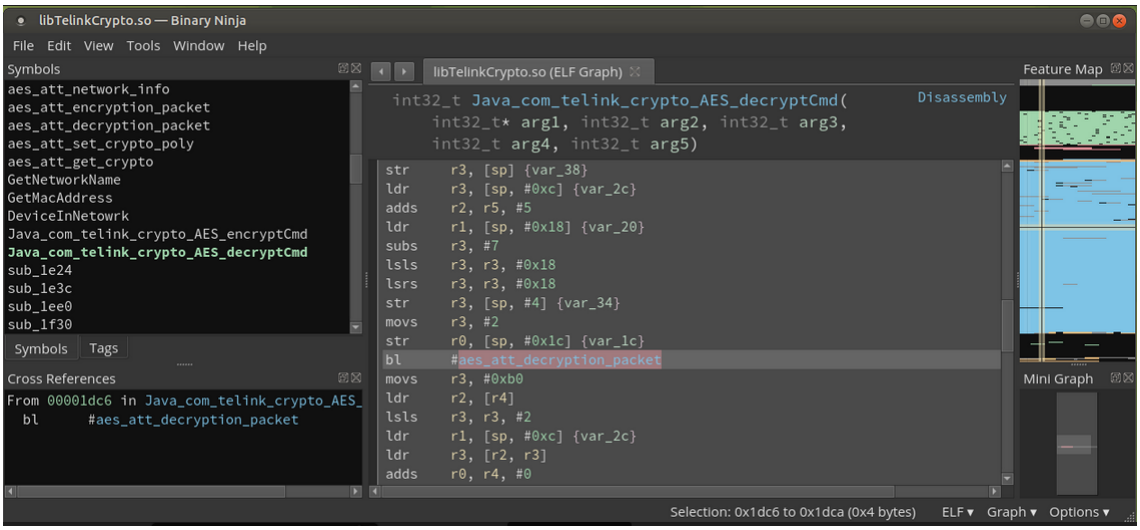


Ilustración 37: binary ninja output

A pesar de la 'dificultad' encontrada para continuar con la ingeniería inversa del código con este método, hay otras opciones que permiten seguir con el análisis del comportamiento del sistema a más alto nivel.

3.2.4.2 Instrumentación del código

Para esta nueva aproximación, se instrumenta el código de la APK original con la herramienta Frida [26], lo que en términos prácticos permite añadir código propio a ciertas funciones dentro de una APK modificada.

Ilustración 38: Frida – Instrumentación dinámica

Frida es una herramienta de instrumentación dinámica utilizada para facilitar la ingeniería inversa, posibilitando la introducción de scripts propios dentro de procesos tipo 'caja negra'. Soporta varias plataformas, incluida Android, lo cual es muy conveniente en este caso.

Se instrumenta el código de las funciones *decryptCmd/encryptCmd* de la siguiente manera:

```
CMDUtils.decryptCmd.implementation =
function (paramD1, paramD2, paramD3) {
        console.log('[+] Inside Decrypt()
        ======');
        var paramD1 =
        this.decryptCmd(paramD1, paramD2,
        paramD3);
        console.log('paramD1:')
        log_byte_array(paramD1)
        console.log('paramD2:')
        log_byte_array(paramD2)
        console.log('paramD3:')
        log_byte_array(paramD3);
        return paramD1;
        };
```

```
CMDUtils.encryptCmd.implementation =
function (paramE1, paramE2, paramE3) {
        console.log('[+] Inside Encrypt()
        ======');
        var paramE1 =
        this.encryptCmd(paramE1, paramE2,
        paramE3);
        console.log('paramE1:')
        log_byte_array(paramE1)
        console.log('paramE2:')
        log_byte_array(paramE2)
        console.log('paramE3:')
        log_byte_array(paramE3);
        return paramE1;
        };
```

Ilustración 39: instrumentación con Frida

En términos prácticos, esta instrumentación permite sustituir las llamadas a las funciones originales *decryptCmd* y *encryptCmd* por las funciones propias que se han escrito (mostradas en la ilustración superior). Como se puede ver en el código de estas funciones 'instrumentadas', lo que hacen es simplemente llamar a la función original y mostrar por pantalla tanto los parámetros de entrada como de salida.

