Universidad
Carlos III de Madrid
www.uc3m.es

Extracción de Información Semántica en Redes Inalámbricas

Título: Extracción de Información Semántica en Redes Inalámbricas

Autor: Alejandro Calleja Cortiñas

Titulación: Grado en Ingeniería Informática

Tutores: Jorge Blasco Alís y Guillermo Suárez De Tangil-Rotaeche

Fecha de la defensa: 30/9/2013

"Your time is limited, so don't waste it living someone else's life. Don't be trapped by dogma - which is living with the results of other people's thinking. Don't let the noise of others' opinions drown out your own inner voice. And most important, have the courage to follow your heart and intuition".

Steve Jobs

AGRADECIMIENTOS

Por fin, después de cuatro intensos años de formación, presento mi trabajo de fin de estudios, desarrollado con toda la ilusión que he podido depositar en él. Es momento de dedicar un sincero agradecimiento a todas las personas que han hecho posible que haya llegado hasta aquí.

En primer lugar, a la Universidad Carlos III de Madrid, por la excelente formación que me ha permitido alcanzar en el campo de la ingeniería informática, lo que me ha hecho depositar mi confianza en ella una vez más para continuar mi formación cursando el "Máster Universitario en Ciencia y Tecnología Informática". En particular me gustaría agradecer a los profesor Jorge Blasco y Guillermo Suárez del grupo de investigación COSEC la ayuda que me han aportado durante los meses que ha durado el desarrollo de este proyecto.

En segundo lugar, a mi familia, sin cuyo ánimo, cariño, apoyo y motivación, alcanzar este logro habría sido imposible

En tercer lugar, a mis amigos, a los de dentro de la universidad y los de fuera, que han compartido conmigo los mejores y peores momentos que he vivo durante los últimos cuatro años.

Y por último, me gustaría agradecer al sistema de educación pública, tan menospreciado en los últimos tiempos, la financiación de mis estudios superiores, que ha facilitado en gran medida la consecución de este título.

TRIBUNAL

PRESIDENTE: Tomás Eduardo de la Rosa Turbides

SECRETATRIO: Andrés Segura Aragonés

VOCAL: José Arturo Mora Soto

Tutores: Jorge Blasco Alís y Guillermo Suárez De Tangil-Rotaeche

Fecha de la defensa: 30/09/2013

Lugar: Escuela Politécnica Superior, Campus de Leganés.

RESUMEN

Desde hace unos años, los dispositivos móviles y en particular los *Smartphone* y *tablets* se han consolidado como el producto estrella en el mundo de la electrónica de consumo para el gran público. Un estudio publicado en el verano de 2013 por la consultora Gartner asegura que las ventas de teléfonos móviles inteligentes superaron a las ventas de teléfonos móviles convencionales por primera vez y a nivel mundial durante el segundo trimestre de 2013 [1].

Uno de los principales motivos que hacen que este tipo de dispositivos sea tan popular entre los usuarios es que ofrece la oportunidad de disfrutar de un ordenador en la palma de la mano. Permiten navegar por Internet, enviar y recibir correos, chatear con nuestros contactos o compartir las últimas novedades en las redes sociales, esto unido al hecho de que las redes WiFi cada vez tienen más presencia en los lugares de ocio, han hecho posible que estemos conectados las 24h del día con nuestro entorno digital.

Sin embargo, para la gran mayoría de los usuarios la seguridad de estos dispositivos no supone un tema preocupante. A pesar de esto, es un aspecto que no conviene olvidar dado que a través de nuestros dispositivos compartimos gran cantidad de información privada y de carácter personal, información que pude acabar en manos de terceras personas si no se presta atención a la seguridad de los dispositivos y de las redes

En este documento se detallará el proceso de desarrollo de dos nuevos módulos para la aplicación dSploit, orientada a la auditoría y a pruebas de penetración en redes inalámbricas. Además, se expondrán los mecanismos en los que se sustentan las comunicaciones móviles a través de Internet y como una red insegura puede poner en riesgo nuestra privacidad en la red.

ABSTRACT

In recent years, mobile devices and, in particular, smartphones and tablets have emerged as the star product in the market of consumer electronics for the general public. A study published in the summer of 2013 by technological consultancy Gartner says smartphone sales have surpassed sales of conventional mobile phones for the first time and worldwide in the second quarter of 2013 [1].

One of the main reasons why these devices are so popular is that they offer users the opportunity to enjoy a computer in their hands, as it enables you to browse the Internet, send and receive emails, chat with your contacts or share the latest news in social networks. This coupled with the fact that wireless networks are increasingly present in our entertainment places have allow us to be online all long the day.

However, for most of the people, the security on mobile devices is not a serious matter. Despite of this, this is an issue that should not be unnoticed about because while using our mobile devices, we share a lot of private information which could be available to third persons if you don't take care about security on devices and networks.

In this document, it's detailed the development process of two new modules for the dSploit app, used for network analysis and penetration testing. Besides, the mechanisms behind the security of mobile networks will be explained and how an unsecured network could put our online privacy on a jeopardy situation.

Índice de Contenidos

Índice de Tablas

Índice de Ilustraciones

1. Introducción

En esta introducción se presentan al lector la motivación inicial, los objetivos y la estructura del documento.

1.1. Motivación y Objetivos

Los objetivos que se han pretendido alcanzar con la realización de este proyecto así como las motivaciones principales quedan recogidos a continuación:

- Demostrar la sencillez con la que un tercer sujeto, en principio ajeno a la comunicación, puede interceptar la información intercambiada mediante nuestros dispositivos móviles conectados a través de un canal no seguro como los que frecuentemente pueden encontrarse en centros de ocio o universidades. Nos centraremos en particular en la interceptación de los datos generados durante una sesión HTTP, ya que son susceptibles de transportar una mayor cantidad de información acerca del usuario o del dispositivo que esté utilizando.

- Presentar los distintos modelos de ataques típicos que pueden llevarse a cabo en una red inalámbrica. Dichos ataques permitirían la interceptación de paquetes, la sustitución o la violación de la integridad del contenido de los mismos, así como poner en peligro la disponibilidad de los servicios prestados en la red atacada.

- Dar a conocer al usuario sin conocimientos técnicos los mecanismos de protección y las contramedidas que pueden ayudarle a advertir y/o impedir los robos de información así como ciertos tipos de ataques de modificación.

- Ampliar los conocimientos previos obtenidos durante el Grado en Ingeniería Informática acerca del desarrollo de aplicaciones para la plataforma Android.

- Dotar de nuevas funcionalidades a la suite de seguridad móvil dSploit, orientada al análisis y la auditoría de entornos LAN inalámbricos.

- La participación activa en un proyecto de software libre, y la adquisición de los conocimientos técnicos que son necesarios para dicho cometido (sistemas de control de versiones, comunicación con los administradores del proyecto, etc.)

1.2. Estructura del Documento

El presente documento está estructurado en siete secciones principales, contando cada una de ellas con las debidas subsecciones. Dicho documento cuenta además con un anexo que se compone de la bibliografía empleada en el documento y del manual de usuario del módulo desarrollado.

El contenido de los distintos apartados queda expuesto a continuación:

- La sección 1. Introducción: Incluye la motivación principal en la que se ha sustentado el trabajo realizado, así como los objetivos que se han pretendido alcanzar con la realización de este proyecto de fin de grado. Por otro lado, contiene el desglose del contenido del documento.

- La sección 2. Estado del Arte: Recoge el estado actual del mercado de los sistemas operativos móviles, incluyendo su cuota de mercado y una breve descripción de cada uno de ellos. Se incluye un breve estudio de las aplicaciones disponibles orientadas a la seguridad y el análisis de redes inalámbricas. Por último se realiza una amplia exposición acerca de la arquitectura de las redes inalámbricas, las distintas capas de abstracción que las componen, los mecanismos de seguridad presentes en las distintas capas (incluyendo las peculiaridades de la plataforma Android) y por último se presentan los distintos tipos de ataques más comunes propios de este tipo de redes.

- La sección 3.Análisis: Incluye los principales casos de uso así como la especificación de requisitos que deberán quedar cubiertos por las nuevas funcionalidades.

- La sección 4.Diseño: Se centra en exponer cómo se construirán los nuevos módulos, así como la forma en que éstos interactuarían con el resto de la aplicación. Para ello se lleva a cabo un estudio previo de la

arquitectura del sistema y se plantean los diagramas y modelos que guiarán el desarrollo.

- La sección 5.Implementacion y pruebas: Cubre los aspectos particulares del desarrollo, tales como la tecnología y particularidades de la plataforma en la que se ejecutará el resultado final. Además recoge un informe acerca de las pruebas realizadas y sus resultados.

- La sección 6. Gestión del Proyecto: Agrupa los aspectos referentes a la metodología seguida, la planificación y el ciclo de vida de desarrollo. Incluye también los recursos, tanto humanos como materiales, empleados en la realización del proyecto y coste estimado de los mismos.

- La sección 7. Conclusiones y Trabajos Futuros: Se concluye este documento ofreciendo un análisis de las principales contramedidas que permitirían al usuario protegerse de los ataques detallados en secciones anteriores. Se exponen también las principales conclusiones a la finalización del proyecto y se proponen las principales líneas futuras.

2. Estado Del Arte

A continuación se repasa el panorama actual de los sistemas operativos para dispositivos móviles y de las aplicaciones móviles orientadas a la seguridad, centrándonos particularmente en la plataforma Android. Por último se explica en profundidad el funcionamiento de las comunicaciones móviles basadas en redes WiFi, sus vulnerabilidades y vectores de ataque más comunes.

2.1. Panorámica de los SSOO Móviles

Conviene comenzar esta sección proporcionando una definición concisa de qué consideramos un dispositivo móvil. Un dispositivo móvil está concebido como un computador, con la capacidad de ser transportado y utilizado sin la necesidad de depender de una fuente de energía o un origen de datos que límite su movilidad. Según esto, multitud de objetos de nuestra vida cotidiana podrían considerarse dispositivos móviles, desde calculadoras de bolsillo hasta relojes digitales, pasando por teléfonos móviles o elementos más sofisticados como las redes de sensores.

La definición es, en definitiva, muy amplia. Sin embargo, en este documento nos limitaremos a reducir esta definición a los dispositivos que han adquirido una mayor importancia en nuestras vidas durante los últimos años: los teléfonos inteligentes o smartphones y las tabletas electrónicas o tablets. La versatilidad de estos dispositivos, así como la posibilidad de conexión a Internet a través de redes móviles como WiFi o GPRS/3G y su máxima portabilidad, han hecho que se conviertan en los reyes del mercado de la tecnología. Multitud de fabricantes han realizado cuantiosas inversiones con el objetivo de desarrollar productos cada vez más potentes y con más funcionalidades, con el objetivo de atraer a un mayor número de usuarios.

De manera pareja a la aparición de esta nueva familia de computadores portátiles aparecieron de mano de las principales empresas de tecnología los sistemas operativos diseñados particularmente para estos dispositivos. A continuación, describiremos las

características más relevantes de cada una de las plataformas móviles más populares en la actualidad.

2.1.1. BlackBerry OS

Podemos considerar a la empresa canadiense *Research In Motion* (hoy en día conocida como BlackBerry) la pionera en el mundo de los teléfonos inteligentes. RIM estrenó su familia de smartphones BlackBerry en el año 2003, se trataba de dispositivos que integraban pantalla a color y teclado *qwerty* lo que facilitó que se hicieran sumamente populares, sobre todo en entornos empresariales. Si bien los primeros modelos de la familia no incorporaban la posibilidad de conexión a redes WiFi (las cuales por aquel entonces no gozaban del grado de implantación actual), sin embargo si permitía la conexión a Internet a través de redes celulares, en concreto GPRS. Esta conexión abría un mundo de posibilidades, tales como la sincronización de correo o la navegación web.

El sistema operativo que gobierna los dispositivos BlackBerry fue desarrollado por la misma compañía bajo el nombre BlackBerry OS. Inicialmente e igual que el dispositivo, estaba orientado a un público perteneciente al ámbito empresarial, sin embargo a medida que el mercado evolucionaba, el producto se enfocó hacia un público más generalista y a partir del año 2010 con el estreno de la sexta versión, se popularizó entre los usuarios.

En cuanto a la arquitectura del sistema operativo se sabe que está basado en un kernel apoyado sobre la máquina virtual de Java y que está desarrollada en su mayoría en C++. El sistema contiene una capa de aplicaciones en la que pequeños programas desarrollados bajo el API *Java Micro Edition* ofrecían funcionalidades extra. Actualmente BlackBerry OS se encuentra en su séptima versión de desarrollo.

BlackBerry se ha visto desplaza de las primeras posiciones del mercado en favor de las plataformas iOS y Android. Durante el segundo cuatrimestre del 20013, Windows Phone superó por primera vez a BlackBerry en cuota de mercado. Actualmente sólo el 2,9 % de los smartphones vendidos pertenece a esta plataforma.

2.1.2. Apple iOS

La siguiente empresa que se incorporó al mercado de la telefonía móvil de nueva generación fue la californiana Apple. En el año 2007 la compañía presentó el iPhone, un nuevo dispositivo que revolucionó el mercado. Presentaba grandes avances en el hardware destacando la inclusión de una pantalla multitáctil y la posibilidad de conectar el dispositivo a la red mediante red WiFi o red de Celular.

En cuanto al software, la familia de dispositivos iPhone cuenta con el sistema operativo iPhone OS (renombrado como iOS a partir de su cuarta versión) y es desarrollado por la propia Apple. Este sistema está basado principalmente en el kernel Darwin BSD, también basado en OS X, el sistema operativo para escritorio de Apple. iOS se diseñó *ad-hoc* para obtener el máximo rendimiento del iPhone, por lo tanto, se consiguió un producto final altamente optimizado. Esta característica, junto con el alto grado de innovación permitió que el dispositivo se popularizara entre el público y sus ventas se disparasen, particularmente en los Estados Unidos.

En julio de 2008 Apple lanzó App Store, su *market* o tienda de aplicaciones online, que permitía a los usuarios descargar de manera gratuita o comprar aplicaciones móviles que añadían nuevas funciones al dispositivo, popularizando así el concepto de *app* y asentando un precedente que sería básico para las plataformas que surgirían más tarde. Sin embargo, la venta de aplicaciones por parte de Apple está controlada por una política de desarrollo muy restrictiva que sólo permite que se pongan a la venta y se instalen en los dispositivos las aplicaciones que la propia empresa considera adecuadas. Para poder esquivar estas políticas y otras restricciones del sistema operativo, surgió el proceso de *jailbreak*, mediante el cual estas limitaciones podían burlarse, permitiendo finalmente la instalación de aplicaciones no certificadas o la modificación de ciertos aspectos del propio sistema. Aun así, el modelo de distribución de aplicaciones de Apple ha tenido gran éxito y en el mes de mayo de 2013 se alcanzaron los cincuenta mil millones de descargas. La App Store cuenta actualmente con cerca de un millón de aplicaciones disponibles.

Si bien iPhone OS se diseñó inicialmente para ser instalado en los dispositivos iPhone, ha sido adaptado para estar presente en todos los dispositivos del ecosistema Apple, como por ejemplo en su tablet, el iPad, que vio la luz en el año 2010 y que combina las funcionalidades de un iPhone con una mayor pantalla, o en los diferentes modelos de iPod y iPod touch. Actualmente iOS goza de una posición privilegiada en los rankings de ventas y se sitúa en un segundo puesto por detrás de Android con un 13,2% de cuota de mercado.

2.1.3. Windows Phone

Microsoft lanzó su propuesta para dispositivos móviles en el año 2010. Windows Phone nació como un sustituto de la antigua plataforma Windows Mobile, que tuvo un gran éxito en dispositivos PDA durante la primera década de los 2000. A diferencia de BlackBerry y Apple, Windows Phone no se diseñó ad-hoc para ninguna familia de dispositivos en concreto, si no que se buscaba seguir la misma estrategia que el sistema operativo de escritorio Windows, adaptándose al mayor número posible de dispositivos.

El diseño del sistema operativo está claramente enfocado en la compatibilidad con los productos de escritorio de Microsoft, como el sistema Windows o la plataforma de entretenimiento XBOX, buscando así la constitución de un ecosistema similar al construido por Apple. El núcleo del sistema está basado en el kernel Windows NT, utilizado por los sistemas Microsoft Windows hasta la aparición de Windows Vista, y al igual que estos está desarrollado en C/C++. Actualmente el SO se encuentra en su octava versión y se espera el lanzamiento de la revisión 8.1 durante los próximos meses.

El sistema operativo de Microsoft está preparado para proporcionar las funcionalidades típicas de una Tablet o Smartphone, pero además puede proporcionar características propias de un ordenador de escritorio si se cuenta con el hardware adecuado, ya que está diseñado para ejecutarse en procesadores basados tanto en arquitecturas ARM como x86. Cabe destacar la innovación en la interfaz gráfica del

sistema, METRO UI, que se aleja bastante de las interfaces típicas de sus rivales basadas en iconos.

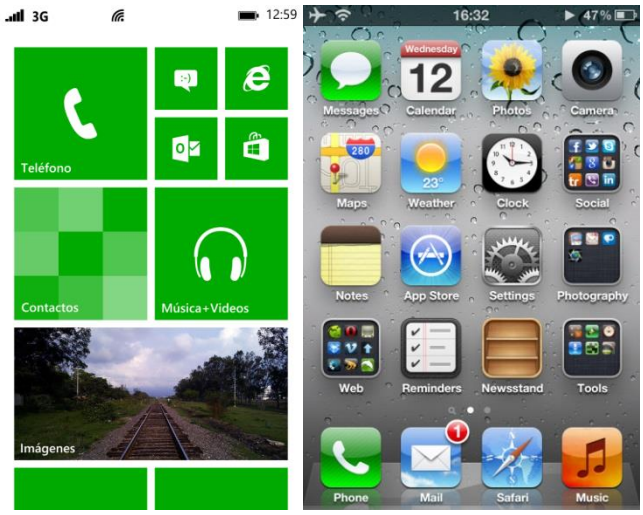


Ilustración 1 - Interfaces de Windows Phone e iOS

Al igual que Apple, Microsoft también cuenta con una tienda online Microsoft Phone Store, en la que se pueden adquirir aplicaciones así como contenido Multimedia. Los desarrolladores que deseen publicar Apps en la tienda, han de contar con una suscripción anual que permite dar de alta un máximo de cien aplicaciones gratuitas. Del mismo modo que en el caso de la App Store, la publicación de aplicaciones y otros contenidos queda sujeta a políticas internas de la compañía, basadas en la naturaleza de los contenidos o la calidad de los mismos.

Windows Phone ha adquirido una gran popularidad en un tiempo relativamente corto y ha llegado a asentarse como la tercera plataforma móvil con más éxito en el mercado por detrás de Apple. Actualmente cuenta con una cuota de mercado del 3,7 %.

2.1.4. Android

Android es sin duda alguna el sistema operativo para dispositivos móviles con más éxito del mercado. Inicialmente su desarrollo estaba al cargo de la empresa californiana Android, que fue adquirida por Google en el año 2005. Android tuvo su presentación oficial en 2007 de mano de la *Open Handset Alliance*, un conglomerado de empresas de telefonía y fabricantes de dispositivos que se dedica al desarrollo de nuevos

estándares abiertos para dispositivos móviles. El primer terminal que se lanzó al mercado con Android como sistema operativo fue el HTC Dream, fabricado por la empresa taiwanesa HTC y que salió a la luz en octubre de 2008. El número de dispositivos que incorporarán Android como sistema operativo crecería exponencialmente durante los próximos años.

El sistema operativo tiene una estructura separada en distintas capas:

La primera de ellas es la capa de aplicación en la que conviven las distintas aplicaciones. El sistema operativo cuenta con una serie de aplicaciones preinstaladas que permiten al usuario asociar el dispositivo a su perfil de Google si este existe. Las aplicaciones más destacables son el cliente de correo Gmail, la aplicación de contactos o la aplicación de mapas, todas ellas apoyadas sobre la tecnología de Google. Las aplicaciones desarrolladas para el sistema operativo están desarrolladas íntegramente en lenguaje Java.

La segunda capa o nivel de abstracción es el marco de aplicaciones o *Application Framework*, este nivel es el responsable de la comunicación entre las distintas aplicaciones así como del arbitraje en el uso de los distintos subsistemas como el gestor de la interfaz gráfica, el servicio de telefonía o las notificaciones. Este arbitraje se lleva a cabo mediante la Propia API de Android, utilizada por los desarrolladores a la hora de escribir nuevas aplicaciones, de tal manera que exista interoperabilidad y compatibilidad entre ellas.

La tercera capa esta compartida por las librerías del sistema y el entorno de ejecución de Android. Las librerías del sistema escritas en C/C++, proporcionan a las capas superiores gran parte de la funcionalidad necesaria para la ejecución de las aplicaciones de la misma manera que las librerías del lenguaje Java lo hacen en un ordenador de escritorio. El entorno de ejecución de Android está conformado por la máquina virtual Dalvik, que es la encargada de ejecutar las Apps en última instancia. Para cada App, se crea un nuevo proceso que se corresponde con una nueva instancia de la máquina virtual, de manera que a este nivel las Apps se ejecuta en modo *"stand-alone"* evitando así que un fallo en la ejecución de una aplicación afecte al resto. La máquina virtual Dalvik es a menudo comparada con la máquina virtual de Java, ya que

ambas interpretan código Java compilado. Sin embargo, a diferencia de la JVM, la máquina Dalvik ejecuta el *bytecode* compilado en formato DEX (Dalvik's Executable) en lugar de utilizar el formato *class* típico de Java. La conversión entre el código de una aplicación compilada como ficheros class y el formato ejecutable de la máquina Dalvik se realiza mediante la herramienta *Dx* incluida en el kit de desarrollo de Android.

Por último, el nivel de abstracción más bajo del sistema operativo está ocupado por el kernel de Linux con ligeras modificaciones. Proporciona al sistema operativo las funcionalidades típicas de su núcleo, como gestión de memoria y entrada/salida, red, drivers, planificación etcétera. Desde la primera versión de Android hasta la cuarta, el kernel utilizado estaba basado en Linux 2.6, momento en el cual se comenzó a utilizar la versión 3.0 del mismo.

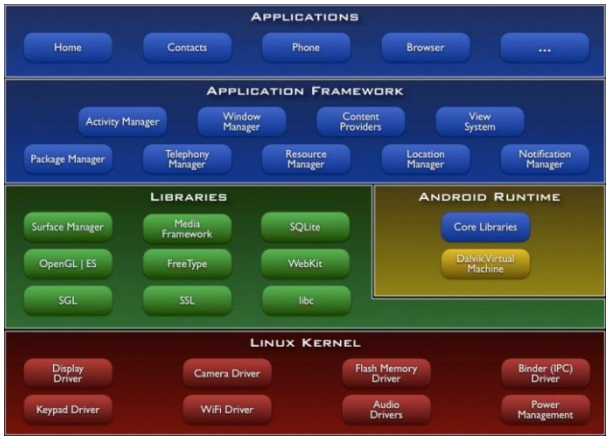


Ilustración 2 - Estructura interna del SO Android [2]

Contar con el kernel de Linux ha facilitado enormemente que Android esté presente en multitud de dispositivos, en gran medida debido al alto grado de compatibilidad que ofrece el uso de este núcleo, permitiendo que Android no sólo se utilice en numerosos modelos de tablets y smartphones, si no que amplíe su presencia a otros dispositivos como sintonizadores de televisión u otros dispositivos empotrados. Además, gracias a la licencia bajo la cual Android está desarrollado, numerosos usuarios han publicado modificaciones sobre las imágenes o ROMS del sistema operativo publicados por los distintos fabricantes. A menudo, estas modificaciones incluyen un kernel más optimizado para ganar más autonomía o gozar de una mejor cobertura, o una capa de interfaz gráfica con distintas características que la original.

Desde que apareció en el mercado, Android ha contado con su propia plataforma de distribución de aplicaciones llamada en un principio Android Market y redenominada en el año 2010 como Google Play, que acabaría por integrar todo un centro de venta de contenido multimedia pensado para ser consumido en los propios dispositivos móviles como por ejemplo libros música o películas. En ella, los desarrolladores pueden publicar sus aplicaciones, las cuales quedan disponibles para su compra o descarga gratuita, de manera similar al resto de tiendas. Como en las demás plataformas, la publicación de aplicaciones se decide en base a una serie de pruebas realizadas por la compañía que evalúan el rendimiento, la seguridad y los contenidos de la misma. Además, Android permite la instalación en los dispositivos de aplicaciones provenientes de fuentes externas a Google sin necesidad de realizar ninguna modificación en el sistema operativo, como el *jailbreak* en caso de iOS. Esta relativa libertad a la hora de desarrollar e instalar aplicaciones junto con la aparición de nuevos *markets* alternativos, ha facilitado el surgimiento de una gran comunidad de desarrollo y ha provocado que el número de aplicaciones publicadas crezca muy rápidamente, sobrepasando las setecientas mil aplicaciones a finales del año 2012. Esto mismo ha dado lugar a que Android se convierta en la plataforma móvil con más malware.

Si Android se ha convertido en el sistema operativo más utilizado es en gran parte debido a que fue diseñado con el objetivo de ser instalado en dispositivos de numerosas marcas y modelos. Esto ha provocado que exista un alto grado de fragmentación en las versiones instaladas del sistema operativo y que únicamente un reducido número de dispositivos cuenten con la última versión del software de Google. Si bien la versión 4.2 fue lanzada en octubre del 2012, únicamente un 8,5% de los terminales que usan Android actualmente cuentan con ella, estando muy extendidas aún, en torno a un 31%, las versiones 2.3.3 y 2.3.7. Sin contar con la versión 3 del sistema que fue especialmente diseñada para su uso en tablets, todas las versiones se han lanzado para smartphones y tablets.

Version	Codename	API	Distribution
2.2	Froyo	8	2.4%
2.3.3 - 2.3.7	Gingerbread	10	30.7%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	21.7%
4.1.x	Jelly Bean	16	36.6%
4.2.x		17	8.5%

Tabla 1 - Fragmentación dentro de la plataforma Android [3]

En estos momentos Android cuenta con una cuota de mercado del 79,3%, siendo el líder indiscutible en cuanto a ventas.

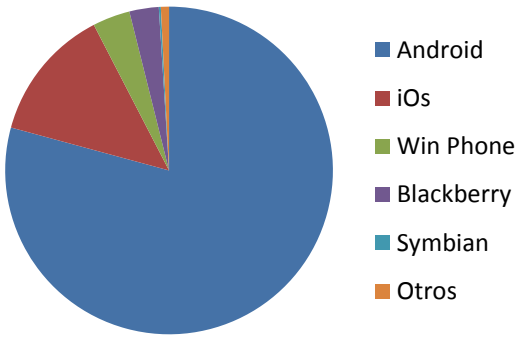


Ilustración 3 - Cuotas de mercado de las distintas plataformas en abril de 2013[4]

Si bien se ha intentado cubrir en esta sección el panorama actual en el terreno de las plataformas móviles más populares, cabe destacar que existen otras alternativas como Symbian, de la empresa finlandesa Nokia, y Bada, de la coreana Samsung. Ambas plataformas, basadas en Java Micro Edition, se vieron radicalmente desplazas con la aparición de las opciones ya repasadas y actualmente se sitúan en los puestos más bajos de los recuentos de ventas.

2.2. Aplicaciones Móviles Orientadas a la Seguridad y el Análisis de Redes

Cuando se habla de aplicaciones de escritorio orientadas a la seguridad informática y la auditoría de redes podemos contar con un extenso catálogo, Encontramos incluso sistemas operativos desarrollados bajo esta motivación como es el caso de Backtrack o Kali Linux. A medida que se popularizó el uso de los dispositivos móviles, las aplicaciones que se desarrollaban a menudo trasladaban al dispositivo móvil funcionalidades típicas de programas de escritorio como procesadores de texto, herramientas de retoque fotográfico o clientes de mensajería. Gracias a que prácticamente la totalidad de dispositivos cuentan con el hardware de red necesario para conectarse a Internet, las herramientas de auditoría y análisis de seguridad en redes fueron ganando presencia en el terreno de las aplicaciones móviles.

La proliferación de este tipo de herramientas tuvo una manifestación más evidente en Android. Los principales motivos son: la licencia de desarrollo bajo la cual se crean las nuevas aplicaciones y que la propia plataforma sea más abierta en lo que se refiere al conocimiento de sus componentes internos. A continuación, se presentan cuatro aplicaciones para la plataforma Android. Nos centraremos especialmente en dSploit, en la que se ha basado este proyecto.

2.2.1. DroidSheep

DroidSheep está pensada con el objetivo de analizar la seguridad en redes inalámbricas y de los sistemas utilizados para la autenticación de usuarios a través de protocolo HTTP por parte de páginas web. Para realizar este análisis, DroidSheep utiliza la técnica denomina *ARP Spoofing* (que repasaremos en las siguientes secciones) para llevar a cabo un ataque de hombre interpuesto entre el dispositivo de la víctima y el punto de acceso que le sirve de acceso a Internet. Esto permite que se capturen las cookies HTTP del objetivo, y se pueda tener acceso a información, como por ejemplo, los identificadores de sesión y otros datos que finalmente permitan una suplantación de la identidad de la víctima en la web a la que pertenece la cookie. DroidSheep está

desarrollado por Andreas Koch bajo licencia GNU y su código se encuentra accesible en los repositorios del proyecto [5].

2.2.2. FaceSniff

Similar a DroidSheep, FaceSniff permite la ejecución de ataques de tipo sesión Hijacking. Si bien inicialmente se desarrolló con el objetivo de atacar únicamente los inicios de sesión en la red social *Facebook*, se ha ampliado su desarrollo y permite capturar sesiones web de numerosas redes sociales y aplicaciones web conocidas como por ejemplo *Twitter*, *Youtube*, *Amazon* o *Tumblr*. La principal novedad que ofrece FaceSniff ante DroidSheep, es que permite que la aplicación opere en redes protegidas mediante los protocolos WEP, WPA o WPA2-PSK. FaceSniff está desarrollado por el ingeniero polaco Bartosz Ponurkiewicz [6].

2.2.3. WifiKill

Desarrollado por el mismo autor que FaceSniff, la herramienta WifiKill cubre distintos objetivos, por un lado permite conocer información sobre los dispositivos conectados a la misma red que el atacante o cuya IP se encuentre dentro de un rango específico, como por ejemplo, el nombre de red de los mismos (NetBios name), el fabricante del dispositivo según la MAC del mismo o las páginas web visitadas por la víctima. Permite además a través de un ataque de hombre interpuesto, descartar todo el tráfico procedente de la víctima con destino a Internet, de tal manera que este queda desconectado de la red [7].

2.2.4. dSploit

La suite dSploit es sin duda una de las herramientas de análisis y *pentesting* más completas actualmente para dispositivos móviles. dSploit permite realizar un mapeo sobre la red a la que estamos conectados con el objetivo de encontrar posibles objetivos contra los que llevar a cabo distintas acciones tales como captura de sesiones, búsqueda

de vulnerabilidades, inyección de código JavaScript en el tráfico HTTP, *ARP spoofing*, manipulación del tráfico HTTP en tiempo real o la inyección de paquetes en la red. Dispone además de herramientas más concretas orientadas para su uso en routers, como por ejemplo, la obtención de pares login/password de administración. dSploit destaca porque aglutina gran cantidad de funcionalidades en unas sola aplicación, lo que permite convertir el dispositivo en una herramienta muy eficaz a la hora de llevar a cabo análisis o tests en las redes inalámbricas a las que podemos conectar el dispositivo. El código de dSploit se encuentra disponible en la página de *GitHub* [9] del proyecto, cuenta con una estructura modular, lo que permite añadir fácilmente nuevas utilidades a la aplicación. Puede encontrarse más información sobre la aplicación en su página Web [8].

Todas las aplicaciones que se han analizado en esta sección, requieren de permisos de *superusuario* o *root* para ejecutarse, o lo que es lo mismo, necesitan que el dispositivo en el que se instalen cuente con acceso libre para el usuario root. Esta característica típica de todos los sistemas basados en Unix, no está disponible en Android por defecto y en un grupo reducido de dispositivos es un escenario imposible de conseguir debido a ciertas restricciones del software o el hardware impuestas por los fabricantes. Por lo tanto, es posible que el uso de estas herramientas sea inviable en un pequeño conjunto de dispositivos. Más detalles acerca de los privilegios de *superusuario* quedan explicados en la subsección 2.3.3.4

2.3. Comunicaciones Móviles

En este punto se cubren las principales características y protocolos típicos de las redes inalámbricas.

