Universitat Oberta de Catalunya

uoc.edu

Pruebas de Seguridad en implementaciones 802.11k/v

Javier Contreras Albesa
Plan de Estudios del Estudiante
Seguridad en la Internet de las cosas

Carlos Hernández Gañán
Victor Garcia Font

Junio 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Pruebas de Seguridad en implementaciones 802.11k/v*
Nombre del autor:	*Javier Contreras Albesa*
Nombre del consultor/a:	*Carlos Hernández Gañán*
Nombre del PRA:	*Victor Garcia Font*
Fecha de entrega (mm/aaaa):	06/2018
Titulación::	*Master Interuniversitario en Seguridad de las TIC* *(MISTIC)*
Área del Trabajo Final:	*Seguridad en la Internet de las cosas 'A medida'*
Idioma del trabajo:	*Castellano*
Palabras clave	*IOT Wireless Security*

Resumen del Trabajo (máximo 250 palabras):

Las implementaciones de redes WiFi basadas en los estándares 802.11-2012, contemplan nuevas funcionalidades para optimizar la transición de un cliente de un punto de acceso a otro (roaming). Dentro de este estándar destacan 802.11k (mecanismos de medición de radio), 802.11v (mecanismos de gestión de radio) y 802.11r (roaming seguro)

La mayor parte de los mensajes utilizados en estos protocolos, no utilizan mecanismos de seguridad disponibles como el 802.11w, por lo que son susceptibles a ataques de suplantación. Cabe además destacar que no existen actualmente herramientas comerciales especializadas en la validación de estos protocolos

Este trabajo tiene como finalidad crear bancos de prueba pruebas contra diferentes implementaciones de 802.11k/v en productos comerciales basados en los sistemas operativos Android y Apple IOS, con el fin de detectar fallos de seguridad

Para realizar las pruebas se realizaron capturas de tramas en los protocolos de interés, para luego crear diferentes bancos de validación que permitieran enviar tramas simuladas con diferentes violaciones de los protocolos, buscando un fallo en el dispositivo, por ejemplo, desconexiones o reinicios

Dentro de las pruebas realizadas, no fue posible obtener fallos consistentes en ninguna de las implementaciones sujetas a validación

Los resultados conllevan a proponer futuros bancos de prueba más sofisticados que permitan interferir de forma más controlada sobre la máquina

de estado de los diferentes dispositivos. Dada la carencia de protección de tramas en las implementaciones actuales, se presupone que existe un riesgo de seguridad que podría ser abusado, y que podría ser encontrado con las pruebas adecuadas

Abstract (in English, 250 words or less):

Different WiFi implementations based on the 802.11-2012 standard have new functionalities designed to optimize roaming of the device between access points and do a better use of radio resources. On this area, we take specially focus on 802.11k (radio resource monitoring), 802.11v (radio control) and 802.11r (secure transition)

Most of the messages used on 802.11k/v protocols are not protected by default, and current implementations do not normally use security mechanisms like 802.11w, so they are susceptible to spoofing and injections attacks around these areas. Additionally, we can make emphasis that there are no commercial test products available nowadays to validate these implementations

This research was done with the objective of creating different test banks against commercial implementations of 802.11k and 802.11v like Android and Apple IOS, looking for possible security vulnerabilities

The creation of test banks were based on collected frames used around the protocols of interest, modifying them to create a set of spoofed frames under control, simulating different protocol violations, looking to create failures like disconnections or resets

As result of the different tests, it was not possible to obtain a consistent failure point on any of the implementations under analysis

The results provide a basis to propose future test banks with more sophisticated state machine control, in order to perform a more controlled impact into the implementations under test. Given the lack of protection on most of the analyzed protocol frames, we can assume that possible security risks should exist waiting to be found with additional testing

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Las implementaciones de redes WiFi basadas en los estándares 802.11-2012 [1], contemplan nuevas funcionalidades que buscan optimizar la transición de un cliente de un punto de acceso a otro (roaming).

Estas funcionalidades fueron previamente definidas en:

- 802.11k [2]: Aprobada en el 2008, añade mecanismos de medición de radio (Radio Resource Measurements). Permite el intercambio de información entre un cliente y la red, sobre cuáles son los puntos de acceso alrededor de un cliente, y cuál es la visión del cliente sobre los puntos de acceso inalámbricos que le rodean. Esto tiene crucial importancia para la optimización de los tiempos de búsqueda de destinos al hacer un roaming

- 802.11v [3]: Aprobada en el 2011, incluye mecanismos para la gestión de la red inalámbrica (Wireless Network Management), permite que las estaciones obtengan información sobre el estado y configuración de la red, además de información de ubicación, y permitiendo técnicas de ahorro de energía al tener un mejor control sobre cuando “dormir” la radio

- 802.11r [4], aprobado en el 2008, provee nuevos mecanismos para la transición segura y rápida del estado de conexión de los dispositivos inalámbricos de un punto de acceso a otro (Fast Transition). Está enfocado en la gestión de autenticación segura al hacer un roaming

El problema a plantear en este trabajo es el análisis desde un punto de vista de seguridad de la implementación actual de los protocolos 802.11v y 802.11k en un subconjunto de dispositivos disponibles en el mercado

Existen varios puntos de interés:

1. Verificar si es posible utilizar 802.11k/v para impedir el correcto funcionamiento de una red inalámbrica
2. Cómo reaccionan los dispositivos a mensajes 802.11k/v manipulados, ya sea pruebas puntuales o mediante fuzzing
3. Cómo reaccionan los dispositivos a mensajes enviados fuera de su estado correspondiente (mensajes no solicitados)

No se abordará análisis del protocolo 802.11r, para reducir el ámbito de la investigación, y enfocarse principalmente en temas de gestión de radio y no de autenticación

Es de destacar que para este autor, no existen en el mercado actual herramientas comerciales u open source automatizadas para la validación de la implementación de los protocolos 802.11k/v en dispositivos [5] [7] [8]. Esto hace pre-suponer que es factible que existan posibles deficiencias a ser encontradas. Solo se encuentran herramientas enfocadas a pruebas básicas de 802.11 ab/g/n/ac y sobre 802.1x (autenticación)

Hay que hacer énfasis que la información provista por los protocolos 802.11k/v se envía mediante “action frames”, que no están protegidos en redes WPA2 y son fácilmente susceptibles a spoofing, independientemente de la protección utilizada en la red inalámbrica, a menos que se utilice el protocolo 802.11w (PMF) para firmar las tramas de gestión.

La utilización de 802.11w podría prevenir posibles ataques, pero su utilización es limitada, debido a su limitada disponibilidad en los dispositivos en el mercado, así como múltiples problemas de interoperabilidad reportados entre distintos fabricantes

1.2 Objetivos del Trabajo

1. Documentar las tramas y elementos de información (Information Elements, IE) 802.11k/v utilizadas dentro de una situación de funcionamiento normal, por los dispositivos bajo estudio

2. Generación de una batería de pruebas para los protocolos 802.11k/v que cubran:
    a. Spoofing de tramas válidas 802.11k/v enviadas por puntos de acceso y clientes utilizando Python/Scapy
    b. Fuzzing para tramas 802.11k/v enviadas por puntos de acceso y clientes, utilizando Python/ Scapy
    c. Pruebas de envío de 802.11k/v cuando no es esperado por el cliente o punto de acceso (por ejemplo, si el punto de acceso indica que no soporta 802.11k, y el cliente recibe una petición 802.1k de Beacon Request)
    d. Pruebas de envío de 802.11k/v con información errónea. Por ejemplo, envío de listas de APs con canales inválidos

3. Detección y documentación de situaciones que impliquen:
    a. Que el dispositivo bajo estudio interrumpa su normal funcionamiento
    b. Que se genere una modificación de los parámetros de funcionamiento que pueda implicar una disrupción de servicio o afectación de prestaciones
    c. Alteraciones de uso de memoria o de utilización de CPU que puedan afectar a otros servicios o funcionalidades prestadas por

el dispositivo

1.3 Enfoque y método seguido

Se propuso crear una batería de pruebas enfocadas al estudio de los protocolos 802.11k/v, y cuál es su impacto al aplicarse en diferentes dispositivos

La propuesta consistió en:

- Pruebas basadas en el análisis inicial de cuál es el modelo "implementado" de los protocolos en varios dispositivos comerciales que indican soporte a estas funcionalidades

- Creación de herramientas para el estudio de que tan robustos son las implementaciones actuales de 802.11k/v

- Creación de pruebas con información falsa de canales/potencia, etc, para tratar de subvertir el uso de 802.11k/v

- Aplicación de las baterías de pruebas a los dispositivos elegidos, capturando su estado y funcionamiento, en búsqueda de disrupciones o afectaciones a la prestación de servicio

Parte de la selección de equipos y recursos, viene dada por la ayuda presentada por Cisco Systems, empresa donde el investigador trabaja habitualmente, que puso a disposición diferentes equipos de red y clientes inalámbricos para la realización de este trabajo

Para la captura de las tramas en uso por los dispositivos, se configuró una red inalámbrica donde se anuncia un SSID con soporte a 802.11k/v. Se utilizaron equipos de Cisco Systems, específicamente un controlador WLC 3504, y puntos de acceso 1800i y 1702i, con el sistema operativo 8.5, que dispone de funcionalidades activas en dichos protocolos.

La captura de tramas se realizó con diferentes tarjetas inalámbricas USB que soportan modo monitor (AirPCap NX, EnGenius 1200AC), conectadas a una máquina virtual con Kali Linux

La selección inicial de equipos a probar, se basó en información disponible de los diferentes fabricantes:

- Listado genérico de clientes: http://clients.mikealbano.com
- Apple: https://support.apple.com/en-us/HT202628
- Intel: https://www.intel.com/content/www/us/en/support/articles/000021562/network-and-i-o/wireless-networking.html

Según el listado, se seleccionaron los dispositivos basados en su disponibilidad en el laboratorio de trabajo.