Tras esta instrumentación y la realización de varias pruebas de cambio de color en la bombilla, se obtiene la siguiente salida por consola:

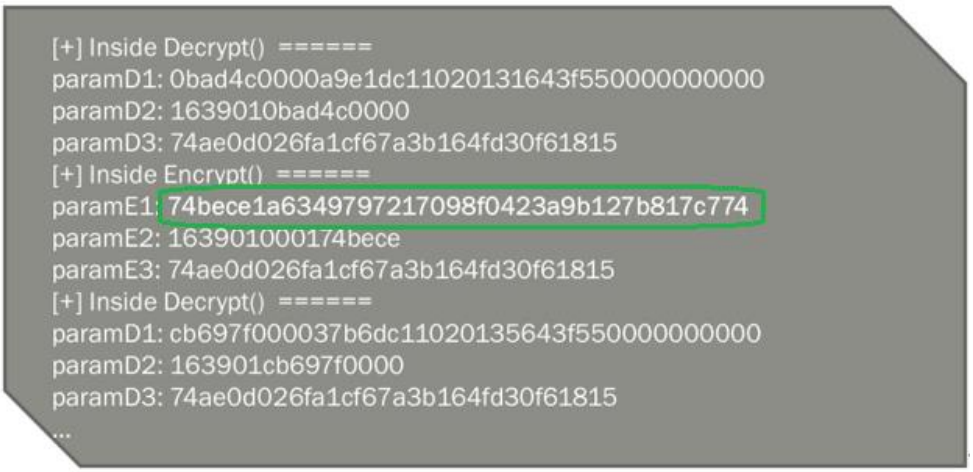


Ilustración 40: salida de instrumentación con Frida

De la cual se deducen ciertos patrones que dan una posible dirección para seguir con el análisis:

- Los parámetros D1 y E1 son la salida de las funciones de 'Decript()' y 'Encript()' respectivamente.
- El parámetro E1 tiene el mismo valor que el encontrado en el payload de la traza analizada (ver apartado 3.2.2.1): '74bece1a6349797217098f0423a9b127b817c774'
- El parámetro D3 y E3 coinciden en ambas funciones de encriptado/ desencriptado. Por lo tanto, se sospecha que ésta sea efectivamente la clave usada tanto para encriptar como para desencriptar el payload.

- El parámetro D2 y E2 no está identificado, pero parece estar construido de la siguiente manera: siempre empieza por 163901 + <inicio del D1 anterior> para la parte de Decript, y 1639010001 + <inicio del E1 anterior> para la parte de Encript. Esto podría indicar que efectivamente se ha incluido cierta aleatoriedad o serialización en este elemento que parece influir en la construcción del payload. (163901 pueden ser efectivamente los valores de RRGGBB para el caso probado).

Llegados a este punto, si se quisiera realizar un reply-attack para esta nueva versión de bombilla inteligente 'más segura' habría que seguir examinando el código tanto de la APK como de las librerías externas, para identificar cómo se construyen exactamente los mensajes con los diferentes 'comandos de control', posiblemente instrumentalizando otras partes del código. Esto, debido a la ofuscación parcial del código y a que la lógica para la construcción de dichos 'comandos' no es sencilla (incluye cierta 'aleatoriedad'/'serialización' como ya se ha visto), es una tarea compleja de 'reverse engineering' que, en cualquier caso, no es el objeto principal de este trabajo.

En cualquier caso, este análisis ha demostrado la potencia (y el riesgo que introducen) las herramientas de 'reverse engineering' y de 'instrumentación' de código que se han utilizado, con lo cual se evidencia de la necesidad de contar con múltiples elementos que puedan añadir seguridad a un sistema.

3.2.5 Resultados / Análisis del sistema 'más seguro

Como se ha visto en este segundo caso de uso, esta segunda versión de la bombilla inteligente es mucho más difícil de vulnerar que la anterior, aunque parece que, haciendo un análisis más profundo, basado en las técnicas exploradas de 'reverse engineering' y de 'instrumentación de código' pueda ser posible.

Aunque sigue habiendo elementos que no son seguros (se sigue sin realizar una Autenticación en las conexiones permitidas y las comunicaciones BLE en sí no están

cifradas), sí que se han encontrado elementos que han añadido seguridad a este nuevo sistema. Entre estos elementos que añaden seguridad se encuentran:

1. los 'comandos de control' mandados sí están cifrados
2. los 'comandos de control' encontrados en las trazas BLE capturadas siguen una lógica compleja, difícil de replicar y de inferir a través de ingeniería inversa
3. se ha añadido cierto nivel de ofuscación al código, con lo que se dificulta la ingeniería inversa

Cabe destacar que esta última medida coincide con una de las recomendaciones de OWASP para esta vulnerabilidad.