2.3.1. El Modelo de Capas y TCP/IP

Antes de entrar a analizar un sistema tan complejo como una red de comunicación, conviene empezar describiendo en profundidad su funcionamiento.

Podemos situar el origen de las redes de comunicación basadas en conmutación de paquetes en la década de los 70, durante la cual se desarrolla ARPANET, que acabaría, tras años de evolución, en convertirse en el actual Internet moderno. Durante los siguientes años, la creación de nuevas redes entró en un periodo de auge, y fueron numerosas las empresas, entidades y centros de investigación los que desarrollaron sus propios estándares y protocolos de intercambio de datos. Sin embargo a medida que las redes aumentaban su tamaño, se hacía más necesaria la capacidad de estas para compartir datos con redes basadas en otros protocolos, propios de otras entidades e incompatibles entre sí en la mayoría de los casos. Comenzó entonces el desarrollo de una normalización oficial que permitiera subsanar esta situación de incompatibilidad creando un estándar o familia de protocolos que normalizara el intercambio de datos.

En 1980 la organización de estándares internacional (ISO) propuso el uso de una estructura común para todas las redes, basada en una descomposición en siete capas o niveles de las distintas etapas que atraviesan los paquetes de datos al ser enviados a través de una red. Se creó así el modelo de referencia o estándar ISO 7498-1, comúnmente conocido como OSI (*Open System Interconection*). Las 7 capas propuestas por el modelo se correspondían con los distintos niveles de abstracción presente en una comunicación, desde la etapa de generación de datos hasta la fase de envió a través del canal. Cada una de las capas cubría una funcionalidad específica del proceso de comunicación.

La primera de estas capas, la capa de aplicación, ofrece a los distintos programas presentes en un sistema conectado acceder a los servicios prestados por la red. Estos servicios son ofrecidos a través de los distintos protocolos que operan en esta capa.

La capa de presentación se encarga de la representación de la información y la compatibilización de los distintos formatos de datos utilizados durante la comunicación, con el objetivo de que la información no pierda su significado aunque los participantes en la conexión cuentas con mecanismos de representación distintos.

La capa de sesión se encarga de mantener y administrar el estado de la comunicación entre dos puntos. En algunos casos la funcionalidad de la capa de sesión es asumida por protocolos de la capa inmediatamente inferior, la capa de transporte.

La capa de transporte se encarga del envió de los datos, así como de la gestión de la comunicación en ciertas ocasiones. La capa de transporte debe realizar su cometido con independencia de los protocolos de la capa inferior que se estén utilizando, con el objetivo de mejorar la interoperabilidad.

La capa de red es la encargada de establecer las rutas entra cada uno de los extremos de la comunicación, en caso de que emisor y receptor no se encuentren conectados directamente, si no que se encuentren conectados a través de numerosos dispositivos de red, el caso más habitual. En esta capa operan distintos algoritmos de descubrimiento de rutas, muchos de ellos con posibilidad de ser optimizados o modificados en base a las características de la red en la que operen.

La capa de enlace se ocupa de la gestión de acceso por parte del sistema conectado al medio físico a través del cual se realiza el intercambio de datos, asume también las tareas de detección de errores así como de control de flujo . En un escenario típico el punto de acceso a una red es compartido por diferentes sistemas que se hallan en una situación de competición por el medio, la misión de los protocolos de la capa de enlace es arbitrar el acceso al mismo así como encaminar los datos a través de los distintos sistemas. Esta capa asume tres funcionalidades bien diferenciadas: la funcionalidad de control del medio físico, el control de acceso al medio y la funcionalidad de control del enlace.

Por último, la capa física cumple la función de gestión del envió de datos convertidos en señales de eléctricas o de radio a través del canal sobre el que está sustentada la red. Además define las características físicas (voltaje, intensidad de la señales eléctricas) y materiales (cables, componentes) de la transmisión de datos.

Gracias a la descomposición en capas, se presentan una serie de requerimientos o etapas que deben estar mínimamente presentes en las redes de comunicación, independientemente de su implementación o de la tecnología subyacente.

Sin embargo, la descomposición en capas no es algo exclusivo del modelo OSI. En los años anteriores a su creación, los investigadores Vinton Cerf y Robert Elliot Kahn del departamento de defensa de los Estados Unidos desarrollaron el modelo de descripción de red TCP/IP definido en el *Request For Coments* número 1122 y que fue implantado efectivamente por primera vez en la red ARPANET. Tanto las redes de área extensa (*Wide Area Networks*) como las redes de área local (*Local Area Networs*) están basadas en la descomposición por capas TCP/IP.

El modelo TCP/IP es similar al modelo OSI y es frecuentemente comparado con él. TCP/IP propone una descomposición en 4 capas jerarquizadas, estando cada una construida sobre su predecesora y utilizando los servicios que esta le brinda, esta descomposición permite que cada capa se encargue de una función específica y sirva su resultado a la capa inmediatamente superior. Las capas propuestas por TCP/IP son las siguientes:

- Capa 4 o capa de aplicación: La capa de aplicación aglutina en un solo nivel de abstracción las funcionalidades de las capas de aplicación, presentación y sesión. Es decir, proporciona un entorno en el que las aplicaciones presentes en el equipo pueden acceder a los servicios proporcionados por la red. Asume también la tarea de asegurar la compatibilidad de formatos y el inicio de la conexión entre extremos. Algunos de los protocolos enmarcados en este nivel pueden ser HTTP, SMTP, FTP, SSH o DNS. Estos protocolos son utilizados por las distintas aplicaciones disponibles en los equipos para prestar un servicio a los usuarios.

- Capa 3 o capa de transporte: La capa tres se asocia con la capa de transporte del modelo OSI. En ella se lleva a cabo la gestión lógica de la comunicación, lo que incluye la apertura y cierre de conexiones bajo de manda de los procesos de la capa superior. La capa tres proporciona a la capa de aplicación un servicio de multiplexación entre ella y la capa de red, la cual simplemente se encarga del enrutado de paquetes, sin importar que proceso los haya generado. Este servicio

se realiza mediante la asignación de una dirección IP y un puerto de origen y una dirección IP y un puerto de destino a cada uno de los paquetes a enviar. Esta información adicional que permite él envió efectivo de información, se añade en la llamada cabecera, generando así lo que se conoce como un segmento de la capa de transporte. Los principales protocolos que operan en este nivel son TCP, que es un protocolo orientado a conexión, es decir, mantiene información acerca del estado de la conexión en todo momento y UDP que no lo es, pero por el contrario es más rápido y menos costoso de utilizar.

- Capa 2 o capa de Red: El nivel de red se encarga del enrutado y el reenvío de los segmentos generados en la capa superior así como del control del estado de la red. En una red típica, los distintos sistemas conectados pueden estar separados por complejas redes formadas por nodos intermedios y en muy pocos escenarios se encuentran conectados directamente entré sí. Esto requiere que los segmentos de la capa de transporte sean guiados por los distintos equipos intermedios mediante un protocolo que gestione las rutas a utilizar, algunos ejemplos de protocolos de enrutamiento son BGP o RIP. En este nivel opera también el protocolo IP, el cual proporciona un servicio *best-effort* de entrega de paquetes así como fragmentación y reensamblado del contenido. Para identificar cada uno de los nodos de una red, IP asigna a cada uno una dirección lógica de 4 bytes, que lo identifica unívocamente dentro de su red. La versión más extendida del protocolo IP es la cuatro (IPv4), estando preparada la implantación de su sexta versión (IPv6) para los próximos años.

- Capa 1 o capa de enlace: La capa uno o capa de enlace es el nivel de la pila TCP/IP que opera a más bajo nivel, agrupa en una sola capa los niveles 1 y 2 del modelo OSI. Sus principales tareas son la gestión de acceso al medio, el control de errores y el control de flujo. Algunos de los protocolos que operan en esta capa son Ethernet, PPPoE utilizado ampliamente por los proveedores de servicios de banda ancha y la especificación 802.11, que estudiaremos en la siguiente sección. En este nivel todos los dispositivos de red quedan identificados mediante una dirección MAC (Media Acces Control) de 6 bytes única para cada dispositivo, que permitirá a las tramas enviadas a través del

medio conocer el dispositivo al que deben llegar. A diferencia de la dirección IP, esta dirección está asociada físicamente a un dispositivo y no de manera lógica. Para permitir la comunicación entre la capa de enlace y la capa de red, se debe utilizar un protocolo que resuelva una dirección MAC o física a partir de una dirección IP o lógica. Esta es la misión del protocolo ARP (*Address Resolution Protocol*) que estudiaremos más adelante.

2.3.2. La Familia de Estándares 802.11

En esta sección pasaremos a analizar y describir el funcionamiento y la evolución de las redes de área local inalámbricas, comúnmente conocidas como WLAN. Este tipo de red goza a día de hoy de una gran difusión y una gran presencia en nuestro alrededor debido fundamentalmente a la facilidad con la que pueden desplegarse en numerosos entornos. A través de estas redes se realiza la gran mayoría del intercambio de datos que se realiza desde nuestro dispositivo móvil, gracias a la velocidad notablemente superior que ofrecen con respecto a las redes celulares típicas como GPRS, UMTS o HSDPA.

802.11 es la familia de estándares y normas que describe la especificación utilizada para el intercambio de datos en redes WLAN (Wireles Local Area Network). A diferencia de las conexiones cableadas, las redes inalámbricas se sirven de ondas de radio para el intercambio de datos a través del medio aéreo. Este cambio de medio conlleva una reformulación de los protocolos de la capa de las capas 6 y 7 del modelo OSI, quedando los demás niveles sin modificar. Esta reformulación se produce fundamentalmente a dos niveles. En primer lugar, a nivel de control de acceso al medio, al que se añaden protocolos de acceso al medio inalámbrico como CSMA y de transporte confiable. En segundo lugar, a nivel físico, se han de añadir funcionalidades de regulación de frecuencia y de codificación de la señal. Los estándares 802.11 operan en la banda de 2.4 a 5 GHz, que queda dividida en 11 canales, el uso de los mismos queda regulado por las normativas de los países en los que se utilice el estándar.

El grupo de trabajo IEEE 802.11 para el desarrollo de las redes WLAN se formó en el año 1997 y desde entonces ha publicado numerosos estándares que han ido evolucionando hasta alcanzar altas velocidades de transferencia de datos y gran fiabilidad. El primer estándar que contó con una amplia aceptación fue 802.11b, aprobado en 1999 y que permitía velocidades de transferencia de hasta 11 Mb/s. En ese mismo año un grupo de empresas del sector de las comunicaciones se asociaron para crear la WECA (*Wireless Ethernet Compatibility Alliance*) actualmente conocida como *Wifi Alliance* con el objetivo de fomentar la compatibilidad de las nuevas redes WiFi con las redes basadas en Ethernet como protocolo del nivel de enlace .La palabra WiFi es el nombre de la marca comercial bajo la cual quedan certificados los equipos compatibles con el conjunto de estándares IEEE 802.11.

Versión del estándar 802.11	Año de publicación	Banda de frecuencias (España)	Velocidad	Estado actual
802.11 a	1999	5 / 3.7 GHz	Hasta a 54 Mb/s	Obsoleto
802.11 b	1999	2.4 GHz	Hasta a 11 Mb/s	En uso
802.11 g	2003	2.4 GHz	Hasta a 54 Mb/s	En uso
802.11 n	2009	2.4 / 5 GHz	Hasta a 150 Mb/s	En uso
802.11 ac	2012	5 GHz	Hasta a 866.7 Mb/s	En uso

Tabla 2 - Estándares de la familia 802.11 más comunes [10]

Las redes inalámbricas de área local ofrecen versatilidad en cuanto a modos de operación se refiere. En primer lugar, podemos considerar una red trabajando en modo infraestructura. En este modo, los clientes (estaciones) se conectan mediante un enlace inalámbrico a un punto de acceso (AP) que puede o no proporcionar conexión con redes externas a la LAN formada por el conjunto de estaciones y punto de acceso. El conjunto de estaciones y punto de acceso conforman lo que se denomina un BSS (*Basic Service Set*). En segundo lugar, las redes bajo la especificación 802.11 pueden operar en modo *ad-hoc*, en este escenario varias estaciones se conectan entre sí con la intención de compartir datos sin contar con la presencia de un punto de acceso, todas las estaciones presentes en la red asumen simultáneamente el papel de punto de acceso y estación.

El escenario más típico de uso y de mayor interés en este caso es el de las redes WLAN con infraestructura. En este tipo de situaciones el intercambio de datos con el punto de acceso es bidireccional y podemos clasificarlo en dos categorías según el tipo de la información intercambiada. La primera categoría es la compuesta por todas las *frames* o tramas enviadas por los equipos conectados con el objetivo de gestionar aspectos de la comunicación tales como la negociación de las velocidades de transferencia, la autenticación de estaciones o la búsqueda de redes disponibles, que puede realizarse de manera activa por parte de las estaciones, emitiendo un tipo especial de trama llamado *probe*, o de manera pasiva por parte de los estaciones, siendo el AP el que emite tramas especiales denominadas *beacons*.

La otra gran categoría está compuesta por las llamadas tramas de datos que transportan el contenido útil de la comunicación además de las distintas cabeceras correspondientes de las capas superiores como la cabecera IP o TCP/UDP.

Durante la comunicación de una estación con un punto de acceso podemos distinguir también dos fases. En primer lugar antes de toda comunicación, se lleva a cabo una fase de asociación. En esta etapa la estación cliente explora los distintos canales en busca de un punto de acceso que se anuncie emitiendo *beacons*. En un momento dado la estación comienza el proceso de asociación mediante el envío de una solicitud, si la solicitud es validada por el punto de acceso (decisión que se toma típicamente en base a la dirección MAC del cliente) la estación y el AP quedan asociados y la estación pasa a formar parte de la red inalámbrica local. La segunda fase consiste en la transferencia efectiva de datos entre cliente y estación basada en los parámetros negociados durante la asociación, el tráfico intercambiado en esta fase se corresponde con los datos generados por la capa de aplicación que se envían a través de la red.

Existen varios aspectos críticos que diferencian las redes inalámbricas de las redes que se apoyan en infraestructura cableada. Quizás el más evidente de ellos sea el uso de un canal abierto como es el aire, más susceptible a interferencia y a ruido que un canal acotado como es cable de cobre. Todo esto hace que la velocidad de transmisión de datos a través del canal aéreo sea sensiblemente inferior comparada con la

transmisión convencional a través de cable y que la latencia o retardo de la comunicación sea mucho mayor. Otro aspecto importante de las redes inalámbricas radica en su relativa inseguridad si las comparamos con las redes cableadas. En una red cableada, el medio a través del cual se envían y reciben los datos pertenece únicamente a los participantes del intercambio, sin embargo cuando se trata de una red inalámbrica cualquiera que cuente con el hardware apropiado, es decir con una tarjeta de red inalámbrica compatible con 802.11 puede escuchar el canal e interceptar el contenido de las comunicaciones. En la siguiente subsección analizaremos en profundidad la seguridad de la familia de estándares 802.11.

2.3.2.1. La seguridad en la especificación 802.11

La normativa 802.11 original tuvo en cuenta la seguridad de la comunicación e incluyó en la especificación el protocolo WEP (*Wired Equivalent Privacy*) con el objetivo de dotar la autenticación y el intercambio de datos de una capa de seguridad a nivel de enlace que asegurara la integridad y la confidencialidad de la información de la misma forma que se garantizan en una red cableada (*Wired Equivalent*). WEP está basado en el cifrador de flujo RC4 y en el uso de claves de una extensión variable entre los 40 y los 104 bits, dicha clave es conocida por la estación y el punto de acceso. Para el control de la integridad, WEP usa código de redundancia cíclica de 32 bits (CRC32).

La autenticación en el protocolo 802.11 bajo WEP constituye un paso previo a la asociación y al envío de paquetes de las capas superiores, puede realizarse de dos maneras. El primer caso contempla una situación sin intercambio de secretos entre estación y AP, básicamente se autentica una estación en base a su dirección MAC, si bien la clave WEP puede ser utilizada para cifrar la información intercambiada. Este método recibe el nombre de autenticación abierta.

El segundo caso está basado en una intercambio reto respuesta de dos fases entre el AP y la estación que desea autenticarse. En primer lugar, el punto de acceso genera un número aleatorio (conocido como reto) y lo envía a la estación que desea autenticarse, en segundo lugar la estación envía de vuelta el reto cifrado mediante RC4 con la clave compartida por AP y estación, el punto de acceso descifra entonces el reto,

si se corresponde con el original, la estación queda autenticada. En caso negativo la autenticación no se produce. Este mecanismo recibe el nombre de autenticación de clave precompartida.

En WEP la confidencialidad se proporciona mediante el cifrado del cuerpo de las tramas de datos, sin incluir los datos de la cabecera. Este cifrado se lleva a cabo usando el algoritmo RC4 y la clave conocida por estación y punto de acceso. El cifrado consiste en realizar una operación XOR entre cada byte de la trama de datos y un byte de la serie cifrante generada por RC4, dicha serie se genera mediante la concatenación de un vector de inicialización (IV) aleatorio de 24 bits a la clave compartida, lo que produce una serie cifrante teóricamente lo suficiente aleatoria como para evitar la recuperación de la clave o del contenido original del paquete.

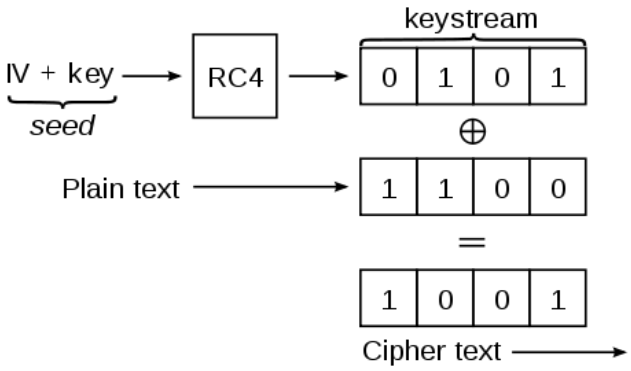


Ilustración 4 - Proceso de cifrado en el protocolo WEP [11]

La integridad en WEP queda cubierta mediante el uso del algoritmo CRC32. Para cada paquete generado se calcula su CRC y se acompaña adjunta a la trama de datos cifrando junto con el resto de la información aviada, de tal manera que en el destino se pueda recalcular este CRC y validar o desecharse según el resultado de la comparación entre ambos códigos.

La seguridad de WEP quedó comprometida seis años después de su publicación, en el año 2001. Los investigadores Scott Fluhrer, Itsik Mantin y Adi Shamir publicaron el artículo "*Weaknesses in the Key Scheduling Algorithm of RC4"* [12], en él se exponen los detalles de una vulnerabilidad de la implementación del algoritmo RC4 utilizada en WEP que permite una recuperación efectiva de la clave utilizada para el cifrado de los

datos y la autenticación de las estaciones. Multitud de publicaciones han demostrado las numerosas vulnerabilidades del protocolo WEP por lo que actualmente el IEEE desaconseja su uso.

En el año 2002 salió a la luz el protocolo WPA (*Wifi Protected Acces*), desarrollado como una solución temporal a las vulnerabilidades de WEP y que implementaba la gran mayoría de normas contenidas en el nuevo estándar 802.11i. Con la especificación WPA se buscaba crear una normativa compatible con el hardware de red preparado para soportar WEP y subsanar las debilidades del protocolo anterior. Al igual que WEP, WPA estaba basado en el cifrador RC4, pero incluía novedades como el protocolo de variación de claves TKIP (*Temporal Key Integrity Protocol*) que establecía un mecanismo de regeneración de claves para WEP, lo cual solucionaba los ataques basados en la repetición de los IV, o una función MIC (*Messsage Integrity Check*) conocida como *Michael* que sustituyó CRC32 como función de control de integridad. WPA aporto también algunas mejoras en el proceso de autenticación, como la posibilidad de hacer uso de autenticación externa mediante el protocolo RADIUS o la modificación del tamaño de los IVs y de la clave. Finalmente en el año 2004 el nuevo estándar 802.11i fue ratificado por completo y la especificación de WPA2 fue lanzada.

La publicación de los protocolos WPA y WPA2 supuso una notable mejora respecto a WEP en la seguridad de las redes inalámbricas de área local, estableciendo dos modos de operación diferenciados, uno especialmente diseñado para su uso en entornos empresariales denominado WPA2- Enterprise y otro diseñado para el entorno doméstico denominado WPA2-Personal o WPA2-PSK. La principal diferencia entre ambos modos, residía en el proceso de autenticación y la distribución de claves, si bien en el modo WPA2-Enterprise la autenticación se lleva a cabo a través de un servidor RADIUS o EAP (*Extensible Authentication Protocol*), en el modo personal se llevaba a cabo mediante una clave precompartida (*pre-shared key*) de modo similar a como se hace en el caso de WEP.

WPA2 incluye el protocolo de cifrado CCMP (*Counter Cipher Mode with Block Chaining-Message-Authentication-Code Protocol*) basado en AES y que mejoraba notablemente la seguridad proporcionada por RC4. Se estableció también un proceso de intercambio de secretos entre la estación y el punto de acceso conocido como *four way*

handshake, que permite el establecimiento de una jerarquía de claves que finalmente da lugar a la creación de la clave de sesión utilizada para el cifrado del tráfico entre estación y AP posterior a la autenticación y que expira cada cierto intervalo de tiempo, un proceso similar es utilizado para generar la clave que se utilizará para cifrar el tráfico multicast.

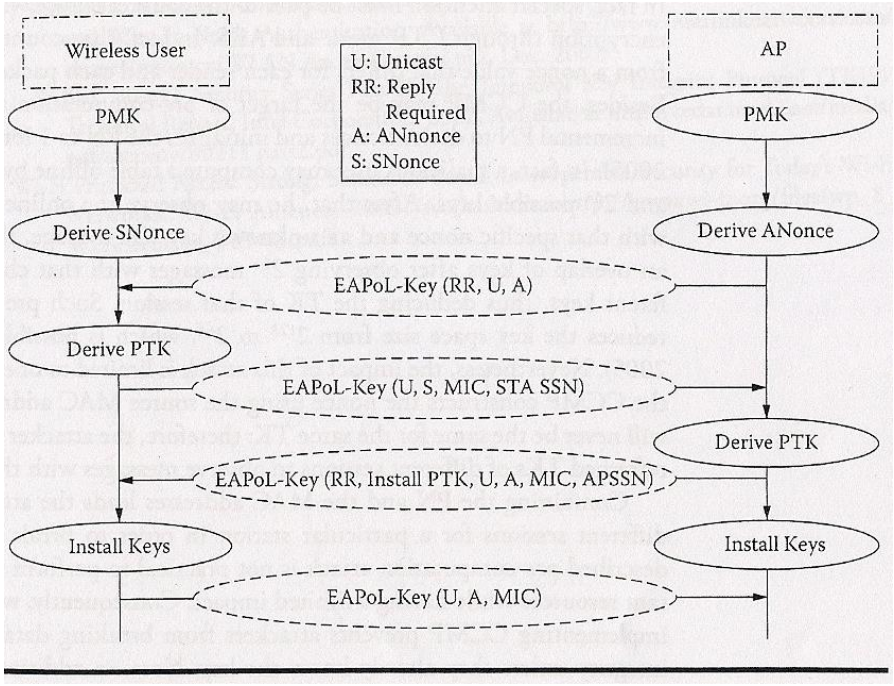


Ilustración 5 - Esquema de negociación de claves en 4 pasos utilizado en WPA2 [13]

Si bien WPA/WPA2 supone un gran avance en la seguridad de redes WLAN en ciertos casos pueden llevarse a cabo ataques de fuerza bruta basados en diccionarios si se cuenta con los paquetes intercambiados durante el *handshake*, sin embargo, el éxito de este tipo de ataques depende en gran medida de la longitud, y de los caracteres que conforman la clave empleada en el punto de acceso.

Protocolo	Lanzamiento	Confidencialidad	Integridad	Autenticación
WEP	1995	RC4 104-40bit	CRC32	PSK
WPA	2001	RC4,TKIP	MIC	PSK, EAP
WPA2	2004	CCMP(AES-CBC),TKIP	MIC,HMAC	PSK, EAP

Tabla 3 - Diferentes algoritmos criptográficos utilizados en los protocolos WEP, WPA y WPA2 [13]

2.3.3. Seguridad en la Capa de Aplicación. Particularidades de la Plataforma Móvil Android.

La capa de aplicación es uno de los niveles de abstracción más críticos en lo que a seguridad se refiere, esto es debido principalmente a que en este nivel operan las distintas aplicaciones que hacen uso de subsistemas críticos como la red, o el almacenamiento. Además la capa de aplicación es la principal entrada de malware a un sistema y es por ello que debe ser vigilada con especial atención, incluso en numerosas ocasiones las aplicaciones que se instalan en un dispositivo pueden no estar diseñadas con la finalidad de causar un perjuicio al usuario, pero debido a que la seguridad no ha sido aplicada a lo largo del proceso de desarrollo, suponen un riesgo para el usuario, sobre todo si hacen uso de información de carácter personal.

Los procesos de actualización de los sistemas operativos juegan un papel fundamental a la hora de prevenir los ataques originados a este nivel, ya que en muchas ocasiones la razón de ser de dichas actualizaciones es solucionar una cierta vulnerabilidad detectada en el sistema.

En la siguiente subsección estudiaremos las particularidades de la seguridad en la capa de aplicación en la plataforma Android, Gran parte de la seguridad del sistema está construida sobre los pilares de la seguridad del kernel de Linux , un sistema que ha sido empleado en gran cantidad de entornos y cuya seguridad ha quedado certificada en numerosas ocasiones. Así por ejemplo se estudiarán mecanismos que se apoyan en los modelos típicos de control de acceso empleados en Unix o el uso de la API criptográfica integrada en el propio núcleo.

2.3.3.1. Process sandboxing

Como se ha comentado en secciones anteriores, las aplicaciones en Android se ejecutan mediante la máquina virtual Dalvik, encargada de interpretar el código *bytecode* generado durante la etapa de desarrollo. En un momento dado, numerosas

aplicaciones pueden ejecutarse en un dispositivo móvil, cada una utilizando un conjunto de datos concretos.

Es tarea del sistema operativo asegurar que los datos que utiliza una determinada aplicación permanezcan visibles únicamente para ella, con el objetivo de impedir que sean accedidos por potenciales aplicaciones maliciosas que convivan en el dispositivo junto con aplicaciones legítimas. Este medida de seguridad esta implementada en Android mediante lo que se conoce como *process isolation* o *process sandboxing*. Cuando una aplicación se inicia, el kernel de Android asigna un nuevo proceso a la instancia de la máquina Dalvik que ejecuta la aplicación, a este proceso se le asigna un UUID (Unix User ID) individual, concreto y único para el proceso. Desde que se produce esta asignación, todos los datos generados y manipulados en la aplicación quedan recluidos a su proceso impidiendo de esta manera que sean accesibles desde fuera del mismo. Este mecanismo es una modificación del modelo de usuarios del kernel de Linux, en el cual las aplicaciones de un determinado usuario se ejecutan bajo un único UUID, en el caso de Android cada aplicación se trata como un usuario al asignarse un UUID distinto para cada una.

A pesar de emplear este modelo de aislamiento entre procesos, el sistema Android contempla la posibilidad de permitir que exista comunicación entre ellos siempre y cuando esta intención se declare explícitamente mediante el uso del esquema de permisos del que se hablará en la siguiente subsección.

2.3.3.2. El esquema de permisos

Además de la técnica de *sandboxing* utilizada en Android, las competencias de cada aplicación así como el acceso a los componentes del sistema operativo y el hardware del dispositivo quedan regulados por un sistema de permisos. En este sistema, una aplicación cuenta con distintos permisos que le autorizan a utilizar servicios o acceder a recursos o incluso a comunicarse con otros procesos. Por ejemplo una aplicación que necesite leer los contactos de agenda o hacer uso de la cámara o la

conexión a Internet del dispositivo, necesitará contar con los permisos adecuados. La asignación de permisos queda en manos de los desarrolladores, que son los encargados de decidir que permisos necesitan las distintas aplicaciones para funcionar correctamente. Los distintos permisos de los que una aplicación necesita hacer uso quedan declarados en el fichero Manifest.xml asociado a la misma, en la sección 5 se explicarán más detalles sobre este componente de la aplicación.

Cuando un usuario procede a la instalación de una nueva aplicación en su dispositivo, debe dar su consentimiento a la lista de permisos que la aplicación necesita, esta autorización ha de hacerse a la totalidad de ellos y no puede hacerse de manera individual, en este sentido o se aceptan todos los permisos o no se instalará la aplicación. En otras plataformas como iOS que también cuentan con un sistema de permisos para el control de acceso de las aplicaciones, la aceptación por parte del usuario se realiza durante la primera ejecución y es persistente, aunque en algunos casos los permisos se solicitan bajo demanda de la aplicación.

El modelo de permisos de Android ha sido frecuentemente explotado por malware de distinta naturaleza. El punto claramente más débil del sistema, es la autorización por parte del usuario, que en un gran número de los casos no repara en el nivel de privilegios que está concediendo a una determinada aplicación, por ejemplo una aplicación que se anuncie como una aplicación multimedia y que requiera acceso al sistema de envío de SMS del sistema, puede estar intentando abusar del modelo de permisos para suscribir al usuario a ciertos servicios sin su consentimiento.

2.3.3.3. Confidencialidad

Desde la versión 3 de Android, el sistema cuenta con mecanismo para asegurar la confidencialidad de los datos almacenados en el dispositivo. Este mecanismo viene implementado mediante el uso de la API criptográfica del kernel de Linux, concretamente a través de la interfaz *dmcrypt*, que cifra todo el sistema de archivos en el que se encuentra los datos de los usuarios.

El cifrado disponible consiste en el empleo del algoritmo AES en modo CBC y claves de 128 bits. La clave que emplea AES se obtiene a partir de la clave que utiliza el usuario para desbloquear el terminal a la que se le aplica un algoritmo de derivación de clave con el objetivo de obtener una clave con niveles más altos de aleatoriedad y entropía, se trata por tanto de un esquema de cifrado que emplea *password bassed encryption*.

Para asegurar la confidencialidad de la clave, esta no se almacena en claro en el dispositivo, si no que se almacena el resultado de aplicar repetidamente un hash SHA1 sobre la concatenación de la clave y un *salt* aleatorio. De esta manera la clave queda protegida ante posibles ataques de recuperación de clave, como los que se basan por ejemplo en el uso de *rainbow tables* o fuerza bruta.

2.3.3.4. El Usuario Root en Android

En los sistemas de la familia UNIX, el usuario root es el usuario que cuenta con acceso a todos los comandos y ficheros del sistema operativo. Es el usuario que se encuentra por encima de los demás jerárquicamente y su función principal es administrar el sistema. En Android, por defecto únicamente un conjunto de aplicaciones próximas al núcleo del sistema ejecutan con privilegios de root.

Sin embargo, en Android puede conseguirse acceso al usuario root realizando ciertas modificaciones al kernel o reinstalando una imagen del sistema operativo que lo permita, esto hace posible que aplicaciones que típicamente operan a nivel de usuario puedan utilizar los permisos del usuario root. Por un lado, esto ofrece mayores posibilidades sobre todo a los desarrolladores con el objetivo de realizar tareas de depuración sobre el sistema operativo, la ejecución de binarios nativos o desarrollar aplicaciones que trabajen en nivel más próximo al kernel. Por otro lado, utilizar el usuario root puede hacer posible que aplicaciones maliciosas lleven a cabo acciones mucho más dañinas contra el dispositivo de las que podrían realizar si no contara con este nivel de privilegios.

En general el uso del usuario root en esta plataforma está desaconsejado para la gran mayoría de usuarios y su uso se reserva para la comunidad de desarrolladores.

2.3.4. Ataques Comunes en WLANS

En esta subsección analizaremos los ataques típicos que pueden llevarse a cabo dentro de una red inalámbrica y las vulnerabilidades que estos aprovechan. Comenzaremos definiendo el concepto de ataque.

Cuando se habla de un ataque, en el ámbito de la seguridad esto se refiere al intento (que puede llegar a ser exitoso o no) de sobrepasar la seguridad de un sistema por parte de un actor interno o externo al mismo, conocido como atacante aprovechando una vulnerabilidad del mismo. El éxito de un ataque depende en gran medida de la gravedad de la debilidad del sistema que permita llevar a cabo el ataque (lo que se conoce como vulnerabilidad) y de la efectividad de las contramedidas desplegadas.

