Análisis de dispositivos IoT: Smart Switchs Review

Cristian Borghello[1], Emanuel Orzuza[2], Walter Cerrudo[3], Aldo Sigura[4], Mariano Wasinger[5], Valentin Baez[6]

Abstract. Los dispositivos IoT han venido a cambiar la forma en que el usuario maneja sus más sensibles datos e información. Los Smart Switch permiten convertir en *inteligentes* a dispositivos que no estaban preparados para tal fin, anexando la posibilidad de encender o apagar aparatos de la más variada índole: desde luces hasta alarmas, sensores o regadores, entre tantos más, tomando control a distancia del mismo desde alguna aplicación. Este artículo pretende evaluar la seguridad de estos denominados *Smart Swtichs*, tomando como premisa los valores de confidencialidad, integridad y disponibilidad

Keywords: Smart Switch, IoT, hacking, Smart House, seguridad.

Abstract. IoT devices have come to change the way the user handles their most sensitive data and information. The Smart Switches allow devices that were not prepared for this purpose to be made intelligent, attaching the possibility of turning on or turning off devices of the most varied nature: from lights to alarms, sensors or sprinklers, among many others, taking remote control of that from an app. This article aims to evaluate the security of these Smart Swtichs, taking as a premise the values of confidentiality, integrity and availability

Keywords: Smart Switch, IoT, hacking, Smart House, security.

1 Licenciado (UTN) en Sistemas y certificado internacional en seguridad de la información. Creador y Director de los sitios Segu-Info -www.segu-info.com.ar- y Segu-Kids -www.segu-kids.org- especializados en Seguridad de la Información.
Contacto: cborghello@segu-info.com.ar

2 Licenciado en Sistemas de Información (UADER). Docente Universitario - UADER. Integrante del Laboratorio de Seguridad de la Información (LASI) – UADER. Contacto: emaorzuza@gmail.com

3 Licenciado en Sistemas Informáticos (UADER). Integrante de LASI – UADER. Contacto: waltercerrudo@gmail.com

4 Ingeniero electromecánico (UTN). Master en Informática Aplicada a la Ingeniería y la Arquitectura (Universidad de La Habana). Docente universitario UADER- UNER. Integrante de LASI – UADER. Contacto: aldo.sigura@gmail.com

5 Estudiante de la carrera de Licenciatura en Sistemas de Información. Integrante de LASI – UADER. Contacto: waltermarianow@gmail.com

6 Estudiante de la carrera de Licenciatura en Sistemas de Información. Integrante de LASI – UADER. Contacto: valentin.baez2@gmail.com

Introducción

El término IoT (Internet de las Cosas) se refiere a escenarios en los que la conectividad de red y la capacidad de cómputo se extienden a objetos, sensores y artículos de uso diario que habitualmente no se consideran computadoras, permitiendo que estos dispositivos generen, intercambien y consuman datos con una mínima intervención humana [1].

Internet de las cosas se presenta como un gran avance que permite que diferentes productos de uso personal, electrodomésticos, automóviles y demás bienes, utilicen información proveniente de internet para su funcionamiento. Esto trae aparejado muchísimas ventajas, pero también exhibe nuevas ventanas de ataques: **IoT introduce una amplia gama de nuevos riesgos y desafíos de seguridad para los propios dispositivos IoT, sus plataformas y sistemas operativos e incluso los sistemas a los que están conectados.**

Los dispositivos de IoT impulsan y seguirán impulsando ataques dirigidos a explotar sus debilidades de configuración, los errores de seguridad y la escasa interacción de los consumidores con sus configuraciones [2].

Los enchufes inteligentes o *Smart Switch* vienen a transformar equipos que no poseían esta capacidad, brindando al cliente una herramienta que permite encender o apagar un dispositivo conectado al Smart Switch desde cualquier lugar, mediante una aplicación residente en algún dispositivo móvil. Esta nueva funcionalidad debería garantizar que la conexión entre el dispositivo y la aplicación encargada de manejarlo sea segura y confiable; dado la sensibilidad y variabilidad de los equipos que pueden disponerse para su conexión.

Casos de estudio

Este artículo toma dos Smart Switch diferentes, de distintos proveedores, marca y modelo; con el objeto de analizar la forma en que cada uno de ellos maneja la información que procesa, transmite y/o almacena.

Se examina a lo largo del documento un dispositivo HS100 de la empresa TP-Link y un Smart Switch Basic de la empresa Sonoff. Se han elegido estos dos dispositivos dado que se trata de las marcas más vendidas y comercializadas a nivel mundial[7]. Son equipos de bajo costo, disponibles en cualquier tienda en línea.