El proceso de captura consistió en conectar los dispositivos disponibles al SSID configurado para las pruebas, y en validar en que anunciaban soporte a 802.11k/v en las tramas de asociación a la red, así como realizar pruebas de roaming y transmisión de datos sobre la red configurada

Luego de tener una colección de capturas suficiente, se seleccionaron las tramas de interés, para realizar una lista de posibles pruebas a realizar, dependiendo del dispositivo, y funcionalidades soportadas

En función del contenido presente en cada una de las tramas analizadas, se procedió a crear un conjunto de validaciones, cada una con modificaciones específicas basadas en alteraciones del contenido (payload) de la información enviada al dispositivo bajo validación

El interés principal de las pruebas consistía en poder determinar si era posible crear alteraciones en el dispositivo, por lo que se establecieron los siguientes mecanismos de monitorización para la detección de fallos:

1. Observación de la pantalla para la detección de reinicios o mensajes de error
2. Envió continuo de pings (ICMP Echo Request), como comprobación simple del estado de la conexión inalámbrica del dispositivo
3. Captura inalámbrica simultanea de cada prueba, para la detección y análisis posterior de fallos
4. Uso del dispositivo a intervalos, para evitar que entre en modo de espera, que podría afectar la recepción de tráfico

La implementación de las pruebas se realizó en Python en conjunto con la librería Scapy [6], ya que dispone de modelos de datos que facilitan la creación de tramas a medida

La ejecución de las pruebas consistió en:

1. Conectar el dispositivo a ser validado a la red inalámbrica
2. Establecer un flujo continuo de ping contra su dirección IP
3. Iniciar una captura inalámbrica en el canal usado por el punto de acceso
4. Iniciar una prueba, enviando las tramas modificadas desde una máquina virtual Linux, utilizando el script Python correspondiente al modelo de alteraciones definido previamente
5. Utilizar (tocar) la pantalla del dispositivo a intervalos regulares
6. Monitorizar el flujo de tráfico y la pantalla del dispositivo para detectar cualquier interrupción
7. En caso de interrupción del tráfico, guardar el estado de la prueba, la captura, y volver a ejecutarla, para descartar falsos positivos

La metodología y desarrollo de las pruebas, fue basado en experiencias previas del investigador, para la obtención de vulnerabilidades de seguridad sobre equipos inalámbricos Cisco. Por ejemplo:
CVE-2016-6375:
https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20160831-wlc-1

CVE-2016-6376:
https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20160831-wlc-2

1.4 Planificación del Trabajo

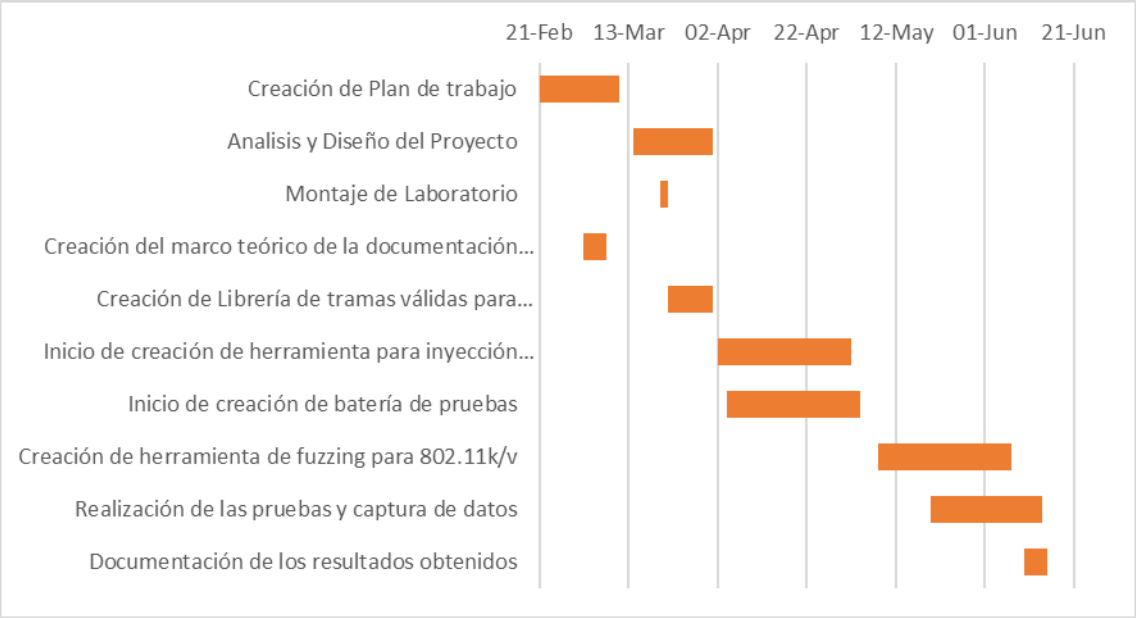


Figura 1. Planificación - Diagrama Gantt

Para la realización de las investigaciones, se utilizarán dispositivos presentes en el laboratorio personal y que servirán para acotar que elementos serán sometidos a las baterías de pruebas

Basados en la información del fabricante, se utilizarán:

- Portátil Lenovo con Intel 8260, Windows 10
- Apple iPad Air
- Samsung S7
- Samsung S8
- Iphone 7

Como infraestructura, se utilizarán:

- Controladora Cisco 3504, con AireOS 8.5
- Puntos de acceso 3700, 3800, 1800

Para la creación de las herramientas de pruebas, se utilizó:

- Python + Scapy: para construir una herramienta que permita inyectar tramas válidas de 802.11v/11k, bajo el entorno PyCharm de desarrollo
- Tarjetas USB con soporte a Radiotap Header: EnGenius 1200AC, AirPcap NX

Esta selección viene dada por experiencia previa personal en la realización del modelaje de otros protocolos inalámbricos

1.5 Breve sumario de productos obtenidos

En función de las tramas capturadas, se crearon 14 pruebas específicas con diferentes alteraciones, las cuales se ejecutaron de forma secuencial sobre cada uno de los dispositivos pre-seleccionados en el laboratorio. Todas las pruebas fuero implementadas como una aplicación Python “11kv.py”

Pruebas	Descripcíon
all11	Fuerza Bruta sobre todos las categorías posibles
neighreport	Neighbor Report básico
neighreport_randomSSID	Neighbor Report. IE de SSID aleatorio
neighreport_largeSSID	Neighbor Report. IE de SSID con longitud máxima
neighreport_invalidSSID	Neighbor Report. IE de SSID inválido
neighreport_largeSSIDnull	Neighbor Report. IE de SSID nulo y longitud máxima
neighreport_nullSSID	Neighbor Report. IE de SSID nulo
neigh_response	Respuesta de Neighbor list
measurementreport	Reporte de Medición de radio
measurementreport_nomeasurement	Reporte de Medición de radio con contenido incompleto
measurementreport_nobeacon	Reporte de Medición de radio con alteración al campo de beacon
measurementreport_noapple	Reporte de Medición de radio con campos de fabricante incompleto
measurementreport_randommeasurement	Reporte de Medición de radio con contenido aleatorio
Measurementreport_randomapple	Reporte de Medición de radio con campo de contenido de fabricante aleatorio
bsstrans_req	Petición de transición (roaming)
bsstrans_random	Petición de transición con contenido aleatorio

Figura 2. Resumen de Pruebas

Adicionalmente, se creó una prueba final tipo “fuerza bruta” (all11), donde se hace un recorrido sobre todos las categorías de action frames disponibles en el estándar 802.11-2012, con contenido totalmente aleatorio, enviandose unas 100 tramas diferentes por cada uno de ellos

Las pruebas fueron implementadas en un script Python, que acepta diferentes opciones en su línea de comando para ofrecer variaciones posibles sobre cada una de las pruebas a realizar:

Comando	Descripcíon
--interface	Nombre del dispositivo a usar para transmitir
--source	Dirección origen de la trama
--destination	Dirección destino de la trama
--bssid	Dirección a usar para el BSSID
--count	Cuantas tramas distintas a envíar
--delay	Retardo entre cada evento de transmisión
--test	Nombre de la prueba a ejecutar

Figura 3. Opciones de Linea de Comandos

Las pruebas realizadas también tenían variaciones de las direcciónes de transmisión, con el fin de validar si esto tenía algún efecto en el procesamiento de las mismas:

- Dirección cliente a infraestructura de red (lo especificado en el estándar)
- De infraestructura de red a cliente , BSSID asociado a dirección del cliente (fuera del estándar)
- De dirección aleatoria al cliente
- De dirección aleatoria al AP, BSSID válido

En el contexto de las pruebas realizadas, no fue posible detectar fallos consistentes en ninguno de los dispositivos analizados, quedando pendiente realizar modificaciones adicionales en función del conocimiento adquirido durante la generación de este trabajo, que no fue posible implementar por limitaciones de tiempo

La metodología empleada ya había sido utilizada por el investigador en tareas similares anteriores, con buenos resultados (por ejemplo CVE-2015-6258, CVE-2016-6375, CVE-2016-1460, CVE-2016-9194), por lo que se hará enfasis en modificaciones posteriores a los bancos de prueba, para hacer otras variaciones que puedan dar mejores resultados

Fue posible detectar un fallo en la conexión de red en los dispositivos Android al recibir tramas con action frames inválidos, el cliente no se vuelve a conectar hasta que se haga un reinicio de su tarjeta de red. Sin embargo, dado que el fallo no fue reproducible consistentemente, no se incluye como resultádo válido de las pruebas, quedando pendiente su investigación posterior

1.6 Breve descripción de los otros capítulos de la memoria

2.1 **Investigaciones Previas:** Contiene un breve listado de diferentes vulnerabilidades y trabajos de investigación destacados, hasta la fecha, con principal enfoque en la seguridad de las redes inalámbricas y sus protocolos.

2.2 **Laboratorio**: Descripción detallada del laboratorio utilizado para las pruebas

2.3 **Actividad 802.11k/v de los dispositivos**: Tramas capturadas sobre cada dispositivo, utilizadas como base para la creación de las pruebas posteriores

2.4 **Batería de Pruebas**: Descripción de las pruebas realizadas

A continuación se listan los capítulos con descripciones detalladas sobre cada una de las tramas estudiadas, y las pruebas realizadas
2.5. Pruebas sobre 802.11k Neighbor Report Request
2.6. Pruebas sobre 802.11k Neighbor Report Response
2.7 Pruebas sobre 802.11k Measurement Report
2.8 Pruebas sobre 802.11v BSS Transition Request
2.9 Prueba de Fuerza Bruta sobre Action Frames

En los anexos, se incluye el código fuente de la herramienta, así como los pasos previos de configuración de la tarjeta de red para poder utilizarla

2. Resto de capítulos

2.1 Investigaciones Previas

Las redes WiFi han sido sujetas a largos análisis exhaustivos de seguridad, principalmente enfocándose en las áreas de encriptación, y autenticación.
Podemos encontrar extensa bibliografía y trabajos previos, aquí solo citaremos algunos más recientes y que tuvieron mucha visibilidad pública:

Key Reinstallation Attacks:
https://www.krackattacks.com/
Enfocado en la reutilización de material criptográfico cuando se envían mensajes fuera del estado donde se esperan. Especialmente famoso por sus implicaciones en el protocolo WPA2

TKIP Cryptographic analisis:
http://download.aircrack-ng.org/wiki-files/doc/tkip_master.pdf
Se exploran problemas criptográficos con la implementación de del protocolo TKIP, utilizado en 802.11i (comúnmente conocido como WPA-TKIP). Explotado en : http://www.aircrack-ng.org/doku.php?id=tkiptun-ng

Hole 196:
https://www.mojonetworks.com/wpa2-hole196-vulnerability
Posibles ataques al protocolo WPA2, debido a la reutilización de la clave de encriptación de tramas broadcast/multicast (GTK), que permitiría la inyección de tráfico y redirección del mismo en situaciones específicas

Practical attacks against WEP and WPA:
https://dl.aircrack-ng.org/breakingwepandwpa.pdf
Describe técnicas aún mejores para optimizar el descubrimiento de claves de encriptación en WEP, y ataques para desencriptrar tráfico en una red TKIP con PSK

Wireless Fuzz Testing
https://www.blackhat.com/presentations/bh-europe-07/Butti/Presentation/bh-eu-07-Butti.pdf
Ya en 2007, se mostraron en la conferencia de Black Hat, varios ejemplos de pruebas de seguridad realizadas por el equipo de R&D de Orange, utilizando conexiones Wireless con ataques de Fuzzing sin embargo, no ha habido nuevos progresos recientes en está área