Aparte de estas medidas, ya que en la última salida del código instrumentado se puede ver una cierta lógica en la construcción de la clave de cifrado, sería interesante considerar para este caso elementos adicionales como:

4. gestión 'inteligente' de la clave de cifrado – por ejemplo, compartiendo la clave solo en el momento de emparejamiento inicial del dispositivo, con lo cual, aunque se intercepten las trazas intermedias no se podrá ya capturar esa clave, o cambiar la clave de manera periódica y/o dependiendo de mensajes intercambiados anteriormente (*rolling logic*), dificultando el análisis de dichos mensajes.

Se puede concluir que, en esta nueva versión de la bombilla inteligente, se han introducido medidas que efectivamente hacen el sistema más seguro, posiblemente aún posible de vulnerar, pero como se ha demostrado, no de manera sencilla.

Capítulo 4 - Conclusiones y trabajo futuro

Los sistemas IoT incluyen múltiples elementos que se pueden incluir en tres capas diferentes: la capa de Percepción que incluye los sensores, actuadores y elementos hardware que capturan los datos e interaccionan con el mundo físico; la capa de Red que se encarga de las comunicaciones entre dispositivos y con las aplicaciones de la capa superior; y por último la capa de Aplicaciones que es la que expone los servicios IoT a los usuarios finales.

En este tipo de sistemas hay un aspecto que afecta a todos los elementos y que es en el que se enfoca este trabajo: la seguridad.

Para evaluar la seguridad en un sistema IoT, resulta conveniente realizar una Auditoria de Seguridad sobre las posibles vulnerabilidades del sistema con el objetivo de ofrecer posibles soluciones. Para identificar qué vulnerabilidades se pueden encontrar, resulta conveniente apoyarse en la información facilitada por diferentes organismos y entidades internacionales, como son OWASP o NIST, que evalúan constantemente los riesgos de seguridad actuales, e incluso facilitan herramientas para su gestión.

Realizar una Auditoría de Seguridad completa es una tarea compleja, normalmente realizada por un equipo multidisciplinar. Aparte de ello, para elementos concretos sobre los que se quiera hacer un análisis de seguridad es posible realizar un Test de Penetración (*pentesting*) que consiste en una exploración exhaustiva de los elementos tanto HW como SW con el objetivo de para encontrar las debilidades de seguridad y poder proponer soluciones.

En este trabajo se ha realizado un pentesting sobre un dispositivo IoT: una bombilla inteligente, que es una bombilla que tiene conectividad con una aplicación móvil, a través de la cual se pueden controlar varias funciones como por ejemplo el encendido/ apagado de la bombilla o el cambio de color de su luz.

El pentesting realizado se ha enfocado especialmente en los riesgos de seguridad relacionados con la capa de comunicaciones, que en el caso analizado se basa en el protocolo Bluetooth Low Energy (BLE).

El protocolo BLE es una tecnología de comunicaciones inalámbrica, muy usada en entornos IoT, especialmente por su bajo consumo de energía manteniendo alcances aptos para su uso en redes de área personal (PAN, Personal Area Network), y que en capas inferiores se basa en el protocolo GATT para la exposición y gestión de los servicios (en formas de características) disponibles en un dispositivo.

En los casos estudiados, se han considerado dos versiones de la bombilla inteligente, con agujeros de seguridad conocidos. El estudio de estas vulnerabilidades se ha realizado siguiendo un manual llamado 'IoT Exploitation Lab' que ha facilitado una empresa de seguridad – Attify - para la realización de diferentes tipos de pentesting.

Los pasos seguidos para realizar dicho pentesting relacionada con las comunicaciones BLE son: descubrimiento del dispositivo y de los servicios expuestos, captura y análisis del tráfico BLE generado al realizar interacciones con el dispositivo para finalmente ver si es susceptible a un 'reply-attack'.

Como se ha comentado, se han estudiado dos versiones de bombilla inteligente: una 'básica' en la que los agujeros de seguridad no han sido mitigados, y una 'más segura' en la que sí.