El análisis que se realiza en ambos dispositivos es el mismo: contempla una inspección visual, recolección de información, análisis de puertos abiertos y el tráfico de los datos transmitidos entre la aplicación y el dispositivo; indagando sobre el cifrado aplicado.

1. Caso 1: Smart Switch TP-Link HS 100

El HS100 es categorizado y promocionado por TP-LINK como un *smart plug*, que permite *"controlar dispositivos conectados al Enchufe Inteligente en cualquier sitio en*

[7] https://www.amazon.es/gp/bestsellers/tools/15399077031. Dispositivos más vendidos en Amazon; consultado el 15 de abril de 2019

el que tengas internet utilizando en tu smartphone la app gratuita Kasa"[8]; además de brindar programación de eventos automáticos, que permitirían encender y apagar otros dispositivos conectados al smart plug.

1.1 Inspección visual

A partir del FCCID (número de identificación de la Comisión Federal de Comunicaciones por sus siglas en inglés) publicado e impreso sobre el dispositivo, podemos identificar características propias y obtener, por ejemplo, fotos externas[9] e internas[10] del dispositivo.

Mediante esta inspección, podemos observar que el HS100 está dotado de un *chip Qualcom QCA4010.* Este MCU es conocido y podemos encontrar su descripción en la página oficial del fabricante[11]. Con una simple observación, podemos entonces conocer el chip que usa para conectividad y la memoria y procesador que tiene el mismo, entre otras particularidades.

Otras características extraíbles a simple vista del dispositivo, son:

- Modelo HS100(us)
- FCCID TE7HS100V2
- Versión del Hardware 2.0.
- S/N: 2179812005950
- MAC: 704F57F9C7CF

1.2 Software KASA

KASA es el nombre de la aplicación de la empresa TP-LINK que permite la administración del dispositivo HS100 desde el celular, tablet o cualquier otro dispositivo que tenga Android o iOS como sistema operativo.

La aplicación permite vincular el dispositivo (HS100) a nuestra cuenta de TP-LINK (es mandatario crear una cuenta para la administración), de manera tal de poder gestionarlo desde cualquier lugar, a través de la aplicación KASA[12]. Para poder examinar y analizar en detalle la aplicación, se descargó la misma en su *versión 2.13.0.858*, desde el sitio público APK PURE[13].

La aplicación fue descomprimida y descompilada con la herramienta Open Source "*jadx"*, para poder recuperar el código en JAVA y analizarlo. Las rutinas de cifrado y

---

[8] TP-LINK HS100, características. https://www.tp-link.com/es/home-networking/smart-plug/hs100/ . visitado el 28/3/2019

[9] Fotos externas del dispositivo HS100: https://fccid.io/TE7HS100V2/External-Photos/External-Photos-3476539 ; visitado el 28/3/2019

[10] Fotos internas del dispositivo HS100: https://fccid.io/TE7HS100V2/Internal-Photos/Internal-Photos-3476540 ; visitado el 28/3/2019

[11] Página Oficial: https://www.qualcomm.com/products/qca4010 ; visitado el 29/3/2019

[12] Descripción de KASA : https://www.tp-link.com/us/kasa-smart/kasa.html ; consultado el 2 de Julio de 2019

[13] Descarga de KASA: https://apkpure.com/es/kasa-smart/com.tplink.kasa_android/download ; consultado el 2 de Julio de 2019

descifrado (*public static encode y public static decode*), como puede observarse en la captura de abajo, están en la clase:

com.tplinkra.tpcommon.tpclient.TPClientUtils

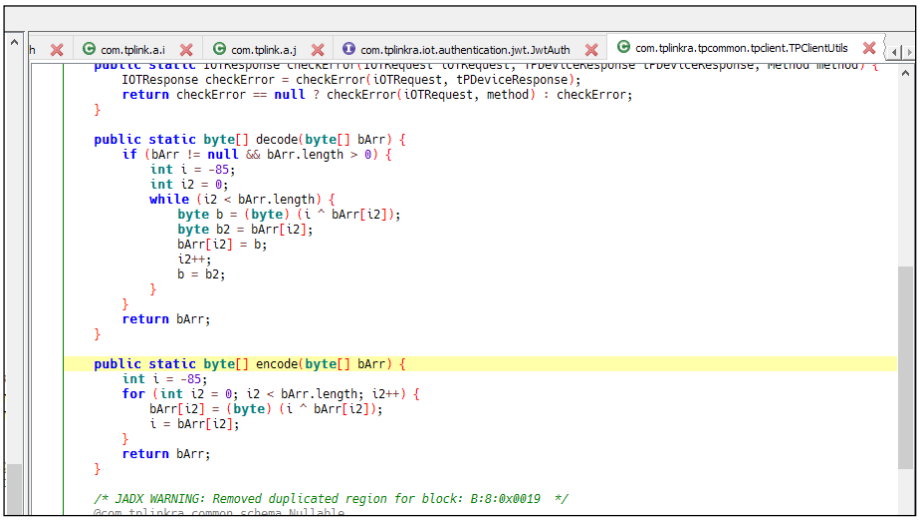


Imagen 1. Funciones de encriptación y desencriptación que usa KASA de TP-Link

En la función de cifrado *(encode)* podemos observar que la clave inicial (vector de inicialización) *"i"* tiene un valor en duro ("*harcodeado")* de "*-85"*. El primer *byte* del texto plano es **XORed** con la clave; en la siguiente línea, la clave es reemplazada con el valor *byte* del carácter sin cifrar. Durante la siguiente iteración, el siguiente *byte* de texto plano es **XORed** con el *byte* de texto sin formato anterior.

El descifrado funciona de la misma manera, con el flujo de claves formado por *bytes* de *cyphertext*. Esto se conoce como cifrado automático y, si bien tiene mejores propiedades estadísticas que el cifrado XOR simple con una clave de repetición, puede romperse fácilmente con ataques de texto simple conocidos.

1.3 Escaneo de puertos y análisis de comunicación

Conociendo como cifra la comunicación el software KASA, interesa conocer los puertos que tiene abierto por defecto el dispositivo HS100, para indagar además que servicios presta y mediante qué protocolos.

Lo primero que se realizó fue un escaneo básico con la herramienta Open Source *"nmap"* dentro de nuestra red privada, para conocer que puertos abiertos tiene el dispositivo, mediante el siguiente comando:

```
nmap -p- 192.168.0.17 ; conociendo que la IP local tomada por el
dispositivo es la que figura a la derecha del comando
```

El resultado obtenido indica que el puerto *"9999/TCP"* es el único que se encuentra abierto. Este puerto es el que se debería utilizar entonces, para comunicar el dispositivo con la aplicación KASA. Utilizando la técnica de *MitM (Man in the Middle*, por sus siglas en inglés), se procedió a interceptar y analizar el tráfico; utilizando los aplicativos Open Source *Ethercap* para realizar *spoofing* y *Wireshark* para monitorizar el tráfico.

Se seleccionó los dos objetivos para el ataque de *ARP Spoofing* (el dispositivo HS100 y el dispositivo celular donde reside la aplicación KASA) y posteriormente se procedió a "*sniffear*" la red local. Como se puede observar en la siguiente imagen, la aplicación Kasa Smart se comunica con el dispositivo HS100 a través del puerto *9999* utilizando el protocolo TCP y UDP, enviando y recibiendo datos.

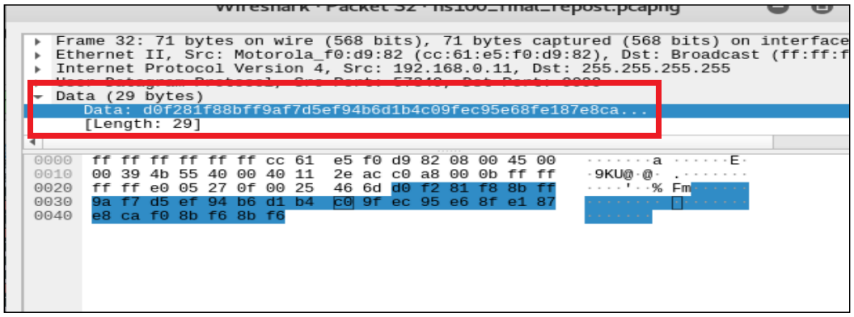


Imagen 2. Captura de tráfico entre el dispositivo HS100 y la aplicación KASA

El tráfico, como se suponía, está cifrado (utilizando la función descripta en 1.2). Para poder identificar a simple vista el tráfico, se utilizó un *dissector* de Wireshark escrito en el lenguaje LUA[14]; de manera que descifre automáticamente los paquetes de la comunicación entre Kasa Smart y el dispositivo HS100 a través del puerto *9999*.

Realizando la captura nuevamente, se puede observar el *dissector* en acción, permitiendo ver en texto plano la comunicación interceptada, que en este caso es un comando de encendido/apagado, con *"state: 1"* para encender y *"state: 0"* para apagar el dispositivo.