Por ahora, no se han encontrados trabajos de investigación sobre la seguridad de los protocolos 802.11k/v

2.2 Laboratorio

Para la realización del proyecto, se configuró un laboratorio con las siguientes características:

Infraestructura de red:

- Puntos de acceso Cisco 1800I
- Punto de acceso Cisco 1702i
- Controlador WLC 3504, utilizando la versión 8.5.120.0, posteriormente migrado a 8.5.130.0

Clientes:

- iPhone 7, con IOS 11.3
- Samsung S7, Android 7
- Samsung S8, Android 7
- Ipad Pro, con IOS 11.3

Desarrollo

- Máquina virtual Kali Linux 2018-1, Pycharm 2018-1
- Python 3.6
- Scapy-python3 0.23
- Lenovo X1, tarjeta Intel 8260, Windows 10
- Tarjeta wireless USB con chipset AR9001U, soporta cabeceras radiotap, e inyección de tramas

La topología física se pensó para que existiera una separación real RF entre al menos dos de los puntos de acceso, y poder realizar pruebas reales de roaming

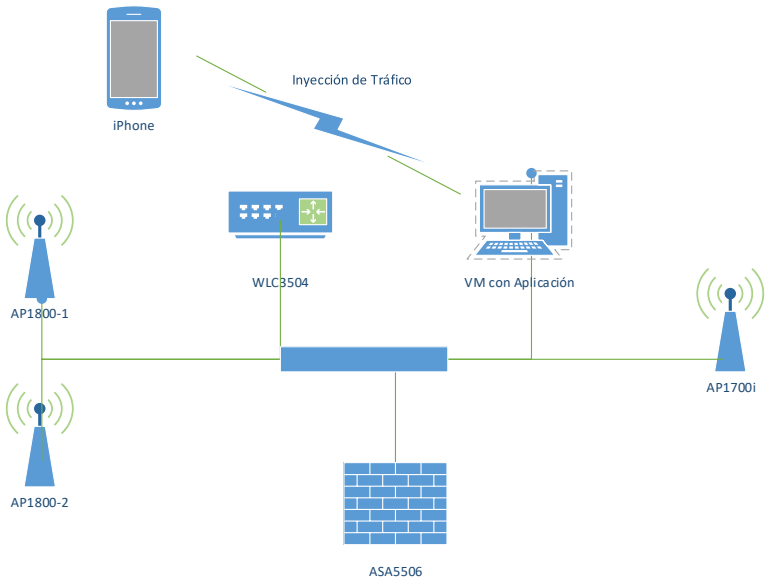


Figura 4. Topología del Laboratorio

Figura 5. Vista de los Equipos

Configuración de SSID con 802.11k (neighbor reports) y 802.11v (BSS Transition, DMS)

Figura 6. Ejemplo de configuración SSID

2.3 Actividad 802.11k/v de los dispositivos

Resultados – iPhone 7

El dispositivo reporta soporte a 802.11k/11v en la solicitud de asociación y existe actividad observable relacionada.

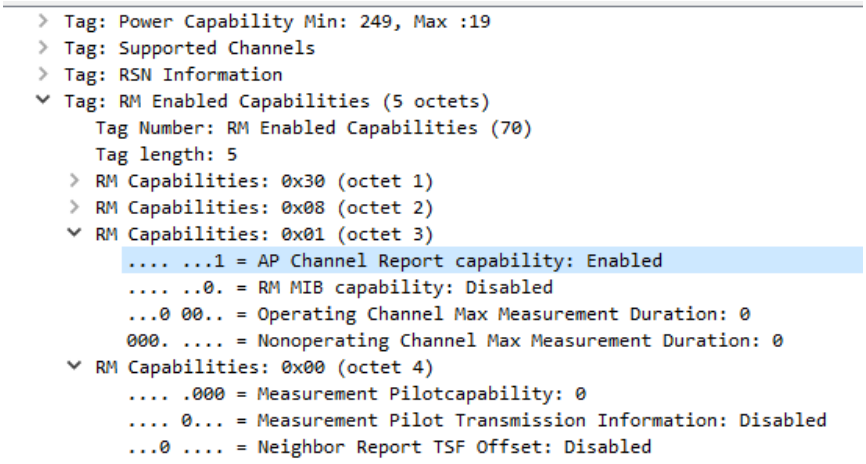


Figura 7. iPhone7 :Soporte a 802.11k, parcial

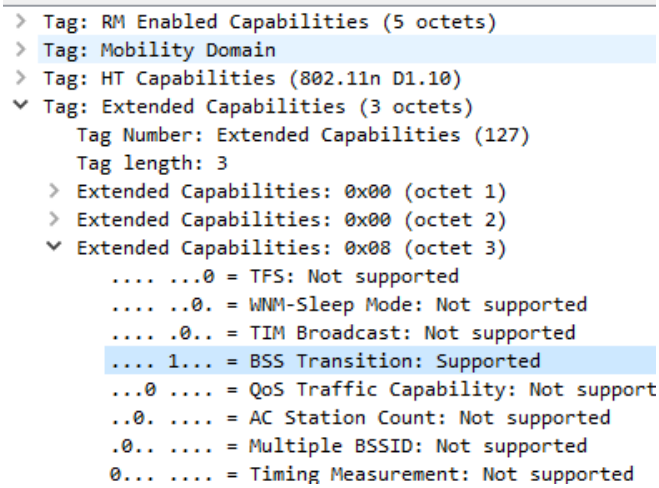


Figura 8. iPhone7: Soporte a 802.11v, parcial

Solicitud de reporte de vecinos (802.11k) realizada por el iPhone:

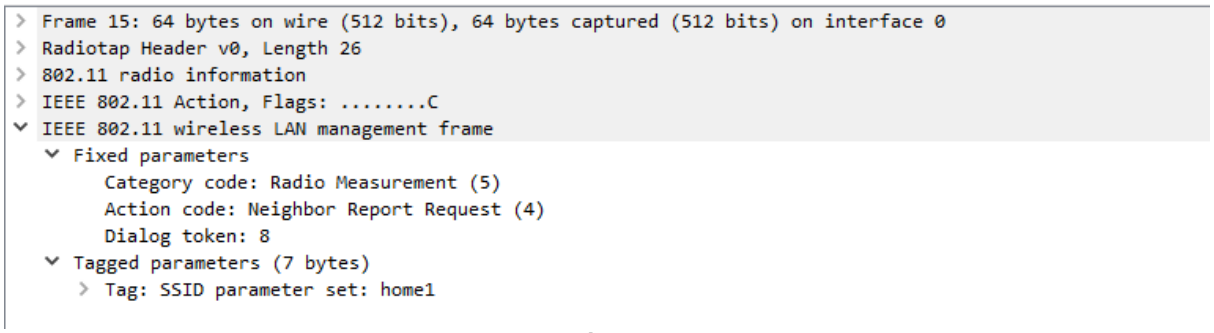


Figura 9. Petición de Vecinos

Respuesta del Punto de Acceso:

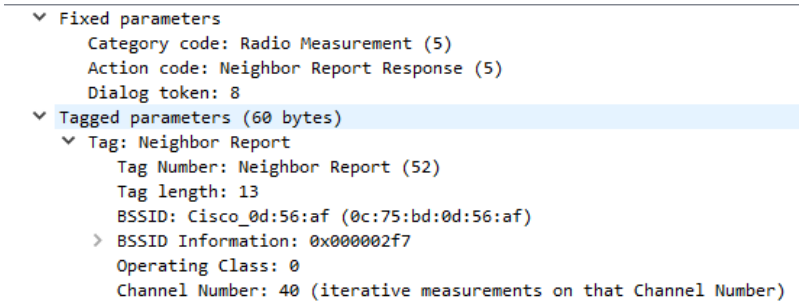


Figura 10. Respuesta a Petición de Vecinos

Resultados – Samsung S7

El dispositivo reporta soporte a 802.11k/11v en la solicitud de asociación, sin embargo en las pruebas iniciales no se observa actividad relacionada, se está pendiente de probar a generar peticiones desde el AP simuladas en la siguiente fase, para confirmar:

Indica soporte a Neighbor Reports, mediciones de Beacons y de Aps por canal:

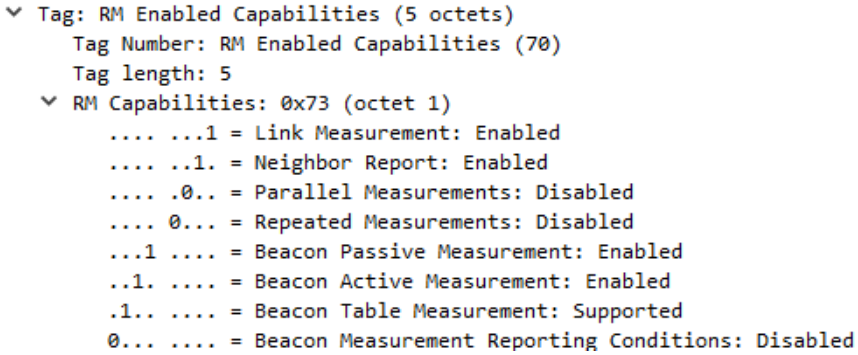
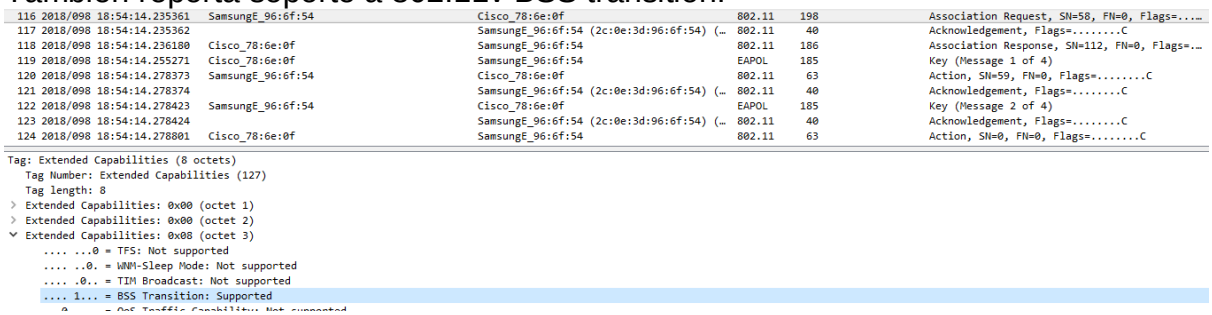


Figura 11. Samsung: 11k

```
   > RM Capabilities: 0x73 (octet 1)
   > RM Capabilities: 0x08 (octet 2)
   v RM Capabilities: 0x01 (octet 3)
        .... ...1 = AP Channel Report capability: Enabled
        .... ..0. = RM MIB capability: Disabled
        ...0 00.. = Operating Channel Max Measurement Duration: 0
        000. .... = Nonoperating Channel Max Measurement Duration: 0
   > RM Capabilities: 0x00 (octet 4)
   > RM Capabilities: 0x00 (octet 5)
> Tag: HT Capabilities (802.11n D1.10)
> Tag: Extended Capabilities (8 octets)
> Tag: VHT Capabilities (IEEE Std 802.11ac/D3.1)
```