Las principales vulnerabilidades encontradas para el caso de bombilla inteligente 'básica' han sido:

1. la conexión usada en este dispositivo no usa las capacidades de añadir seguridad del protocolo: la conexión BLE no está cifrada → esto implica que, tras interceptar las trazas de comunicación, éstas se pueden leer y analizar sin problema
2. el dispositivo no tiene capacidades de entrada/salida para realizar un emparejamiento (ni se fuerza exclusividad en el emparejamiento) → esto implica que cualquiera puede emparejarse e interactuar con el dispositivo.
3. los 'comandos de control' para gestionar la funcionalidad del dispositivo, encontrados en el payload de las trazas capturadas no están cifrados y siguen una lógica de construcción sencilla → esto implica que es posible construir otros 'comandos de control' similares tras un análisis sencillo y usarlos en un 'reply-attack' para tomar el control del dispositivo.

Cabe destacar que este tipo de vulnerabilidades coinciden de hecho con la vulnerabilidad identificada por OWASP en el #3 de su Top10 de vulnerabilidades IoT: 'Interfaces del ecosistema inseguras'.

Las propuestas de OWASP para mitigar estas vulnerabilidades, incluyen añadir mecanismos de Autenticación segura, técnicas de ofuscación o anti-tampering (para impedir su modificación) en la aplicación móvil. Aparte de estas propuestas, habría que considerar también añadir cifrado (tanto a nivel de protocolo como en el payload con los comandos de control).

En el caso de la bombilla inteligente 'más segura', las vulnerabilidades 1 y 2 siguen presentes, pero en el caso de la 3 se ha encontrado el siguiente escenario:

3. los 'comandos de control' encontrados en las trazas BLE capturadas sí están cifrados y siguen una lógica compleja, difícil de replicar y de inferir a través de ingeniería inversa

A lo que además añadimos

4. se ha añadido cierto nivel de ofuscación al código

En este último caso, relativos al punto 3 y 4, se han probado diferentes herramientas de 'Ingeniería Inversa' (Reverse Engineering) o de 'Instrumentación' del código, a pesar de lo cual no ha sido fácil encontrar una posible lógica para la construcción de 'comandos de control' que permitiesen tomar el control de la bombilla mediante un reply-attack.

Es por esto por lo que cabe concluir que añadir cifrado (a nivel de protocolo y de payload) así como considerar las mejoras indicadas por OWASP para este tipo de vulnerabilidades – especialmente las relacionadas con añadir ofuscación – resultan en sistemas IoT basados en BLE considerablemente más seguros, en los que no será fácil para un atacante tomar el control del dispositivo.

Para finalizar, como trabajo futuro relacionado con este Pentesting para la Bombilla Inteligente con comunicaciones basadas en BLE, se podría:

- por una parte, ampliar la información para el pentesting presente, haciendo un estudio más profundo para el segundo caso de la bombilla inteligente 'más segura', con las herramientas de Ingeniería Inversa estudiadas, para ver si efectivamente el nivel de seguridad añadido es suficiente o si se podría eventualmente llegar a un nivel de conocimiento suficiente para realizar un 'reply-attack'.
- por otra parte, ampliar la presente Auditoría de Seguridad para añadir otros elementos que no se han considerado en el presente trabajo, como por ejemplo el estudio de vulnerabilidades hardware – accediendo al dispositivo físico en sí (abriendo la bombilla) para poder estudiar sus componentes hardware o SoC (System on Chip).

Introduction

Motivation

The term Internet of Things (IoT - Internet of Things) was coined by Kevin Ashton as early as 1999 [1], but it is still very much in force in current information systems. He described a world where all 'things' have a digital identity of their own and are organized and managed by means of computers - in what we can call 'IoT System'.

From another perspective, the IoT is an environment where these 'things' have the ability to detect, control, send and transfer data to other elements and where the collected data will be visible through different applications.

Therefore, when designing and developing an IoT system, multiple elements must be taken into account, including: what hardware devices and sensors to use (and with what firmware), what communication protocols to use between the different components or what processing (middleware) and data will be exposed at a higher level (applications).