Una vez aclarado este aspecto, podemos pasar a realizar una descripción en profundidad de los ataques típicos de este tipo de redes. En este caso, clasificaremos los distintos tipos de ataque según su actuación, considerando los cuatro siguientes tipos:

- Ataques de interceptación: En este tipo de ataques, el atacante se interpone entre los extremos de la comunicación, recibiendo todo el flujo de datos generado por ambas partes, con la posibilidad de almacenarlo y utilizarlo posteriormente para futuros análisis.

- Ataques de manipulación: En este tipo de ataque, el atacante intenta modificar la información producida por uno de los extremos. En la mayoría de los casos esto se consigue a través de un ataque de interceptación al que se le añade una fase de modificación de la información capturada para posteriormente renviarla al receptor.

- Ataques de interrupción: El objetivo de este tipo de ataques es el de impedir el acceso a un sistema o un servicio por parte de sus usuarios legítimos, esto se consigue habitualmente mediante la saturación de los recursos del objetivo, sometiendo a esto a niveles altos de tráfico hasta que este deja de responder.

A continuación, pasaremos a describir en profundidad cada uno de estos ataques, aportando ejemplos y escenarios de uso de cada uno de ellos.

2.3.4.1. Ataques de Interceptación

Este tipo de ataques, también conocidos como ataques *man in the middle* o ataques de hombre interpuesto, tienen el objetivo de permitir al atacante escuchar la comunicación entre un cliente conectado al punto de acceso sin que ninguna de las partes lo perciba. Este ataque es particularmente fácil de llevar a cabo dentro de una red inalámbrica, fundamentalmente debido a las características del canal, como se ha razonado en la sección anterior referente a la familia de protocolos 802.11.

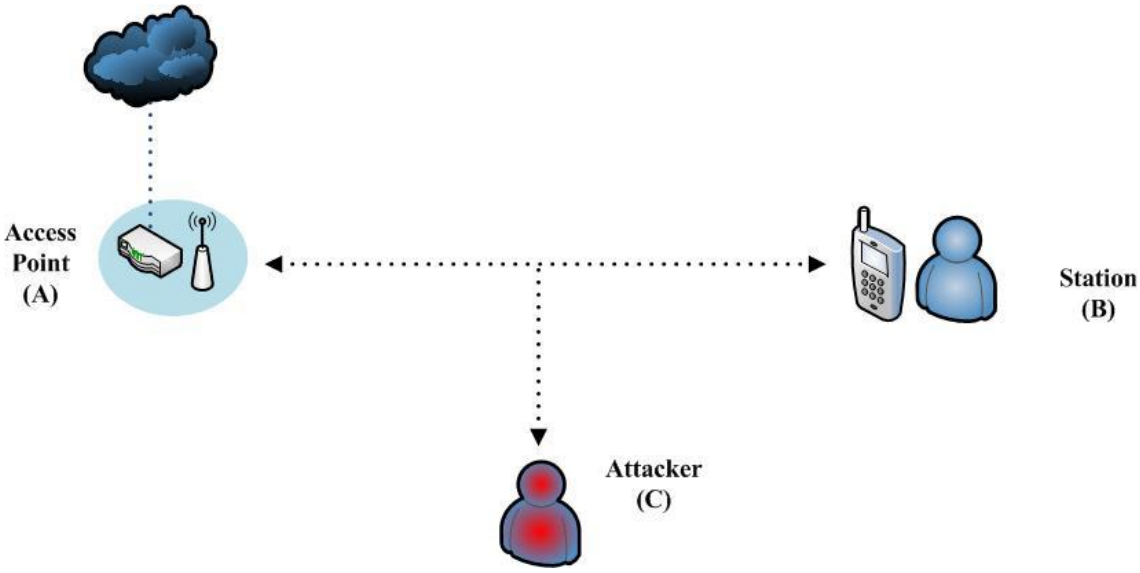


Ilustración 6 - Ataque *man in the middle* típico

Las motivaciones de un atacante a la hora de llevar a cabo un ataque de estas características se corresponde comúnmente con la intención de obtener información

acerca del contenido de los paquetes intercambiados, por lo que si no se cuenta con algún sistema que garantice su confidencialidad, este ataque puede resultar muy provechoso al atacante.

Existen numerosos escenarios que favorecen este ataque, por ejemplo en una WLAN típica en la que no exista ningún mecanismo que regule la autenticación de las estaciones frente al punto de acceso, cualquier usuario malintencionado conseguiría acceso al punto de acceso y podría comenzar a escuchar las comunicaciones que las estaciones cercanas mantengan con él. Sin embargo la información que el atacante conseguiría capturar dependerá directamente de la arquitectura de la red. Esto se debe a que cuando un paquete abandona el equipo de origen, tiene asignada una dirección MAC y una dirección IP de destino, que identifica a nivel de enlace y a nivel de red respectivamente el equipo receptor del paquete en cuestión. Debido a esto, el paquete no pasará por el atacante sino que este solo escuchará el tráfico multicast de la red, es decir el tráfico que se envía a todas las estaciones. Para conseguir capturar todo el tráfico que generan la estación y el punto de acceso, el atacante debe recurrir a la técnica conocida como *ARP Spoofing* o *ARP Poissoning* [14].

El protocolo ARP, es encargado de traducir una dirección IP en una dirección MAC de la capa de enlace. Cuando se va a enviar un paquete a través de la red, el equipo emisor envía un mensaje *broadcast* a la red, que será recibido por todos los equipos de la misma, en el que pregunta que equipo físico tiene asignada la dirección IP de destino, este paquete es denominado *ARP request* y tiene la siguiente estructura:

```
Hardware type: Ethernet (1)
Protocol type: IP (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: AsustekC_42:e0:c0 (60:a4:4c:42:e0:c0)
Sender IP address: 192.168.1.2 (192.168.1.2)
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 192.168.1.1 (192.168.1.1)
```

Ilustración 7 - Consulta ARP

Si en la red existe algún equipo que tenga asignada la IP consultada, este envía un mensaje de respuesta o *ARP reply*, indicando que la dirección IP por la que pregunta

el host emisor está asignada a su tarjeta de red mediante el envío de la dirección MAC de la misma. En caso de no existir ningún equipo dentro de la red con esa IP asignada, no se producirá ninguna respuesta.

```
Hardware type: Ethernet (1)
Protocol type: IP (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: reply (2)
Sender MAC address: Vmware_62:b4:52 (00:0c:29:62:b4:52)
Sender IP address: 192.168.1.1 (192.168.1.1)
Target MAC address: AsustekC_42:e0:c0 (60:a4:4c:42:e0:c0)
Target IP address: 192.168.1.2 (192.168.1.2)
```

Ilustración 8 - Respuesta ARP

Una vez que la dirección IP queda resuelta a una dirección MAC, esta correspondencia se almacena en la caché ARP del dispositivo, que se encarga de almacenar las resoluciones que han sido utilizadas en los últimos minutos, esta caché se refresca cada cierto tiempo.

Según se entiende el proceso de resolución de direcciones IP a direcciones MAC, se puede observar que en ningún momento se comprueba la legitimidad de la respuesta ARP, es decir, un equipo de la red podría enviar información falsa. Además la existencia de una consulta ARP no es requisito obligado para que se genere una respuesta. Estos dos hechos hacen posible que se lleven a cabo el ataque conocido como ARP spoofing o *ARP poissonning*. Este ataque consiste en la emisión por parte de un equipo de la red, de respuestas ARP falsas, con el objetivo de manipular la ruta de los paquetes enviados por la víctima. Veámoslo con un ejemplo:

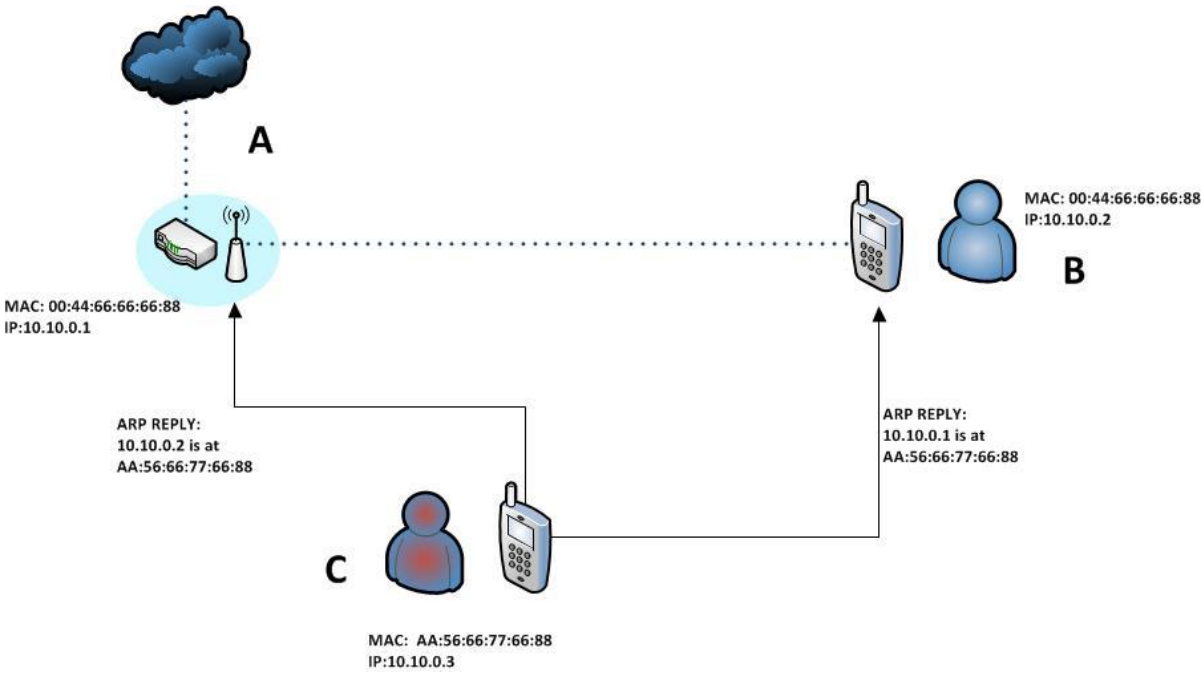


Ilustración 9 - Ataque *ARP Spoof*

En imagen se muestra una red inalámbrica formada por un punto de acceso A, que tiene asignadas la IP 10.10.0.1 y cuya dirección MAC es 00:44:66:66:66:88 y dos estaciones conectadas a él, B y C con direcciones IP 10.10.0.2 y 10.10.0.3 y cuyas direcciones MAC son 00:44:66:66:66:88 y AA:56:66:65:AC:FE respectivamente. En este caso, ambas estaciones se encuentran autenticadas ante el AP. El atacante, mediante la estación C, pretende llevar a cabo un ataque de *ARP spoofing* contra la estación B, para ello comienza a generar masivamente dos mensajes *ARP reply*, uno de ellos dirigido al punto de acceso y el otro dirigido a la víctima. En el mensaje que va dirigido al punto de acceso A, se anuncia que el host con IP 10.10.0.2 (la IP original de la estación B) se corresponde con la dirección MAC de la estación C (la estación atacante). En el mensaje dirigido a C, se anuncia que el host con IP 10.10.0.1, la IP original del punto de acceso, se corresponde con la MAC de la estación atacante. De esta manera, las tablas ARP de ambos dispositivos contienen las entradas falsas que forzarán a que el tráfico que intercambien A y B pase primero por C, que se encargará de activar el renvió de paquetes para que la comunicación siga fluyendo en ambas direcciones.

Mediante este ataque se consigue desviar el tráfico según las necesidades del ataque, una vez que todo el tráfico pasa por el atacante, este puede analizarlo sobre la

marcha o almacenar la captura para un posterior análisis. Este ataque además es relativamente fácil de realizar y es tremendamente eficaz si no se ha implementado algún tipo de contramedida. En la actualidad existen multitud de herramientas que automatizan todo el proceso de envío de *replies* ARP y consiguen realizar un envenenamiento efectivo de las cachés de los dispositivos. Este ataque sirve en algunas ocasiones de para la realización de otras ofensivas más complejas. En el siguiente apartado nos centraremos en los ataques de manipulación que hacen uso de un ataque *man in the middle* como paso intermedio.

2.3.4.2. Ataques de Manipulación

Estos ataques tienen como meta la modificación de los datos intercambiados por parte de la víctima. Esta modificación puede perseguir distintos fines muy diversos, tales como forzar ciertos comportamientos en el equipo de la víctima o alterar la representación de los datos una vez llegan al equipo, constituyen por lo tanto una violación de la integridad de los datos. En general este tipo de ataques pueden ser difíciles de detectar si se realizan bien, y a priori pueden parecer fáciles de evitar mediante el uso por ejemplo de funciones hash o sumas de verificación. Sin embargo en multitud de ocasiones, estas medidas no han sido suficientes para evitarlos, como por ejemplo en el caso de los ataques a la integridad en el protocolo WEP.

Frecuentemente en este tipo de redes, estos ataques suelen apoyarse en un ataque previo del tipo *man in the middle*, logrado mediante la técnica *ARP spoofing*. Uno de los más típicos es el *DNS spoofing* o falsificación DNS, que pasaremos a explicar a continuación.

El protocolo DNS (*Domain Name System*) es el encargado de llevar a cabo la traducción de nombres de dominio, fáciles de recordar para los humanos como *google.com* o *uc3m.es* en direcciones IP de la capa de red, sencillos de manipular para los equipos de red. DNS utiliza principalmente UDP como protocolo de la capa de transporte, principalmente para agilizar el proceso de petición respuesta aunque también permite trabajar sobre TCP. Basa su funcionamiento en una base de datos distribuida a nivel mundial que almacena las correspondencias entre nombres de dominio y

direcciones IP, para el almacenaje de estas correspondencias se utilizan distintos tipos de registros. Algunos de los más importantes son los siguientes:

- Registros A: Almacenan directamente la correspondencia entre un dominio y su dirección o IPv4. Los registros AAAA son similares pero almacenan la dirección IPv6.

- Registros CNAME: Estos registros conocidos también como registros de nombre canónico, almacenan la correspondencia entre un alias, que puede utilizarse para identificar un servicio determinado dentro de un dominio (por ejemplo matriculas.uc3m.es), y un dominio determinado. De esta manera, matriculas.uc3m.es se resolvería al registro CNAME uc3m.es.

- Registros MX: Estos registros almacenan la correspondencia entre un nombre de dominio y un servidor de intercambio de correo electrónico. Pueden definirse numerosos servidores de correo para un solo dominio asignando a cada uno de ellos un nivel de prioridad.

- Registros NS: Los registros NS o *Name Server*, asocian a un determinado dominio los servidores de nombres que utiliza para difundir la información DNS del dominio al resto del sistema.

Cada equipo de una red, ha de contar en su configuración con la dirección IP de un servidor DNS, ya que para cada vez que establece una conexión o envía algún dato a través de la red, necesita resolver una o varias peticiones que son contestadas por el servidor DNS correspondiente mediante una respuesta que contiene uno o más registros. La estructura de un mensaje de consulta DNS, está compuesta de 4 zonas principales. En primer lugar el *Transaction ID* que identifica el diálogo con el servidor DNS, cada par consulta respuesta está identificado con un *Transaction ID* de 2 bytes. Cada consulta cuenta con un conjunto de *flags* o parámetros de la consulta, mediante su uso se especifican opciones como por ejemplo si la consulta se resolverá recursivamente, si el mensaje es una respuesta o el código de operación del mensaje DNS. Se incluye

también un recuento de las consultas, y respuestas que incluye el datagrama, en el caso de un mensaje *DNS request* únicamente se cuantifica el número de consultas que porta el mensaje. Por último la petición DNS cuenta con un listado de los nombres de dominio de los cuales queremos obtener su registro, para cada entrada de este listado contiene el tipo de registro esperado, y las clase del mismo.

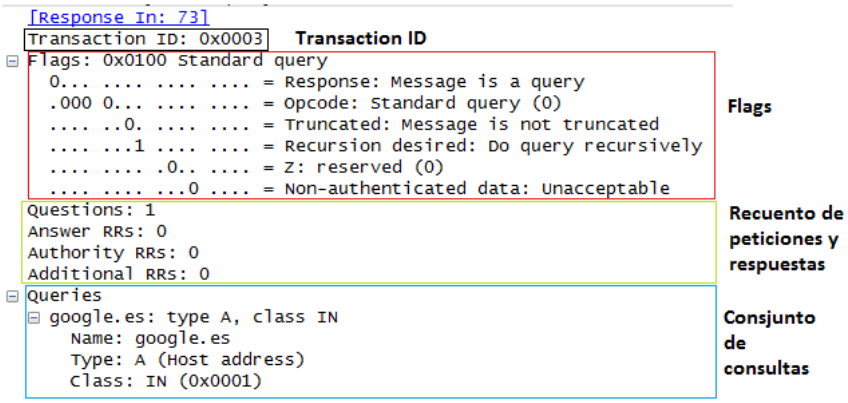


Ilustración 10 - Consulta DNS tipo A del dominio google.es

La imagen superior muestra el datagrama correspondiente a una consulta DNS de tipo A sobre el dominio google.es, como puede verse en ella se distingue claramente las cuatro zonas del mensaje.

El formato de un mensaje DNS de respuesta o DNS response, es similar al formato de la consulta con algunos añadidos. En primer lugar contiene un *Transaction ID*, que debe ser idéntico al que contiene la petición asociada. El conjunto de *flags* de la respuesta incluye las que aparecen en la consulta además de opciones adicionales como la posibilidad del servidor de resolver peticiones recursivamente o el código de respuesta, que se utilizará para indicar estados anómalos del diálogo como posibles errores de formato en la consulta. Al igual que en el mensaje de consulta en este mensaje se incluye el recuento de peticiones y respuestas contenidas en el datagrama, que ahora incluye el número de consultas y de respuestas, también aparece el listado de peticiones tal y como se envía en la consulta. Por último el mensaje incluye una sección adicional en la que se listan las respuestas, cada entrada de este listado se corresponde con un registro del tipo solicitado al servidor.

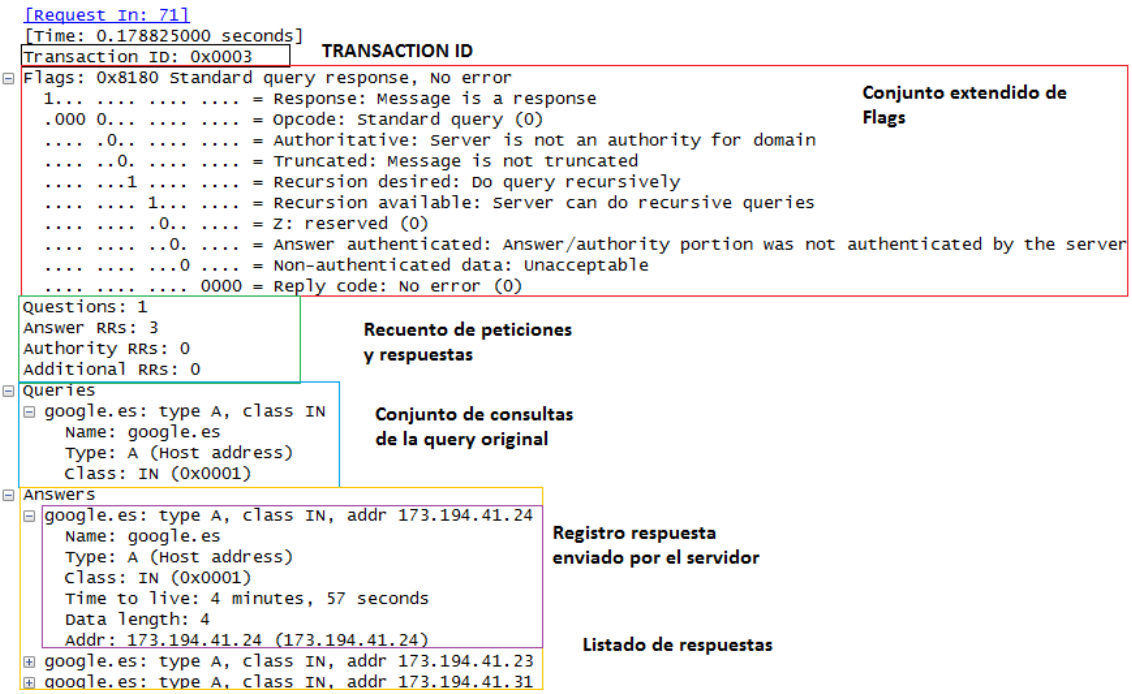


Ilustración 11 - Respuesta DNS tipo A del dominio google.es

Como puede verse, el servidor DNS contesta a la petición anterior con un listado de tres registros tipo A. Una vez que el sistema conoce la IP del dominio google.es puede establecer la conexión con él. El protocolo DNS, es un mecanismo realmente complejo, en esta sección nos hemos limitado a describir brevemente su funcionalidad básica, una explicación más detalla del mismo puede encontrarse en el documento RFC número 1034.

Una vez explicado los aspectos básicos de este protocolo podemos pasar a explicar el ataque en sí. El *DNS Spoofing* consiste en una modificación de los mensajes DNS intercambiados entre cliente y servidor con el objetivo de alterar el resultado de la resolución de un determinado dominio. Esto puede resultar especialmente provechoso para realizar ataques de suplantación de sitios web con el objetivo de llevar a cabo un robo de los credenciales de los clientes. Existen numerosas maneras de llevar a cabo este ataque, alguna de ellas consiste en la inyección de registros falsos en los servidores de nombres, aprovechando vulnerabilidades del protocolo, sin embargo existen maneras más simples de llevar a cabo ataques de este tipo.

En este proyecto se ha implementado un ataque de modificación consistente en alterar el nombre de dominio por el que se pregunta al servidor DNS en un mensaje de petición. Supongamos un escenario en el que un cliente B se conecta a Internet a través de un punto de acceso A, de forma transparente para el punto de acceso y el cliente, un atacante C ha conseguido llevar a cabo un ataque *man in the middle* entre ambos extremos. El cliente B, quiere acceder a la página web de su periódico favorito a través de su navegador, para ello, su dispositivo resuelve el nombre de dominio *miperiodico.com* mediante una petición DNS a los servidores predeterminados en el dispositivo del usuario B. La consulta DNS pasa por el equipo del atacante, que la altera para que esta pregunte al servidor acerca del dominio *periodicodelacompetencia.com* en lugar de preguntar por *miperiodico.com.* Una vez realizada esta modificación, encapsula la petición en un paquete UDP al que asigna como dirección de origen la IP de B y como puerto de origen el puerto que utilizo B para enviarla (ya que será al puerto al que el servidor envíe la respuesta). Una vez que la petición modificada llega al servidor, este la resuelve hacia la IP de *periodicodelacompentencia.com*, haciendo llegar la respuesta a B que finalmente se encuentra visitando una página web que no solicitó.

Este ataque permite realizar de manera efectiva una falsificación de la petición DNS generada por el cliente, sin embargo tiene ciertas limitaciones, como por ejemplo que el dominio por el que se sustituye el dominio original ha de estar registrado, y el servidor de nombres ha de ser capaz de recuperar el registro de tipo A referente a él. Por último, para que este taque sea imperceptible para la víctima, es necesario que al recrear el paquete NDS por parte del atacante, se conserven la IP y el puerto de origen así como la *Transaction ID* asignada a la petición, en caso contrario, el servidor retornará una respuesta con la *fla*g indicadora de error de formato activada.

Es posible llevar a cabo este ataque debido a que dentro del mensaje DNS no se incluye ningún tipo de mecanismo para garantizar la integridad de su contenido como podrían ser el uso de una función hash o una suma de verificación similar a la que si incluye UDP en sus paquetes.

2.3.4.3. Ataques de Interrupción

La finalidad de este tipo de ataques es comprometer la disponibilidad de un servicio o recurso de la red, mediante la destrucción o el impedimento del acceso al mismo. Estos ataques, conocidos como ataques de denegación de servicio o ataques DoS (*Denial of Service*) son especialmente fáciles de realizar ya que pueden tener como objetivo distintas capas de la red. Típicamente, los objetivos principales de estos ataques han sido la capa de transporte y la capa de red, en particular, los protocolos TCP e ICMP.

La forma más típica de ataque de denegación de servicio, es la conocida como inundación TCP SYN. Como se ha explicado en secciones anteriores, TCP es un protocolo orientado a conexión, es decir, necesita que se establezca una conexión lógica entre los extremos de la comunicación como paso previo al intercambio de datos entre ellos. Este establecimiento de conexión, se realiza mediante el proceso conocido como negociación TCP de 3 pasos, en el cual, el equipo que desea iniciar la conexión, envía al equipo destinatario un mensaje TCP con la opción SYN activada, indicando la intención de crear un nuevo vínculo de comunicación. El destinatario ha de responder con un mensaje TCP con las opciones SYN y ACK activas, para comunicar que está dispuesto a aceptar la conexión, en este momento el equipo reserva los recursos necesarios (memoria RAM, ancho de banda, etcétera) que serán necesarios a la hora de mantener el estado de la conexión. Finalmente, el equipo que inicia el diálogo, envía un mensaje TCP con la opción ACK activa y la comunicación queda establecida entre ambos extremos.

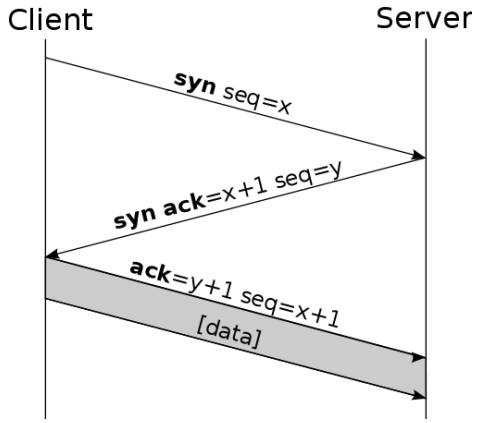


Ilustración 12 - Negociación en tres pasos TCP [15]

Si un atacante comienza a inundar la red de solicitudes de conexión TCP (mensajes SYN), el destinatario comenzará a reservar recursos para cada conexión, hasta que en un momento dado quede saturado y no pueda recibir más conexiones entrantes por parte de cualquier usuario, aun tratándose de un usuario legítimo. Un ataque similar puede ser realizado mediante el abuso del protocolo ICMP, pensado para obtener información del estado de la red, puede ser utilizado para generar masivamente mensajes de tipo *ECHO request* enviados hacia el equipo víctima, que dedicará una pequeña porción de su ancho de banda para contestarlos, hasta que finalmente su conexión quede ocupada por el tráfico ICMP.

En el entorno de una red WLAN, otros tipos de ataque de interrupción pueden llevarse a cabo atacando las capas inferiores de la red. En redes en las que no existe ningún mecanismo de autenticación ante el punto de acceso, cualquier cliente conectado a él puede acceder y generar tramas administrativas, en particular las empleadas en la asociación de nuevos clientes. En un momento dado un atacante autenticado en la red puede generar tráfico en el que suplantando al punto de acceso. Por ejemplo, puede enviar a un usuario legítimo una trama de desautenticación, quedando así la conexión con el punto de acceso finalizada. Este proceso puede repetirse con el objetivo de impedir la asociación entre punto de acceso y estación cliente. Este tipo de ataques pueden impedirse implementando mecanismos de autenticación de los clientes ante en punto de acceso, en este sentido WEP es un protocolo que implemente este mecanismo pero que no resulta efectivo debido a que existen vulnerabilidades que permiten falsificar el proceso de autenticación.

Otro tipo de ataque de denegación que puede llevarse a cabo en entornos inalámbricos es la saturación de la banda de frecuencias utilizada en 802.11 (de 2.4GHZ a 5GHZ) mediante la inyección de ruido en el medio. Si se consigue generar suficiente ruido, el canal quedará inutilizado en lo que se refiere al intercambio de datos de forma inalámbrica. Este ataque es particularmente dañino, ya que es difícil por parte del hardware de red, discriminar las señales legítimas del ruido.

3. Análisis

En este capítulo se abordan las principales tareas llevadas a cabo enmarcado dentro del proceso de análisis. En los de casos de uso se exponen con los distintos pasos que se dan a la hora de utilizar una funcionalidad concreta del sistema. En la especificación de requisitos se recogen las necesidades técnicas que deben quedar cubiertas por el producto desarrollado y las restricciones al diseño.

3.1. Casos de Uso

Un caso de uso queda definido como la ejecución de una funcionalidad concreta del sistema en un cierto escenario. En el caso de uso participan dos entidades bien diferenciadas, en primer lugar el actor que es la entidad que participa o realiza de forma activa la acción. En segundo lugar el sistema que es sobre el que se realiza la acción, por lo tanto el caso de uso describe la interacción entre actor y sistema.

Un actor queda definido por un nombre, el conjunto de casos de uso en los que toma lugar y una descripción de la situación en la que se encuentra el usuario con respecto al sistema. En el caso que nos ocupa, solo se ha definido un solo actor, correspondiente con la figura del atacante:

Nombre	**Atacante**
Casos de Uso	UC-1, UC-2, UC-3, UC-4, UC-5, UC-6, UC-7, UC-8, UC-9.
Descripción	Usuario de la aplicación. Se encuentra conectado a una red de área local inalámbrica e interceptas las comunicaciones entre cualquiera de los demás clientes y el punto de acceso.

Tabla 4 - Actor atacante

A la hora de definir un caso de uso, debemos especificar los actores que participan en él, así como las precondiciones y las postcondiciones esto es, el estado inicial del sistema y el estado del mismo una vez se ha finalizado el caso de uso, una descomposición secuencial de los pasos que el actor realiza durante la interacción con el sistema y la descripción de las posibles situaciones de carácter excepcional que pueden darse durante la realización del caso de uso. Además de estas propiedades típicas en esta especificación cada caso de uso contendrá los siguientes campos:

- Identificador: Cada caso de uso tendrá asociado un ID unívoco. Este identificador estará conformado por el prefijo UC seguido del número que ocupa en la especificación.
- Nombre: Identificador semántico del caso de uso.
- Descripción: Una breve descripción general del caso de uso
- Prioridad: Prioridad del caso de uso dentro del proceso de análisis Se establecen tres niveles: Alta, Media o Baja

A continuación quedan tabulados los principales casos de uso del sistema:

ID	**UC-1**
Nombre	Falsificación de consultas DNS a un determinado servidor.
Descripción	Captura de una petición DNS y sustitución del dominio a traducir por el especificado en el fichero hosts, almacenado en el dispositivo y cuya estructura se define en el requisito no funcional NFR-4.
Prioridad	Alta
Actores	• Atacante
Precondiciones	El dominio a traducir contenido en la consulta se encontrado en la lista de dominios que deben der modificados.
Postcondiciones	El servidor resuelve la petición y envía la respuesta correspondiente al dominio modificado en la petición en lugar del servidor original.
Secuencia de	1. El usuario navega a través de la interfaz de la aplicación hasta el menú

Acciones	de selección de ataque, donde selecciona el módulo DNS Spoof. 2. El usuario comienza la interceptación de consultas DNS 3. Si se intercepta una consulta DNS cuyo campo de consulta coincida con uno de los dominios contenidos en la lista de dominios a modificar, este será sustituido según proceda y la petición será reenviada. 4. El servidor responde al cliente con la dirección IP del dominio falseado.
Excepciones	1. La sustitución del dominio original por el dominio falseado en la consulta DNS no se realiza de forma correcta. 2. El servidor devuelve al cliente una respuesta DNS alertando de un error de formato en la petición.

Tabla 5 - Caso de uso UC-1

ID	**UC-2**
Nombre	Selección de objetivo.
Descripción	Selección del objetivo (víctima) que genera tráfico HTTP/HTTPS que será capturado.
Prioridad	Alta
Actores	• Atacante
Precondiciones	El objetivo genera tráfico HTTP/HTTPS.
Postcondiciones	La captura de los paquetes HTTP/HTTPS originarios del objetivo seleccionado comienza.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. Comienza la búsqueda de dispositivos que se encuentren generando tráfico HTTP o HTTPS. 3. El usuario elige uno de los objetivos que se le muestran en la interfaz.
Excepciones	1. Tras dar comienzo la búsqueda, no se encuentra ningún dispositivo que genere el tipo de tráfico deseado.