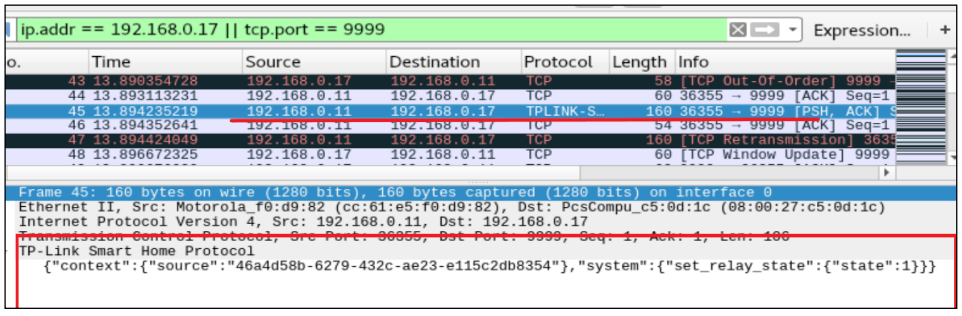


Imagen 3. Captura de tráfico entre el dispositivo HS100 y la aplicación KASA, utilizando el *Dissector*

Es factible, entonces, monitorear las comunicaciones entre la aplicación Kasa y el Smart Plug en una red local, por lo que podríamos comenzar a relevar los distintos comandos que son enviados desde la aplicación Kasa Smart al dispositivo HS100.

En primera instancia se ha podido notar que la carga útil (*payload)* enviado en las comunicaciones consiste un objeto JSON, donde se detalla el comando, el tipo de comando, y algunos parámetros. El dispositivo en su respuesta también devuelve un objeto JSON con información o el resultado de la operación.

[14] Dissector for TP-LINK smart https://github.com/softScheck/tplink-smartplug/blob/master/tplink-smarthome.lua ; consultado el 06 de junio de 2019

Algunos ejemplos, son:

`{"system":{"get_sysinfo":{}}};` *Devuelve información del dispositivo*

`{"system":{"reboot":{"delay":1}}};` *Reinicia el dispositivo*

1.4 ONOFF Con Python

Conociendo como cifra y descifra el software KASA la información transmitida hacia el dispositivo, analizar la comunicación entre ambos y develar que entre el aplicativo y el dispositivo (o al menos entre las órdenes enviadas desde uno hacia el otro) no existe una validación de una relación de confianza, se desarrolló un *Script en Python v3.6*. Esto permite aprovechar esta vulnerabilidad, explotándola de manera tal que, por ejemplo, se pueda encender o apagar el dispositivo sin haber generado una relación de confianza entre quien envía la orden y el smart plug, siendo el único requisito estar conectados en la misma red LAN. Las imágenes siguientes grafican esta situación:

```
+ IP = '192.168.0.22'
+ Puerto = 9999
+ Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
+ Socket.connect((IP, Puerto))
+ data = codecs.decode("0000002ad0f281f88bff9af7d5ef94b6c5a0d48bf99cf091e8b7c4b0d1a5c0e2d8a381f286e793f6d4eedfa2dfa2","hex")
+ Socket.send(data)
+ data = codecs.decode("00000025d0f281e28aef8bfe92f7d5ef94b6d1b4c09ff194ec98c7a6c5b1d8b7d9fbc1afdab6daa7da","hex")
+ Socket.send(data)
+ Socket.close()
```

Imagen 4. Script que enciende el HS100, estableciendo una conexión a través del puerto 9999, siendo la IP del dispositivo 192.168.0.22

El *script* envía la cadena codificada en formato hexadecimal, obtenida en el análisis con Wireshark, a la IP correspondiente al Plug y especificando el puerto *9999*; provocando el encendido (imagen 4) o el apagado (imagen 5) sin establecer ninguna relación de confianza entre el dispositivo y quien envía la orden.

```
+ IP = '192.168.0.22'
+ Puerto = 9999
+
+ Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
+ Socket.connect((IP,Puerto))
+ data = codecs.decode("0000002ad0f281f88bff9af7d5ef94b6c5a0d48bf99cf091e8b7c4b0d1a5c0e2d8a381f286e793f6d4eedea3dea3","hex")
+ Socket.send(data)
+ data = codecs.decode("0000002dd0f281f88bff9af7d5ef94b6c5a0d48bf99cf091e8b7c4b0d1a5c0e2d8a381e496e4bbd8b7d3b694ae9ee39ee3","hex")
+ Socket.send(data)
+ Socket.close
```

Imagen 5. Script que apaga el HS100, estableciendo una conexión a través del puerto 9999, siendo la IP del dispositivo 192.168.0.22

1.5 Primeras conclusiones

En una conexión LAN no existe autenticación alguna necesaria para acceder al dispositivo, por lo cual es posible realizar acciones como encender/apagar el dispositivo, o especificar la URL desde donde realizar la actualización del *firmware* del dispositivo, esto podría permitir que atacantes realicen actualizaciones de *firmware* modificados.