Figura 12. Samsung: 11v

También reporta soporte a 802.11v BSS transition:



```
116 2018/098 18:54:14.235361  SamsungE_96:6f:54   Cisco_78:6e:0f                  802.11   198  Association Request, SN=58, FN=0, Flags=....
117 2018/098 18:54:14.235362                      SamsungE_96:6f:54 (2c:0e:3d:96:6f:54) (…  802.11    40  Acknowledgement, Flags=........C
118 2018/098 18:54:14.236180  Cisco_78:6e:0f      SamsungE_96:6f:54               802.11   186  Association Response, SN=112, FN=0, Flags=....
119 2018/098 18:54:14.255271  Cisco_78:6e:0f      SamsungE_96:6f:54               EAPOL    185  Key (Message 1 of 4)
120 2018/098 18:54:14.278373  SamsungE_96:6f:54   Cisco_78:6e:0f                  802.11    63  Action, SN=59, FN=0, Flags=........C
121 2018/098 18:54:14.278374                      SamsungE_96:6f:54 (2c:0e:3d:96:6f:54) (…  802.11    40  Acknowledgement, Flags=........C
122 2018/098 18:54:14.278423  SamsungE_96:6f:54   Cisco_78:6e:0f                  EAPOL    185  Key (Message 2 of 4)
123 2018/098 18:54:14.278424                      SamsungE_96:6f:54 (2c:0e:3d:96:6f:54) (…  802.11    40  Acknowledgement, Flags=........C
124 2018/098 18:54:14.278801  Cisco_78:6e:0f      SamsungE_96:6f:54               802.11    63  Action, SN=0, FN=0, Flags=........C

Tag: Extended Capabilities (8 octets)
   Tag Number: Extended Capabilities (127)
   Tag length: 8
 > Extended Capabilities: 0x00 (octet 1)
 > Extended Capabilities: 0x00 (octet 2)
 v Extended Capabilities: 0x08 (octet 3)
      .... ...0 = TFS: Not supported
      .... ..0. = WNM-Sleep Mode: Not supported
      .... .0.. = TIM Broadcast: Not supported
      .... 1... = BSS Transition: Supported
```

2.4 Batería de Pruebas

Las pruebas a realizar se enfocarán principalmente sobre la actividad en 802.11k/11v de los clientes observados, creando variaciones específicas de cada trama, además de crear un set de pruebas “genérico” cubriendo todas las combinaciones de tipos de trama posibles en ambos protocolos, según lo documentado en el estándar 802.11-2012, con un payload aleatorio (prueba de fuerza bruta)

La idea es crear conjunto de tramas lo suficientemente válidas para ser aceptadas y procesadas por el equipo bajo estudio, pero con variaciones significativas fuera de lo documentado en el estándar. Esto forma parte de un proceso típico de “pruebas negativas” (negative testing)

Cada prueba recibirá un nombre, según la opción necesaria en el CLI para ejecutar la herramienta de pruebas

Detección de Fallos

Cada equipo bajo prueba (DUT por sus siglas en inglés), será monitorizado mediante:

- Ping continuo a su dirección IP
- Salida de consola al dispositivo, si aplica
- Inspección visual de la pantalla, si aplica
- Uso de la pantalla

2.5. Pruebas sobre 802.11k Neighbor Report Request

Descripción de la trama:

- Dirección usual: Cliente al AP
- Tipo: Radio Measurement (5), Action code: Neighbor Request (4)
- Dialog Token: Valor que incrementa a lo largo de la conexión, suele ser 0 o 1
- Payload: Information Element con SSID, de 0 a 32 bytes
- Opcionales: Ninguno

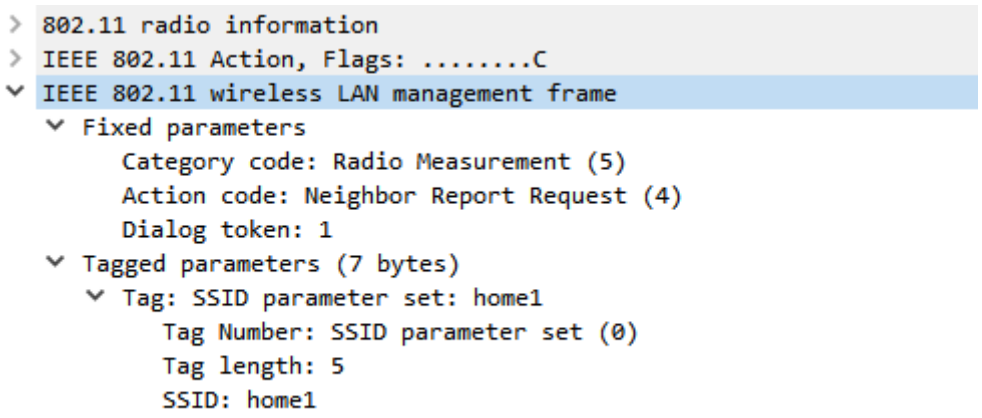


Figura 13. Ejemplo de Petición de Vecinos

La trama se utiliza para solicitar que un dispositivo informe sobre los posibles vecinos que ofrecen servicios bajo un SSID dado

Pruebas Realizadas

Al ser una trama simple, son sólo un elemento incluido (SSID), ofrece pocas variaciones para probar posibles vulnerabilidades.

Dirección de las pruebas

Se harán variaciones de cada prueba, según la dirección asociada a la trama:

- Dirección cliente a infraestructura de red (lo especificado en el estándar)
- De infraestructura de red a cliente , BSSID asociado a dirección del cliente (fuera del estándar)
- De dirección aleatoria al cliente
- De dirección aleatoria al AP, BSSID válido

Variaciones propuestas:

- Envío a dirección broadcast, buscando ver si todos los APs en el canal responden (amplificación de tráfico)
- Anexar IE SSID con nombre inválido, diferente al que el cliente está asociado. En herramienta: neighreport_invalidSSID
- Anexar IE SSID, con longitud cero. En herramienta: neighreport_nullSSID
- Anexar IE con longitud inválida (máximo es 32 bytes según estándar), incluir 255 caracteres. neighreport_largeSSID
- Anexar IE con longitud inválida, no incluir ningún carácter. neighreport_largeSSIDnull

- Anexar IE con longitud aleatoria, incluir un número aleatorio de caracteres. neighreport_randomSSID

Implementación de petición básica:
Copia de trama típica enviada por cliente

```
def payload_neighreport():
       #Valid neigh report, hard coded SSID as per lab
       neigh_report= b"\x05"  # Radio Measurament
       neigh_report += b"\x04"  # Neighbor Report Request
       neigh_report += b"\x01"  # Dialog Token
       neigh_report += b"\x00" # SSID IE
       neigh_report += b"\x04"  # length
       neigh_report += "3504".encode()  # SSID sting
       return neigh_report
```

Implementación de petición con SSID inválido:
Modificación incluyendo un nombre de SSID diferente al configurado

```
def payload_neighreport_invalidSSID():
    # SSID null length

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token

    neigh_report += b"\x00"  # SSID IE
    neigh_report += b"\x04"  # length
    neigh_report += "none".encode()  # SSID sting

    return neigh_report
```

Implementación de petición tamaño máximo de SSID:
Envío de nombre de SSID, excediendo el tamaño máximo del estándar (32 bytes)

```
def payload_neighreport_largeSSID():
    # large SSID length, random payload

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token

    neigh_report += b"\x00"  # SSID IE
    neigh_report += b"\xFF"  # length

    for entry in range(0, 255):  # SSID content
        neigh_report += random.randint(0, 255).to_bytes(1, "little")
    return neigh_report
```

Implementación de petición con SSID aleatorio:
Modificación que incluye un IE de SSID con contenido aleatorio

```
def payload_neighreport_randomSSID():
    # random SSID length, 0 to

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token
    neigh_report += b"\x00"  # SSID IE

    neigh_report += random.randint(1, 255).to_bytes(1, "little") #random size

    for entry in range(0, random.randint(0, 255)):  # SSID content random
        neigh_report += random.randint(0, 255).to_bytes(1, "little")

    return neigh_report
```

Implementación de petición con SSID nulo, tamaño máximo:
Envió de IE de SSID, pero truncando la trama (información no incluída)

```
def payload_neighreport_largeSSIDnull():
    # large SSID, not present

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token
    neigh_report += b"\xFF"  # SSID IE


    return neigh_report
```

2.6. Pruebas sobre 802.11k Neighbor Report Response

Descripción de la trama:

- Dirección usual: AP al Cliente
- Tipo: Radio Measurement (5), Action code: Neighbor Response (5)
- Dialog Token: Valor en función de la petición recibida
- Payload: Serie de Information Elements con lista de vecinos, incluyendo su BSSID, canal, tipo, etc

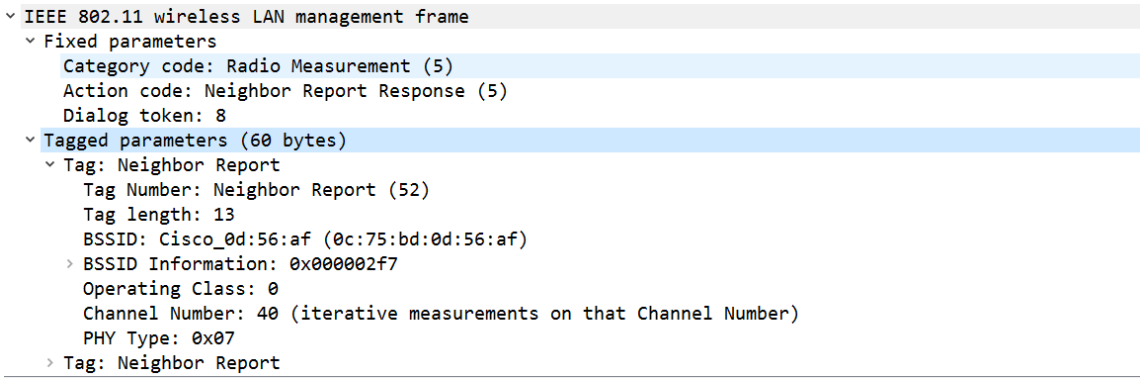


Figura 14. Ejemplo de Respuesta de Vecinos

La trama contiene la respuesta a una petición de vecinos, y puede incluir información sobre cero o más puntos de acceso

Pruebas Realizadas

Existen varias alteraciones posibles a ejecutar, modificando el contenido de la trama, la cantidad de vecinos disponibles, sus valores de canal y características físicas.