Tabla 6 - Caso de uso UC-2

ID	UC-3
Nombre	Captura de cookies enviadas al servidor WEB en una petición HTTP GET
Descripción	El atacante captura una petición HTTP GET que contiene cookies enviada por el objetivo a un determinado servidor Web
Prioridad	Alta
Actores	• Atacante
Precondiciones	El objetivo genera tráfico HTTP/HTTPS a través del cual se produce el envío de cookies por parte del cliente.
Postcondiciones	Las cookies capturadas quedan expuestas al atacante.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. El usuario elige uno de los objetivos encontrados durante la búsqueda. 3. Si el objetivo ha enviado alguna cookie a través de una petición HTTP GET, esta aparecerá en la interfaz.
Excepciones	1. El objetivo no genera tráfico HTTP/HTTPS 2. El objetivo no envía ninguna cookie al servidor Web

Tabla 7 - Caso de uso UC-3

ID	UC-4
Nombre	Lectura de pares clave valor en las cookies enviadas al servidor WEB.
Descripción	El atacante lee las cookies capturadas como resultado del caso de uso anterior.
Prioridad	Alta
Actores	• Atacante
Precondiciones	El objetivo ha realizado una petición HTTP GET contenedora de una cookie y dicha petición se muestra en la interfaz.

Postcondiciones	El atacante conoce los valores de las cookies enviadas.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. El usuario elige uno de los objetivos encontrados durante la búsqueda. 3. Si el objetivo ha enviado alguna cookie a través de una petición HTTP GET, esta aparecerá en la interfaz. 4. El atacante selecciona una de las peticiones mostradas 5. Los pares clave valor se muestran en la interfaz.
Excepciones	6. El Usuario no genera tráfico HTTP o no genera ninguna cookie.

Tabla 8 - Caso de uso UC-4

ID	**UC-5**
Nombre	Exportación de pares clave valor de las cookies capturadas al almacenamiento del dispositivo atacante.
Descripción	El atacante almacena el contenido de las cookies interceptadas en el almacenamiento interno del dispositivo en el que se ejecuta la aplicación.
Prioridad	Alta
Actores	• Atacante
Precondiciones	El objetivo ha realizado una petición HTTP GET contenedora de una cookie y dicha petición se muestra en la interfaz.
Postcondiciones	El contenido de las cookies enviadas se almacena en el dispositivo.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. El usuario elige uno de los objetivos encontrados durante la búsqueda. 3. Si el objetivo ha enviado alguna cookie a través de una petición HTTP GET, esta aparecerá en la interfaz. 4. El atacante selecciona una de las peticiones mostradas 5. El usuario selecciona la opción exportar

Excepciones	1. El objetivo no genera tráfico HTTP/HTTPS. 2. El objetivo no envía ninguna cookie al servidor Web. 3. El almacenamiento externo no está disponible o se ha alcanzado su máxima capacidad.

Tabla 9 - Caso de uso UC-5

ID	**UC-6**
Nombre	Capturar información enviada en petición HTTP al servidor WEB.
Descripción	El atacante intercepta una petición HTTP y tiene acceso a los campos de la misma.
Prioridad	Alta
Actores	• Atacante
Precondiciones	El objetivo ha realizado una petición HTTP a un determinado servidor y esta se muestra en pantalla.
Postcondiciones	El atacante conoce la información enviada en la petición.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. El usuario elige uno de los objetivos encontrados durante la búsqueda. 3. Las peticiones HTTP GET realizadas por el objetivo se muestran en pantalla.
Excepciones	1. El objetivo no genera tráfico HTTP.

Tabla 10 - Caso de uso UC-6

ID	**UC-7**
Nombre	Capturar información enviada en petición HTTPS al servidor WEB.

Descripción	El atacante intercepta una petición HTTPS y tiene acceso a los campos de la misma.
Prioridad	Alta
Actores	• Atacante
Precondiciones	El objetivo ha realizado una petición HTTPS a un determinado servidor y esta se muestra en pantalla.
Postcondiciones	El atacante conoce la información enviada en la petición.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. El usuario elige uno de los objetivos encontrados durante la búsqueda. 3. Las peticiones HTTPS realizadas por el objetivo se muestran en pantalla.
Excepciones	1. El objetivo no genera tráfico HTTPS.

Tabla 11 - Caso de uso UC-7

ID	**UC-8**
Nombre	Creación del fichero PCAP de la sesión.
Descripción	Creación de un fichero de formato pcap que contiene los paquetes capturados en la sesión de extracción.
Prioridad	Alta
Actores	• Atacante
Precondiciones	Existe tráfico HTTP o HTTPS generado por el objetivo en la red local.
Postcondiciones	El fichero de la captura se hace persistir en el almacenamiento externo.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. El usuario elige uno de los objetivos encontrados durante la búsqueda. 3. El Usuario selecciona la opción de almacenar los paquetes capturados

	en la memoria del dispositivo.
Excepciones	1. El objetivo no genera tráfico HTTP/HTTPS. 2. El almacenamiento externo no está disponible o se ha alcanzado su máxima capacidad.

Tabla 12 - Caso de uso UC-8

ID	**UC-9**
Nombre	Obtención del perfil de un determinado usuario de la red.
Descripción	El usuario obtiene los posibles perfiles que cumple el tráfico HTTP generado por el objetivo. Dichos perfiles quedan definidos dentro del archivo profiles.xml, cuya estructura se detalla en el requisito no funcional NFR-3.
Prioridad	Alta
Actores	• Atacante
Precondiciones	El atacante ha de contar con suficiente tráfico generado por el objetivo.
Postcondiciones	El atacante obtiene los perfiles en los que encaja la víctima según el tráfico que ha generado.
Secuencia de Acciones	1. El usuario navega a través de la interfaz hasta el menú de selección de ataque donde escoge el módulo de extracción de información semántica. 2. El usuario elige uno de los objetivos encontrados durante la búsqueda. 3. El usuario aplica los perfiles obtenidos del fichero profiles.xml
Excepciones	

Tabla 13 Caso de uso UC-9

Todos los casos de uso tabulados quedan recogidos en el siguiente diagrama de casos de uso:

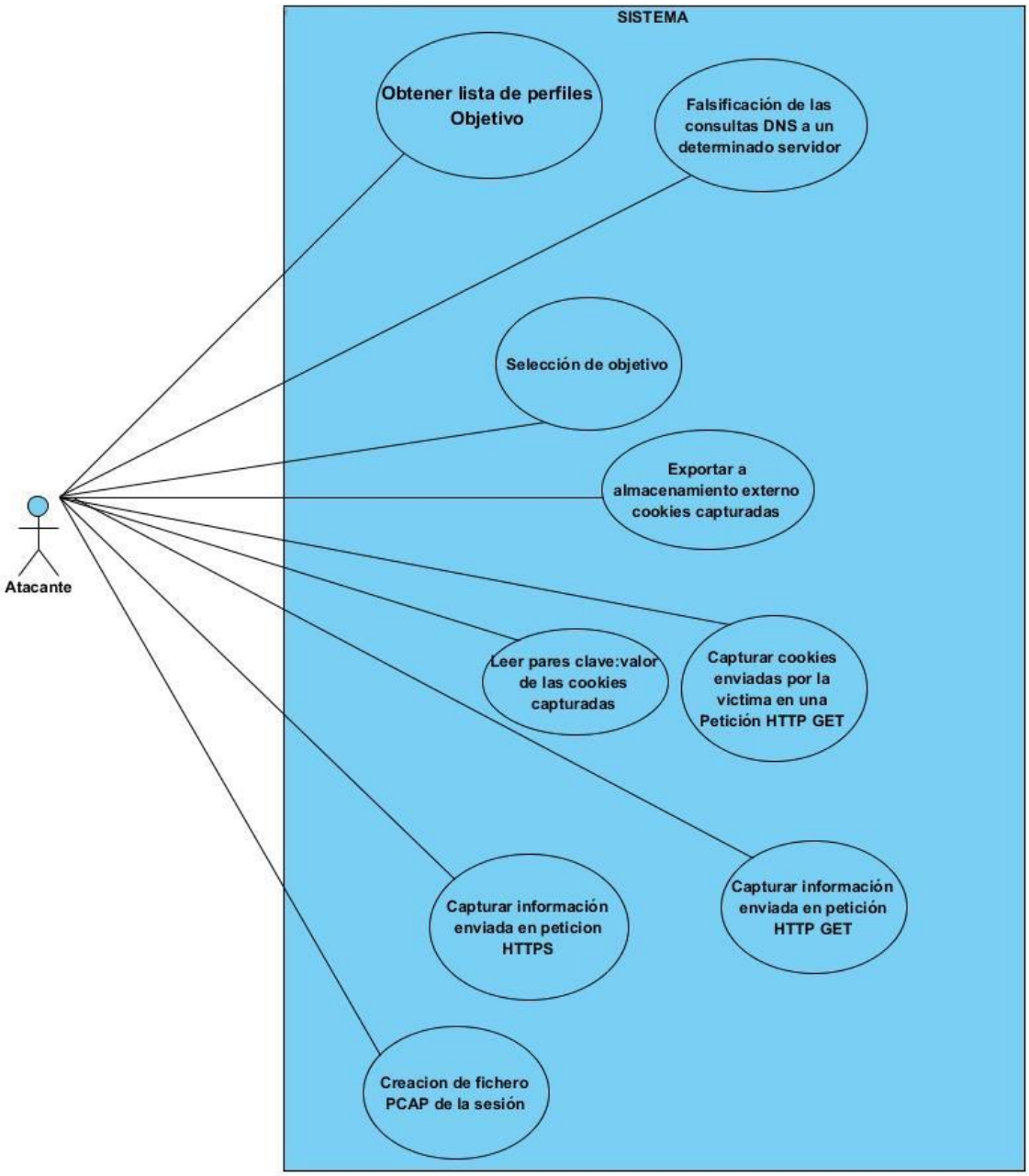


Ilustración 13 - Diagrama de casos de uso

3.2. Especificación de Requisitos

En esta fase del proceso de análisis se consideran tres tipos fundamentales de requisitos, requisitos funcionales, requisitos no funcionales y restricciones. Se entienden como requisitos funcionales aquellos que definen una funcionalidad cubierta por el sistema o alguno de sus componentes. Los requisitos no funcionales especifican

aspectos del sistema que no son apreciables durante la operación del mismo de manera directa tal y como ocurre con los requisitos funcionales, si no que definen una característica no funcional, como por ejemplo el cumplimiento de un estándar o el rendimiento del sistema. Por último las restricciones son reglas que debe cumplir el producto final del desarrollo para considerar que la implementación ha sido correcta. Se considera que una especificación de requisitos está bien formulada si sus requisitos están definidos de manera concisa, atómica y no ambigua.

Cada uno de los requisitos del sistema incluidos en esta especificación contará con los siguientes atributos

- Identificador del requisito**:** Establece un identificador unívoco dentro de la especificación. Los posibles ID seguirán un formato determinado. Según se trate de requisitos funcionales, no funcionales o restricciones, su ID se corresponderá con el patrón **FR-X**, **NFR-Y** o **RR-Z** respectivamente, donde X, Y, Z indican la posición en la que fueron incluidos dentro de la especificación en orden cronológico.

- Tipo: Indica si se trata de un requisito de tipo funcional, no funcional o de una restricción.

- Nombre: Identificador semántico del requisito.

- Prioridad: Indica la prioridad del requisito dentro del desarrollo, se establecen tres niveles: Baja, Media o Alta.

- Descripción: Breve descripción del aspecto cubierto por el requisito.

- Fuente: Origen del requisito acompañado del medio a través del cual se obtuvo.

- Estabilidad: Indica las probabilidades de un requisito de varias durante el proceso de desarrollo, cuanto más alta es su estabilidad menores son las probabilidades de variación.

3.1.1. Requisitos Funcionales

Los requisitos funcionales quedan tabulados a continuación:

ID	FR-1
Tipo	Funcional
Nombre	Extracción de información semántica.
Prioridad	Alta
Descripción	La aplicación permitirá al usuario la obtención de información con significado para el atacante mediante la interceptación del tráfico HTTP generado por la víctima.
Fuente	Cliente
Estabilidad	Alta

Tabla 14 - Requisito funcional FR-1

ID	FR-2
Tipo	Funcional
Nombre	Búsqueda de objetivos.
Prioridad	Alta
Descripción	Como paso previo a la captura, el módulo de extracción permitirá al usuario buscar dispositivos conectados a la red que generen tráfico HTTP o HTTPS.
Fuente	Cliente
Estabilidad	Alta

Tabla 15 - Requisito funcional FR-2

ID	FR-3
Tipo	Funcional
Nombre	Selección de objetivo.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica permitirá al usuario la selección de un objetivo de la lista de objetivos encontrados.
Fuente	Cliente
Estabilidad	Alta

Tabla 16 - Requisito funcional FR-3

ID	FR-4
Tipo	Funcional
Nombre	Obtención de perfiles.

Prioridad	Alta
Descripción	La aplicación permitirá al usuario la clasificación de cada víctima dentro de unos perfiles previamente definidos en el fichero profiles.xml.
Fuente	Cliente
Estabilidad	Alta

Tabla 17 - Requisito funcional FR-4

ID	**FR-5**
Tipo	Funcional
Nombre	Escucha en puerto 80 para el módulo de extracción de información semántica.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica escuchará las comunicaciones que se produzcan en la red cuyo puerto de destino sea el 80 asociado al protocolo HTTP.
Fuente	Cliente
Estabilidad	Alta

Tabla 18 - Requisito funcional FR-5

ID	**FR-6**
Tipo	Funcional
Nombre	Escucha en puerto 443 para el módulo de extracción de información semántica.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica escuchará las comunicaciones que se produzcan en la red cuyo puerto de destino sea el 443 asociado al protocolo HTTPS.
Fuente	Cliente
Estabilidad	Alta

Tabla 18 - Requisito funcional FR-6

ID	**FR-7**
Tipo	Funcional
Nombre	Visualización de las cookies de la petición.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica permitirá al usuario visualizar las cookies HTTP de la última petición GET a un servidor concreto siempre que dicha petición las contenga.
Fuente	Cliente
Estabilidad	Alta

Tabla 19 - Requisito funcional FR-7

ID	FR-8
Tipo	Funcional
Nombre	Obtención del nombre de la red destino utilizando su IP.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica obtendrá mediante whois el nombre de la red asociada a una determinada IP.
Fuente	Cliente
Estabilidad	Alta

Tabla 20 - Requisito funcional FR-8

ID	FR-9
Tipo	Funcional
Nombre	Representación de la información capturada.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica permitirá al usuario visualizar los siguientes datos acerca de una petición HTTP a un servidor concreto: • Hora y fecha de la última petición GET al servidor. • URI de la última petición GET. • Número de peticiones. • *User-Agent* del origen de la petición • Si la última petición contenía o no cookies. • El nombre del servidor así como su dirección IP. • El *favicon* en los casos que proceda.
Fuente	Cliente
Estabilidad	Alta

Tabla 21 - Requisito funcional FR-9

ID	FR-10
Tipo	Funcional
Nombre	Representación de la información capturada.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica permitirá al usuario visualizar los siguientes datos acerca de una petición HTTPS a un servidor concreto: • El nombre del servidor así como su dirección IP • El nombre de la red de destino • El número de peticiones

	• La hora y fecha de la última petición
Fuente	Cliente
Estabilidad	Alta

Tabla 22 - Requisito funcional FR-10

ID	**FR-11**
Tipo	Funcional
Nombre	Visualización de las cookies de la petición.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica permitirá al usuario visualizar las cookies de la última petición GET a un servidor concreto siempre que dicha petición las contenga.
Fuente	Cliente
Estabilidad	Alta

Tabla 23 - Requisito funcional FR-11

ID	**FR-12**
Tipo	Funcional
Nombre	Posibilidad de exportar las cookies.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica permitirá al usuario exportar a un fichero las cookies de la última petición GET a un servidor concreto siempre que dicha petición las contenga.
Fuente	Cliente
Estabilidad	Alta

Tabla 24 - Requisito funcional FR-12

ID	**FR-13**
Tipo	Funcional
Nombre	Escucha en puerto 53 para el módulo *DNS Spoofing*.
Prioridad	Alta
Descripción	El módulo *DNS Spoofing* escuchará las consultas DNS que se envían a través de la red local a la que el dispositivo se conecte.
Fuente	Cliente
Estabilidad	Alta

Tabla 25 - Requisito funcional FR-13

ID	FR-14
Tipo	Funcional
Nombre	Escucha de las consultas DNS tipo A.
Prioridad	Alta
Descripción	El módulo *DNS Spoofing* escuchará las consultas DNS de clase A que se envíen a través de la red local.
Fuente	Cliente
Estabilidad	Alta

Tabla 26 - Requisito funcional FR-14

ID	FR-15
Tipo	Funcional
Nombre	Falsificación de consultas DNS.
Prioridad	Alta
Descripción	El módulo *DNS Spoofing* sustituirá de las peticiones DNS tipo A capturadas el dominio consultado siempre que este aparezca en el fichero hosts.
Fuente	Cliente
Estabilidad	Alta

Tabla 27 - Requisito funcional RF-15

ID	FR-16
Tipo	Funcional
Nombre	Reenvío de consultas DNS.
Prioridad	Alta
Descripción	El módulo *DNS Spoofing* reenviará las consultas DNS manipuladas al servidor DNS de destino desde el puerto original.
Fuente	Cliente
Estabilidad	Alta

Tabla 28 - Requisito funcional FR-16

3.1.2. Requisitos no Funcionales

Los requisitos no funcionales quedan tabulados a continuación:

ID	NFR-1
Tipo	No Funcional - Interfaz de Usuario
Nombre	Presentación del *favicon* del servidor al que se dirige las peticiones.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica mostrará el *favicon* del servidor destino siempre que dicho servidor pertenezca a una de las siguientes organizaciones: • Facebook • Twitter • Google • Amazon • Universidad Carlos III de Madrid
Fuente	Cliente
Estabilidad	Alta

Tabla 29 - Requisito no funcional NFR-1

ID	NFR-2
Tipo	No Funcional - Interfaz de Usuario
Nombre	Representación de peticiones HTTPS.
Prioridad	Alta
Descripción	La interfaz a la hora de mostrar una petición HTTPS variará para indicar la diferencia con una petición HTTP. Las variaciones consistirán en alterar el *favicon* mostrado y un cambio en el color de la interfaz.
Fuente	Cliente
Estabilidad	Alta

Tabla 30 - Requisito no funcional NFR-2

ID	NFR-3
Tipo	No Funcional - Definición de datos
Nombre	Estructura fichero profiles.xml.
Prioridad	Alta
Descripción	El fichero profiles.xml contiene la definición de los perfiles de usuario con los que se intenta encontrar una equivalencia. Su estructura está compuesta por una serie de nodos

	XML: El nodo principal <profiles> que contiene al resto y una serie de nodos <profile> por cada uno de los perfiles que contenga el fichero. Los nodos <profile> estarán formados por una etiqueta <name>, la etiqueta opcional <allow-or> y una serie de nodos <rule>, que contendrán la definición de las reglas incluidas en el perfil. Dentro del nodo <rule>, se incluirán la etiquetas <host> y <number>.
Fuente	Cliente
Estabilidad	Alta

Tabla 31 - Requisito no funcional NFR-3

ID	**NFR-4**
Tipo	No Funcional - Definición de datos
Nombre	Estructura del fichero hosts.
Prioridad	Alta
Descripción	El fichero hosts tendrá una estructura bien definida, formada por entradas que contienen el primer lugar el dominio a sustituir y en segundo la representación hexadecimal del dominio sustituto, separados ambos por un espacio en blanco.
Fuente	Cliente
Estabilidad	Alta

Tabla 32 - Requisito no funcional NFR-4

ID	**NFR-5**
Tipo	No Funcional –Lectura de datos
Nombre	Lectura del archivo profiles.xml.
Prioridad	Alta
Descripción	El módulo de extracción de información semántica leerá el fichero profiles.xml, cuya estructura queda definida en el requisito no funcional NFR-3 en busca de definiciones de nuevos perfiles cada vez que el módulo sea iniciado.
Fuente	Cliente
Estabilidad	Alta

Tabla 33 - Requisito no funcional NFR-5

ID	NFR-6
Tipo	No Funcional –Lectura de datos
Nombre	Lectura del archivo host.
Prioridad	Alta
Descripción	El módulo *DNS Spoofing* analizará el fichero cuya estructura queda definida en el requisito no funcional NFR-4, en busca de los host que debe sustituir en las consultas DNS capturadas.
Fuente	Cliente
Estabilidad	Alta

Tabla 34 - Requisito no funcional NFR-6

3.1.3. Restricciones

Las restricciones del proyecto quedan tabuladas a continuación:

ID	RR-1
Tipo	Restricción
Nombre	Idioma.
Prioridad	Alta
Descripción	La información, los botones y otras etiquetas mostradas en el dispositivo durante la ejecución de los nuevos módulos han de estar escritos íntegramente en inglés.
Fuente	Cliente
Estabilidad	Alta

Tabla 35 - Restricción RR-1

ID	RR-2
Tipo	Restricción
Nombre	Lenguaje.
Prioridad	Alta
Descripción	El lenguaje utilizado en la implementación será Java.
Fuente	Plataforma
Estabilidad	Alta

Tabla 36 - Restricción RR-2

ID	RR-3
Tipo	Restricción
Nombre	Versión.
Prioridad	Alta
Descripción	La configuración de versiones de la aplicación es la siguiente: • Mínima versión del SO: 2.2 • Compilación para versión del SO: 2.2 • Compilado con versión SDK: 4.0.3
Fuente	Cliente
Estabilidad	Alta

Tabla 37 - Restricción RR-3

ID	RR-4
Tipo	Restricción
Nombre	Cota de utilización de CPU.
Prioridad	Alta
Descripción	Los nuevos Módulos Desarrollados no deben provocar que la carga de CPU asociada a la ejecución de dSploit supere el 10%.
Fuente	Cliente
Estabilidad	Alta

Tabla 38 - Restricción RR-4

4. Diseño

En este capítulo se discutirán las principales decisiones de diseño tomadas previamente al comienzo del proceso de desarrollo, para ello nos apoyaremos en diferentes diagramas realizados bajo el modelo UML (*Unified Modeling Language*). Antes de abordar en profundidad en las cuestiones relativas al diseño, repasaremos las funcionalidades concretas de los módulos a desarrollar:

- Módulo de extracción de información semántica: Este módulo tiene por objetivo dotar a dSploit de una funcionalidad que permita obtener información tanto del dispositivo que utiliza como de las páginas web que el objetivo está visitando. Para ello se capturarán las cabeceras de las peticiones GET realizadas por el objetivo. En este módulo se incluirá también una funcionalidad de creación de perfiles de usuario, que permitirán clasificar a los usuarios según el tipo de webs que visiten.

- Módulo de falsificación DNS: Este módulo tiene por objetivo la modificación de las consultas DNS enviadas por los usuarios de la red. Para ello se lleva a cabo un ataque *man in the middle* y se modifican las peticiones DNS que sean de tipo A y consulten un nombre de dominio incluido en un listado gestionado por la aplicación, para que resuelvan otro nombre de dominio distinto.

Ambos módulos se desarrollarán como nuevos *plugins* de la aplicación, cuya estructura favorece este tipo de implantación ya que está altamente modularizada. Otro aspecto a repasar antes de comenzar a hablar de la arquitectura, es el conjunto de dependencias o paquetes adicionales que necesita la aplicación para funcionar.

En primer lugar la aplicación necesita contar con la aplicación BusyBox instalada en el dispositivo. BusyBox es un conjunto de herramientas como las que podemos encontrar en un sistema Unix típico, pero compiladas para ser utilizadas en sistemas empotrados. Estas herramientas permiten utilizar los binarios de algunos

comandos como *su, iptables, netstat,* etcétera, de la misma forma que en un ordenador convencional pero desde el dispositivo móvil. dSploit hace uso de ellas para gestionar distintos aspectos del estado del teléfono, tales como el acceso al usuario *root* con *su* o la modificación del firewall con *iptables.*

En segundo lugar, la aplicación necesita también la librería Sherlock ActionBar. Esta librería está pensada para facilitar el uso del componente ActionBar de la interfaz de Android. Es necesario referenciar esta librería en tiempo de compilación como un proyecto externo a la propia aplicación dSploit.

4.1. Arquitectura del Sistema

Como se ha mencionado anteriormente, la aplicación tiene una estructura fundamentalmente modular, esto quiere decir que está compuesta de varios componentes que realizan tareas concretas y que proporcionan interfaces a otros componentes. Este patrón de diseño, favorece la reutilización y la implantación de nuevos módulos sin la necesidad de realizar grandes modificaciones sobre el código base. Los distintos módulos que conforman dSploit quedan listados a continuación, acompañados de la descripción de su funcionalidad básica.

- **Archivos binarios:** Estos son los archivos ejecutables y las librerías nativas contenidas en dSploit así como las que proporciona Busybox. Estos archivos realizan tareas directamente sobre la interfaz de comandos del sistema operativo y proporcionan datos de entrada a los niveles superiores de la aplicación.

- **Módulo *core:*** Este módulo contiene la lógica principal de la aplicación, en ella están contenidos cuatro componentes de gran importancia. El primero de ellos es El gestor de actualizaciones, que se encarga de comprobar si existen actualizaciones disponibles, su descarga y su instalación. El segundo de estos componentes es el módulo Plugin, en él se definen los atributos y métodos que han de tener todos los *plugins* de la aplicación, a nivel de clases los nuevos *plugins* que implementen heredarán de esta clase. Otro componente importante es el módulo Shell, que se encarga de gestionar las operaciones de entrada salida

con las herramientas que se ejecutan en la línea de comandos del sistema. Por último, el módulo Tools se encarga de proporcionar las interfaces para la ejecución de los binarios por parte de los demás subsistemas.

- **Módulos Net y Wifi:** El módulo Wifi, es el contenedor de la funcionalidad relativa a los ataques a puntos de acceso inalámbricos que se incluyen en dSploit. El módulo Net, contiene la lógica que abstrae elementos de una red como una estación, un punto de acceso o un router. Además contiene un pequeño servidor HTTP y un proxy que son utilizados por los *plugins* que llevan a cabo ataques de *hijacking*.

- **Plugins:** El módulo responsable de contener la funcionalidad de los distintos *plugins* de la aplicación, dentro de este módulo conviven las distintas extensiones de la aplicación que hacen uso de los recursos de niveles inferiores. Algunos ejemplos de componentes de esta capa son el buscador de *exploits*, el inyector de paquetes o los nuevos módulos añadidos.

- **Módulo Gui:** Este módulo es el encargado de la interfaz de usuario de la aplicación y de ofrecer acceso a las herramientas que contiene. Utiliza componentes de la librería Sherlock para permitir la navegación entre las distintas actividades mediante el componente ActionBar.

El siguiente gráfico muestra al detalle los distintos componentes y las relaciones entre ellos:

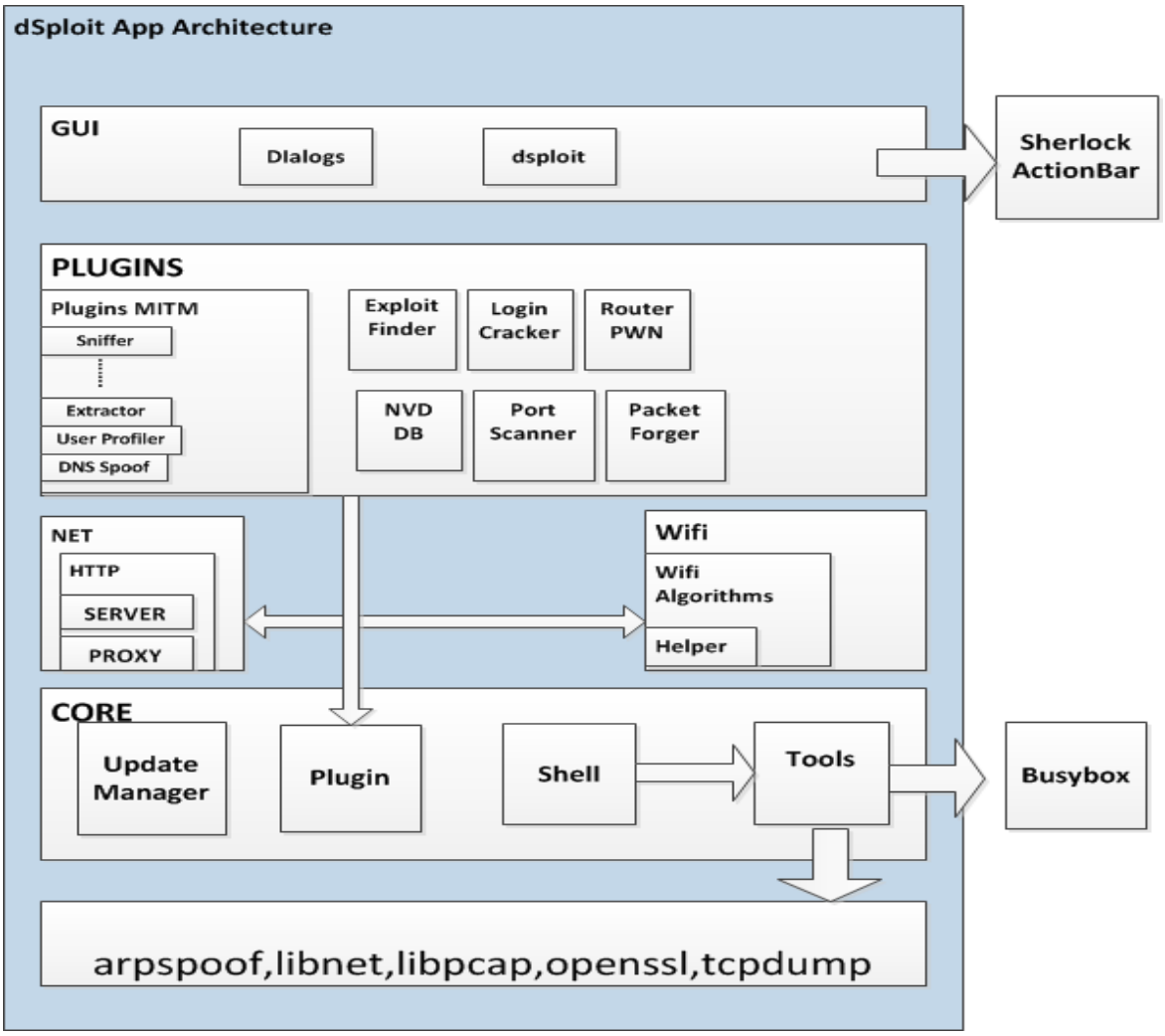


Ilustración 14 - Estructura de componentes de dSploit

4.2. Diagrama de Componentes

El diagrama de componentes es un elemento de la especificación UML que representa las dependencias entre los diferentes componentes de un sistema, como interactúan entre si y los datos que comparten. La finalidad última es ilustrar la estructura de una entidad compleja mediante su descomposición en componentes más sencillos.

En este caso se han definidos dos diagramas de componentes, uno para cada módulo desarrollado. El primero de ellos es el diagrama de componentes del módulo de

DNS spoofing, en él puede observarse la dependencia del componente *Tools* con el módulo de *DNS spoofing* a través de las herramientas *tcpdump* y *arpspoof*.

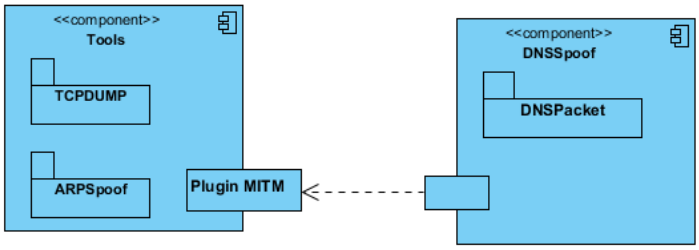


Ilustración 15 - Diagrama de componentes del módulo de DNS spoofing

El objeto DNSPacket en el módulo DNS spoof, representa la abstracción del paquete de consulta DNS que se manipula durante la ejecucion de la funcionalidad de este módulo.

El segundo diagrama representa la estructura en componentes del módulo de extracción de información semántica. En el aparecen 4 componentes distintos. En primer lugar, el componente *Tools*, del que depende el módulo de selección de objetivos o *Target Chooser*. Este módulo contiene un objeto *EndPointTarget* que abstrae las caracteristicas del objetivo de la extracción. En siguinete componente, *Extactor*, mantiene una dependencia con *Target Choose*r a través de el objeto *EndPointTarget*. Este componente se encarga de capturar la información generada por el objetivo y representarla en objetos *BrowserRequest.* Por último, el componente *User Profiler* que mantiene una dependencia con el componente *Extractor* mediante el objeto BrowserRequest, se encarga de la obtención de perfiles de usuario en base a la información extraída.