Estando conectados a la misma red Wi-Fi del dispositivo, es posible hacerse del control total sobre el dispositivo: el mismo no valida quien envíe la petición, siempre que la misma pueda ser legible (descifrada) por el Smart Switch

2. Caso 2: Sonoff Smart Switch Basic

Sonoff publicita a su Smart Switch básico como *"un interruptor inteligente Wi-Fi que proporciona a los usuarios control del hogar, pudiendo conectar el mismo a una amplia gama de electrodomésticos; transmitiendo datos a una plataforma en la nube, permitiendo a los usuarios controlar de forma remota todos los dispositivos conectados a través de la aplicación móvil eWeLink"*[15].

2.1 Inspección visual

A través del FCCID[16] se ha podido observar características propias del Sonoff. Este Smart Switch está dotado de un módulo ESP8285, muy conocido y utilizado dentro de IoT por su bajo coste y altas prestaciones. A diferencia de TP-LINK, Sonoff no imprime visualmente la MAC en el dispositivo, ni tampoco su número de serie. Solo especifica que soporta las normas *802.11 b/g/n* y su capacidad máxima de carga (10A).

2.2 eWeLink Software

eWeLink es la aplicación que recomienda Sonoff para el uso de sus dispositivos. Soporta múltiples protocolos y fabricantes de hardware. Para realizar su análisis, nuevamente se desempaqueto la aplicación con *Jadx*, como se hizo en 1.2.

Dentro de la aplicación desempaquetada se ha observado que existe un archivo llamado *index.android.bundle* que contiene múltiples funciones en lenguaje JavaScript, entre las cuales se encuentran algunas de cifrado y descifrado; además de los parámetros de apareamiento de la Red Wi-Fi generada por el dispositivo de manera *hardcodeada*, tal cual se demuestra en imagen 6**.**

```
    S = y.sent, E.default.log(D, 'doGetRequestFromServer response:', S), F.default.hiddenLoading(), S && S.h
        deviceid: u,
        deviceSSID: f,
        pairMode: S.data.pairMode,
        deviceIp: 0 === S.data.pairMode ? '10.10.7.1' : '192.168.1.1',
        deviceSsidPwd: 0 === S.data.pairMode ? '12345678' : ''
    }, S.data.hasOwnProperty('secretKey') && (O.secretKey = S.data.secretKey), 1 === S.data.pairMode && (O.i
        pairType: p.default.PAIR_TYPE.SOFT_AP
    })) : o.navigate('PairErrorScreen', {
        title: h.default.getErrorCode().AP_PAIR.GetDeviceMetaFromServerFail.TITLE,
        content: h.default.getErrorCode().AP_PAIR.GetDeviceMetaFromServerFail.CONTENT,
        supportFunction: h.default.getErrorCode().AP_PAIR.GetDeviceMetaFromServerFail.FUNCTION
    });
case 24:
```

Imagen 6. Datos hardcodeados que incluyen la password de la red Wi-Fi que genera el dispositivo

El equipo genera su propia red Wi-Fi para aparearse al dispositivo que lo configurará (en este caso un celular con eWeLink instalado). La red generada tiene *SSID "ITEAD-10000XXX"* donde los últimos dígitos se corresponden al ID del dispositivo; la contraseña tal cual puede observarse en el código fuente de la figura 6 es *"12345678"*.

[15] https://sonoffargentina.com/sonoff-basic/

[16] https://fccid.io/2APN5BASICR2/Internal-Photos/Internal-Photos-4642803 ; consultado 10 de abril de 2020

Vinculando cualquier dispositivo a esta red Wi-Fi, podemos obtener datos como los detallados a continuación.

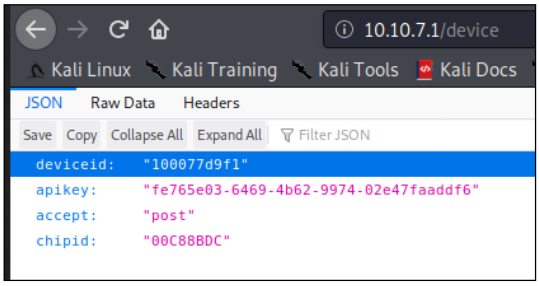
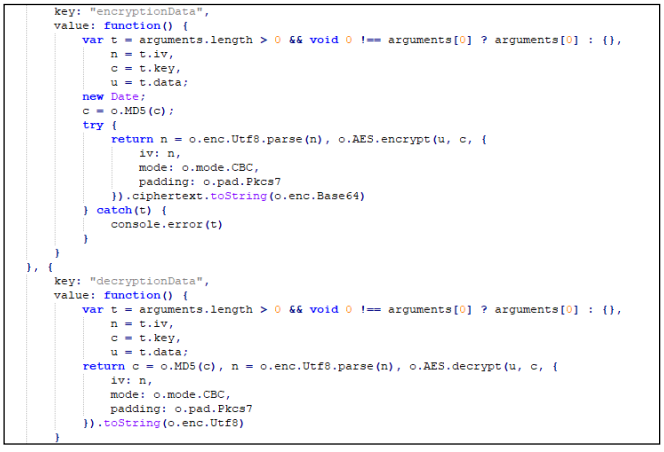