In the following image, we can see an example of an IoT system architecture, at a high level:

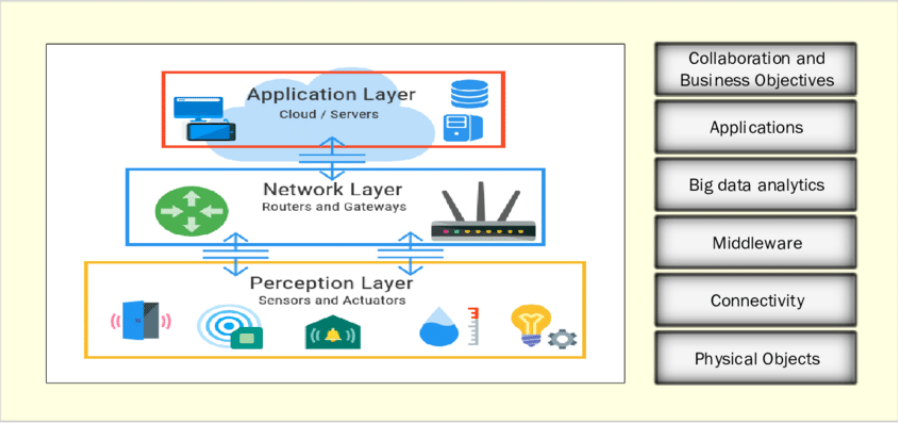


Ilustración 41: IoT Architecture example [2]

In this architecture it is observed how there are three differentiated layers:

- Perception layer that includes physical devices such as sensors and actuators

- Network layer: in which we can include both communication between devices, as well as with the applications of the upper layer (middleware, as a communication layer between hardware elements and software applications)
- Application layer: which in an IoT environment can be of a different nature depending on the need they cover (e.g. Smart City, Smart Home, Smart Lightning, etc.). This layer may or may not be supported by 'Big Data Analytics' components.

The selection criteria of some or other components in these layers will vary depending on the use case, but there is one aspect that is always important to consider, and on which we focus in this work: security.

In complex systems, with multiple layers and components, as in the case of IoT systems, all its components can be a source of security risks, therefore it is necessary to make a global analysis of it (security audit or 'pentesting') to be able to estimate with a certain degree of confidence if a system is secure or not.

There are international organizations, such as OWASP [3], which is an independent non-profit international foundation, made up of different companies, organizations and academic communities, which are dedicated to researching the different vulnerabilities in software systems. In addition, they provide a series of indications and tools to mitigate these vulnerabilities. OWASP in particular provides us with a list of 'top 10' vulnerabilities in IoT that can undoubtedly serve as a reference when conducting a security audit in an IoT system.

The vulnerabilities that apply at any given time are changing (new attacks and mitigations appear often), but in general, and at a high level we can divide them according to the nature of the analyzed component, including:

- Hardware (chipset, I / O ...)
- Software (Firmware, Web App, APIs ...)
- Communications (protocols: Wifi, BLE, Sigfox ...)

Therefore, in order to identify vulnerabilities in a system or specific element of it, a pentesting ('penetration testing') can be carried out. A pentesting is an attack on a computer system with the intention of finding its security weaknesses and everything that could have access to it, its functionality and data.

A pentesting will serve both to see if a system has exploitable security vulnerabilities, and to identify possible solutions to improve its security.

Objectives

The objective of this Master's thesis is to carry out a pentesting or security audit on an IoT device - in this case for a Smart Bulb (SmartBulb), of which we have considered 2 different models.

Ilustración 42: SmartBulb

A 'smart' light bulb is simply a light bulb that has connectivity with a mobile application, with which you can control both its on / off, as well as change the light colour.

In a full safety audit, all the components of the system would have to be analyzed, including both the hardware and the firmware of the bulb, which is a complex task, normally carried out by a multidisciplinary team. In my case, I have focused mainly on the field of communications used by the bulb (BLE or Bluetooth Low Energy) as well as a part of the code of the mobile application that controls the bulb.

The objectives of this pentesting are, on the one hand, to know if the device is exploitable or has any vulnerability, and, on the other hand, knowing said vulnerability to be able to identify possible solutions or mitigations to avoid it, and to be able to have mechanisms to develop safer systems in the field studied.

Workplan

To carry out this pentesting, I have relied on the 'IoT Exploitation Lab' manual from the company Attify [4]. This manual greatly facilitates pentesting in different areas by indicating which steps - usually at a high level - must be replicated to achieve it.

Ilustración 43: logo of Attify

Apart from the manual, this pentesting 'kit' includes the IoT devices on which the pentesting will be performed (and other types of hardware devices if necessary).

In my case, I have had 2 smart bulbs with known security holes and explained in the manual, but as seen during the tests they have been corrected in one of them (although not in the other).