Dirección de las pruebas

Se harán sólo pruebas de tramas de AP hacia los clientes bajo validación:

- De infraestructura de red a cliente , BSSID asociado a dirección del cliente (fuera del estándar)
- De dirección aleatoria al cliente

Variaciones propuestas:

- Inclusión de número aleatorio (0 a 40) de repeticiones del mismo vecino
- Envió de respuesta, con contenido aleatorio para los IE de cada vecino (canal, info, phy)

Envío de respuesta básico

Copia de trama típica enviada por AP

```
    def payload_neighponse(kvtest):

        neigh_report= b"\x05"  # Radio Measurament
        neigh_report += b"\x05"  # Neighbor Report Response
        neigh_report   +=   random.randint(0,   2).to_bytes(1,   "little")
    #dialog token

        neigh_report += b"\x34"  # Neigh Report
```

```
        neigh_report += b"\x0d"  # length
        neigh_report += kvtest.bssidbinary  # target AP
        neigh_report += b"\xf7\x02\x00\x00"  # BSSID Information
        neigh_report += b"\x00"  # Operating class
        neigh_report += b"\x40"  #channel num
        neigh_report += b"\x07"  #phy type

        return neigh_report
```

Envío de respuesta, con número aleatorio de vecinos, mismo canal

Trama modificada para repetir de forma aleatoria, el mismo IE describiendo un vecino válido

```
def payload_neighponse(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x05"  # Neighbor Report Response
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    for count in range(0,random.randint(0, 40)):
        neigh_report += b"\x34"  # Neigh Report
        neigh_report += b"\x0d"  # length
        neigh_report += kvtest.bssidbinary  # target AP
        neigh_report += b"\xf7\x02\x00\x00"  # BSSID Information
        neigh_report += b"\x00"  # Operating class
        neigh_report += b"\x40"  #channel num
        neigh_report += b"\x07"  #phy type

    return neigh_report
```

Envío de respuesta, con contenido aleatorio en los IE

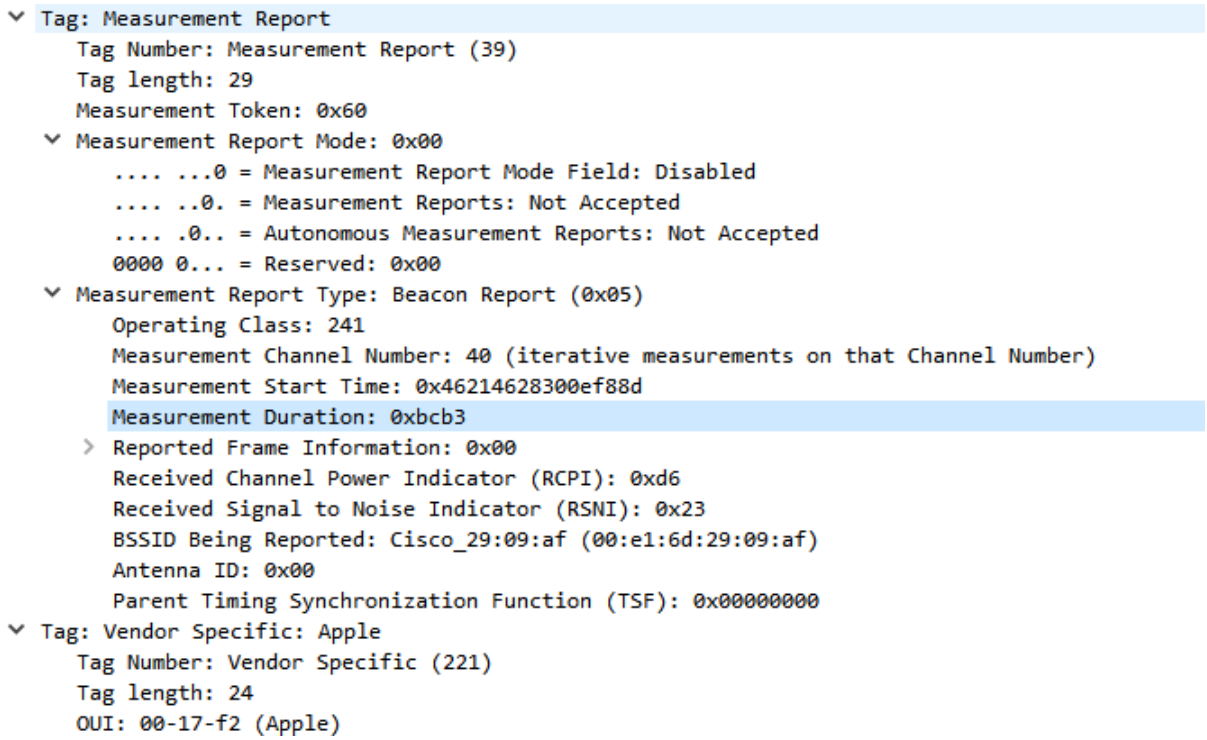
Modificación que envía un número aleatorio de IE describiendo un vecino, que a su vez es definido con características aleatorias

```
def payload_neighponse_randomIEcontent():

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x05"  # Neighbor Report Response
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    for count in range(0,random.randint(0, 40)):
        neigh_report += b"\x34"  # Neigh Report
        neigh_report += b"\x0d"  # length
        neigh_report += random_mac() # BSSID
        neigh_report += random_mac()  # BSSID info random
        neigh_report += random.randint(0, 255).to_bytes(1, "little")
             # Operating class
        neigh_report += random.randint(0, 255).to_bytes(1, "little")
             #channel num
        neigh_report += random.randint(0, 255).to_bytes(1, "little")
             #phy type

    return neigh_report
```

2.7 Pruebas sobre 802.11k Measurement Report

Descripción de la trama:

- Dirección: Cliente al AP
- Tipo: Radio Measurement (5), Action code: Radio Measurement Report (1)
- Dialog Token: Normalmente 0
- Payload: Varios elementos en formato TLV (Type, Length Value), entre ellos:
    - Measurement Report (39), 29 bytes de longitud
    - Beacon Report (5)
    - Vendor Specific, Apple (221)



La trama se utiliza para solicitar que un dispositivo inalámbrico realice una o más mediciones en uno o más canales

Pruebas Realizadas

Esta trama ofrece un contenido más complejo, con múltiples puntos donde introducir variaciones susceptibles de causar vulnerabilidades

Variaciones propuestas:

- No inclusión de elemento de Measurement Report. En herramienta: measurementreport_nomeasurement
- No inclusión de elemento de Beacon Report. En herramienta: measurementreportnobeacon
- No inclusión de elemento Vendor Specific (Apple): En herramienta: measurementreportnoapple
- Valor aleatorio de Measurement Token. En herramienta: measurementreportrandommeasurement
- Valor aleatorio para Measurement Report Mode

- Contenido aleatorio para Beacon report
- Inclusión de elemento de Measurement Report, con longitud 0
- Inclusión de elemento de Measurement Report, con longitud 255, con contenido aleatorio
- Inclusión de elemento de Measurement Report, con longitud 255, sin contenido
- Inclusión de elemento de Vendor Specific, tipo Apple, para vendor specific data, con contenido aleatorio. En herramienta: measurementreportrandommeasurement
- Inclusión de elemento de Vendor Specific, tipo Apple, longitud 0
- Inclusión de elemento de Vendor Specific, con longitud 255, con contenido aleatorio
- Inclusión de elemento de Vendor Specific, con longitud 255, sin contenido

Envío básico de Radio Measurement Report

Copia de trama típica enviada por cliente

```
def payload_measurementreport(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    neigh_report += b"\x27"  # Measurement Report
    neigh_report += b"\x1d"  # length
    neigh_report += b"\x60"  # token
    neigh_report += b"\x00"  # report mode
    neigh_report += b"\x05"  # beacon report
    neigh_report += b"\xf1"  # class
    neigh_report += b"\x28"  # channel
    neigh_report += b"\x46\x21\x46\x28\x30\x0e\xf8\x8d"  # time
    neigh_report += b"\xb3\xbc"  # duration
    neigh_report += b"\x00"  # frame information
    neigh_report += b"\xd6"  # power indication
    neigh_report += b"\x23"  # noiseindication
    neigh_report += kvtest.bssidbinary# BSSID reported
    neigh_report += b"\x00"  # antenna ID
    neigh_report += b"\x00\x00\x00\x00"  # TSF
    neigh_report += b"\xdd"  # Vendor Specific tag
    neigh_report += b"\x18"  #  length
    neigh_report += b"\x00\x17\xf2"  # apple
    neigh_report += b"\x0a"  # type
    neigh_report                                                    +=
b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64\x37\x2
c\x33"  # data
```

No inclusión de elemento de Measurement Report.

Modificación que omite IE obligatorios

```
def payload_measurementreport_nomeasurement(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token

    neigh_report += b"\xdd"  # Vendor Specific tag
    neigh_report += b"\x18"  #  length
    neigh_report += b"\x00\x17\xf2"  # apple
```

```
    neigh_report += b"\x0a"  # type
    neigh_report +=
b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64\x37\x2
c\x33"  # data

    return neigh_report
```

No inclusión de elemento de Beacon Report.
Modificación que omite IE obligatorios

```
    def payload_measurementreportnobeacon(kvtest):

        neigh_report= b"\x05"  # Radio Measurament
        neigh_report += b"\x01"  # Radio Measurement Report
        neigh_report   +=   random.randint(0,   2).to_bytes(1,   "little")
    #dialog token
        neigh_report += b"\x27"  # Measurement Report
        neigh_report += b"\x05"  # length
        neigh_report += b"\x60"  # token
        neigh_report += b"\x00"  # report mode
        neigh_report += b"\x00"  # frame information
        neigh_report += b"\xd6"  # power indication
        neigh_report += b"\x23"  # noiseindication
        neigh_report += b"\xdd"  # Vendor Specific tag
        neigh_report += b"\x18"  #  length
        neigh_report += b"\x00\x17\xf2"  # apple
        neigh_report += b"\x0a"  # type
        neigh_report                                                    +=
    b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64
    \x37\x2c\x33"  # data

    return neigh_report
```

No inclusión de elemento Vendor Specific
Modificación que elimina la información específica de fabricante

```
    def payload_measurementreportnoapple(kvtest):
        neigh_report = b"\x05"  # Radio Measurament
        neigh_report += b"\x01"  # Radio Measurement Report
        neigh_report  +=  random.randint(0,  2).to_bytes(1,  "little")    #
    dialog token
        neigh_report += b"\x27"  # Measurement Report
        neigh_report += b"\x1d"  # length
        neigh_report += b"\x60"  # token
        neigh_report += b"\x00"  # report mode
        neigh_report += b"\x05"  # beacon report
        neigh_report += b"\xf1"  # class
        neigh_report += b"\x28"  # channel
        neigh_report += b"\x46\x21\x46\x28\x30\x0e\xf8\x8d"  # time
        neigh_report += b"\xb3\xbc"  # duration
        neigh_report += b"\x00"  # frame information
        neigh_report += b"\xd6"  # power indication
        neigh_report += b"\x23"  # noiseindication
        neigh_report += kvtest.bssidbinary  # BSSID reported
        neigh_report += b"\x00"  # antenna ID
        neigh_report += b"\x00\x00\x00\x00"  # TSF
```

```
    return neigh_report
```

Valor aleatorio de Measurement Token.