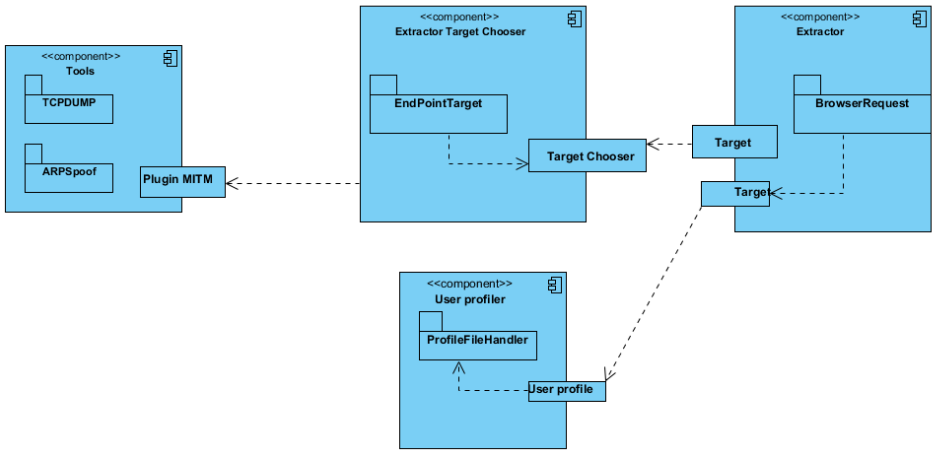


Ilustración 16 - Diagrama de componentes del módulo de extracción de información semántica

4.3. Modelo de Datos

El modelo de datos define las distintas estructuras presentes en el diseño y como estas interactúan entre sí. En este diseño se han creado dos categorías en las que clasificaremos las estructuras de datos que se han utilizado, estructuras de datos residentes en memoria y estructuras de datos persistentes en almacenamiento principal.

Las estructuras de datos en memoria son principalmente los objetos involucrados en el diseño. En este caso se consideran los siguientes elementos:

- **Objeto EndPointTarget:** Almacena información relativa al objetivo cuya comunicación va a ser escuchada. Los datos que almacenan son la dirección IP y la dirección MAC del objetivo.

- **Objeto BrowserRequest:** Contiene información relativa a la petición realiza por el navegador del objetivo. La información que contiene varía según se trate de una petición HTTP o HTTPS, pero en ambos casos almacena la hora, el número de peticiones que se han producido y la dirección IP del destino.

- **Objetos Rule y profile**: Son los objetos que almacenan la abstracción del contenido del fichero profiles.xml. *Rule* contiene dos campos, el

nombre del dominio y el número de veces que debe ser visitado. *Profile* contiene una lista de *rules* y una variable booleana que controla el modo de validación del perfil.

- **Objeto DNSPacket:** Contiene la representación hexadecimal de un paquete de consulta DNS generado por el objetivo.

- **Objeto hostFileEntry:** Similar a los objetos *Rule* y *Profile*, almacena el contenido de una entrada del fichero hosts.

Los ficheros persistentes en almacenamiento principal son las estructuras de datos contenidas en archivos, que se almacenan en la memoria no volátil del dispositivo. En este caso se consideran estos dos elementos:

- **Archivo hosts:** Este archivo contiene una lista de reglas que especifican que nombres de dominio serán sustituidos y los dominios por los que se reemplazarán en las consulta DNS. El formato de cada regla o cada entrada es sencillo; en la primera columna se especifican los nombres de dominio que de ser encontrados en una consulta DNS interceptada, serán sustituidos por el contenido de la segunda columna que contiene la representación hexadecimal del valor del campo *Name* contenido en una consulta DNS tipo A hacia el dominio por el que se quiere sustituir el original. Un ejemplo de este fichero puede ser el siguiente:

```
#Domain             #name field of type A fake DNS query (Hexadecimal)
yahoo.es            0377777706676f6f676c6502657300
uc3m.es             0377777706676f6f676c6502657300
```

Ilustración 17 - Estructura del fichero hosts

En este caso, los dominios yahoo.es y uc3m.es serán sustituidos por la representación hexadecimal de google.es, que se corresponde con la cadena hexadecimal 0x0377777706676f6f676c6502657300.

- **Archivo profiles.xml:** Este archivo almacena en una estructura XML información acerca de los perfiles de usuario que quieren validarse. Está compuesto por nodos <Profile>, que a su vez pueden contener nodos <rule> tantas veces como reglas tenga el perfil, un nodo <name> que indica el nombre del perfil y un nodo <AllowOr>, que indica que la validación de una regla bastará para validar el perfil completo. Un ejemplo de este fichero puede ser el siguiente:

```
<profiles>
    <profile>
        <name>Google_User</name>
        <allowOr></allowOr>
        <rule>
            <host>www.google.es</host>
            <number>1</number>
        </rule>
        <rule>
            <host>google.es</host>
            <number>1</number>
        </rule>
    </profile>
</profiles>
```

Ilustración 18 - Estructura del fichero Profiles.xml

4.4. Diagrama de Clases

Los diagramas de clase son los elementos de la especificación UML dedicados a mostrar los atributos de cada clase así como la manera en que estas pueden interactuar con otras clases, es decir el conjunto de métodos del que disponen. En este diagrama también pueden observarse las distintas relaciones de agregación y /o composición que existen entre clases.

Para ilustrar el diseño de clases correspondiente a los nuevos módulos desarrollados, se han generado tres diagramas de clase. Los dos primeros se

corresponden con la funcionalidad de extracción de información semántica, el tercero ilustra el diseño del subsistema de obtención de perfiles de usuario, y por último, el cuarto diagrama se corresponde con la funcionalidad de DNS Spoofing.

Este primer diagrama ilustra la descomposición en clases del subsistema de selección de objetivos, contenido en el módulo de extracción de información semántica. Como puede observarse cuenta con un total de cuatro clases, *EndPointTarget*, *EndPointTargetAdapter*, *ElementHolder* y *ExtractorTargetChooser*. La clase *EndPointTarget*, de la que ya se ha hablado en el modelo de datos, es utilizada para almacenar la información sobre un objetivo en concreto. Las clases *EndPointTargetAdapter* y *ElementHolder*, se emplean en Android para mostrar en una lista la información contenida en un objeto (en este caso, el objeto *EndPointTarget*). Mediante los métodos de esta clase se añaden nuevas entradas a la lista, se obtienen referencias directas a las entradas de la misma o se obtiene el recuento de elementos que contiene. Por último, la clase *ExtractorTargetChooser*, contiene los objetos que se representan en la interfaz gráfica, además de referencias a instancias de las demás clases del subsistema. Su diagrama UML es el siguiente:

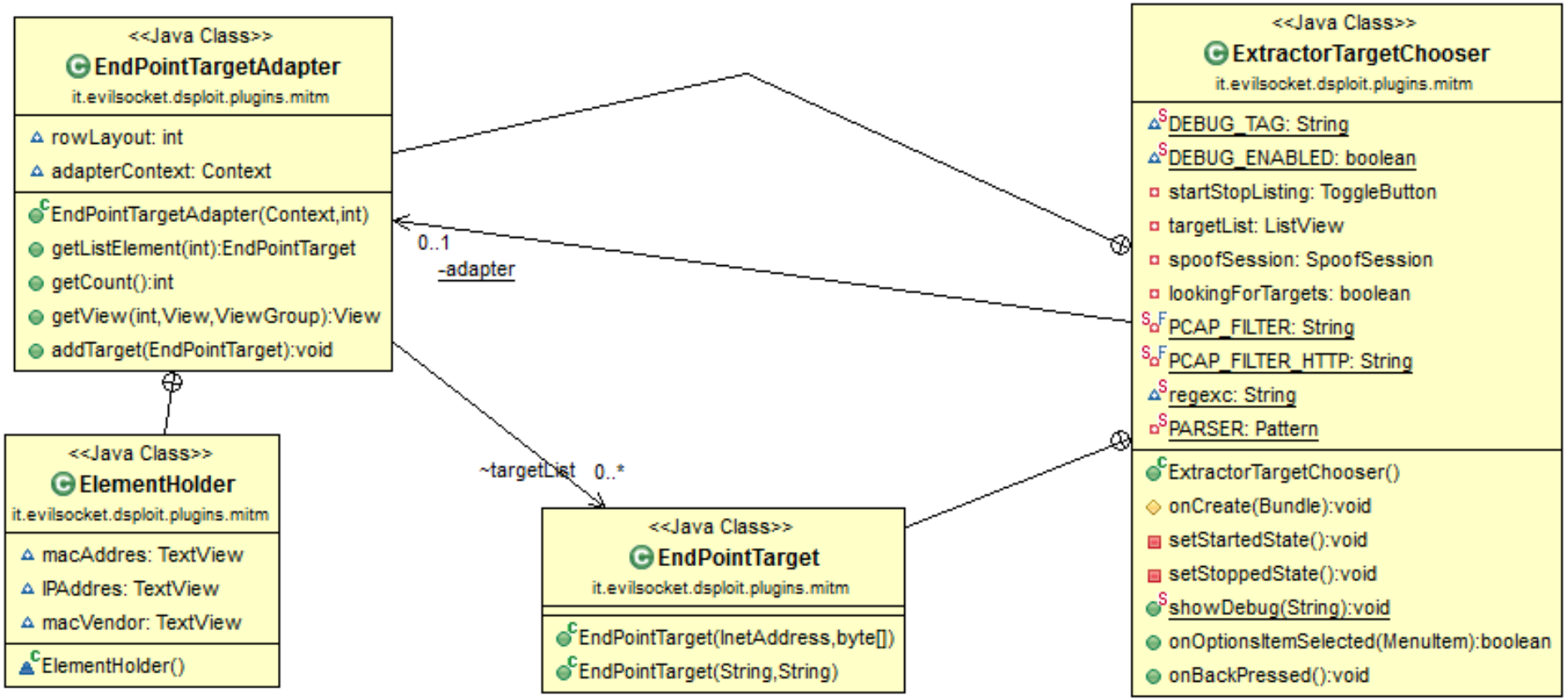


Ilustración 19 - Diagrama de clases I

El segundo diagrama muestra la descomposición en clases del subsistema de obtención de información. Está compuesto por un total de cinco clases, *BrowserRequest*,*NetworkAsynchronousTask*, *BrowserRequestListAdapter*,*ListElement Holder* y *Extractor*. Las clases *BrowserRequestListAdapter* y *ListElementHolder* tienen una funcionalidad similar que sus clases homónimas del apartado anterior, gestionar las entradas en la lista de objetos *BrowserReqeuest*, *BrowserRequestList*. La clase más interesante de este subsistema es NetWorkAsynchronousTask, que se encarga de solcitar mediante whois el nombre de la red a la que van dirigidas las peticiones HTTPS. Todo el proceso de consulta se realizará en un hilo independiente del proceso principal. El diagrama de clases de este subsistema es el siguiente:

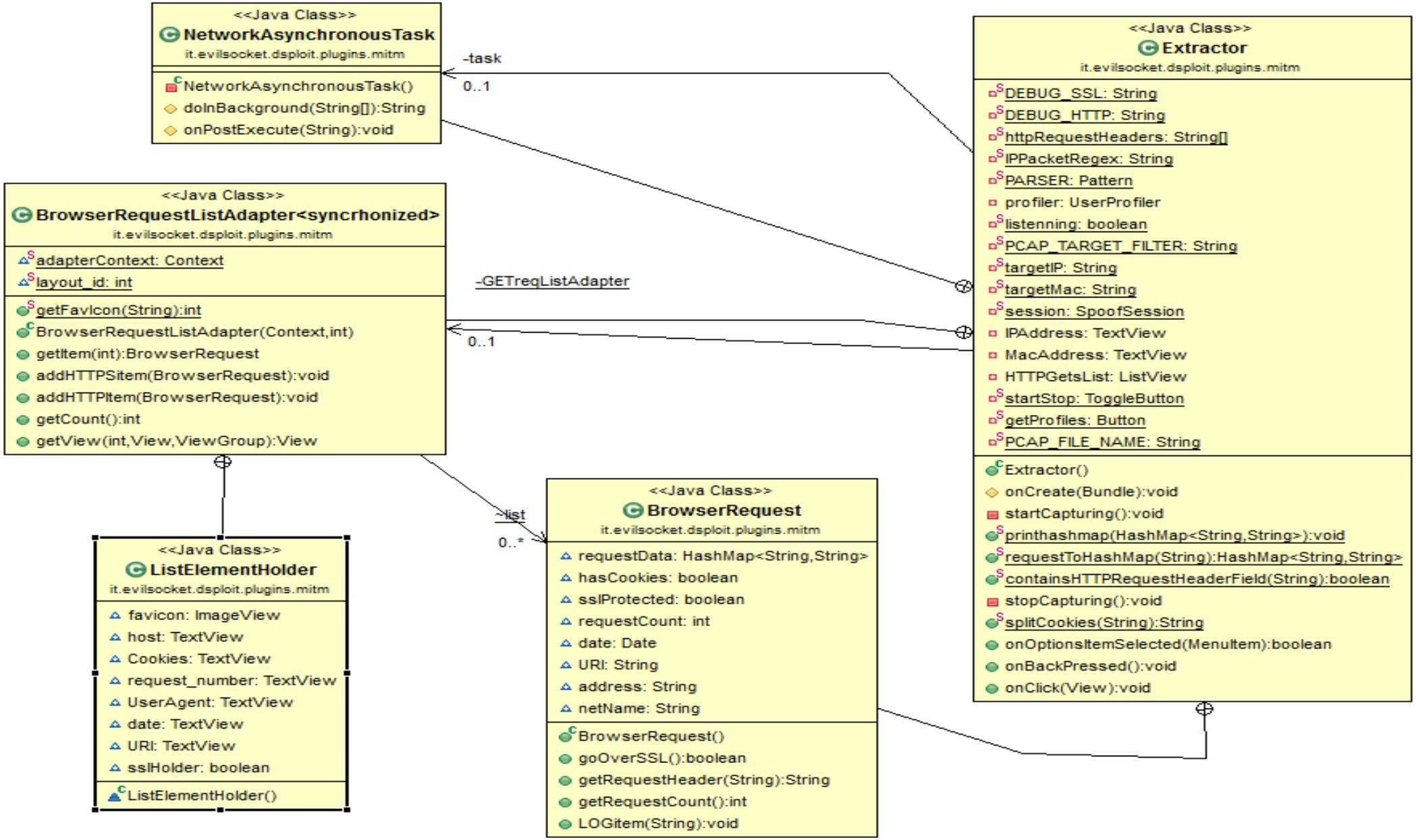


Ilustración 20 - Diagrama de clases II

En el tercer diagrama se muestran las distintas clases y las relaciones que conforman el subsistema de obtención de perfiles. Las clases declaradas en este diagrama son las siguientes: *Profile*, *Rule*, *UserProfiler* y *ProfilesFileHandler*. *Profile* y *Rule* han sido documentadas en la sección 4.3, su función es almacenar las características de un perfil de usuario determinado. *UserProfiler* se encarga de aplicar la lista de perfiles recuperados del fichero profiles.xml a los datos capturados por la aplicación, de este proceso se obtendrá la lista de perfiles que validan los datos extraídos. Por último la clase *ProfilesFileHandler* se encarga de las tareas de entrada salida relacionadas con la apertura lectura y cierre del fichero, así como de parsear los datos que contiene. El diagrama de clases asociado a este subsistema es el siguiente:

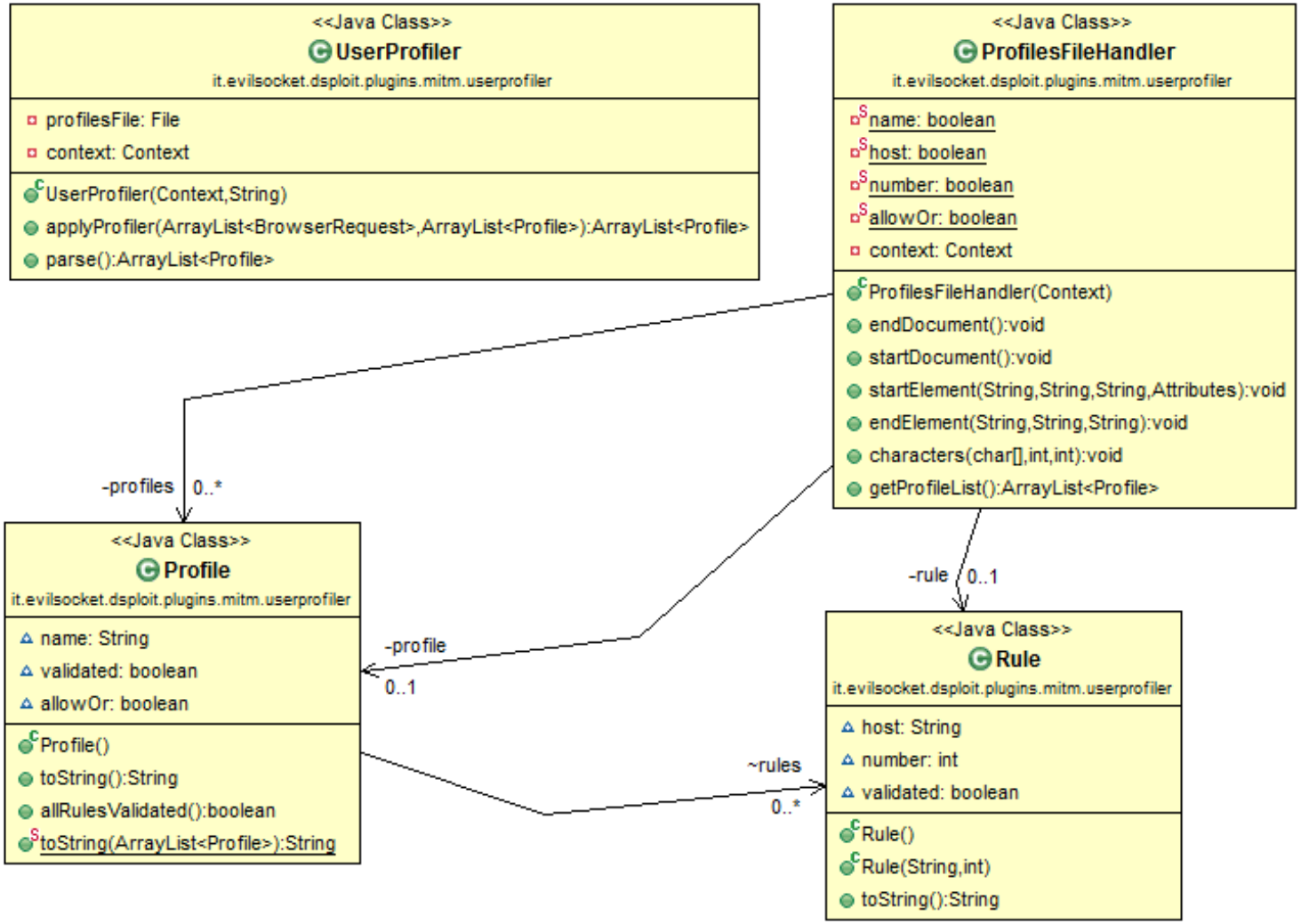


Ilustración 21 - Diagrama de clases III

El cuarto y último diagrama de clases documenta las clases contenidas en el módulo de DNS Spoofing. Este componente está integrado por tres clases: *hostFileEntry*, DNSSpoof y DNSPacket. Como se ha dicho en la sección 4.3, hostfileEntry abstrae a un objeto el contenido de cada línea del dichero hosts y DNSPacket se utiliza para representar el contenido de las peticiones DNS que envíen los dispositivos conectados a la red. Por último la clase DNSSpoof contiene la interfaz gráfica y las llamadas a los métodos de las demás clases del módulo. El diagrama de clases de este subsistema es el siguiente:

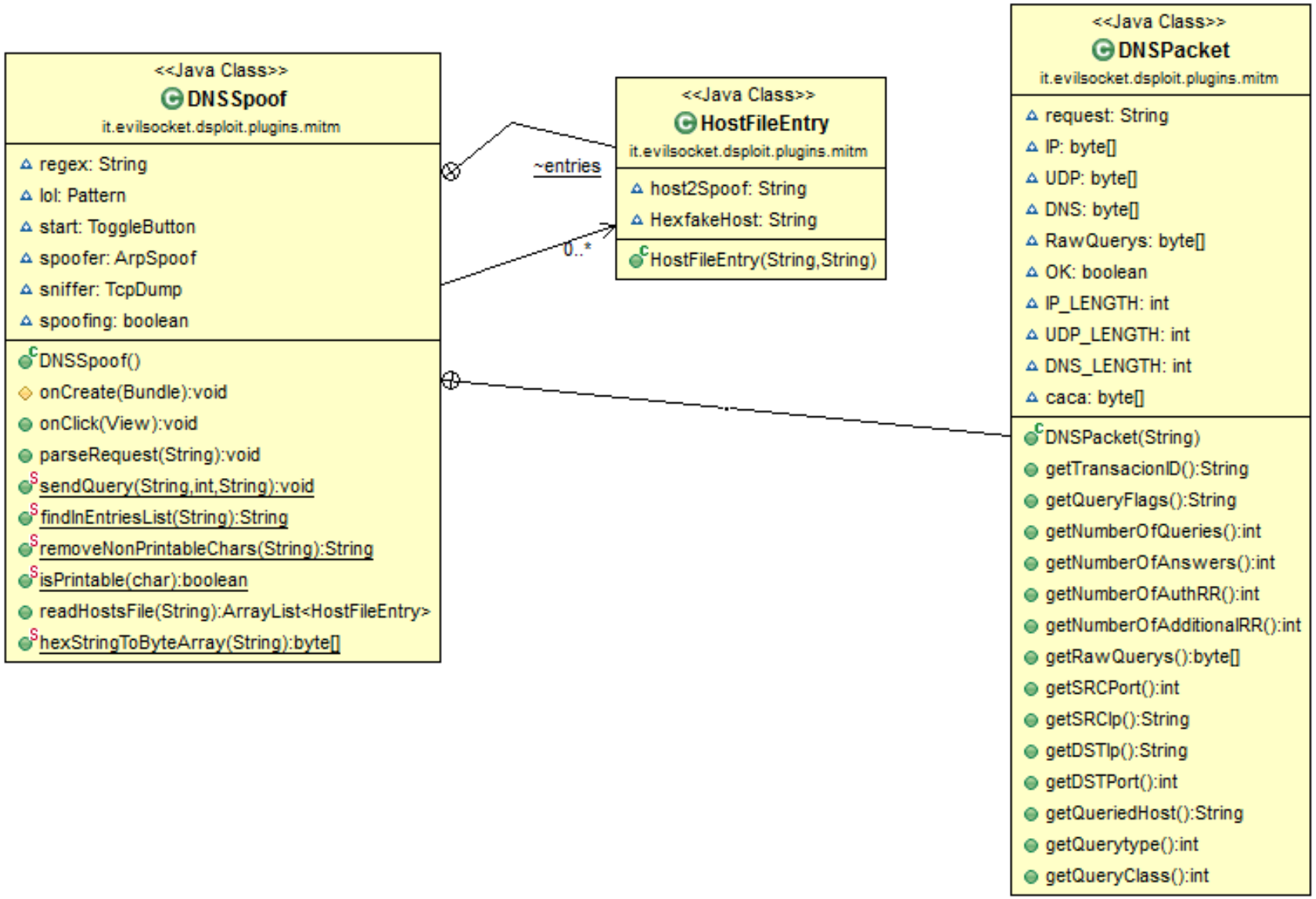


Ilustración 22 - Diagrama de clases IV

5. Implementación y Pruebas

En este capítulo se analizarán los principales aspectos referentes al proceso de implementación y a las pruebas realizadas, se incluye además un pequeño repaso por los conceptos básicos del desarrollo de aplicaciones en la plataforma Android, tales como la composición de un proyecto, los principales componentes del API y la ejecución de código nativo en Android mediante Java JNI.

5.1. Aspectos Concretos del Desarrollo de Aplicaciones Móviles en la Plataforma Android

El desarrollo de aplicaciones Android se realiza bajo el API de la plataforma construido sobre el lenguaje Java. Este API ha ido creciendo desde que se publicaron las primeras versiones, incorporando nuevas posibilidades a medida que los dispositivos han ido contando con nuevos componentes como sensores, y otros elementos. El Android Development Kit (ADK) contiene el conjunto de herramientas necesarias para comenzar el desarrollo de una aplicación, sus componentes básicos son los siguientes:

- **Herramientas de depuración y desarrollo:** Son los componentes del SDK empleados para la depuración y la implementación de las aplicaciones, entre las más destacadas se encuentran *adb* (*Android Debug Bridge*) que permite llevar a cabo tareas de *debugging* sobre un dispositivo físico o *Android Emulator* que permite virtualizar un dispositivo Android en distintas arquitecturas (ARM o x86), sobre el que probar las aplicaciones si no se cuenta con un dispositivo físico.

- **Librerías de la plataforma:** Parta cada versión de Android, existen una versión de las librerías incluidas en su API. El SDK proporciona la totalidad de librerías compiladas dentro de un archivo Jar. Este archivo Jar contiene una implementación de las librerías de Android para una versión concreta, es un

componente fundamental del SDK, ya que todas las aplicaciones desarrolladas basarán su funcionamiento en este conjunto de librerías.

- **Imágenes del sistema Android:** El emulador del que se ha hablado en el punto sobre las herramientas, necesita contener una instalación completa del sistema operativo, por ello se incluyen las imágenes del sistema precompiladas para las distintas arquitecturas sobre las que se puede emular.

- **Código fuente de las librerías de la plataforma:** En el SDK se incluye el código fuente de las librerías de Android, esto es especialmente útil para llevar a cabo tareas de depuración.

- **APIs de servicios de Google:** Las APIS de Google proporcionan acceso a los servicios prestados por Google tales como Maps, el acceso a la red social Google+ o la comunicación con el contenedor de aplicaciones App Engine.

Como componente adicional al ADK es necesario contar con un entorno de desarrollo para llevar a cabo el desarrollo propiamente dicho. Android recomienda el uso de Eclipse y además proporciona un *plugin* que dota al entorno de una interfaz para utilizar las herramientas del ADK.

Las aplicaciones Android se construyen sobre en 4 componentes fundamentales definidos en el API. Estos elementos son las Actividades o *activities*, los servicios, los *Intents* y los *Content Providers*.

- **Actividades:** Son los componentes de la aplicación encargados de mostrar la interfaz de usuario. Una aplicación puede contener diferentes actividades, cada una de ellas diseñada para permitir al usuario realizar una serie de interacciones diferentes según la funcionalidad de la aplicación. En toda aplicación existe una actividad principal o *Main Activity* que se corresponde con la primera interfaz mostrada por la App. Para una actividad se definen distintos estados según este siendo mostrada, haya sido sustituida por otra actividad o sea eliminada de memoria por el recolector de basura, el conjunto de estados por los que pasa una actividad se denomina ciclo de vida. Para cada cambio de estado se produce un evento que provoca la ejecución de un método en la clase *Activity*. El siguiente diagrama ilustra el ciclo de vida de una actividad y los eventos que se producen durante el mismo:

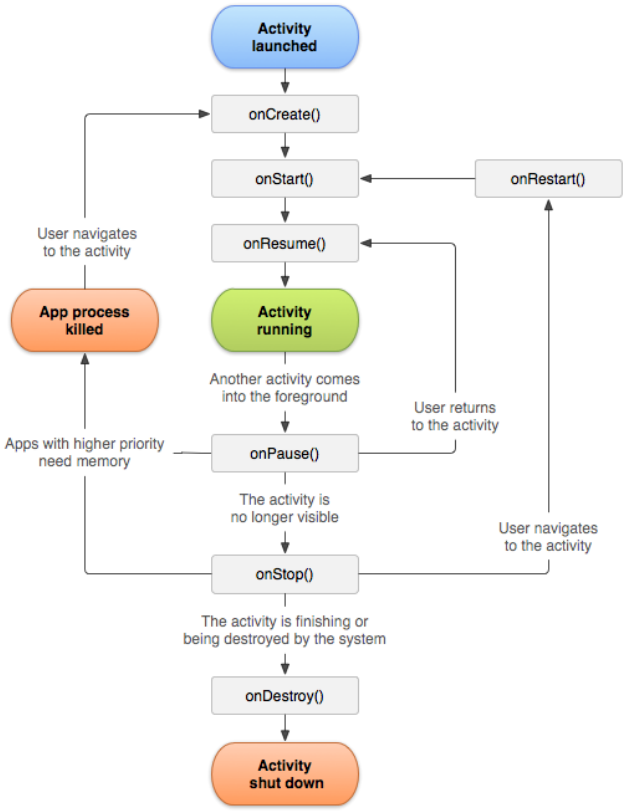


Ilustración 23 - Ciclo de vida de una *Activity* [16]

- **Servicios:** Los servicios son componentes pensados para ejecutar tareas de larga duración tales como la recepción de datos desde Internet, la reproducción de música u otras tareas de entrada/salida. Típicamente un servicio es utilizado por otro componente para realizar una función determinada. Los servicios se ejecutan en el mismo proceso que la aplicación que los ha creado, sin embargo aunque la aplicación ya no esté activa (ninguna de sus actividades se esté mostrando) el servicio continuará su ejecución en segundo plano, por otro lado, si el proceso de la aplicación se termina, la ejecución de los posibles servicios asociados se detendrá.

- **Intents:** Los *intents* son el mecanismo principal de interacción entre componentes de una aplicación. Un *intent* es en esencia un mensaje que un componente envía a otro para solicitar una determinada operación, por ejemplo cuando una actividad A necesita mostrar la interfaz de otra Actividad B, genera un *intent* con un conjunto de datos asociados, la actividad B recibirá el *intent* y será mostrada en pantalla sustituyendo a la actividad que la llamó.

- **Content providers:** Los *content providers* son un mecanismo de comunicación entre procesos, mediante ellos una aplicación puede acceder a los datos generados o almacenados por otra aplicación de forma controlada (es decir siempre que se cuente con los permisos necesarios). Este mecanismo se utiliza en aplicaciones del sistema operativo como por ejemplo la agenda de contactos, además puede utilizarse para añadir a las nuevas aplicaciones un vínculo a través del cual otras aplicaciones puedan acceder a los datos que esta gestiona.

Para todos los proyectos Android, se establece una estructura de directorios y archivos común para todos ellos durante la etapa de desarrollo. Algunos de los elementos que forman parte de esta estructura, como por ejemplo el archivo Manifest.xml, son de gran importancia para el resultado final de la etapa de

implementación cuando el proyecto se compile a un fichero *apk*. A continuación quedan descritos brevemente los principales componentes de un proyecto Android:

- **Directorio src:** Este directorio contiene toda la estructura de paquetes en los que se organiza el código fuente de la aplicación.

- **Directorio bin:** En este directorio se almacenan los ficheros binarios generados por la compilación del proyecto, principalmente el fichero *apk* instalador de la aplicación.

- **Directorio jni:** Contiene los programas y librerías escritos en código nativo que serán compilados con la herramienta NDK.

- **Directorio gen:** Contiene el código generado por las herramientas de desarrollo del SDK, como por ejemplo el fichero R.Java que permite al código escrito por el programador el acceso a los distintos recursos del proyecto.

- **Directorio assets:** En este directorio el desarrollador podrá almacenar recursos que serán utilizados por la aplicación tales como certificados, contenedores de datos, etcétera.

- **Directorio res:** En este directorio se almacenan ficheros estáticos de distinto tipo que serán utilizados por la aplicación con varios fines. Contiene los gráficos que utilizara la aplicación incluyendo el icono que la identificara una vez se instale en el dispositivo, además de una serie de ficheros XML en los que se definen las interfaces de usuario de cada actividad y valores estáticos como los rótulos mostrados en los botones de la interfaz.

- **Directorio libs:** En este directorio se almacenan las librerías externas que necesita el proyecto, en formato Jar. Android también permite referenciar un proyecto externo como una librería en lugar de almacenarla en este directorio.