The steps to follow, described in the manual to perform this pentesting, can be summarized as:

1. Capture BLE traffic between the SmartBulb and the mobile app when interacting with the bulb
2. Analyze the communication traces, to discover what characteristics and services are written to carry out said interaction. In addition, it is sought if there is any logic in the construction of these captured 'control commands' in which, for example, the color of the light of the bulb is changed.

3. Make a reply-attack based on the above information. A reply-attack consists of simply forwarding the captured information, in the same format, to the device and checking if that way it can be taken over.

.

Conclusions and future work

IoT systems include multiple elements that can be included in three different layers: the Perception layer that includes the sensors, actuators and hardware elements that capture data and interact with the physical world; the Network layer that is responsible for communications between devices and with the applications of the upper layer; and finally, the Applications layer, which is the one that exposes the IoT services to the end users.

In this type of system there is an aspect that affects all the elements and that is what this work focuses on: safety.

To evaluate the security in an IoT system, it is convenient to carry out a Security Audit on the possible vulnerabilities of the system in order to offer possible solutions. To identify which vulnerabilities can be found, it is convenient to rely on the information provided by different international organizations and entities, such as OWASP or NIST, which constantly assess current security risks, and even provide tools for their management.

Carrying out a complete Security Audit is a complex task, usually carried out by a multidisciplinary team. Apart from this, for specific elements on which a security analysis is to be carried out, it is possible to carry out a Penetration Test (pentesting) that consists of an exhaustive exploration of both HW and SW elements in order to find security weaknesses and to be able to propose solutions.

In this work, a pentesting has been carried out on an IoT device: a smart light bulb, which is a light bulb that has connectivity with a mobile application, through which various functions can be controlled, such as the on / off of the light bulb or the color change of your light.

The pentesting carried out has focused especially on the security risks related to the communications layer, which in the case analyzed is based on the Bluetooth Low Energy (BLE) protocol.

The BLE protocol is a wireless communications technology, widely used in IoT environments, especially due to its low energy consumption, maintaining ranges suitable for use in personal area networks (PAN, Personal Area Network), and that in lower layers is based on the GATT protocol for the exposure and management of services (in feature forms) available on a device.

In the cases studied, two versions of the smart bulb have been considered, with known security holes. The study of these vulnerabilities has been carried out following a manual called 'IoT Exploitation Lab' provided by a security company - Attify - to carry out different types of pentesting.

The steps followed to carry out said pentesting related to BLE communications are: discovery of the device and the exposed services, capture and analysis of the BLE traffic generated when making interactions with the device to finally see if it is susceptible to a 'reply-attack'.

As mentioned, two versions of the smart light bulb have been studied: a 'basic' one in which security holes have not been mitigated, and a 'more secure' one in which they have.

The main vulnerabilities found for the 'basic' smart bulb case have been:

1. The connection used in this device does not use the capabilities to add security of the protocol: the BLE connection is not encrypted → this implies that, after intercepting the communication traces, they can be read and analyzed without problem
2. The device does not have input / output capabilities to perform a pairing (nor is exclusivity enforced on pairing) → this implies that anyone can pair and interact with the device.
3. The 'control commands' to manage the functionality of the device, found in the payload of the captured traces, are not encrypted and follow a simple construction logic → this implies that it is possible to construct other similar 'control commands' after an analysis simple and use them in a 'reply-attack' to take control of the device.

It should be noted that these types of vulnerabilities actually coincide with the vulnerability identified by OWASP in # 3 of its Top10 IoT vulnerabilities: 'Insecure Ecosystem Interfaces'.

OWASP's proposals to mitigate these vulnerabilities include adding Secure Authentication mechanisms, obfuscation or anti-tampering techniques (to prevent their modification) in the mobile application. Apart from these proposals, we should also consider adding encryption (both at the protocol level and in the payload with the control commands).

In the case of the 'more secure' smart bulb, vulnerabilities 1 and 2 are still present, but in the case of 3 the following scenario has been found:

3. The 'control commands' found in the captured BLE traces are encrypted and follow a complex logic, difficult to replicate and infer through reverse engineering

To which we also add

4. Some level of obfuscation has been added to the code

In the latter case, relative to points 3 and 4, different 'Reverse Engineering' or 'Instrumentation' tools of the code have been tested, despite which it has not been easy to find a possible logic for the construction of 'control commands' that allowed to take control of the light bulb by means of a reply-attack.