Imagen 7. Datos del dispositivo al invocar la URL 10.10.7.1/device

El Smart Switch envía a la aplicación, mediante un JSON, información referente al dispositivo: *deviceid*, *apikey* y *chipid*. Esto se utiliza para agregar el switch a la lista de dispositivos gestionados por la aplicación móvil e identificar el mismo. Además, mediante una consulta HTTP la aplicación transfiere los datos de la red Wi-Fi a la cual el dispositivo luego se conectaría para funcionar. Esto último es logrado haciendo una consulta a la URL *http://10.10.7.1/ap*.

eWeLink tiene dentro de su estructura funciones de cifrado y descifrado, como se ha comentado arriba. La aplicación usa el método de cifrado *AES-CBC PKCS7* y utiliza un vector de inicialización no aleatorio que recibe como parámetro. Se ha podido inducir, también, que la llave *(t.key)*, el vector de inicialización *(t.iv)* y la información a cifrar/descifrar *(t.data)* es recibida por parámetros; advirtiendo además que a la clave se le aplica un hash MD5 y que el vector de inicialización es codificado a UTF8.

La librería utilizada para generar el *hashing MD5*, el cifrado mediante AES, y otras funciones criptográficas se corresponde a la librería Open Source *CryptoJS*[17], lo que brindaría la posibilidad de poder analizar cómo funcionan los algoritmos de cifrado que utiliza, si se trata de una clave de 128 o 256 bits, entre otros detalles.



```
    key: "encryptionData",
    value: function() {
        var t = arguments.length > 0 && void 0 !== arguments[0] ? arguments[0] : {},
            n = t.iv,
            c = t.key,
            u = t.data;
        new Date;
        c = o.MD5(c);
        try {
            return n = o.enc.Utf8.parse(n), o.AES.encrypt(u, c, {
                iv: n,
                mode: o.mode.CBC,
                padding: o.pad.Pkcs7
            }).ciphertext.toString(o.enc.Base64)
        } catch(t) {
            console.error(t)
        }
    }
}, {
    key: "decryptionData",
    value: function() {
        var t = arguments.length > 0 && void 0 !== arguments[0] ? arguments[0] : {},
            n = t.iv,
            c = t.key,
            u = t.data;
        return c = o.MD5(c), n = o.enc.Utf8.parse(n), o.AES.decrypt(u, c, {
            iv: n,
            mode: o.mode.CBC,
            padding: o.pad.Pkcs7
        }).toString(o.enc.Utf8)
    }
```

Imagen 8. Funciones de cifrado y descifrado que usa eWeLink

[17] Librería CryptoJS https://cryptojs.gitbook.io/docs/; consultado 20 de mayo de 2020

2.3 Escaneo de puertos y análisis de comunicación

Tal como se ha procedido en 1.3, utilizando *"nmap"*, para escanear todos los puertos abiertos en el dispositivo (IP local *192.168.1.33*) se logró detectar que el único puerto abierto originalmente es el *8180/TCP*.

Utilizando la misma técnica que con el HS100 (*ARP Spoofing* con *Ettercap* y captura de paquetes con *Wireshark*), teniendo en cuenta que el dispositivo tiene *IP 192.168.1.33* y el celular donde reside la app eWeLink tiene dirección *192.168.1.11*, se ha podido observar que existen múltiples paquetes que se envían entre estos nodos, alguno de ellos utilizando el protocolo HTTP y teniendo como dato un objeto de tipo JSON. El puerto destino, si se trata del dispositivo, siempre se corresponde al *8180/TCP*.

Tomando un paquete en particular y realizado análisis de su contenido, se ha podido identificar que el JSON envía datos del dispositivo (conocidos); con el anexo de un campo *data* que tiene contenido cifrado. Este contenido, por su estructura, utiliza el método AES: tiene un vector de inicialización *(iv)*; por lo que es asumible que la función detallada en 2.2 sea la que cifre este contenido.