- **Archivo AndroidManifest.xml:** El fichero AndroidManifest constituye la principal fuente de información sobre la aplicación que utiliza el sistema operativo. En este documento XML se definen multitud de propiedades del proyecto, como por ejemplo el nombre de la aplicación, los distintos componentes que conforman el proyecto, la versión mínima del sistema necesaria para ejecutar la aplicación y los permisos que está autorizada a utilizar, aspecto de vital importancia.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.VIBRATE" />
```

Ilustración 24 - Permisos a utilizar por la aplicación definidos en el Android Manifest

5.1.1. Ejecución de Código Nativo. Android y JNI

Desde la versión 1.5 del SDK, Android incluye la herramienta NDK (Native Development Kit) que permite desarrollar partes de la aplicación utilizando los lenguajes C/C++ en lugar de Java o utilizar binarios nativos de la plataforma Unix que hayan sido compilados para la arquitectura ARM. El NDK es un conjunto de herramientas entre las que se incluye el compilador gcc, que permite la generación de librerías y archivos ejecutables que pueden ser utilizados por una App Android.

La especificación que describe como el código escrito en Java hace uso de los binarios y librerías nativas es Java Native Interface conocida como JNI, en esta especificación se documentan la sintaxis a utilizar a la hora de invocar las funciones programadas en código nativo. Debido a que el framework JNI puede elevar el nivel de complejidad del desarrollo, a la hora de utilizar binarios nativos en Android se suele hacer uso de las clases *Runtime* y *Process*, mediante las cuales puede crearse un proceso que ejecute un determinado archivo binario, de este proceso podremos obtener su salida, por lo que podremos capturar los datos que genere. Esta metodo resulta una vía más simple que el uso de JNI.

La aplicación dSploit utiliza este mecanismo para ejecutar las herramientas de las que hacen uso los diferentes módulos, algunos ejemplos de estas herramientas son *tcpdump*, *ettercap* o *nmap*.

El uso de código nativo en Android está recomendado en contadas ocasiones, ocasiones, y se recomienda su uso sobre todo cuando se está desarrollando una aplicación que haga un uso intensivo de la CPU, como por ejemplo aplicaciones que utilicen renderizado de imagenes tales como juegos o reproductores multimedia, o como en este caso que necesiten ejecutar alguno de sus componentes a un nivel de abstracción menor que en el que opera la máquina virtual Dalvik.

5.2. Implementación

En esta sección se describen los aspectos más destacados de la etapa de implementación de las funcionalidades diseñadas anteriormente. Para una mejor organización, se ha dividido este apartado en varias subsecciones dedicada cada una a un módulo específico.

5.2.1. Módulo de Extracción

Como ha quedado documentado en la fase de análisis, la finalidad de este módulo es obtener en tiempo real la información que un usuario de la red envía a un servidor web, contenida en peticiones HHTP o HTTPS. Este módulo se descompone en tres subsistemas, que son: la selección del objetivo, la extracción y la obtención de perfiles. A continuación quedan detallados los procesos de implementación de cada uno de ellos.

5.2.1.1. Selección de objetivo

Antes de comenzar la obtención efectiva de información es necesario seleccionar el origen de la misma, es decir, de qué cliente se desean obtener datos que resulten interesantes para usuario de la aplicación. Para llevar a cabo este proceso de selección, se ha creado la clase *ExtractorTargetChooser*. Dentro de esta categoría, se ha definido una clase adicional denominada *EndPointTarget*, utilizada para abstraer los atributos de los potenciales objetivos que se encuentran en la red, cada vez que se detecte un dispositivo enviando mensajes HTTP o HTTPS se instanciará un objeto de esta clase. La implementación de este objeto se basa en extender la clase *Endpoint*, contenida dentro del módulo NET de dSploit. Esta clase representa un dispositivo conectado a la red y contiene los atributos que permiten identificarlo, la dirección IP y la dirección MAC.

```
public  class EndPointTarget extends Endpoint
{
    public EndPointTarget(InetAddress address, byte[] hardware) {
        super(address, hardware);
    }

    public EndPointTarget(String address, String hardware) {
        super(address, hardware);
    }
}
```

Ilustración 25 - Detalle de la clase EndPointTarget

Para permitir al usuario elegir entre todos los objetivos que se encuentren en la red, estos se le presentarán en una lista, donde podrá elegir el que prefiera. Para mostrar los objetivos, es necesario que se defina en la interfaz de la actividad una lista o *listview*. Esto se hace a través de la edición del fichero de superficie o *layout* asociado a la actividad. En este caso se ha creado una lista con los siguientes atributos:

```
<ListView
    android:id="@+id/targetList"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/separatorBar"
    android:layout_alignParentBottom="true"
    android:paddingLeft="5dp"
    android:paddingRight="5dp" >
</ListView>
```

Ilustración 26 - Definición XML de la lista de objetivos

Para poder utilizar esta lista en la actividad, es necesario crear un objeto que la referencie. En concreto, un objeto de la clase *Listview* que debe ser inicializado en el método *onCreate()* de la actividad contenedora, llamando al método findViewByID(). Este es un proceso común para todos los objetos que quiera añadirse en la interfaz de una actividad.

```
//initialize the activity views
startStopListing =(ToggleButton)findViewById(R.id.startTargetListingButton);
targetList =(ListView)findViewById(R.id.targetList);
```

Ilustración 27 - Referencia de los componentes de la interfaz en la actividad de selección de objetivos

Una vez creada la lista, necesitamos definir el aspecto de sus entradas. Si bien Android cuenta con una serie de modelos base, también permite crear nuestro propio diseño, mediante la implementación de tres componentes: la interfaz de la entrada, el objeto que contenga sus atributos y el adaptador que permita a la lista gestionar cada una de sus entradas. El segundo de estos componentes ya está desarrollado y no es otro que el objeto definido en la clase *EndPointTarget*. Para crear la interfaz de la entrada, procedemos del mismo modo que en el caso de la actividad y definimos su estructura en un fichero *layout*. Estará compuesto por tres campos de texto, uno para mostrar la dirección IP del objetivo, otro para la MAC y otro para mostrar el fabricante del chip de red del dispositivo:

```
<TextView
    android:id="@+id/IPAddresField"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:textColor="#FFFFFF" />
<TextView
    android:id="@+id/MacAddresField"
    android:layout_below="@id/IPAddresField"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:textColor="#FFFFFF" />
<TextView
    android:id="@+id/MacVendorField"
    android:layout_below="@id/MacAddresField"
    android:layout_marginTop="2dp"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:textColor="#FFFFFF" />
```

Ilustración 28 - Definición del aspecto de las entradas de la lista de objetivos

Por último necesitamos desarrollar la clase adaptador, que cuenta con una estructura de datos donde se almacenan el conjunto de objetos que queremos mostrar y con funciones que permiten agregar, eliminar y modificar elementos de la lista. Para esta tarea es necesario crear una clase que extienda *ArrayAdapter*, clase utilizada por

defecto en Android como adaptador y cuyo método principal es *getView()*, que es invocado cada vez que se muestra una nueva entrada de la lista.

```
public static class EndPointTargetAdapter extends ArrayAdapter<EndPointTarget>
{
    int rowLayout;
    Context adapterContext  =   null;
    ArrayList<EndPointTarget> targetList;
    static class ElementHolder{
        TextView macAddres;
        TextView IPAddres;
        TextView macVendor;
    }
    public EndPointTargetAdapter(Context context,int layout) {
        super(context, layout);
        this.adapterContext=context;
        this.rowLayout=layout;
        targetList  =   new ArrayList<EndPointTarget>();
    }
```

Ilustración 29 - Definición de la clase adaptador y de su constructor principal.

```
//create a new target list adapter
adapter=new EndPointTargetAdapter(getApplicationContext(), R.layout.end_point_target_list_item);

//set the adapter for the list
targetList.setAdapter(adapter);
```

Ilustración 30 - Creación y asignación del nuevo adaptador para la lista

En concreto el método *getView()* se encarga de dos funciones principales: Asociar los elementos de la interfaz de cada entrada de la lista con su definición en el fichero *layout* y de poblar los elementos de la interfaz del contenido almacenado en los objetos que están contenidos en la lista. El proceso de mostrar cada una de las entradas en la lista es delegado a un servicio del API denominado *LayoutInflater*:

```
LayoutInflater inflater=(LayoutInflater) adapterContext.getSystemService(LAYOUT_INFLATER_SERVICE);
listElement=inflater.inflate(rowLayout, null,false );
holder=new ElementHolder();
holder.IPAddres=(TextView)listElement.findViewById(R.id.IPAddresField);
holder.macAddres=(TextView)listElement.findViewById(R.id.MacAddresField);
holder.macVendor=(TextView)listElement.findViewById(R.id.MacVendorField);
```

Ilustración 31 - Inflado de los componentes de la interfaz

```
holder.IPAddres.setText(getListElement(position).getAddress().toString().replace("/", ""));
holder.macAddres.setText(targetList.get(position).getHardwareAsString());
if(System.getMacVendor(targetList.get(position).getHardware())==null)
{
    holder.macVendor.setText("Unknown MAC vendor");
}else{
holder.macVendor.setText(System.getMacVendor(targetList.get(position).getHardware()));
}
```

Ilustración 32 - Asignación del contenido a las entradas de la lista

En listas en las que se van a mostrar muchas entradas, este proceso puede ser realmente costoso y es conveniente optimizarlo. Esta optimización puede llevarse a cabo mediante el uso de una clase *holder*, que actué como caché de los elementos de la interfaz, con el fin de no tener que inflarlos cada vez que se muestran, de tal manera que solo sea necesario actualizar su contenido. Aunque en esta lista no se espera que se muestren muchas entradas, la optimización se ha implementado mediante la clase ElementHolder.

Mediante estos tres pasos, la lista queda implementada. Es el momento de desarrollar la búsqueda de objetivos. Para este proceso se han utilizado dos herramientas ya implementadas en dSploit, la función de *arpSpoof* y la herramienta de captura de análisis de paquetes *tcpdump*. Mediante la herramienta *arpspoof* se lleva a cabo un ataque *man in te middle* entre las estaciones y el punto de acceso mientras que con *tcpdump* se capturan y filtran los paquetes según sus características. Para utilizar la herramienta *arpspoof*, se ha utilizado una instancia de la clase *SpoofSession*, que actúa de interfaz sobre la funcionalidad de ARP Spoofing y facilita el uso de la herramienta. Este objeto debe ser inicializado durante la creación de la actividad, en el método *onCreate():*

```
//initialize the spoofSession object, not using proxy or server
spoofSession       = new SpoofSession( false, false, null, null );
```

Ilustración 33 - Creación de una nueva instancia de la clase *Spoofsession*

Para arrancar la búsqueda de paquetes, se ha desarrollado el método *setStartedState()*, que se encarga de iniciar la sesión de ARP spoofing, comenzar el filtrado de paquetes con *tcpdump* y de añadir nuevos elementos a la lista de objetivos. Este método se lanza cuando se pulsa sobre el botón de la interfaz rotulado como “Look

For Targets", para lograr esto, se ha añadido un manejador de eventos de tipo *OnClick* al botón, que se encarga de la ejecución del método *setStartedState()* o *setStoppedState()* según proceda:

```
//set new OnclickListener for the button
startStopListing.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        if(lookingForTargets){
            setStoppedState();
        }else{
            setStartedState();
        }
    }
});
```

Ilustración 34 - *OnClickListener* del botón LookForTargets

```
private void setStartedState( ) {
    //if the re is a tcpdump session alive, we first kill it
    System.getTcpDump().kill();
    lookingForTargets=true;
    spoofSession.start( new OnSessionReadyListener() {
        @Override
        public void onError( String error ) {

        }

        @Override
        public void onSessionReady() {

            System.getTcpDump().sniff( PCAP_FILTER_HTTP, null, new OutputReceiver()
```

Ilustración 35 - Comienzo de la búsqueda de objetivos

Como se observa en la ilustración, se solicita una instancia de *tcpdump*, sobre la cual se invoca el método *sniff()*, pasando como parámetro una cadena, que contiene las características de los paquetes que debe capturar. El filtro utilizado es el siguiente:

```
-vveA dst port 80 or dst port 443 or src port 80 or src port 443
```

Esto hace que *tcpdump* muestre los paquetes cuyo puerto de destino u origen sea el 80 o el 443, utilizados por HTTP y HTTPS respectivamente y además capture la cabecera Ethernet (para tener acceso a la dirección MAC del emisor) y su carga útil. Cada uno de los paquetes capturados, se evalúa bajo una expresión regular. Esta expresión regular obtiene los grupos de caracteres que representan las direcciones IP y

MAC contenidas en las cabeceras. Para cada par de valores IP, MAC distintos que se detecten se creará un objeto de la clase *EndPointTarget*, que se añadirá a la lista:

```
Matcher matcher = PARSER.matcher( line.trim() );

    if(matcher!=null && matcher.find()){
        EndPointTarget source= new EndPointTarget(matcher.group(8), matcher.group(1));
        //EndPointTarget destiny= new EndPointTarget(matcher.group(9), matcher.group(4));

        if(System.getNetwork().isInternal(matcher.group(8))){
        //  showDebug("internal node (SRC)");
            //save it to the list
            final EndPointTarget target=source;

            ExtractorTargetChooser.this.runOnUiThread(new Runnable() {

                @Override
                public void run() {
                    adapter.addTarget(target);
                    adapter.notifyDataSetChanged();

                }
            });

        }
```

Ilustración 36- Creación del objeto *EndPointTarget* e inclusión en la lista

Las nuevas entradas se añaden a la lista mediante una llamada el método *addTarget()* de la clase adaptadora de la lista, este método recibe el objeto como parámetro y comprueba si ya ha sido en añadido a la lista, en caso afirmativo el objeto queda descartado y si por el contrario la lista no contiene una entrada con esos datos, el objeto se añade:

```
public synchronized void addTarget(EndPointTarget newtarget){
    boolean targetAdded=false;
    for(EndPointTarget target:targetList){
        if( target.getAddress().equals(newtarget.getAddress()) ){
            targetAdded=true;
            break;
        }
    }
    if(!targetAdded){
        showDebug("Added Target!");
        targetList.add(newtarget);
        showDebug(String.valueOf(getCount()));
    }
}
```

Ilustración 37- Método *addTarget*

Cuando la lista se ha poblado de posibles objetivos, la búsqueda se puede detener pulsando de nuevo el botón, cuyo manejador de eventos *onClick* invoca al método *setStoppedState()*. Esta función detiene la sesión de ARP spoofing y destruye la instancia de *tcpdump* creada al comienzo:

```
private void setStoppedState( ) {
    //stop the current spoof session
    spoofSession.stop();
    //kill the current tcpdump session
    System.getTcpDump().kill();
    //update the search state
    lookingForTargets= false;
    //uncheck the start/stop button
    this.startStopListing.setChecked( false );
}
```

Ilustración 38 - Detalle del método setStoppedState

Al final del proceso de búsqueda de objetivos, la interfaz muestra todas las estaciones de la red que generan o han generado tráfico web durante la sesión. En este momento, el usuario seleccionará una de las entradas de la lista y comenzará la fase de obtención de información. El aspecto de la interfaz cuando la lista de objetivos ha sido poblada es el siguiente:

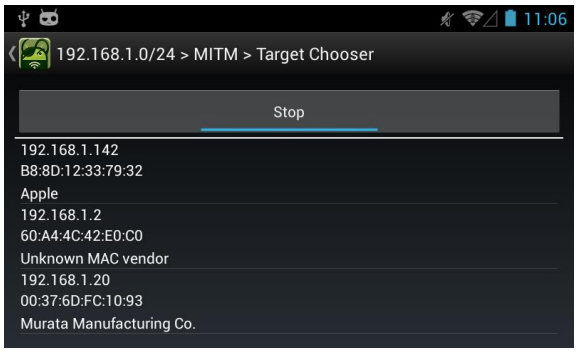


Ilustración 39 - Lista de objetivos detectados

Para acceder a la siguiente actividad, se ha signado a la lista un manejador de tipo *onItemClick*, de tal manera que al pulsar el usuario sobre el objetivo que le interese, se produzca el cambio de actividad. Dentro del manejador, se ha instanciado un *intent*, al cual se le agregan como datos la dirección IP y Mac del objetivo seleccionado, para que estén disponibles en la siguiente actividad:

```
targetList.setOnItemClickListener(new OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> arg0, View element, int position,long id) {
        //in case of active search, stop it
            setStoppedState();
            Intent intent=new Intent(getApplicationContext(), Extractor.class);
            intent.putExtra("target_IP", adapter.getListElement(position).getAddress().toString().replace("/", ""));
            intent.putExtra("target_MAC", adapter.getListElement(position).getHardwareAsString());
            startActivity(intent);
    }
});
```

Ilustración 40 - Detalle del manejador OnItemClick de la lista de objetivos

5.2.1.2. Extractor

Una vez que el objetivo ha sido elegido mediante la actividad anterior, podemos comenzar a interceptar el tráfico web que este genere. La implementación de esta actividad es en gran medida similar a la anterior. En lo que se refiere a la interfaz, también cuenta con una lista que contendrá las peticiones HTTP y HTTPS capturadas y cuenta con dos botones. El primero de ellos se utiliza para iniciar la obtención de perfiles y el otro se utiliza para arrancar o detener la captura.

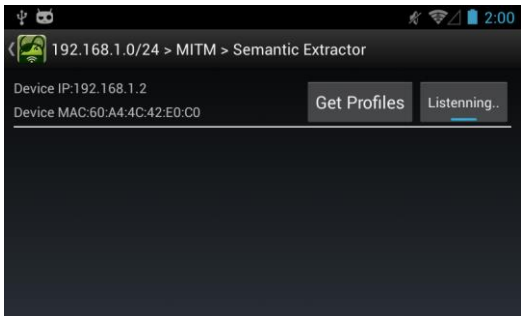


Ilustración 41 - Interfaz de la Actividad Extractor

En lo que se refiere a la codificación, es también muy similar a la de la anterior actividad. Se ha definido una clase *BrowserRequest*, que será utilizada para almacenar los datos de las peticiones capturadas y que se mostrarán en la lista. Los principales atributos de esta clase son la fecha de la petición, la URI del recurso solicitado, el nombre de dominio al que iba dirigida, un objeto de la clase *HashMap* que contiene las cabeceras de la petición, una cadena que contendrá las cookies en caso de que existan y una variable de control que permitirá conocer si la petición está formulada en claro o bajo HTTPS:

```
public static class BrowserRequest{
    HashMap<String, String> requestData;
    boolean hasCookies=false;;
    boolean sslProtected=false;
    int requestCount;
    Date date;
    String URI="";
    String address="";
    String netName="";
    public BrowserRequest(){
        requestCount=0;
        URI="";
        address="";
        netName="";
    }
```

Ilustración 42 - Detalle de la clase *BrowserRequest*

La interfaz contiene también una lista, definida en el fichero *layout* correspondiente, de la misma manera que en el caso anterior. Se ha creado una clase adaptadora *BrowserRequestListAdapter* para que la lista maneje los objetos de la clase *BrowserRequest* que se mostrarán en ella. El aspecto que tendrán las entradas de esta lista se ha definido en otro fichero *layout* y está compuesta por seis campos de texto y una pequeña imagen, que mostrará el *favicon* de la web a la que se envió la petición en los casos en los que se ha especificado en los requisitos.

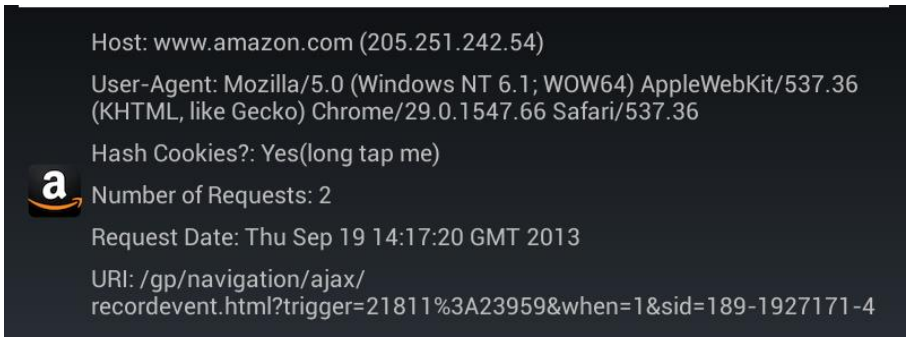


Ilustración 43- Ejemplo del aspecto de las entradas en la lista de peticiones

En la codificación de la clase adaptador, se ha prestado especial atención a optimizar el funcionamiento del método *getView()*, empleando mecanismos como el uso de una clase *Holder* para cachear los elementos de la interfaz, o una estructura de decisión que impide que se inflen los elementos de la lista que no se muestran en un determinado momento, reduciendo significativamente el número de accesos a memoria. Esto necesario en este caso, debido a que los elementos de la lista serán modificados

con mucha frecuencia, se mostrarán muchas entradas y cada una de las entradas tiene un diseño complejo, que incluye texto e imagen.

La parte más compleja del desarrollo de este subsistema se corresponde con la recuperación de las peticiones generadas por el cliente. Para desviar el tráfico y capturarlo se ha utilizado el mismo método que en el caso anterior, la clase *SpoofSession* y la herramienta *tcpdump*. Para facilitar la captura de los mensajes HTTP y HTTPS se ha distinguido entre dos casos según el puerto destino de los paquetes que se capturaban.

Si el paquete tiene como destino el puerto 80, que se corresponde con HTTP, se comienza a reconstruir la petición mediante las líneas que *tcpdump* devuelve. Cada vez que se aparece un carácter de salto de línea o retorno de carro, el buffer que almacena la salida de la herramienta *tcpdump* libera una línea. Si préstamos atención a la estructura de un mensaje de petición HTTP, podemos observar que cada cabecera termina en un carácter de retorno de carro, por la tanto por cada cabecera *tcpdump* nos devuelve una línea.

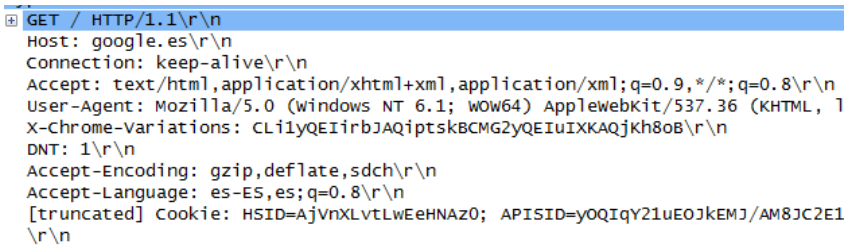


Ilustración 44 - Estructura de un mensaje GET HTTP

Esto obligó a desarrollar un algoritmo que facilitara la recuperación de toda la estructura. Este algoritmo consiste encontrar la primera línea, que contiene el nombre del método, GET. Y a partir de ella almacenar en un buffer todas las cabeceras hasta que se encuentra el comienzo de un nuevo mensaje. De esta manera se pueden recuperar todos los datos de la petición de manera sencilla.

```
//sniffed data contains an plain http request field
if(containsHTTPRequestHeaderField(line)){
    //Log.d(DEBUG_HTTP,"HTTP package");
    //the line contains a GET FIELD
    if(line.contains("GET")){
        //exist previous GET
        if(foundGET){
            /*
             * this means we have reached the end of
             * the last request and a new one starts
             */
            requestFields=requestToHashMap(req);
            httprequest=new BrowserRequest();
            httprequest.requestData=requestFields;
            httprequest.address=address;
            httprequest.hasCookies=requestFields.containsKey("Cookie");
            httprequest.URI=URI;
            httprequest.date=new Date(java.lang.System.currentTimeMillis());
            httprequest.sslProtected=false;

        //Do not exist previous GET
        if(!foundGET){
            //change foundGEt status
            foundGET=true;
            //get request URI
            URI=line.substring(line.indexOf("GET"),line.length()).replace("GE

        }
```

Ilustración 45 - Recuperación del mensaje de petición HTTP

Una vez que se han obtenido todos los valores de las cabeceras, se crea el objeto *BrowserRequest* y se da valor a sus atributos, para almacenarlo en la lista.

```
httprequest.sslProtected=false;
Extractor.this.runOnUiThread(new Runnable(){
    @Override
    public void run() {

        ReqListAdapter.addHTTPItem(httprequest);
        ReqListAdapter.notifyDataSetChanged();
    }
});
```

Ilustración 46 - Inclusión del mensaje HTTP en la lista

Para incluir el objeto, se llama al método *addHTTPItem()* de la clase adaptador que emplea la lista. Este método se encarga de añadir una nueva petición o en el caso de que ya haya sido añadida una petición al mismo dominio, actualizar sus atributos.

```
        if(listRequest.requestData.get("Host").equals(req.requestData.get("Host"))){
            listRequest.requestCount++;
            listRequest.date=req.date;
            listRequest.hasCookies=req.hasCookies;
            listRequest.requestData=req.requestData;
            listRequest.URI=req.URI;
            found=true;
            break;
        }
    }
}
}if(!found){
    req.requestCount++;
    list.add(req);
    //Log.d("","item added");
    req.LOGitem(DEBUG_HTTP);
}
```

Ilustración 47 - Detalle del método *addHTTPItem()*

Por otro lado, si el mensaje tiene como destino el puerto 443, las posibilidades de obtener información de él son bastante limitadas. Por lo tanto nos limitamos a recopilar la dirección IP de destino mediante el uso de una expresión regular, del mismo modo que se hace en la etapa anterior y el nombre de la red de destino. Para obtener este dato se lleva a cabo una consulta a un servidor whois, formulada mediante la clase *NetWorkAsynchronousTask()*, que hereda de la clase *AsyncTask()* del API de Android y que permite realizar tareas que pueden demorarse en el tiempo en un hilo independiente de la interfaz de la aplicación, con el objetivo de impedir que esta se detenga.

```
private class NetworkAsynchronousTask extends AsyncTask<String, Void, String>{
    protected String doInBackground(String... params) {
        String address=params[0];
        String queryResult="";
        Socket socket;
        String serverName = "whois.ripe.net";
        int port = 43;
        try {
            socket = new Socket(serverName, port);
            Writer out = new OutputStreamWriter(socket.getOutputStream());
            out.write(address + "\r\n");
            out.flush();
            DataInputStream theWhoisStream;
            theWhoisStream = new DataInputStream(socket.getInputStream());
            String s;
            while ((s= theWhoisStream.readLine()) != null) {
                //Log.d("ssla",s);

                if(s.contains("netname")){
                    queryResult=s.replace("netname:", "").trim();
                    //Log.d("ssldespues",s);
                    break;
                }
            }
```

Ilustración 48 - Envió petición WHOIS

```
if(line.contains(".443")){
    Matcher matcher =PARSER.matcher(line);
    if(matcher.find()){
        //Log.d(DEBUG_SSL,"HTTPs package");
        httpsrequest=new BrowserRequest();
        httpsrequest.address=matcher.group(3);
        httpsrequest.date=new Date(java.lang.System.currentTimeMillis());
        httpsrequest.requestData=new HashMap<String,String>();
        httpsrequest.hasCookies=false;
        httpsrequest.URI="";
        httpsrequest.sslProtected=true;


        task=new NetworkAsynchronousTask();
        httpsrequest.netName=task.execute(matcher.group(3)).get();


        //request.LOGitem();
        Extractor.this.runOnUiThread(new Runnable(){

            @Override
            public void run() {
                ReqlistAdapter.addHTTPSitem(httpsrequest);
                ReqlistAdapter.notifyDataSetChanged();

            }


        });
    }
```

Ilustración 49 - Creación e inclusión de la petición HTTPS en la lista

Como puede verse en la ilustración anterior, la inclusión en la lista de la nueva petición HTTPS se lleva a cabo mediante una llamada al método *addHTTPSItem()*, similar al método *addHTTP()* explicado anteriormente.

Otro aspecto reseñable de esta actividad, es el método *getView()* de la clase adaptador de la lista. En él se ha diferenciado entre un objeto *BrowserRequests* con la variable booleana *sslProtected* con el valor *true* (es decir una petición HTTPS) y con la variable en *false*, ya que según el caso se mostrara una información u otra y la interfaz tendrá aspectos distintos. Cuando se trata de una petición HTTPS, la interfaz se configura para cambiar de color al rojo y para que la imagen que se muestra se corresponda con la de un candado.

```
holder.favicon.setImageResource(R.drawable.favicon_padlock);
holder.host.setText(Html.fromHtml("<b>Host: </b>")+list.get(position).netName+" ("+list.get(position).address+")");
holder.Cookies.setText(Html.fromHtml("<b>Hash Cookies?: UNAVAILABLE INFORMATION</b>"));
holder.UserAgent.setText(Html.fromHtml("<b>User-Agent: UNAVAILABLE INFORMATION</b>"));
holder.request_number.setText(Html.fromHtml("<b>Number of Requests: </b>")+String.valueOf(list.get(position).requestCount));
holder.date.setText(Html.fromHtml("<b>Request Date: </b>")+String.valueOf(list.get(position).date));
holder.URI.setText(Html.fromHtml("<b>URI: UNAVAILABLE INFORMATION</b>"));
listElement.setBackgroundColor(0xFFE00000);
```

Ilustración 50 - Asignación de valores a los componentes de la interfaz en el caso de HTTPS

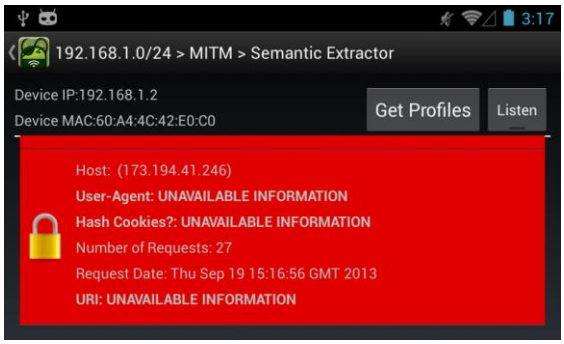


Ilustración 51 - Representación de una petición HTTPS en la interfaz

En caso de tratarse de una petición HTTP normal, se mostrará toda información sobre ella y la interfaz tendrá un aspecto similar al de la ilustración 44. Además, si se trata de una petición que contenga cookies, estas podrán visualizarse si el usuario mantiene pulsada la interfaz, ya que la lista cuenta con un manejador de eventos *OnLongClick* que mostrará el contenido de las cookies en un diálogo, que ofrece al usuario la opción de exportarlas a un fichero.

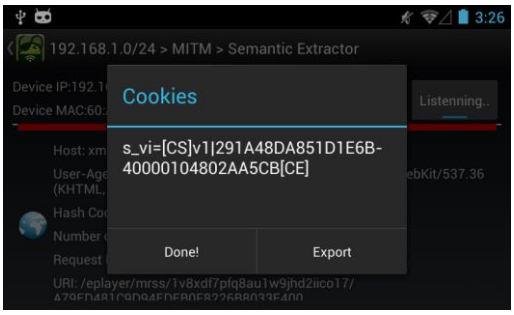


Ilustración 52 - Contenido de las cookies enviadas en una petición HTTP

Por último y enlazando con la siguiente subsección, el botón "Get Profiles" de la interfaz permite al usuario dar comienzo al proceso de búsqueda de coincidencias entre los perfiles definidos en el fichero profiles.xml y el tráfico capturado.

5.2.1.3. Obtención de Perfiles

Esta funcionalidad, no se corresponde con una nueva actividad si no que actúa en la misma actividad que el extractor, pudiendo realizar ambas tareas al mismo tiempo. Como se ha introducido en el apartado anterior, al pulsar sobre el botón "Get Profiles" se comienza el proceso de búsqueda de perfiles. Este proceso tiene dos fases.

En la primera fase se abre el fichero profiles.xml, contenido en la raíz del sistema de archivos y se serializan los perfiles a objetos de las clases *Profile* y *Rule*. Esto se lleva a cabo mediante el uso de un *parser* xml, concretamente *SaxParser* que está incluido en el API de Android. Este *parser* permite obtener el contenido almacenado en las distintas etiquetas XML de un archivo y almacenarlo como un objeto. Todas estas tareas se llevan a cabo cuando se ejecuta el método *parse()* sobre una instancia un objeto de *UserProfiler*.

```
public class Profile {
    String name;
    ArrayList<Rule> rules;
    boolean validated=false;
    boolean allowOr=false;
```

```
public class Rule {
    String host;
    int number;
    boolean validated=false;
```

Ilustración 53 - Atributos de las clases Profile y Rule

```
public ArrayList<Profile> parse()
{
    SAXParserFactory factory = SAXParserFactory.newInstance();
    ArrayList<Profile> result=null;

        SAXParser parser;
        try {
            parser = factory.newSAXParser();
            ProfilesFileHandler handler = new ProfilesFileHandler( context);
            parser.parse(this.profilesFile, handler);
            result=handler.getProfileList();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        } catch (IOException e) {
            Toast.makeText(context, "Error While Reading profile file", Toast.LENGTH_SHORT).show();
        }

    return result;
}
```

Ilustración 54 - Detalle del método *Parse()*

Como se puede apreciar en la ilustración, la recuperación efectiva de los perfiles se realiza mediante la llamada al método *gerProfiles()*, que es el que realmente aplica *SaxParser* sobre el archivo y devuelve la lista de perfiles obtenidos.

La segunda fase consiste en comparar los perfiles obtenidos con el tráfico capturado con el objetivo de encontrar coincidencias que los validen. Se considera que una regla esta validada si se ha encontrado una petición al dominio que contiene el número indicado de veces. Por otro lado se considera que un perfil estará validado si todas las reglas que contiene han sido validadas o si lo ha sido solo una en caso de que la variable booleana *allowOr* tome valor *true*. El algoritmo utilizado para conocer que perfiles se validan en base al tráfico capturado es muy sencillo, se basa en recorrer secuencialmente la lista de perfiles y la lista de peticiones mediante el método *applyprofiler()*, comparando el número de veces que se han envidado peticiones a ciertos dominios. Según el resultado de la comparación se validarán las reglas y los perfiles correspondientes.