Therefore, it can be concluded that adding encryption (at the protocol and payload level) as well as considering the improvements indicated by OWASP for these types of vulnerabilities - especially those related to adding obfuscation - result in BLE-based IoT systems considerably more. safe, in which it will not be easy for an attacker to take control of the device.

Finally, as future work related to this Pentesting for the Smart Bulb with BLE-based communications, one could:

- on the one hand, expand the information for the present pentesting, making a more in-depth study for the second case of the 'safer' smart light bulb, with the reverse engineering tools studied, to see if the added level of security is indeed sufficient or if a sufficient level of knowledge could eventually be reached to carry out a 'reply-attack'.
- on the other hand, expand this Security Audit to add other elements that have not been considered in this work, such as the study of hardware vulnerabilities - accessing the physical device itself (opening the light bulb) in order to study its components hardware or SoC (System on Chip).

.

BIBLIOGRAFÍA

[1] «https://www.rfidjournal.com/that-internet-of-things-thing,» [En línea].

[2] M. &. H. V. &. O. S. Tavana, «IoT-based Enterprise Resource Planning: Challenges, Open Issues, Applications, Architecture, and Future Research Directions. Internet of Things. 11. 10.1016/j.iot.2020.100262,» 2020.

[3] «OWASP,» [En línea]. Available: https://owasp.org/www-project-internet-of-things/.

[4] «Attify - "Offensive IoT Exploitation":,» [En línea]. Available: https://www.attify.com/iot-security-exploitation-training.

[5] «https://www.f5.com/labs/articles/education/what-is-the-cia-triad,» F5labs, 2019. [En línea].

[6] «NIST,» [En línea]. Available: https://nvd.nist.gov.

[7] «IOT Inspector report,» [En línea]. Available: https://cvmdp.ucm.es/moodle/pluginfile.php/1040804/mod_resource/content/1/IoT_Inspector_Sample_Report%20%281%29.pdf.

[8] OWASP, «OWASP-IoT-Top-10,» 2018. [En línea]. Available: https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf.

[9] O. M. I. vulnerability. [En línea]. Available: https://wiki.owasp.org/index.php/Top_10_2014-I7_Insecure_Mobile_Interface.

[10] C. C. A. &. R. D. Kevin Townsed, Getting Started with Bluetooth Low Energy, O'Reilly, 2014.

[11] R. Heydon, Bluetooth Low Energy: The Developer's Handbook, Prentice Hall, 2013.

[12] «BLE [wikipedia],» [En línea]. Available: https://es.wikipedia.org/wiki/Bluetooth_de_baja_energ%C3%ADa.

[13] IoTLab, «BT vs BLE,» [En línea]. Available: https://iotlab.tertiumcloud.com/2020/08/19/classic-bluetooth-vs-bluetooth-low-energy-ble/.

[14] Cifrado-minubeinformática, «http://minubeinformatica.com/cursos/seguridad-informatica/criptografia/,» [En línea].

[15] gatttool, «http://manpages.ubuntu.com/manpages/cosmic/man1/gatttool.1.html,» [En línea].

[16] hcitool, «https://helpmanual.io/help/hcitool/,» [En línea].

[17] HappyLighting, «https://play.google.com/store/apps/details?id=com.xiaoyu.hlight&hl=es&gl=US, » [En línea].

[18] wireshark, «https://www.wireshark.org/,» [En línea].

[19] HaoDeng, «https://play.google.com/store/apps/details?id=com.zengge.telinkmeshlight&hl=es_419&gl=US,» [En línea].

[20] Mozilla, «Firefoz APK downloader plugin,» [En línea]. Available: https://addons.mozilla.org/es/firefox/addon/apkgk-downloader/.

[21] JADX, «https://github.com/skylot/jadx,» [En línea].

[22] AES, «https://es.wikipedia.org/wiki/Advanced_Encryption_Standard,» [En línea].

[23] Wikipedia, «AES ECB CBC,» [En línea]. Available: https://es.wikipedia.org/wiki/Modos_de_operaci%C3%B3n_de_una_unidad_de_cifrado_por_bloques#Modo_ECB_(Electronic_codebook).

[24] rgrep, «https://www.commandlinux.com/man-page/man1/rgrep.1.html,» [En línea].

[25] B. Ninja, «https://binary.ninja/,» [En línea].

[26] Frida, «https://frida.re/,» [En línea].