El objeto JSON enviado desde la aplicación eWeLink hacia el Sonoff, tiene la forma:


```
{
"sequence":"1590459569326",
"deviceid":"100077d9f1",
"selfApikey":"74e6857f-8358-4045-ac6c-39345bd5ae72",
"iv":"ODc2ODE4OTgzNzE3NzA1OA==",
"encrypt":true,
"data":"VDupthSEIc4kLkT1FxHGFg=="
}
```

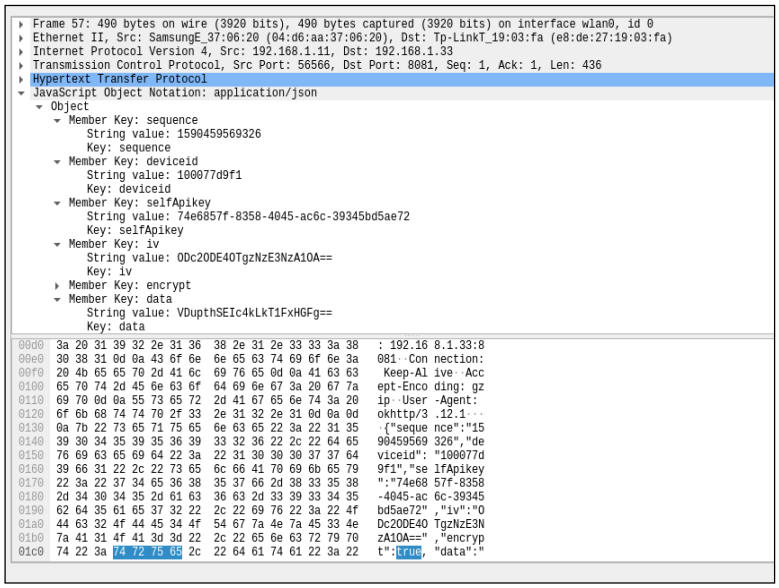



Imagen 9. Captura de un paquete enviado desde la aplicación al Smart Switch Sonoff

También se sabe, por el paquete capturado, que este objeto JSON es enviado mediante POST, a través del puerto *8180*, a *"/zeroconfig/switch";* por lo que resulta interesante enviar esta petición al dispositivo, para conocer cómo se comporta.

Para ello usamos la herramienta Postman; enviado el JSON de arriba tal cual se detalla en la imagen siguiente:

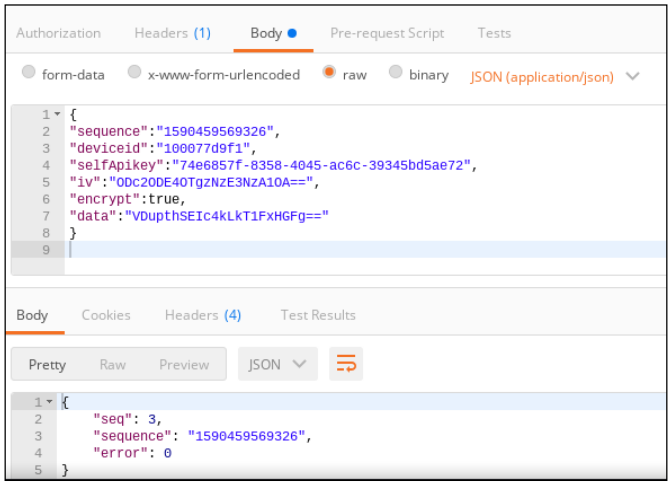


Imagen 10. Envío de un JSON al dispositivo Sonoff usando Postman

En efecto, se ha podido comprobar que el objeto JSON enviado funciona, dando como resultado el encendido del dispositivo, sin autenticación alguna de quien envía la solicitud, dado que se envió desde una PC de la red que nunca había sido apareada al Smart Switch. La respuesta desde el dispositivo es un JSON que contiene el número de secuencia ejecutada desde que se encendió *(seq)* -este dato se reinicia a *"0"* cada vez que se apaga el switch-, el número de secuencia a la que corresponde esta ejecución (secuencia solicitada *sequence*) y si ha arrojado algún error.

Puede observarse que alguno de los datos enviados por POST se corresponde a los campos *data, iv,* y *apikey*. Dado que se trata de datos posiblemente involucrados en el cifrado y descifrado, se han suprimido y modificado estos campos del JSON para analizar el comportamiento de la respuesta.

Luego de una serie de pruebas, se ha podido concluir que, si se cambia alguno de los campos del objeto, la respuesta sigue siendo correcta; salvo que estos campos sean el vector de inicialización, *encrypt* o *data*; pudiendo asumir entonces que estos son, en parte, los datos utilizados para descifrar los paquetes de datos. Los demás campos enviados no interfieren en la orden de encendido/apagado que recibe el dispositivo; al menos en la LAN.