```
public ArrayList<Profile> applyProfiler(ArrayList<BrowserRequest> requests,ArrayList<Profile> profiles){
    ArrayList<Profile> validatedProfiles = new ArrayList<Profile>();

    for (Profile profile : profiles) {

        for (Rule rule : profile.rules) {

            for(BrowserRequest request:requests){
                if(!request.goOverSSL()){
                    if((rule.host).contains(request.getRequestHeader("Host")) && (request.getRequestCount()>=rule.number)){
                        rule.validated=true;}                    }              }       }
           if(profile.allRulesValidated()){
               profile.validated=true;
               validatedProfiles.add(profile);     }    }
    return validatedProfiles;
}
```

Ilustración 55 - Detalle del método *applyProfier()*

Una vez que ha terminado el proceso, la aplicación muestra un diálogo con la lista de perfiles validados:

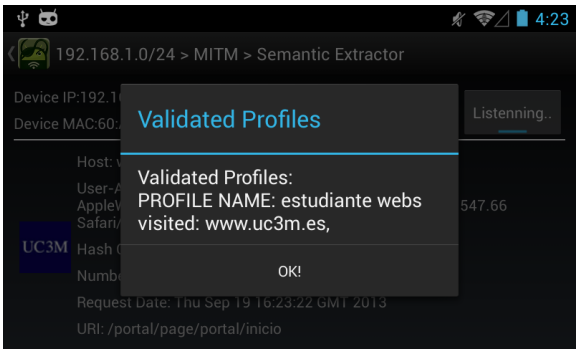


Ilustración 56 - Ejemplo de perfil validado

5.2.2. Módulo DNS Spoofing

La finalidad de este módulo es la manipulación de las consultas DNS que los clientes de la red envían a los servidores de nombre para resolver las direcciones IP con la que su dispositivo necesita comunicarse. Esta manipulación, consiste en sustituir el dominio por el que el cliente pregunta, por otro especialmente elegido por el atacante, de tal manera que al resolver el servidor DNS la consulta, entrega la dirección IP correspondiente al dominio que el atacante desea.

Toda la funcionalidad de este módulo está contenida en la clase DNSSpoof, que contiene una actividad, en la que se desarrolla la interacción con el usuario. El contenido de esta actividad es muy simple, está formada únicamente por un botón que permite iniciar o detener la falsificación de consultas

En lo referente a la codificación de la misma, al igual que en el módulo anterior se ha hecho uso de *arpspoof* y *tcpdump* para la interceptación del tráfico. Se han definido dentro de la clase DNSSpoof, dos clases auxiliares. La primera de ellas es DNSPacket, que está diseñada para almacenar la información referente a las consultas DNS y cuyos principales atributos son las direcciones IP de origen y del servidor DNS al que iba dirigida, el puerto de origen y las cabeceras IP y UDP del paquete original, además contiene los métodos necesarios para extraer estos atributos a partir de la trama IP que recibe su constructor por parámetro obtenida de *tcpdump*.

```
public static class DNSPacket{

    String request;
    byte[]IP;
    byte[]UDP;
    byte[]DNS;
    byte[]RawQuerys;
    boolean OK=false;
    int IP_LENGTH;
    int UDP_LENGTH=16;  //set to 16
    int DNS_LENGTH;
    byte[] caca;
```

Ilustración 57 - Detalle de la clase auxiliar *DNSPacket*

La segunda clase, *HostFileEntry* se utiliza para almacenar en cada una de las entradas del fichero hosts, que almacenan las correspondencias entre los host a suplantar y la cadena hexadecimal que contiene el valor del campo *Name* de una petición DNS hacia el dominio por el que se quiere sustituir el original dentro de la consulta. Estos objetos cuentan por tanto con dos atributos, el dominio cuyas peticiones deben suplantarse y el contenido con el que se deben sustituir.

```
public static class HostFileEntry{
String host2Spoof;
String HexfakeHost;
public HostFileEntry(String host2spoof,String FakeHost){
    this.HexfakeHost=FakeHost;
    this.host2Spoof=host2spoof;
}
}
```

Ilustración 58 - Detalle de la clase auxiliar *HostFileEntry*

Cada vez que se crea la actividad, se ejecuta el método *readHostFile()* que lee el fichero hosts y carga serializa su contenido en objetos de la clase *HostFileEntry* en una lista, que será consultada cada vez que se reciba una petición DNS.

Un paso previo antes de realizar alterar las consultas, es configurar el firewall del dispositivo atacante para conseguir que las peticiones del cliente no se reenvíen hacia el servidor. Recordemos que en un ataque ARP spoofing es necesario que el reenvío IP esté activado y para conseguir que todas las peticiones falseadas que se envíen al servidor contengan como IP de origen la IP del cliente que las generó en lugar de la IP del dispositivo que es quien realmente las ha enviado. Para ello se ha utilizado

la herramienta *iptables* contenida en dSploit para añadir dos reglas que filtren el tráfico saliente. La regla empleada para evitar que las consultas legítimas alcancen el servidor es:

```
-A FORWARD --proto udp --dport 53 -j DROP
```

Esta regla detecta todos los paquetes que se reenvían cuyo puerto de destino es el 53 y los descarta, impidiendo que alcancen su destino. La segunda regla empleada para modificar la IP de origen de los paquetes DNS enviados por el atacante es:

```
-t nat -A POSTROUTING -j SNAT -p udp --dport 53 –to ADDRESS
```

Donde ADDRESS indica la IP que se desea suplantar. Mediante esta regla NAT, todos los paquetes saliente que utilicen el protocolo UDP y cuyo puerto de destino sea el 53 cambiarán su IP a ADDRESS.

Como se ha dicho en párrafos anteriores, esta clase hace uso de *tcpdump* para la captura de las consultas DNS, para diferenciar esto paquetes se ha utilizado el siguiente filtro:

```
-vvvA dst port 53
```

Este filtro, permite obtener la carga útil (en este caso el mensaje de consulta) de los paquetes que tienen como destino el puerto 53, el que operan los servidores DNS, además de las cabeceras IP y UDP del paquete en cuestión. Cada vez que se recibe una petición DNS, se comprueba que sea de tipo A y se extrae el dominio del cual el cliente quiere conocer su IP y se busca en la lista de entradas del fichero hosts. Si se encuentra, se procede a la modificación. Para ello se construye una nueva consulta, que es enviada en a través de un socket UDP con dirección al servidor original a través del mismo puerto de destino. Si no se encuentra el dominio dentro de la lista, es necesario reenviar manualmente la consulta ya que el reenvío hacia el puerto 53 ha sido desactivado. Para ello procedemos igual que en el apartado anterior.

Se consigue de esta manera un ataque que sin ser muy sofisticado es completamente funcional y que puede pasar sin ser detectado por el cliente, ya que las

peticiones DNS que no han de ser falsificadas se envían integras al servidor que finalmente responde al cliente original.

5.3. Pruebas Realizadas y Resultados

En esta sección se expondrán las pruebas realizadas así como los resultados que se han obtenido. Se ha estructurado esta sección en tres subsecciones estando cada una de ellas dedicada a un subsistema de los componentes desarrollados.

El entorno en el que se han desarrollado las pruebas está compuesto por los siguientes elementos:

- **Punto de acceso:** Se trata de un router inalámbrico que sirve de punto de acceso a Internet a los dispositivos de la red.
- **Dispositivo atacante:** Dispositivo móvil en el que se están ejecutando los nuevos módulos de dSploit
- **Dispositivo víctima:** Un ordenador portátil que está siendo utilizado para una sesión de navegación web.

Con este entorno se pretende imitar los escenarios en los que están desplegadas típicamente las redes inalámbricas de acceso público, como restaurantes, aeropuertos o universidades.

5.3.1. Pruebas Sobre el Módulo de Extracción

En esta subsección se probará el módulo de extracción de información. Para ello se pretende obtener datos sobre la sesión de navegación que se está llevando a cabo. Además se intentarán obtener las cookies del usuario en una determinada web, con el objetivo de almacenar una copia de las mismas en el dispositivo.

Para empezar, se ejecuta dSploit, se inicia la búsqueda de objetivos y se elige uno de las estaciones encontradas. Una vez hecho esto, se procede a activar la captura y se esperará a obtener resultados.

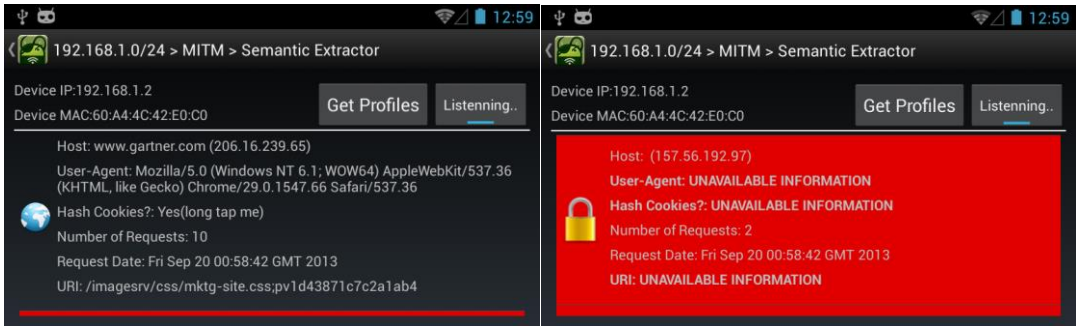


Ilustración 59 - Captura de peticiones HTTP y HTTPS

Como puede verse, el módulo obtiene tanto resultados HTTP como HTTPS, y en cada uno de ellos se muestra la información solicitada. Como indica la interfaz, la petición mostrada en la lista contiene cookies. Para acceder a ellas se debe pulsar detenidamente el elemento de la lista.

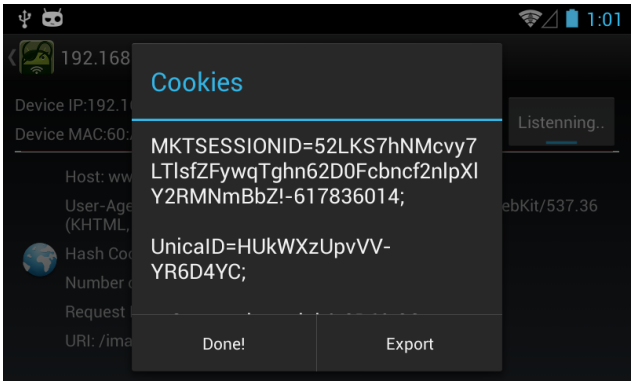


Ilustración 60 - Cookies en una petición HTTP

La interfaz muestra el valor de las cookies. Estas pueden ser exportadas pulsando el botón “export” de la interfaz. Una vez hecho esto, se muestra un mensaje indicando que se han exportado correctamente, para comprobarlo, accederemos al fichero generado a través del explorador de archivos del dispositivo.

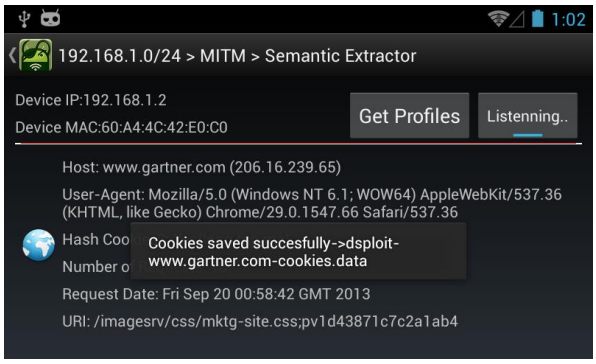


Ilustración 61 - Mensaje indicando el almacenamiento de las cookies

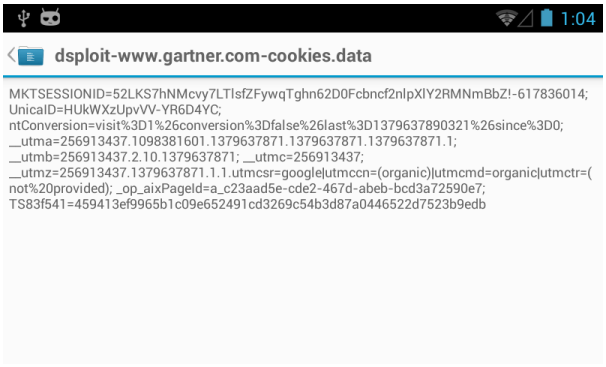


Ilustración 62 - Fichero de cookies abierto desde el dispositivo

Como puede observarse el resultado almacenado en el fichero y el mostrado en la interfaz de dSploit es idéntico, prueba de que la exportación se ha realizado correctamente. Esta información obtenida puede utilizarse para realizar un análisis sobre la seguridad del sitio web a la que pertenecen las cookies.

5.3.2. Pruebas sobre el Módulo de Obtención de Perfiles

En esta subsección se llevarán a cabo pruebas relativas a la obtención de perfiles a partir del tráfico capturado. Para ellos se utilizarán distintos archivos profiles.xml con los que probar esta funcionalidad.

En primer fichero profiles.xml que utilizaremos tendrá únicamente una entrada:

```
<profiles>

    <profile>
        <name>Google_User</name>
        <rule>
            <host>www.google.es</host>
            <number>10</number>
        </rule>
    </profile>
```

Ilustración 63 - Primer archivo profiles.xml de pruebas

Si se captura una petición al dominio google.es y se lleva cabo la búsqueda de perfiles, el perfil definido en el archivo debe ser detectado. Para ello se pone en marcha una sesión de captura y se espera a obtener un paquete enviado al dominio.

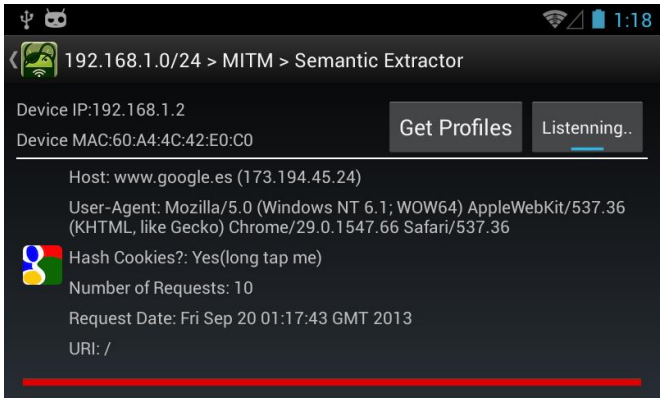


Ilustración 64 - Petición HTTP a google.es capturada

En este instante, se procede a ejecutar la detección de perfiles.

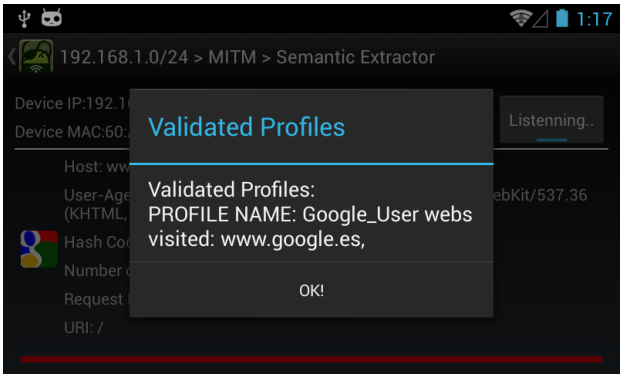


Ilustración 65 - Perfil Google_User validado

Como puede verse el perfil ha sido detectado, y en el diálogo se muestra el dominio que contenía la regla que lo conformaba.

El segundo fichero que se ha utilizado es el siguiente:

```
<profiles>
    <profile>
        <name>Universitario</name>
            <rule>
                <host>www.urjc.es</host>
                <number>1</number>
            </rule>
            <rule>
                <host>www.uc3m.es</host>
                <number>1</number>
            </rule>
    </profile>
</profiles>
```

Ilustración 66 - Segundo archivo profiles.xml de pruebas

Este fichero cuenta con un perfil con dos reglas. Para la validación del perfil se han de encontrar peticiones dirigidas a ambos dominios, para comprobar que el perfil es validado procedemos del mismo modo que en el caso anterior:

Ilustración 67 - Peticiones a www.urjc.es y www.uc3m.es capturadas

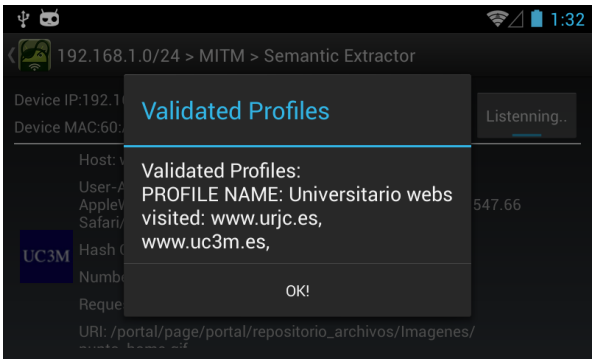


Ilustración 68 - Perfil Universitario validado

Como puede verse el perfil queda validado y en el diálogo se muestran ambos dominios.

El último fichero que utilizaremos será el siguiente:

```
<profiles>
    <profile>
        <allow-or></allow-or>
        <name>Periodicos</name>
        <rule>
            <host>www.elpais.com</host>
            <number>10</number>
        </rule>
        <rule>
            <host>www.elmundo.es</host>
            <number>1</number>
        </rule>
</profiles>
```

Ilustración 69 - Tercer fichero profiles.xml de pruebas

Este fichero contiene un perfil con dos reglas y con la etiqueta <allow-or>. Esta etiqueta indica que únicamente se necesita que una regla se valide para que el perfil sea validado. Para realizar esta prueba se enviarán peticiones a las webs por separado y se comprobará que en ambos casos el perfil ha quedado validado.

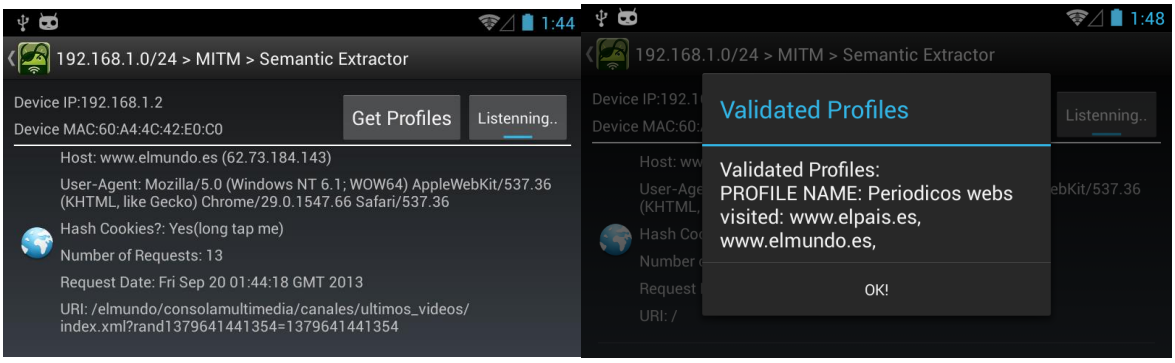


Ilustración 70 - Una sola petición a www.elmundo.es valida el perfil

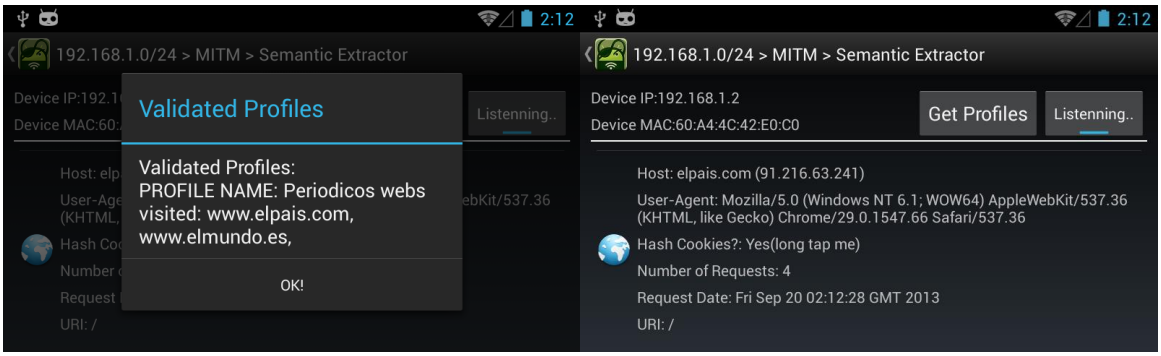


Ilustración 71 - Una petición a www.elpais.com también valida el perfil

Como puede verse en ambos casos el perfil se valida, aunque en ninguna ocasión se validan ambas reglas al mismo tiempo.

5.3.3. Pruebas sobre el Módulo DNS Spoofing

En esta última subsección se probará el funcionamiento del módulo de falsificación DNS. Para ello contaremos con un fichero hosts con el siguiente contenido:

```
##domain     ##fake domain
yahoo.com    03777777066676f6f676c6502657300
```

Ilustración 72 - Fichero hosts de pruebas

En él se indica que todas las peticiones DNS que intenten resolver *yahoo.com*, verán cambiado su dominio por el que aparece en la segunda columna, que en este caso se corresponde con la representación hexadecimal de *google.es*.

Para comprobar que esta modificación se realiza, se utilizará la herramienta de resolución de nombres nslookup integrada en el sistema operativo Windows. Mediante el comando nslookup *yahoo.com* intentaremos obtener la IP del dominio *yahoo.com*, pero en su lugar se deberá obtener la IP de *google.es*.

```
C:\Users\Alex>nslookup yahoo.com
DNS request timed out.
    timeout was 2 seconds.
Servidor:  UnKnown
Address:  80.58.61.250

Respuesta no autoritativa:
Nombre:  www.google.es
Addresses:  173.194.45.31
          173.194.45.23
          173.194.45.24
```

Ilustración 73 - Resolución del dominio yahoo.com

Como podemos comprobar en el resultado de la resolución, la petición a yahoo.es resuelve a las IPs 173.194.45.23, 173.194.45.31 y 173.194.45.24. Si buscamos cualquiera de estas IPs en una base de datos whois, obtenemos los siguientes resultados, asociados a la empresa Google:

```
NetRange:       173.194.0.0 - 173.194.255.255
CIDR:           173.194.0.0/16
OriginAS:       AS15169
NetName:        GOOGLE
NetHandle:      NET-173-194-0-0-1
Parent:         NET-173-0-0-0-0
NetType:        Direct Allocation
RegDate:        2009-08-17
Updated:        2012-02-24
Ref:            http://whois.arin.net/rest/net/NET-173-194-0-0-1
```

Ilustración 74 - Registro whois de Google

.

Por lo tanto la falsificación se ha llevado a cabo de manera exitosa. Efectivamente, si comprobamos el intercambio de paquetes con el sniffer *Wireshark*, podremos ver que el diálogo entre el servidor y la estación ha sido modificado.