Generación de tramas con contenido parcialmente aleatorio

```
def payload_measurementreportrandommeasurement(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report   +=   random.randint(0,   2).to_bytes(1,   "little")
#dialog token
    neigh_report += b"\x27"  # Measurement Report
    length=random.randint(0, 255).to_bytes(1, "little")
    lengthint=int.from_bytes(length, byteorder='big')
    neigh_report += length  # length
    for count in range(0,lengthint):
        neigh_report  +=  random.randint(0,  255).to_bytes(1,  "little")
#payload

    neigh_report += b"\xdd"  # Vendor Specific tag
    neigh_report += b"\x18"  #  length
    neigh_report += b"\x00\x17\xf2"  # apple
    neigh_report += b"\x0a"  # type
    neigh_report                                                    +=
b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64
\x37\x2c\x33"  # data

    return   neigh_report
```

2.8 Pruebas sobre 802.11v BSS Transition Request

Descripción de la trama:

- Dirección: AP al cliente
- Tipo: WMN (10), Action code: BSS Transition Management Request (7)
- Dialog Token: Normalmente 1
- Payload: Varios elementos fijos entre ellos:
    - Campo binario de 1 byte, indicando el estado de varias banderas
    - Tiempo de Desconexión, 2 bytes
    - Intervalo de Validez, 1 byte
    - URL con información de sesión: tamaño variable y opcional

Figura 15. Ejemplo de 802.11v BSS Transition Request

La trama se utiliza para solicitar que un dispositivo realice un cambo de punto de acceso (roaming)

Pruebas Realizadas

Una valoración inicial hacía suponer que este sería una trama que ofrecería la mayor posibilidad de ataques. Sin embargo no fue posible construir una trama que fuera aceptada correctamente por los clientes bajo pruebe

Variaciones propuestas:

- Contenido aleatorio

Envío básico de BSS Transition Request.

Generación básica de tramas

```
def payload_bsstransreq():

    neigh_report= b"\x0A"  # WNM
    neigh_report += b"\x07"  # BSS Transition Management Request
    neigh_report += b"\x01"    #dialog token
    neigh_report += b"\x12"  # request mode, preferred, abridged, no
inminent, no bss termination included, ess dis inminent (url)
    neigh_report += b"\x00\x00"  # dissac timer
    neigh_report += b"\x10"  # validity interval
    neigh_report += b"\x11"  # url lenfth
    neigh_report += "http://google.com".encode()  # url
```

```
        return neigh_report
```

Envío de contenido aleatorio dentro de BSS Transition Request.

Modificación que genera contenido aleatorio

```
    def payload_bsstransreqrandom():
        neigh_report = b"\x0A"  # WNM
        neigh_report += b"\x07"  #  BSS Transition Management Request
        neigh_report += b"\x01"  # dialog token

        for count in range(0, random.randint(0, 100)):
            neigh_report += random_mac()  # BSSID


        return neigh_report
```

2.9 Prueba de Fuerza Bruta sobre Action Frames

Como prueba adicional, se incluyó un escaneo secuencial sobre todas las categorías válidas de action frames definidos en [1], utilizando un rango de enteros típico para cada action code.
Para cada combinación de categoría/action code, se envían 100 tramas con un contenido aleatorio entre 0 y 1400 bytes

La idea es detectar si hay alguna combinación no previamente detectada, que podría causar un error en el dispositivo bajo análisis.

Es una prueba con baja probabilidad de éxito, pero con bajo coste de desarrollo, por lo que se decidió incluir en las baterías de validación realizadas contra cada dispositivo

```
    if options.test == "all11k":
        globalSEQ=0 #carry over the used SEQ to all the tests

        #goes over all frame types inside 11-actions, sending count
        # frames with random payload on each one
        log(STATUS, "Running full 11-action random test")
        # subtype 32=data, 40=qosdata
        for category in range(0,15):
            if category !=3: #skip BAK
                radio_measurament_action_values = {b"\x00", b"\x01", b"\x02",
b"\x03", b"\x04", b"\x05", b"\x06", b"\xff"}


                for action_field in radio_measurament_action_values:
                    for x in range(0, options.count):
                        payload = category.to_bytes(1, "little")
                        # all categories
                        payload += action_field  # Neighbor Report Response
                        payload += random.randint(0, 2).to_bytes(1, "little")
                        # dialog token
                        payload += random_char(random.randint(0, 1400))

                        packet = RadioTap(present=0) /
                        Dot11(addr1=receiver_address, addr2=transmit_address,
                        addr3=bssid, type=00, subtype=0x0d, FCfield=0,
                        SC=calculateSC(globalSEQ))
                        packet /= payload
                        sendp(packet, iface=options.interface,
                          count=options.burst)

                        time.sleep(options.delay)
                        packet.show()
                        print("Category Code: " + str(category))
                        print("Action Field: " + str(action_field))

                        globalSEQ += 1
```

3. Conclusiones

La propuesta de este trabajo se realizó basada en experiencias previas personales del investigador al realizar pruebas de otros protocolos usados en redes WiFi, dentro de un contexto laboral. Como tal, la definición de los tipos de tests, la ejecución y la recolección de resultados, fue basado en estas actividades anteriores. Los procedimientos que dieron resultados exitosos sobre vulnerabilidades en otros dispositivos, no proveyeron los mismos resultados al enfocarse sobre otros protocolos (802.11k/v), y ejecutarlos sobre dispositivos de terceros

Durante este trabajo, se crearon una serie de baterías de pruebas básicas para confirmar que tan robustos son las implementaciones de 802.11k/v de varios dispositivos comerciales. No fue posible detectar ninguna vulnerabilidad consistentemente, por lo que no se obtuvieron resultados importantes a documentar

A pesar de que esto podría verse como un resultado negativo, existen varios puntos que permiten hacer una lectura parcialmente positiva del trabajo realizado:

1. Se pudo ahondar en el marco teórico de los diferentes tipos de tramas 802.11k/v que están implementadas actualmente en dispositivos comerciales
2. Fue posible crear una herramienta para la generación controlada de tramas 802.11k/v contra clientes o puntos de acceso inalámbricos
3. Se pudo demostrar que los dispositivos probados, no contienen vulnerabilidades básicas evidentes, según las pruebas realizadas

La creación de la herramienta, con los distintos tipos de mutaciones sobre tramas de 802.11k/v, en sí misma, un avance significativo, ya que no se ha podido conseguir herramientas similares en el mercado

La herramienta actual hace una generación de tramas en forma “no solicitada - stateless” hacia los clientes, lo que supone una posible limitación significativa para conseguir que la trama inválida sea procesada por el equipo bajo prueba para causar un fallo

La planificación del trabajo sufrió retrasos debido a compromisos laborales, por lo que no fue posible la creación de baterías de pruebas exhaustivas para cada una de las tramas soportadas por los protocolos 802.11k/v, si no que se hizo un enfoque sobre un subconjunto de aquellas que fueran más comúnmente implementadas por los dispositivos

Este es un proyecto que se desea continuar, dentro de las actividades de laborales de este investigador, enfocándose en:

1. Creación de una máquina de estados, que permita escuchar peticiones 802.11k/v y responder de forma automática al cliente, haciendo spoofing del punto de acceso, y respetando el dialog token enviado por el cliente, aumentando las probabilidades de éxito de la prueba
2. Avanzar en la implementación de BSS transition requests, dado que es un mecanismo que podría usarse para impactar negativamente una red inalámbrica, si se consigue que los clientes realicen un roaming incorrecto

4. Glosario

802.11k: Es un estándar que define protocolos que ayudan a realizar una transición más rápida entre diferentes puntos de acceso. Provee información de radio sobre que otros dispositivos puedan ofrecer un SSID y los canales donde se encuentran

802.11r: Es un protocolo que permite la transición segura y rápida de un dispositivo inalámbrico, entre dos puntos de acceso. También puede ser llamado Fast Transition, o Fast Roaming. Está principalmente enfocado a dispositivos en movimiento (teléfonos, PDA, vehículos, etc)

802.11v: Define una serie de extensiones con el fin de permitir la gestión de redes inalámbricas (Wireless Network Management, WMN), incluyendo información sobre la red RF o topología

802.11w: Es una extensión a los estándares 802.11, que permite la protección de tramas de gestión (Protected Management Frames, PMF por sus siglas en inglés). Provee control de integridad, autenticación y anti replay

Fuzzing: Es una técnica de pruebas automatizadas para software, que consiste en proveer datos inválidos, aleatorios o no esperados a un componente bajo prueba

IE: lease *Information Element*

Information Element: Es básicamente un campo dentro de una trama 802.11. Suele estar comprendido de un identificador de tipo, una longitud y datos.

Roaming: Dentro del contexto de redes 802.11, consiste en la transición de conectividad de una estación inalámbrica de un punto de acceso a otro. Esto puede conllevar fases de escaneo de canales, autenticación y establecimiento de nueva asociación de un punto a otro

Spoofing: Dentro del contexto de seguridad informática, es la situación donde un atacante, se hace pasar por otro dispositivo mediante la falsificación de datos

SSID: Por sus siglas en inglés: Service Set IDentifier, un identificador de grupo de servicio. Sirve para dar una etiqueta en texto natural, a una red o servicio inalámbrico

Tramas de Asociación (association request): Es la trama que genera un cliente WiFi, para solicitar el establecimiento de una nueva relación con un punto de acceso.

5. Bibliografía

[1]: IEEE 802.11–2012:
https://standards.ieee.org/findstds/standard/802.11-2012.html

[2]: IEEE 802.11k:
https://standards.ieee.org/findstds/standard/802.11k-2008.html

[3]: IEEE 802.11v:
https://standards.ieee.org/findstds/standard/802.11v-2011.html

[4]: IEEE 802.11r:
https://standards.ieee.org/findstds/standard/802.11r-2008.html

[5]: Implementación Comercial de Fuzzer para wireless:
https://www.synopsys.com/software-integrity/security-testing/fuzz-testing/defensics/protocols/80211c.html

[6]: Uso de Scapy para la generación de Tramas Inalámbricas:
https://scapy.readthedocs.io/en/latest/usage.html?highlight=wireless

[7]: Implementación Comercial de Fuzzer para wireless:
https://www.beyondsecurity.com/dynamic_fuzzing_testing_wifi_protocol.html

[8]: Implementación de fuzzer contral puntos de acceso inalámbricos:
http://blog.emaze.net/2011/09/testing-wireless-access-points.html