Si se suprime alguno de los datos (cualquiera de los atributos del JSON) el dispositivo no puede procesar el objeto incompleto, quedando sin entregar respuesta, permanece tildado sin poder recibir peticiones, desde la aplicación que lo tiene vinculado, provocando entonces una denegación de servicio (DoS).

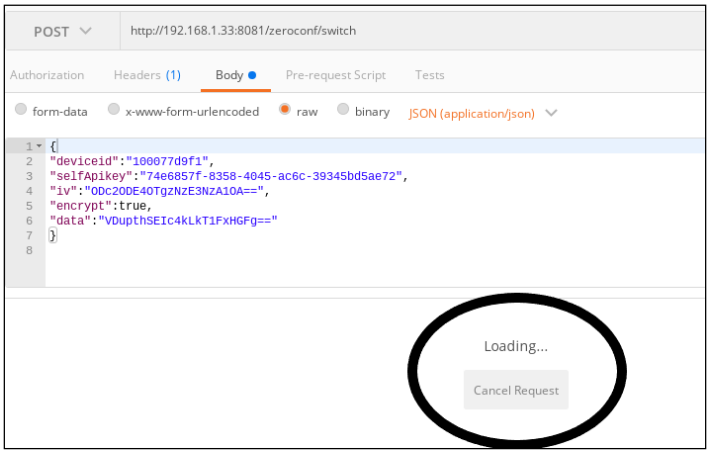


Imagen 11. Cuando se envía un JSON incompleto mediante POST al dispositivo Sonoff, queda tildado sin dar respuesta (se envía en este caso JSON sin campo *sequence*)

Capturando más paquetes y realizando pruebas del tipo ensayo/error, con Postman, mediante el envío POST hacia el puerto *8180* de estos objetos JSON, es fácilmente es deducible cuales cadenas encienden o apagan el dispositivo aun desconociendo qué información transmite (cifrada) en el campo *data*. Por lo que se tendría control total del dispositivo (encendido, apagado, reinicio), en la LAN, simplemente conociendo su dirección IP. Este control, además, permitirá generar una denegación de servicios como ya se ha detallado.

2.4 Conclusiones. Próximos pasos

El dispositivo puede ser controlado desde la LAN sin autenticarse de ninguna forma: el Smart Switch no valida si el dispositivo que envía la solicitud de encendido/apagado es de su confianza. Además, cuando no puede procesar un objeto JSON queda tildado y no recibe otro paquete por un tiempo determinado, pudiendo realizar de manera muy simple una denegación de servicios. Como ocurre con TP-LINK HS100, en la LAN se puede tomar control total, sin autenticación o validación alguna.

Podríamos decir que el vector de inicialización enviado en el paquete forma parte del proceso de cifrado y descifrado de los datos; dado que si sufre alguna alteración el dispositivo tampoco puede procesar el objeto. No sucede esto último con los demás atributos, por lo que no estarían implicados en el proceso de descifrado, al menos de este objeto.

Conociendo que eWeLink utiliza AES-CBC tal como se ha detallado y que el vector de inicialización involucrado en los procesos criptográficos pareciera ser el enviado en el paquete, los próximos pasos a seguir serán estudiar cual es la clave *(key)* utilizada, estudiando el algoritmo y el software en profundidad. Además, también interesa saber cómo son los procesos de envío/recepción de datos cuando no se trabaja en entornos LAN, dado que la aplicación utiliza la nube de Amazon.

3. Conclusiones generales

Los dispositivos inteligentes de bajo costo como los presentados en este documento otorgan nuevas funcionalidades a equipos más viejos; permiten su control de manera remota, pero pueden ser altamente vulnerables. En una misma red es fácil tomar el control del equipo y generar un encendido, apagado, o denegación de servicios ya que en ninguno de los casos se valida quién envía la petición.

Es fundamental que los equipos validen que la petición viene de un dispositivo de confianza, apareado de manera anterior o que haya establecido un vínculo de alguna forma. Considerando que hay muchas redes Wi-Fi vulnerables, si los dispositivos en LAN no tienen autenticación o no validan las peticiones que reciben, esto aumenta la gravedad del problema. Si se tiene en cuenta, además, que el *manejo inteligente de la corriente* mediante estos equipos es extensible a objetos sensibles como alarmas, sensores o cámaras de seguridad, la ventana de posibles objetos de ataque se amplía aún más.

Referencias

1. **Internet Society.** Internet Society. [En línea] 29 de Marzo de 2019. https://www.internetsociety.org/iot.
2. **El internet de las cosas (vulnerables).** Blog de Abast. [En línea] 29 de Marzo de 2019. https://blog.avast.com/es/predicciones-para-2019-el-internet-de-las-cosas-vulnerables