6. Anexos

6.1 Código fuente

```
import argparse
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
from netaddr import EUI, valid_mac

import time
from datetime import datetime
import binascii

version = "0.1"

MAX_SN = 4095
MAX_FRAG = 16

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

ALL, DEBUG, INFO, STATUS, WARNING, ERROR = range(6)
COLORCODES = {"gray": "\033[0;37m",
              "green": "\033[0;32m",
              "orange": "\033[0;33m",
              "red": "\033[0;31m",
              "none": "\033[1;0m"}

global_log_level = INFO


def log(level, msg, color=None, showtime=True):
    if level < global_log_level: return
    if level == DEBUG and color is None:
        color = "gray"
    elif level == WARNING and color is None:
        color = "orange"
    elif level == ERROR and color is None:
        color = "red"
    else:
        color = "none"

    print((datetime.now().strftime('[%H:%M:%S]  ')  if  showtime  else  "  "  *  11)  +
COLORCODES.get(color,

"") + msg + "\033[1;0m")




def calculateSC(sn, frag=0):
    if (frag > MAX_FRAG): frag = 0
    if (sn > MAX_SN): sn = 0
    hexSN = hex(sn)[2:] + hex(frag)[2:]
    SC = int(hexSN, 16)
    return SC


def random_char(length=10):
    """
    Generate a set of random ascii characters.
    :param length:
    :return:
    """
    # randomstring = "".join(chr(random.randrange(256)) for _ in range(length)).lower()
    # return str.encode(randomstring)
    return bytes(random.getrandbits(8) for _ in range(length))
```

```
def random_mac():
    # returns a random mac in byte format
    # ensures it is not a local address or multicast

    filter = b"\xfc"
    mac = random.randint(0, 255).to_bytes(1, "little")[0] & filter[0]
    mac = mac.to_bytes(1, "little")
    for value in range(0, 5):
        mac += random.randint(0, 255).to_bytes(1, "little")

    return mac


class ElevenKVTest():
    def __init__(self, options):
        self.nic_iface = options.interface
        self.nic_mon = options.interface + "mon"
        if options.source=="":
            #no mac provided, pull the physical address
            self.source = scapy.arch.get_if_hwaddr(options.interface)
        else:
            self.source=options.source
        self.sourcebinary = binascii.unhexlify(self.source.replace(':', ''))
        #self.bssid_str = self.getBSSID()
        self.destination = options.destination
        self.destinationbinary = binascii.unhexlify(self.destination.replace(':', ''))
        self.bssid = options.bssid
        self.bssidbinary = binascii.unhexlify(self.bssid.replace(':', ''))

        if self.bssid:
            # if we got address, set the base radio mac
            self.baseRadio  =  binascii.unhexlify((self.bssid[0:16]  +  '0').replace(':',
''))
        '''
        self.configure_interfaces()
        if options.channel !=0:
            #ok, channel is set, configure the interface
            self.set_channel(options.channel)
        '''




    def getBSSID(self):
        from sys import platform
        import subprocess
        if platform == 'linux' or platform == 'linux2':
            # linux
            output   =   subprocess.check_output(['iwgetid',   self.nic_iface,   '-r',   '-
a'])[0:17]
            output = str(output, "utf-8")
            if len(output) == 17:
                return output
            else:
                return None

        else:

            return None
        '''
        #not using other platforms for the moment
        elif platform == 'darwin':
            # OS X
            ps = subprocess.Popen(

('/System/Library/PrivateFrameworks/Apple80211.framework/Versions/Current/Resources/airp
ort', '-I'),
                stdout=subprocess.PIPE)
            output   =   subprocess.check_output(('awk',   '/   SSID/   {print   substr($0,
index($0, $2))}'), stdin=ps.stdout)
        elif platform == 'win32':
            output = subprocess.check_output("netsh wlan show interfaces")
            ps.wait()
```

```
        '''


def payload_neighreport():
    #Valid neigh report, hard coded SSID as per lab

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token

    neigh_report += b"\x00" # SSID IE
    neigh_report += b"\x04"  # length
    neigh_report += "3504".encode()  # SSID sting

    return neigh_report

def payload_neighreport_nullSSID():
    #SSID null length

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token

    neigh_report += b"\x00"  # SSID IE
    neigh_report += b"\x00"  # length

    return neigh_report


def payload_neighreport_invalidSSID():
    # SSID null length

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token

    neigh_report += b"\x00"  # SSID IE
    neigh_report += b"\x04"  # length
    neigh_report += "none".encode()  # SSID sting

    return neigh_report



def payload_neighreport_largeSSID():
    # large SSID length, random payload

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token

    neigh_report += b"\x00"  # SSID IE
    neigh_report += b"\xFF"  # length

    for entry in range(0, 255):  # SSID content
        neigh_report += random.randint(0, 255).to_bytes(1, "little")
    return neigh_report

def payload_neighreport_randomSSID():
    # random SSID length, 0 to

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token
    neigh_report += b"\x00"  # SSID IE

    neigh_report += random.randint(1, 255).to_bytes(1, "little") #random size

    for entry in range(0, random.randint(0, 255)):  # SSID content random
        neigh_report += random.randint(0, 255).to_bytes(1, "little")

    return neigh_report
```

```
def payload_neighreport_largeSSIDnull():
    # large SSID, not present

    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x04"  # Neighbor Report Request
    neigh_report += b"\x01"  # Dialog Token
    neigh_report += b"\xFF"  # SSID IE


    return neigh_report

def payload_neighponse(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x05"  # Neighbor Report Response
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token

    neigh_report += b"\x34"  # Neigh Report
    neigh_report += b"\x0d"  # length
    neigh_report += kvtest.bssidbinary  # target AP
    neigh_report += b"\xf7\x02\x00\x00"  # BSSID Information
    neigh_report += b"\x00"  # Operating class
    neigh_report += b"\x40"  #channel num
    neigh_report += b"\x07"  #phy type

    return neigh_report

def payload_neighponse_randomBSSID():

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x05"  # Neighbor Report Response
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    for count in range(0, random.randint(0, 40)):
        neigh_report += b"\x34"  # Neigh Report
        neigh_report += b"\x0d"  # length
        neigh_report += kvtest.bssidbinary  # target AP
        neigh_report += b"\xf7\x02\x00\x00"  # BSSID Information
        neigh_report += b"\x00"  # Operating class
        neigh_report += b"\x40"  #channel num
        neigh_report += b"\x07"  #phy type

    return neigh_report

def payload_neighponse_randomIEcontent():

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x05"  # Neighbor Report Response
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    for count in range(0,random.randint(0, 40)):
        neigh_report += b"\x34"  # Neigh Report
        neigh_report += b"\x0d"  # length
        neigh_report += random_mac() # BSSID
        neigh_report += random_mac()  # BSSID info random
        neigh_report += random.randint(0, 255).to_bytes(1, "little")# Operating class
        neigh_report += random.randint(0, 255).to_bytes(1, "little") #channel num
        neigh_report += random.randint(0, 255).to_bytes(1, "little") #phy type

    return neigh_report

def payload_bsstransreq():

    neigh_report= b"\x0A"  # WNM
    neigh_report += b"\x07"  # BSS Transition Management Request
    neigh_report += b"\x01"    #dialog token
    neigh_report += b"\x12"  # request mode, preferred, abridged, no inminent, no bss
termination included, ess dis inminent (url)
    neigh_report += b"\x00\x00"  # dissac timer
    neigh_report += b"\x10"  # validity interval
    neigh_report += b"\x11"  # url lenfth
    neigh_report += "http://google.com".encode()  # url


    return neigh_report
```

```
def payload_bstransreqrandom():
    neigh_report = b"\x0A"  # WNM
    neigh_report += b"\x07"  #  BSS Transition Management Request
    neigh_report += b"\x01"  # dialog token

    for count in range(0, random.randint(0, 100)):
        neigh_report += random_mac()  # BSSID


    return neigh_report


def payload_ft_random(kvtest):
    neigh_report = b"\x06"  # FT BSS transition
    neigh_report += b"\x01"  # request
    neigh_report += random_mac()  # STA adress
    neigh_report += kvtest.bssidbinary # target AP
    neigh_report += b"\x30"  # RSN
    neigh_report += random_char(random.randint(0, 200))
    return    neigh_report

def payload_ft(kvtest):
    neigh_report = b"\x06"  # FT BSS transition
    neigh_report += b"\x01"  # request
    neigh_report += random_mac()  # STA adress
    neigh_report += kvtest.bssidbinary  # target AP
    neigh_report += b"\x30"  # RSN
    neigh_report += b"\x26"  # length
    neigh_report += b"\x01\x00"  # version
    neigh_report += b"\x00\x0f\xac\x04"  # Group key, AES
    neigh_report += b"\x01\x00"  # PTK count
    neigh_report += b"\x00\x0f\xac\x04"  # PTK key, AES
    neigh_report += b"\x01\x00"  # AKM count
    neigh_report += b"\x00\x0f\xac\x04"  # AKM, ft
    neigh_report += b"\x0c\x00"  # RSN capabilities
    neigh_report += b"\x01\x00"  # PMKID count
    neigh_report += random_char(16)  # PMKID
    neigh_report += b"\x36"  # mobility domain
    neigh_report += b"\x03"  # length
    neigh_report += b"\x5d\x11"  # identifier
    neigh_report += b"\x01"  # policy
    neigh_report += b"\x37"  # RSN
    neigh_report += b"\x58"  # length
    neigh_report += b"\x00\x00"  # mic control
    neigh_report += random_char(16)  # mic
    neigh_report += random_char(32)  # anonce
    neigh_report += random_char(32)  # snonce
    neigh_report += b"\x03"  # PMK-k0
    neigh_report += b"\x04"  # length
    neigh_report += random_char(4)  # holder

    return neigh_report


def payload_measurementreport(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    neigh_report += b"\x27"  # Measurement Report
    neigh_report += b"\x1d"  # length
    neigh_report += b"\x60"  # token
    neigh_report += b"\x00"  # report mode
    neigh_report += b"\x05"  # beacon report
    neigh_report += b"\xf1"  # class
    neigh_report += b"\x28"  # channel
    neigh_report += b"\x46\x21\x46\x28\x30\x0e\xf8\x8d"  # time
    neigh_report += b"\xb3\xbc"  # duration
    neigh_report += b"\x00"  # frame information
    neigh_report += b"\xd6"  # power indication
    neigh_report += b"\x23"  # noiseindication
    neigh_report += kvtest.bssidbinary# BSSID reported
    neigh_report += b"\x00"  # antenna ID
```

```
    neigh_report += b"\x00\x00\x00\x00"  # TSF
    neigh_report += b"\xdd"  # Vendor Specific tag
    neigh_report += b"\x18"  #  length
    neigh_report += b"\x00\x17\xf2"  # apple
    neigh_report += b"\x0a"  # type
    neigh_report                                                                    +=
b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64\x37\x2c\x33"     #
data

    return neigh_report

def payload_measurementreport_nomeasurement(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token

    neigh_report += b"\xdd"  # Vendor Specific tag
    neigh_report += b"\x18"  #  length
    neigh_report += b"\x00\x17\xf2"  # apple
    neigh_report += b"\x0a"  # type
    neigh_report                                                                    +=
b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64\x37\x2c\x33"     #
data

    return neigh_report

def payload_measurementreportnobeacon(kvtest):

    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    neigh_report += b"\x27"  # Measurement Report
    neigh_report += b"\x05"  # length
    neigh_report += b"\x60"  # token
    neigh_report += b"\x00"  # report mode
    neigh_report += b"\x00"  # frame information
    neigh_report += b"\xd6"  # power indication
    neigh_report += b"\x23"  # noiseindication
    neigh_report += b"\xdd"  # Vendor Specific tag
    neigh_report += b"\x18"  #  length
    neigh_report += b"\x00\x17\xf2"  # apple
    neigh_report += b"\x0a"  # type
    neigh_report                                                                    +=
b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64\x37\x2c\x33"     #
data

    return neigh_report

def payload_measurementreportnoapple(kvtest):
    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  # dialog token
    neigh_report += b"\x27"  # Measurement Report
    neigh_report += b"\x1d"  # length
    neigh_report += b"\x60"  # token
    neigh_report += b"\x00"  # report mode
    neigh_report += b"\x05"  # beacon report
    neigh_report += b"\xf1"  # class
    neigh_report += b"\x28"  # channel
    neigh_report += b"\x46\x21\x46\x28\x30\x0e\xf8\x8d"  # time
    neigh_report += b"\xb3\xbc"  # duration
    neigh_report += b"\x00"  # frame information
    neigh_report += b"\xd6"  # power indication
    neigh_report += b"\x23"  # noiseindication
    neigh_report += kvtest.bssidbinary  # BSSID reported
    neigh_report += b"\x00"  # antenna ID
    neigh_report += b"\x00\x00\x00\x00"  # TSF


    return neigh_report

def payload_measurementreportrandommeasurement(kvtest):
```

```
    neigh_report= b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  #dialog token
    neigh_report += b"\x27"  # Measurement Report
    length=random.randint(0, 255).to_bytes(1, "little")
    lengthint=int.from_bytes(length, byteorder='big')
    neigh_report += length  # length
    for count in range(0,lengthint):
        neigh_report += random.randint(0, 255).to_bytes(1, "little") #payload

    neigh_report += b"\xdd"  # Vendor Specific tag
    neigh_report += b"\x18"  #  length
    neigh_report += b"\x00\x17\xf2"  # apple
    neigh_report += b"\x0a"  # type
    neigh_report                                                                    +=
b"\x01\x01\x11\x01\x06\x31\x31\x2e\x33\x2e\x31\x02\x07\x69\x50\x61\x64\x37\x2c\x33"    #
data

    return   neigh_report

def payload_measurementreportrandomapple(kvtest):
    neigh_report = b"\x05"  # Radio Measurament
    neigh_report += b"\x01"  # Radio Measurement Report
    neigh_report += random.randint(0, 2).to_bytes(1, "little")  # dialog token
    neigh_report += b"\x27"  # Measurement Report
    neigh_report += b"\x1d"  # length
    neigh_report += b"\x60"  # token
    neigh_report += b"\x00"  # report mode
    neigh_report += b"\x05"  # beacon report
    neigh_report += b"\xf1"  # class
    neigh_report += b"\x28"  # channel
    neigh_report += b"\x46\x21\x46\x28\x30\x0e\xf8\x8d"  # time
    neigh_report += b"\xb3\xbc"  # duration
    neigh_report += b"\x00"  # frame information
    neigh_report += b"\xd6"  # power indication
    neigh_report += b"\x23"  # noiseindication
    neigh_report += kvtest.bssidbinary  # BSSID reported
    neigh_report += b"\x00"  # antenna ID
    neigh_report += b"\x00\x00\x00\x00"  # TSF


    neigh_report += b"\xdd"  # Vendor Specific tag
    #length = random.randint(0, 255).to_bytes(1, "little")
    length = b"\x18"
    lengthint = int.from_bytes(length, byteorder='big')
    neigh_report += length  # length

    neigh_report += b"\x00\x17\xf2"  # apple
    neigh_report += b"\x0a"  # type
    for count in range(0, lengthint):
        neigh_report += random.randint(0, 255).to_bytes(1, "little")  # payload


    return   neigh_report


def main():
    # Parse CLI arguments
    parser = argparse.ArgumentParser(description="802.1k/v Test Tool")
    parser.add_argument('--version',         action='version',         version='%(prog)s
{}'.format(version))
    parser.add_argument("-i",  "--interface",  default="wlan0",  help="Interface.  (def:
wlan0), it will be set to monitor mode")
    parser.add_argument("-s", "--source",  default="", help="Spoofs source mac address,
default uses interface address")
    parser.add_argument("-d", "--destination",  required=True, help="Destination mac")
    parser.add_argument("-bs",  "--bssid", help="BSSID,   if not set, will be same as
destination")
    parser.add_argument("-ch", "--channel", type=int, default=0, help="Channel to use,
default: use current configured channel")
    parser.add_argument("-c",  "--count",  type=int,  default=20,  help="Number  of  TX
events. (def: 20)")
    parser.add_argument("-delay", type=float, default=0.1, help="Delay between TX events
(def: 0.4s)")
```

```
    parser.add_argument("-b", "--burst", type=int, default=1,
                        help="Burst of packets to transmit on each TX event. (def 1)")
    parser.add_argument("-t", "--test", required=True, default="neighreport", nargs='?',
const='neighreport',
                        choices=['all11','neighreport',         'neighreport_randomSSID',
'neighreport_largeSSID', 'neighreport_invalidSSID' ,
                                 'neighreport_largeSSIDnull',     'neighreport_nullSSID',
'neigh-response', 'measurementreport',

'measurementreport_nomeasurement','measurementreportnobeacon',
'measurementreportnoapple',
                                 'measurementreportrandommeasurement',
'measurementreportrandomapple',
                                 'bsstrans-req',  'bsstrans-random',  'ft',  'ft-random'],
help="Select the test payload type: neigh-report")


    # parser_test = parser.add_argument_group("Test cases,  must select one" )
    # parser_ex = parser_test.add_mutually_exclusive_group()

    options = parser.parse_args()

    if options.source!=""  and not valid_mac(options.destination):
        log(ERROR, "Please provide valid source MAC")
        return 1

    if not valid_mac(options.destination):
        log(ERROR, "Please provide valid destination MAC")
        return 1

    if options.bssid!="" :
            if not valid_mac(options.destination):
                log(ERROR, "Please provide valid source MAC")
                return 1
    else:
        options.bssid=options.destination

    kvtest = ElevenKVTest(options)



    transmit_address = kvtest.source
    receiver_address = kvtest.destination
    bssid=kvtest.bssid


    seq = 0



    if options.test == "all11":
        globalSEQ=0 #carry over the used SEQ to all the tests

        #goes over all frame types inside 11k, sending count frames with random payload
on each one
        log(STATUS, "Running full 11-action random test")
        # subtype 32=data, 40=qosdata
        for category in range(0,15):
            if category !=3: #skip BAK
                radio_measurament_action_values  =  {b"\x00",  b"\x01",  b"\x02",  b"\x03",
b"\x04", b"\x05", b"\x06", b"\xff"}


                for action_field in radio_measurament_action_values:
                    for x in range(0, options.count):
                        payload = category.to_bytes(1, "little") # all categories
                        payload += action_field  # Neighbor Report Response
                        payload += random.randint(0, 2).to_bytes(1, "little")  # dialog
token
                        payload += random_char(random.randint(0, 1400))

                        packet  =  RadioTap(present=0)  /  Dot11(addr1=receiver_address,
addr2=transmit_address,
```

```
                                                        addr3=bssid,        type=00,
subtype=0x0d, FCfield=0,
                                                        SC=calculateSC(globalSEQ))
                        packet /= payload
                        sendp(packet, iface=options.interface, count=options.burst)
                        time.sleep(options.delay)
                        packet.show()
                        print("Category Code: " + str(category))
                        print("Action Field: " + str(action_field))

                        globalSEQ += 1


    else:
        # repeat the transmit events
        for x in range(0, options.count):
            seq += 1
            if seq > MAX_SN: seq = 0

            # subtype 32=data, 40=qosdata
            for burst in (0, options.burst):
                packet    =    RadioTap(present=0)    /    Dot11(addr1=receiver_address,
addr2=transmit_address,
                                                           addr3=bssid,            type=00,
subtype=0x0d, FCfield=0,
                                                           SC=calculateSC(x))


                if options.test == "neighreport":
                    log(STATUS, "Running Basic Neighbor Report")
                    packet /= payload_neighreport()
                elif options.test == "neighreport_randomSSID":
                    log(STATUS, "Running  Neighbor Report Random SSID")
                    packet /= payload_neighreport_randomSSID()
                elif options.test == "neighreport_largeSSID":
                    log(STATUS, "Running  Neighbor Report Large SSID")
                    packet /= payload_neighreport_largeSSID()
                elif options.test == "neighreport_invalidSSID":
                    log(STATUS, "Running  Neighbor Report  Invalid SSID")
                    packet /= payload_neighreport_invalidSSID()
                elif options.test == "neighreport_nullSSID":
                    log(STATUS, "Running  Neighbor Report  Null SSID")
                    packet /= payload_neighreport_nullSSID()
                elif options.test == "neighreport_nullSSID":
                    log(STATUS, "Running  Neighbor Report  Large SSID, no payload")
                    packet /= payload_neighreport_largeSSIDnull()
                elif options.test == "measurementreport":
                    log(STATUS, "Running  Measurement Report")
                    packet /= payload_measurementreport(kvtest)
                elif options.test == "measurementreport_nomeasurement":
                    log(STATUS, "Running  Measurement Report, no measurement IE")
                    packet /= payload_measurementreport_nomeasurement(kvtest)
                elif options.test == "measurementreportnobeacon":
                    log(STATUS, "Running  Measurement Report, bo beacon IE")
                    packet /= payload_measurementreportnobeacon(kvtest)
                elif options.test == "measurementreportnobeacon":
                    log(STATUS, "Running  Measurement Report, no apple IE")
                    packet /= payload_measurementreportnoapple(kvtest)
                elif options.test == "measurementreportrandommeasurement":
                    log(STATUS, "Running  Measurement Report, random measure IE")
                    packet /= payload_measurementreportrandommeasurement(kvtest)
                elif options.test == "measurementreportrandomapple":
                    log(STATUS, "Running  Measurement Report, random apple IE")
                    packet /= payload_measurementreportrandomapple(kvtest)

                elif options.test == "neigh-response":
                    log(STATUS, "Running  Neighbor Response")
                    packet /= payload_neighponse(kvtest)
                elif options.test == "bsstrans-req":
                    log(STATUS, "Running  BSS Transition Request")
                    packet /= payload_bsstransreq()
                elif options.test == "bsstrans-random":
                    log(STATUS, "Running  BSS Transition Random")
                    packet /= payload_bsstransreqrandom()
```

```
                elif options.test == "ft":
                    log(STATUS, "Running  FT")
                    packet /= payload_ft(kvtest)
                elif options.test == "ft-random":
                    log(STATUS, "Running  FT random")
                    packet /= payload_ft_random(kvtest)

                else:
                    log(ERROR, "INTERNAL ERROR: Uknown test")
                    return 1


                pkt_list = []
                pkt_list.append(packet)

                sendp(packet, iface=options.interface, count=options.burst)
                time.sleep(options.delay)

        packet.show()

    return


if __name__ == "__main__":
    main()
```

6.2 Configuración preliminar para tarjeta de red

Para poder utilizar la herramienta, es necesario realizar una configuración previa y crear una interface auxiliar en modo monitor. Dependiendo del tipo de manejador de dispositivo USB, algunos modelos de tarjetas inalámbricas no suportan estas operaciones

```
iw wlan0 interface add wlan0mon type monitor
iwconfig wlan0mon mode monitor
iwconfig wlan0 mode monitor
iwconfig wlan0mon ch 40
iwconfig wlan0 ch 40
ifconfig wlan0mon up
ifconfig wlan0 up
```

6.3 Comprobación del estado de la tarjeta

Ejemplo de tarjeta correctamente configurada para la realización de las pruebas. El canal ha de coincidir con el utilizado por el punto de acceso inalámbrico

```
root@kali:~# iwconfig
wlan0mon  IEEE 802.11  Mode:Monitor  Frequency:5.2 GHz  Tx-Power=18 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

eth0      no wireless extensions.

wlan0     IEEE 802.11  Mode:Monitor  Frequency:5.2 GHz  Tx-Power=18 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

lo        no wireless extensions.
```