```
Standard query 0x0004  A yahoo.com
Standard query response 0x0004  A 173.194.45.23 A 173.194.45.31 A 173.194.45.24
Standard query response 0x0004  A 173.194.45.23 A 173.194.45.31 A 173.194.45.24
```

Ilustración 75 - Detalle de la resolución DNS. El cliente solicita a IP de yahoo.com pero obtiene las de google.es

6. Gestión del proyecto.

En esta sección se discutirán aspectos relativos a la gestión del proyecto realizado, tales como la metodología utilizada, la planificación del desarrollo, el ciclo de vida del software desarrollado, un resumen de los recursos empleado y una breve cronología del mismo.

6.1. Metodología

La metodología seguida durante el desarrollo de este proyecto ha sido basada fundamentalmente en un desarrollo incremental. En esta metodología se definen hitos dentro del desarrollo que se corresponden con ciertas funcionalidades del producto final, cada uno de estos hitos se conoce con el nombre de iteración.

Al final de cada iteración se cuenta con un producto que cubra un subconjunto del conjunto total de funcionalidades definidas por el cliente durante la etapa de análisis y particularmente en la captura de requisitos. Además este modelo de desarrollo permite que se opere con gran flexibilidad a la hora de añadir cambios, ya que al final de cada una de las iteraciones el cliente puede evaluar los resultados y proponer o no cambios en base a los mismos. El desarrollo incremental es especialmente eficiente cuando los hitos se marcan a corto plazo, ya que se evalúan los resultados con más rapidez y el proyecto avanza de manera más sólida, lo que a efectos finales permite pronosticar con mayor exactitud fechas de finalización.

6.2. Planificación

En este apartado queda expuesta de manera resumida la planificación seguida a lo largo del proyecto. Como puede verse en el siguiente diagrama de Gantt las tareas a las que se imputa la mayor carga de tiempo son la implementación, el estudio del código de dSploit y el estudio de la plataforma Android como paso previo al desarrollo de los nuevos módulos. Puede observarse también que durante todo el proyecto se han mantenido un total de tres reuniones de seguimiento en las que han participado las distintas partes involucradas en el proyecto.

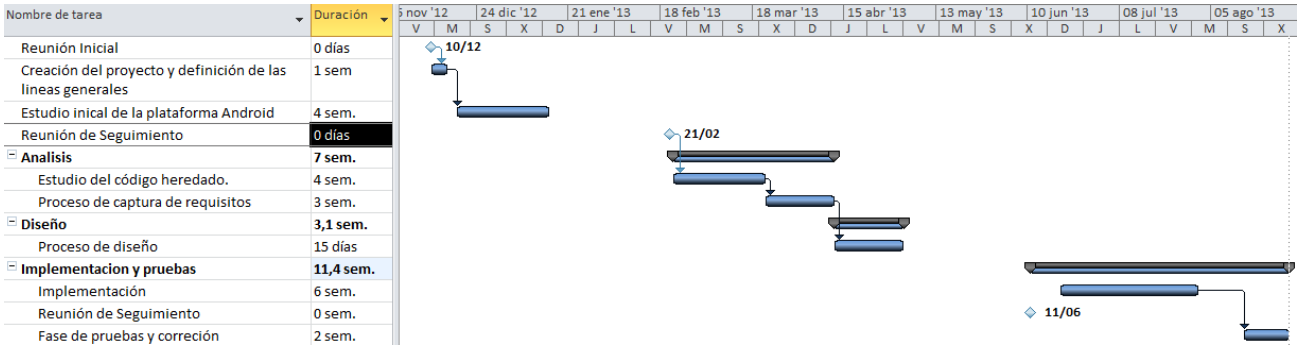


Ilustración 76 - Diagrama de Gantt del proyecto

6.3. Ciclo de Vida

El ciclo de vida de un proyecto establece a las distintas etapas por la que se ha pasado hasta la finalización del mismo y en ocasiones a las etapas futuras. Constituye un mecanismo de aseguramiento de la calidad pues permite a las partes involucradas centrarse en la consecución de cada etapa, lo que favorece el entendimiento entre cliente y desarrollador y la detección temprana de errores en las etapas intermedias.

En este proyecto se ha seguido un ciclo de vida en cascada, que con templa cinco etapas bien diferenciadas y dependiente es entre sí llevadas a cabo de manera secuencial. Otros modelo de ciclo de vida frecuentemente utilizado es el modelo en espiral, en el que se llevan a cabo varias iteraciones sobre las fases de determinación de objetivos, planificación, desarrollo y análisis de riesgo. En el modelo en cascada se definen las fases de análisis, diseño, desarrollo, pruebas y mantenimiento.

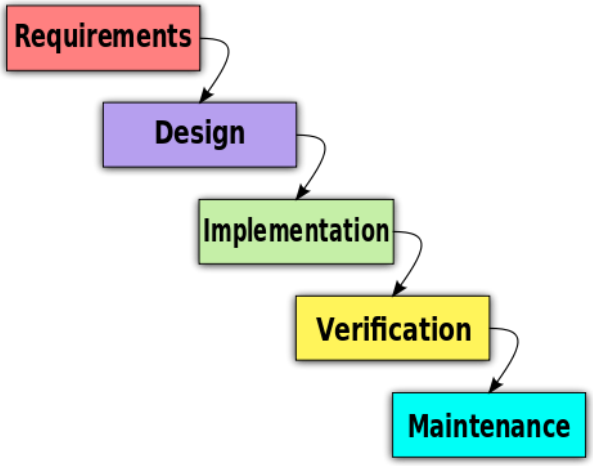


Ilustración 77 - Etapas del desarrollo en cascada [17]

En la Fase de Análisis, se discuten con el cliente las funcionalidades que se desean cubrir con el software desarrollado, para ello se lleva a cabo un proceso de captura de requisitos que finaliza cuando todas las necesidades que debe satisfacer el producto final han sido recopiladas. La facilidad con la que nuevos requisitos podrán ser incorporados al proyecto dependerá en gran medida de la metodología de desarrollo a seguir.

En la fase de diseño se realiza la definición de los componentes lógicos que formarán parte de nuestra aplicación, este proceso se realiza tanto a nivel global como a nivel de cada uno de los subsistemas de la aplicación. El diseño consiste en una traducción de las especificaciones capturadas en la fase de análisis a los mecanismos que se seguirán en la etapa de implementación y que permitirán cubrir dichas especificaciones.

La fase de implementación consiste en la codificación de las funcionalidades solicitadas por el cliente. Esto no se realiza directamente sobre la especificación de requisitos si no que se lleva a cabo sobre el diseño del sistema, de esta manera la traducción de requisito a funcionalidad se lleva a cabo de manera más natural y se minimizan los errores.

En la fase de verificación, se realizan una serie de pruebas con el objetivo de validar los resultados del proceso. Estas pruebas pueden ser de diferentes tipos, según el conjunto de funcionalidades a evaluar y de distinta naturaleza según el enfoque desde el que se realicen, podemos realizar por tanto pruebas unitarias, de integración, de caja blanca, de caja negra, de aceptación, etcétera.

En la fase de mantenimiento, se realiza un seguimiento del uso que los usuarios realizan del producto final con el objetivo de diseñar nuevas funcionalidades y de solucionar problemas que no se han detectado en la fase de pruebas. Esta fase reinicia el ciclo de vida e itera en todas sus fases de nuevo, ya que las nuevas funcionalidades han de pasar por las mismas etapas que el producto inicial.

6.4. Recursos

En esta sección describiremos los principales recursos tanto humanos como materiales empleados en la realización de este proyecto.

6.4.1. Recursos Humanos

El equipo de recursos humanos ha estado formado por tres miembros, dos de los cuales han participado en él desde el mes de diciembre de 2012, incorporándose el tercero en el mes de febrero del 2013 y asumiendo el papel de asesor técnico. La información relativa al equipo queda recogida en las siguientes Tablas

Dirección de proyecto		
Nombre	**Responsabilidad**	**Fecha incorporación**
Jorge Blasco Alís	Director de proyecto	17/12/2012

Tabla 39 - Recursos humanos asociados a dirección de proyecto

Asesoría Técnica		
Nombre	**Responsabilidad**	**Fecha incorporación**
Guillermo Suárez De Tangil-Rotaeche	Asesor Técnico	21/02/2013

Tabla 40 - Recursos humanos asociados a asesoría técnica

Análisis, Diseño y Desarrollo		
Nombre	**Responsabilidad**	**Fecha incorporación**
Alejandro Calleja Cortiñas	Analista	17/12/2012
Alejandro Calleja Cortiñas	Diseñador	17/12/2012
Alejandro Calleja Cortiñas	Desarrollador	17/12/2012

Tabla 41 - Recursos humanos asociados a análisis, diseño e implementación

6.4.2. Recursos Hardware

El Conjunto de los recursos hardware empleados en el proyecto queda desglosado a continuación.

Nombre del Recurso	Tipo	Uso	Horas Uso
Samsung Galaxy (SI9000)	Dispositivo móvil	Desarrollo y pruebas de la aplicación	80
Comtrend WPA 5813	Dispositivo de Red (Router)	Desarrollo y pruebas	80
PC Intel I5	Ordenador de escritorio	Desarrollo y documentación	800
Apple Macbook Pro 13''	Ordenador portátil	Documentación	500

Tabla 42 - Recursos materiales hardware

6.4.3. Recursos Software

Los recursos de naturaleza software utilizados durante este proyecto quedan desglosados a continuación:

Nombre del Recurso	Categoría	Tipo Licencia
Microsoft Windows 7 profesional	Sistema Operativo	Propietaria
Apple Mac OS X mountain Lion	Sistema Operativo	=
Android SDK	Desarrollo	Apache/GNU GPL
Android 4.2.1	Sistema Operativo	=
Eclipse Juno	Entorno de desarrollo	Eclipse Public License
Microsoft Office	Suite ofimática	Propietaria
OpenWrt Barrier Breaker	Firmware Router	GNU GPL
Visual Paradigm	Análisis y diseño	Freeware
Git	Control de versiones	GNU GPL

Tabla 43 - Recursos materiales software

6.5. Análisis de Costes

En base a los recursos analizados anteriormente en esta sección detallaremos el desglose de costes correspondiente a este proyecto. La suma costes final es una mera aproximación del coste exacto. Durante la estimación se considerarán dos principales orígenes del mismo:

- *Costes Directos*: Son los costes derivados de elementos concretos dentro de los recursos, como los recursos humanos y los recursos materiales. El desglose correspondiente a los costes directos queda expuesto a continuación:

Costes Asociados al Empleo de Recursos Humanos					
Nombre	**Rol**	**Trabajo**	**Dedicación (Horas)**	**Coste Hora (€)**	**Coste Total (€)**
Jorge Blasco Alís	Director de Proyecto	Reuniones de Seguimiento	4	45	180
Jorge Blasco Alís	=	Lectura de Documentación	2	45	90
Guillermo Suárez De Tangil-Rotaeche	Asesor Técnico	Reuniones de Seguimiento	3	45	90
Guillermo Suárez De Tangil-Rotaeche	=	Asesoría Estrategias de desarrollo	2	45	90
Alejandro Calleja Cortiñas	Analista	Estudio del trabajo Previo	40	35	1400
Alejandro Calleja Cortiñas	=	Captura De requisitos	240	35	8400
Alejandro Calleja Cortiñas	Diseñador	Diseño Subsistemas y arquitectura.	124	40	4960
Alejandro Calleja Cortiñas	Desarrollador	Desarrollo Funcionalidades acordadas	240	25	6000
Alejandro Calleja Cortiñas	=	Redacción de documentación	80	25	2000
TOTAL RR.HH (€)					23210

Tabla 44 - Costes asociados al empleo de recursos humanos

Costes Asociados A la explotación de recursos hardware			
Nombre del recurso	**Coste Antes Impuestos (€)[1]**	**Coste Después de Impuestos (€)**	**Coste Imputable[2]**
Smartphone Samsung Galaxy S (SI9000)	100	121	20.16
Router Comtrend Wap 5813	30	36.3	6.05
Ordenador personal sobremesa	810.13	983.88	136.98
Ordenador Portátil Apple Macbook Pro	1041.52	1259.61	209.83
TOTAL RECURSOS HARDWARE (€)			373.02

Tabla 45 - Costes asociados a la explotación de recursos hardware

[1] Se imputa un 21% de impuesto sobre el valor añadido (IVA) a todos los recursos de la tabla.

[2] Se calcula el coste imputable (CI) según la siguiente fórmula donde L es la vida útil del recurso (considerada como 60 meses), D es la duración del proyecto (10 meses) y C es el coste del recurso después de impuestos:

$$CI = \frac{C}{L} * D$$

Cotes Asociados al Empleo de recursos Software				
Nombre	**Licencia**	**Coste Antes Impuestos (€)[1]**	**Coste Después Impuestos (€)**	**Coste Imputable (€)[2]**
Microsoft Windows 7 Professional	Propietaria	102.7	120.48	20.08
Microsoft Office	=	298.40	377.73	62.95
Apple Mac OS X mountain Lion	=	14.21	17.99	2.99
Eclipse Juno	Eclipse Public License	-		-
Android SDK	Apache/GNU GPL	-		-
Android 4.2.1	=	-		-
OpenWrt Barrier Breaker	GNU/GPL	-		-
Visual Paradigm	Freeware	-		-
GIT	GNU/GPL	-		-
TOTAL RECURSOS SOFTWARE (€)				86.02

Tabla 46 - Cotes asociados al empleo de recursos software

- *Costes indirectos*: Se trata de costes asociados al uso de un servicio o una infraestructura que no pueden ser imputados directamente a un único elemento dentro de los recursos empleados. Los costes asociados a la

electricidad, el transporte o el alquiler de propiedades pertenecen a este conjunto. Los costes indirectos quedan desglosados a continuación:

Costes Indirectos				
Concepto	**Mensualidad (€)**	**Número de Pagos**	**Total Antes de Impuestos (€)**	**Total (€)**
Electricidad	58	10	580	701.8
Alquiler línea Telefónica Y ADSL	71.77	10	717.70	867.57
TOTAL COSTES INDIRECTOS (€)				1569.37

Tabla 47 - Costes indirectos

Como se desprende del análisis de costes, la mayor parte del mismo está asociado al empleo de recursos humanos. El uso de licencias abiertas como GNU/GPL o Apache ha abaratado sensiblemente los costes asociados al empleo de software.

Costes Totales	
Total Directos (€)	23669
Total Indirectos (€)	1569.37
Coste Final (€)	23669.04

Tabla 48 - Coste final aproximado

6.6. Histórico

Este proyecto fue concebido con la meta de demostrar una vez más las vulnerabilidades típicas de las redes inalámbricas y como pueden ser aprovechadas por un atacante con distintos fines. La primera decisión que se tomó antes de comenzar, en el mes de diciembre, fue la elección de la plataforma más adecuada para llevarlo a cabo, decantando la decisión por la plataforma móvil y concretamente Android debido principalmente a la portabilidad y versatilidad que ofrece. Seguidamente se establecieron las principales funciones del software a desarrollar.

Durante el mes de enero se realizó un estudio a conciencia del sistema Android, tanto a nivel de implementación de aplicaciones como a nivel de sistema operativo, con el objetivo de asentar firmemente los conocimientos sobre esta plataforma. Durante esta etapa se encontraron algunas dificultades con el uso de aplicaciones escritas en código nativo y con la ejecución de programas a nivel de root. Tras una reunión con el director de proyecto y el asesor técnico en febrero, se propuso desarrollar el proyecto como parte de la suite dSploit, reutilizando gran parte de la capa de aplicaciones nativas que se encontraba totalmente desarrollada con el objetivo de reducir la complejidad del proyecto y facilitar el desarrollo. Durante las siguientes semanas se llevó a cabo un análisis del código original y se perfiló la estrategia a seguir para llevar a cabo la integración de los nuevos módulos.

A finales del mes de marzo comenzaron las tareas de análisis y captura de requisitos según las funcionalidades que se acordaron en reuniones previas con la dirección de proyecto, dichas tareas se demoraron hasta mediados del mes de mayo, momento en el que arranco el proceso de diseño.

Una vez objetivo el diseño de los componentes a desarrollar se comenzó la etapa de desarrollo en el mes de mayo. Se programó una reunión para el mes de junio en la que se aclararon dudas acerca de la implementación y las funcionalidades concretas de los módulos que se encontraban en fase de desarrollo. Esta etapa se demoró un total de seis semanas desde su inicio, finalizándose a mediados del mes de agosto. Momento en el cual comenzó la fase de pruebas.

Al final del mes de agosto se comenzó la redacción de la documentación y la memoria del proyecto, que finalizó el día 19 de septiembre.

7. Conclusiones y Trabajos Futuros

En este capítulo repasaremos las conclusiones extraídas en la realización de este proyecto. Se plantearán una serie de contramedidas a los ataques propuestos en secciones anteriores orientadas a incrementar la privacidad de las comunicaciones y se visitará la legislación española involucrada en el desarrollo de este tipo de proyectos.

7.1. Contramedidas a los Ataques Propuestos

A continuación se exponen una serie de contramedidas a los ataques propuestos, aplicables en los distintos niveles del modelo TCP/IP.

7.1.1 TLS/SSL.

SSL (*Secure Socket Layer*) es un protocolo que opera en un nivel intermedio entre las capas de aplicación y transporte y que añade una capa de seguridad a los protocolos típicos del nivel de aplicación como por ejemplo HTTP, cuya implementación sobre SSL se conoce como HTTPS y es ampliamente utilizada en numerosos servicios. SSL fue desarrollado por la empresa de software Netscape en el año 1995 con el objetivo de hacer más seguras las conexiones bajo TCP. En el año 1999 se presentó el sucesor de SSL, TLS (*Transport Layer Security*) que estaba basado en la tercera versión de SSL, presentada en el año 1996 y que introducía nuevos algoritmos de cifrado y mejoraba algunos aspectos de la seguridad del protocolo. Actualmente el protocolo TLS se encuentra en su versión 1.2 que esta especificada en el RFC 5246.

SSL/TLS proporciona confidencialidad e integridad a las comunicaciones a través de la red, estas medidas de seguridad se implementan mediante el uso de criptografía de clave pública y algoritmos criptográficos de clave simétrica.

En una comunicación que haga uso de SSL/TLS podemos distinguir diferentes fases. En primer lugar se produce una fase de *handshake* o salutación entre ambos extremos, en la que se acuerdan los parámetros del intercambio de datos así como los

algoritmos de cifrado, integridad o firma que se utilizarán. De esta forma se crea lo que se denomina una sesión HTPS, que tiene una duración determinada, después de la cual expira y ha de ser renegociada.

La segunda fase, es el intercambio seguro de información. En este proceso el cliente y servidor intercambian información cifrada con los algoritmos y las claves que se han negociado durante la fase de salutación. Los mensajes enviados se acompañan de códigos hash que aseguran la integridad del contenido de los mensajes intercambiados. La sesión que se crea durante la fase de salutación tiene un tiempo de expiración, cuando este tiempo vence se deben de renegociar las claves y protocolos generando así una nueva sesión. Este proceso se lleva a cabo con el objetivo de refrescar los objetos criptográficos que toman partido en la comunicación e impedir que si un tercero tenga acceso a ellos pueda descifrar en tiempo real la comunicación.

Sin embargo SSL/TLS no es un protocolo infalible, en numerosas ocasiones ha sido objetivo de numerosos ataques que han llegado a comprometer de manera efectiva su seguridad. Algunos de los ataques más comunes contra SSL son los ataques de *versión rollback*, mediante los cuales se fuerza a la aplicación que hace uso de SSL a usar versiones del protocolo que cuentan con fallos conocidos y explotables, generalmente versiones antiguas. Sin embargo las vulnerabilidades que hacen posible este ataque han sido parcheadas en las últimas versiones y son cada vez menos comunes, aunque la interoperabilidad entre las distintas versiones del protocolo se ha visto reducida debido a ello. Otro de los ataques más conocidos de contra SSL es conocido bajo el nombre de *SSL stripping*. *SSL stripping* está orientado a inutilizar la seguridad proporcionada por el protocolo en su uso en HTTPS, principalmente se basa en forzar al cliente que intercambia datos con el servidor de manera no segura, sin que este lo perciba. En la siguiente sección describiremos con profundidad este ataque.

7.1.1.1. SSLStripping

En una comunicación bajo SSL ambos extremos acuerdan el uso de unas determinadas claves y protocolos para asegurar la confidencialidad y la autenticidad de la información. El ataque *SSL stripping* está basado en la manera en la que los usuarios hacen uso del protocolo HTTPS durante una sesión de navegación. Cuando nos conectamos a un servidor web a través de HTTPS, la interfaz del navegador advierte al usuario de que la conexión que se está utilizando está protegida por SSL/TLS y muestra los protocolos utilizados así como información referente al certificado del servidor:

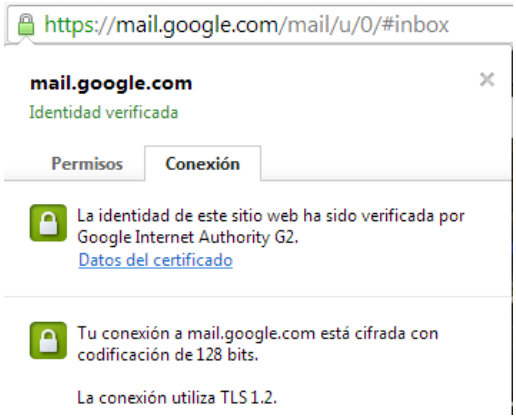


Ilustración 78 - Información sobre la sesión HTTPS mostrada en el navegador

Como puede observarse, la barra de direcciones muestra que se usa el protocolo HTTPS en lugar de HTTP tradicional, sin embargo en la mayoría de escenarios los usuarios no escriben las URLs directamente en el navegador indicando el protocolo a utilizar, sino que simplemente acceden a través de marcadores o escriben el nombre del dominio al que desean conectarse. Por ejemplo, cuando un usuario desea acceder a su buzón de correo protegido por HTTPS, introducirá en la barra de direcciones la URL del servidor, correoseguro.com, lo cual se traduce en el envío de una petición HTTP GET al servidor. Cuando el servidor recibe la petición, advierte que la conexión no es segura y redirige al cliente a una URL alternativa que si utiliza HTTPS puesto que se van a intercambiar datos sensibles como son las credenciales del usuario, para ello envía al equipo del cliente un código de redirección HTTP 301 acompañado de la dirección del servidor seguro.

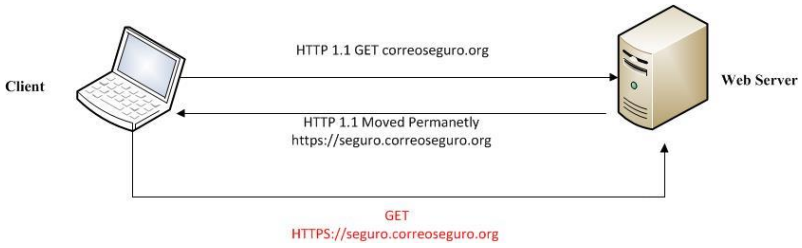


Ilustración 79 - Proceso típico de redirección HTTP a HTTPS

Una vez que el usuario sigue la redirección, se conecta a la nueva URL mediante HTTPS, da comienza la fase de salutación y el túnel queda establecido.

En un escenario en el que el usuario esté conectado a una red insegura en la que un atacante lleve a cabo un ataque *man in the middle* mediante *ARP spoofi*ng, todos los paquetes enviados por el usuario, pasarían primero por el atacante antes de llegar al punto de acceso y viceversa. En este escenario el atacante recibe el tráfico antes que el cliente. Supongamos que el cliente se conecta a un servicio HTTP y el servidor le envía una redirección a una URL que utilice HTTPS, el atacante podría capturar este mensaje y sustituir el protocolo de la nueva URL por HTTP. Cuando este mensaje llega al cliente, se conectara a la nueva URL a través de HTTP, y por lo tanto todo el tráfico que envía queda expuesto al atacante. En este escenario es el atacante el que establece la conexión HTTPS con el servidor web, renviando los datos en claro que el cliente envía por HTTP al servidor seguro, por lo tanto el hombre interpuesto tendrá acceso a las credenciales enviadas en claro por el cliente y podría utilizarlas con la intención de suplantarle ante el servicio.

El ataque *SSL Stripping* fue presentado en el año 2009 en la conferencia *Black Hat* por el investigador Moxie Marlinspike [18] y acorde a lo explicado anteriormente no aprovecha ninguna vulnerabilidad en los protocolos utilizados por SSL si no que aprovecha una debilidad en su diseño y en la manera en que es utilizado por los navegadores. Para demostrar la eficacia del ataque, Marlinspike publico una sencilla aplicación Python que implementaba las líneas teóricas que hacen posible el ataque. A continuación se incluye una demostración práctica, para llevarla a cabo utilizaremos en el servicio de correo *Gmail*, en este ejemplo el usuario con dirección de email

ejemplo.tfg@gmail.com y contraseña "*myPassword*" intenta acceder a su cuenta de correo.

El entorno de pruebas que vamos a utilizar es el siguiente:

- **Equipo víctima:** El equipo desde el que el cliente abre su sesión de correo a través de un navegador comúnmente utilizado como Internet Explorer, particularmente utilizaremos la versión 10.0.9 ejecutándose bajo Windows 7.

- **Equipo atacante**: El equipo desde el que el atacante lleva a cabo el ataque de hombre interpuesto y captura los paquetes enviados desde el cliente, para posteriormente emplear la herramienta *sslstrip* y obtener el tráfico HTTPS generado por el cliente en claro. Este equipo utilizara el sistema operativo Backtrack 5 basado en GNU/Linux.

- **Punto de acceso inalámbrico:** El punto de acceso al que están conectados el equipo de la víctima y el equipo atacante, encontrándose ambos en la misma red local.

El primer paso que debe dar el atacante una vez se encuentra conectado a la red, es conocer las direcciones MAC de los clientes conectados a su mismo punto de acceso. Esto puede llevarse a cabo de numerosas maneras, en este caso el atacante hace uso de la herramienta *nmap* para obtener la lista de clientes conectados al punto de acceso:

Ilustración 80 - nmap mostrando los clientes conectados a la red

El siguiente paso es comenzar el ataque *ARP spoof*, para ello utiliza la herramienta *arpspoof*, invocándola con el siguiente comando:

```
root@bt:~/Desktop# arpspoof -i eth1 -t 192.168.1.2 192.168.1.1
0:c:29:62:b4:52 60:a4:4c:42:e0:c0 0806 42: arp reply 192.168.1.1 is-at 0:c:29:62:b4:52
0:c:29:62:b4:52 60:a4:4c:42:e0:c0 0806 42: arp reply 192.168.1.1 is-at 0:c:29:62:b4:52
```

Ilustración 81 - Ejecución de la herramienta arpspoof

En este ejemplo, 192.168.1.1 es la IP del punto de acceso y 192.168.1.2 la IP de la víctima. Se ha de activar también el IP_FORWARDING en la máquina del atacante para reenviar los paquetes a su dirección IP de destino:

```
root@bt:~# echo "1" > /proc/sys/net/ipv4/ip_forward
```

Ilustración 82 - Activación del reenvió IP en la máquina atacante

Cuando el ataque comienza, se envían masivamente paquetes ARP reply con el objetivo de envenenar la caché ARP del dispositivo víctima tal y como se explicó en la sección 2. En un momento dado, cuando las cachés del objetivo y del punto de acceso contienen la entrada falsa, todo el tráfico que envía y recibe el objetivo pasa por el equipo del atacante. En este momento se debe lanzar, al mismo tiempo que *arpspoof*, la herramienta *sslstrip*. Como paso previo a la ejecución del programa, debemos desviar el tráfico con destino al puerto 80 hacia otro puerto en el que *sslstrip* tenga acceso, para ello incluimos la siguiente regla en el firewall:

```
iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8000
```

Ilustración 83 - Regla NAT para el desvió del tráfico al puerto 8000

Para ejecutar la herramienta se emplea el comando el comando `python sslstrip.py -a -l 8000`, con la opción -a activamos la captura del tráfico HTTP y HTTPS y con la opción -l el puerto al que hemos desviado el tráfico mediante la regla del firewall:

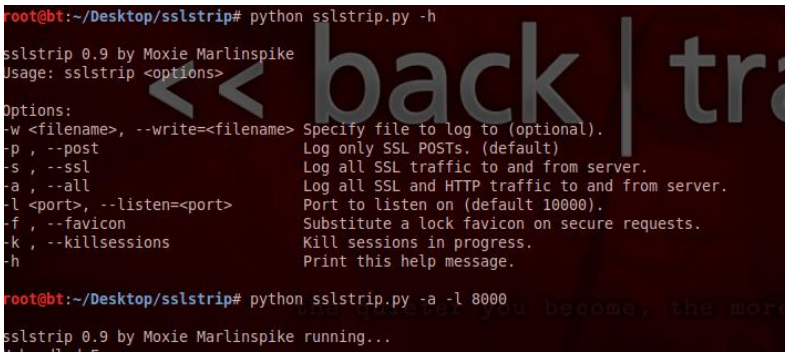


Ilustración 84 - Ejecución de *sslstrip* a través del intérprete de python

De forma paralela a esto, puede utilizarse un *sniffer* como Wireshark para advertir cuando el cliente recibe un mensaje HTTP de redirección (con código 301) lo que supone un indicio de que va a comenzar una comunicación segura:

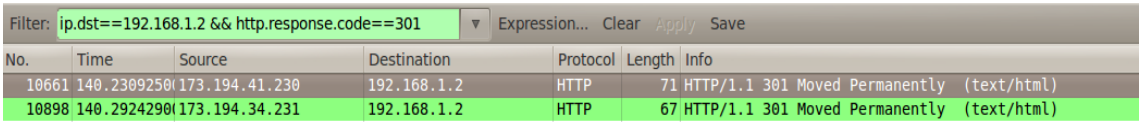


Ilustración 85 - Captura del mensaje HTTP de redirección

Pasados unos instantes, el atacante detiene la herramienta *sslstrip* y comprueba el log generado. Si se ha capturado una sesión SSL, los credenciales enviados por el cliente estarán a disposición del atacante en claro:

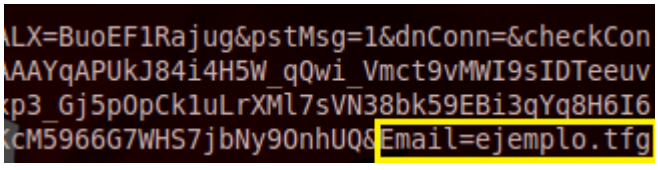


Ilustración 86 - Email de la víctima

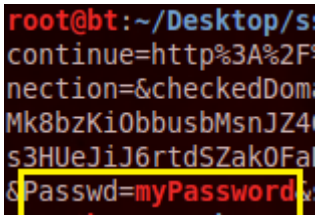


Ilustración 87 - Password de la víctima

En estos momentos el cliente está consultando su correo a través de una conexión HTTP que no está protegida. Si presta atención a la barra de direcciones del navegador, se muestra que el protocolo utilizado es HTTP en lugar de HTTPS:

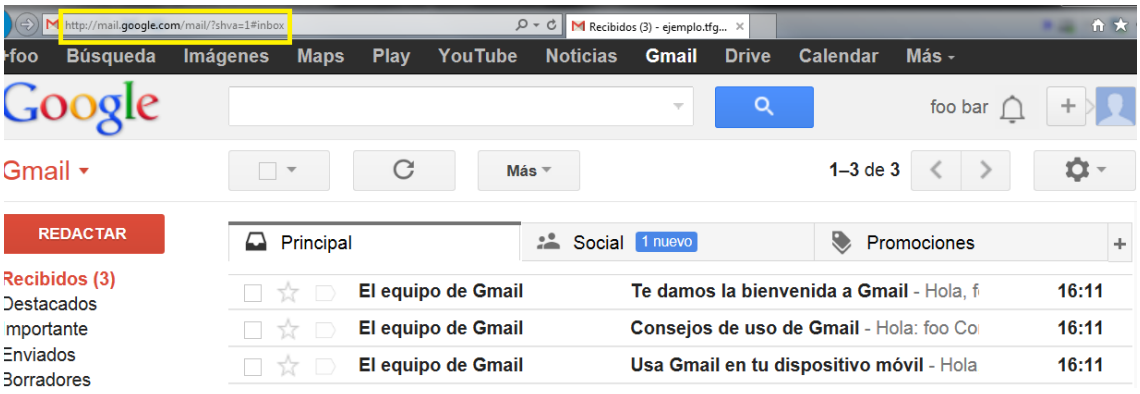


Ilustración 88 - Consulta no segura de correo electrónico

Si bien este ataque no es muy, es altamente eficaz. Para prevenirlo, existen algunas extensiones para los navegadores más comunes que fuerzan la navegación por HTTPS siempre que esté disponible. Sin embargo la práctica más eficaz para prevenir este tipo de ataques es evitar el uso de redes inalámbricas públicas y escribir las URLs completas en la barra de direcciones, especificando HTTPS como protocolo para el intercambio de datos, evitando así las redirecciones.

7.1.1. VPN

Los ataques que se han planteado en este documento, están pensados para llevarse a cabo en escenarios en los que atacante y objetivo se encuentren en la misma red local. Como respuesta a estos ataques pueden establecerse numerosos mecanismos de defensa con el objetivo de paliarlos en la medida de lo posible. Sin embargo, no debemos olvidar lo que sucede con la información que enviamos una vez que esta sale de nuestra red local en dirección a su destino, los paquetes que portan dicha información atraviesan numerosos nodos y pueden no contar con medidas que garanticen la seguridad de los mismos.

Un ejemplo en el que es fácil entender esta situación es aquel en el que enviamos un correo a través de una red local que cuente con alguna media de seguridad que garantice la confidencialidad, como por ejemplo él envió de correo electrónico a través del protocolo SMTP (*Simple Mail Transfer Protocol*) sobre una conexión asegurada con SSL. Cuando el mensaje alcanza el servidor de correo destino, este ha de ser renviado a través de Internet para que llegue a la red del proveedor de correo del destinatario, este envió se hace a través de un canal no seguro, ya que SMTP solo funciona a nivel de aplicación entre el cliente y el servidor de correo.

Una solución a este problema es el uso de las llamadas redes privadas virtuales VPN (*Virtual Private Network*) en estas redes, se forma un túnel entre los distintos equipos de una red pública a través del cual la información enviada permanece confidencial. Esto se logra en numerosas ocasiones mediante el uso del protocolo IPSEC, una extensión del protocolo IP que agrega una capa de seguridad al nivel de enlace, cifrando el contenido de las capas superiores desde el equipo origen antes de entrar en la red insegura. IPSEC ofrece dos modos de operación principales: El primero de ellos es AH (*Autentication Header*) que proporciona a los datagramas de la capa superior integridad de los datos y autenticidad del origen, es decir asegura que los datos no han sido modificados y que el origen de los mismos es quien dice ser, sin embargo este modo no cubre la confidencialidad de los datos. El segundo modo de operación es ESP (*Encapsulation Security Protocol*) que añade confidencialidad a los servicios prestados por el modo AH. Como asegurar la confidencialidad es el principal objetivo de las redes VPN, es el modo que más se utiliza cuando se trata de IPSEC.

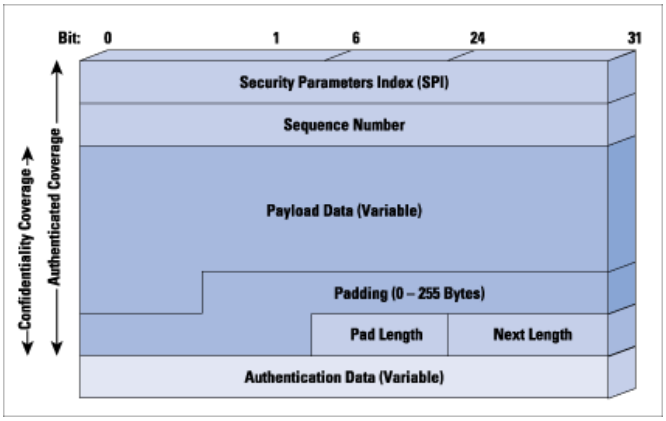


Ilustración 89 - Cabecera IPSEC en modo ESP [19]

Las implicaciones de aplicar mecanismos criptográficos en el nivel de enlace son muy extensas. Algunos de los más importantes son la necesidad de establecer mecanismos de clave pública y de intercambios de claves entre los distintos nodos de la red, con el objetivo de aumentar la eficiencia del cifrado y de los algoritmos de encaminamiento.

7.1.2. Asegurando el Nivel de Enlace

Como se ha explicado en la sección 2.3, la especificación 802.11i proporciona seguridad a nivel de enlace en redes inalámbricas mediante el uso de protocolos de cifrado e integridad, esta seguridad se "garantiza" únicamente en el segmento entre la estación y el punto de acceso. Sin embargo, los protocolos de la familia 802.11 se diseñaron bajo la premisa de que el punto de acceso siempre constituye un eslabón confiable dentro de la red, es decir que siempre va a ser quien dice ser y que no va a ser suplantado bajo ninguna circunstancia. Esta suposición puede ser correcta en algunos casos en los que se aplica la especificación 802.1i, siempre que se utilice autenticación EAP y el cliente y servidor se autentiquen entre sí, pero en muchas ocasiones esto no se cumple.

A menudo el criterio que utilizan los dispositivos móviles para conectarse a redes WLAN depende en gran medida del usuario, sobretodo en entornos en los que se han detectado numerosas redes disponibles. En estos casos los usuarios seleccionan una de las redes y proceden a autenticarse en ella. Sin embargo en numerosas ocasiones, los dispositivos móviles utilizan su propio criterio para conectarse a redes WLAN, basándose en si requieren de un proceso de autenticación y en la potencia de la señal, de tal manera que es posible que si un dispositivo no está correctamente configurado se conecte a una red desconocida por el simple hecho de no requerir de autenticación. Este hecho es algo peligroso, ya que pude dar lugar a un tipo de ataque de suplantación conocido como *Rogue Ap*.

En estos ataques, un punto de acceso es suplantado por otro que se identifica con el mismo SSID, una potencia de señal mayor que el AP original conocido por el usuario

y que se encuentra bajo control de un atacante. En este escenario es habitual que el dispositivo móvil del usuario este configurado para conectarse al AP original de manera predeterminada. Cuando el dispositivo realiza una búsqueda de AP y encuentra ambos puntos de acceso, prioriza la elección en base a la potencia con la que emitan la señal y en este caso elegirá el punto de acceso falso. Esto lleva al cliente a una situación en la que su información se envía directamente al equipo del atacante, que pasa a tener acceso completo a todo el tráfico.

Este tipo de ataques son posibles porque la especificación 802.11 no contempla ningún tipo de protocolo para regular la elección del punto de acceso al que se conecta el cliente. Estos ataques de sustitución pueden ser fácilmente evitados si se comprueba la dirección MAC del punto de acceso, ya que el AP suplantado y el AP falso tendrán direcciones distintas.

Otro aspecto que se debe tener en cuenta a la hora de emplear redes inalámbricas es que quedan expuestas a todo el mundo y si no cuentan con un cifrado lo suficientemente fuerte su uso puede quedar expuesto a terceros que lleven a cabo alguno de los ataques explicados en secciones anteriores y la privacidad de los usuarios legítimos quede expuesta.

Para evitar esto, existe la posibilidad de ocultar el SSID de nuestro punto de accesos. Esto se consigue anulando la emisión de *beacons* por parte del dispositivo, lo que impedirá que sea detectable por las estaciones de las proximidades.

Finalmente, el uso de una contraseña robusta y de un protocolo de cifrado como WPA2 basado en CCMP complica en gran medida los ataques que pueden realizarse en este tipo de entornos que implican que el AP quede comprometido.

7.2. Aspectos Legales

En un proyecto de estas características es necesario incluir una sección que informe al lector acerca de la regulación vigente en España acerca de la seguridad de las comunicaciones y la privacidad.

Actualmente en España el abuso de dispositivos que faciliten o supongan los medios para perpetrar una violación de la integridad la confidencialidad o la disponibilidad de la información ajena o de un sistema informático está considerada delito, según queda tipificado en el convenio europeo de ciberdelincuencia, ratificado por numerosos países europeos en el año 2001 y que ha ido sumando apoyos de otros países extra comunitarios durante los últimos años [20].

Según esto, el uso de un dispositivo móvil para llevar a cabo alguno de los ataques que se han expuesto en esta memoria queda considerado como una violación de la legislación vigente en lo que se refiere a delitos informáticos.

Durante la realización de este proyecto, todas las pruebas que se han realizado han sido llevadas a cabo en un entorno controlado y jamás han implicado poner en peligro o conocimiento público información privada de ninguna persona ajena al mismo. Asimismo, todo el desarrollo realizado ha sido llevado a cabo con fines meramente académicos y en ningún caso con la intención de utilizarlo en un entorno público.

7.3. Líneas Futuras

En esta sección, listaremos algunas de las líneas de trabajo futuras, pensadas para incluir nuevas funcionalidades en la suite dSploit o mejorar las ya incluidas:

- **Implementación de SSLStrip utilizando el servidor HHTP incluido en la aplicación:** Una propuesta interesante seria incluir la funcionalidad necesaria para llevar a cabo un ataque de *SSLStripping* tal y como el que se expone en la sección anterior. Además se podría reutilizar para ello el servidor web que

contiene dSploit para manejar la comunicación por mensajes HTTP con la victima mientras que se mantiene la conexión HTTPS con el servidor.

- **Mejorar el algoritmo de validación de perfiles:** El algoritmo utilizado actualmente para validar los perfiles, se basa en recorrer de forma serial una lista que contiene los diferentes perfiles y por cada entrada de esa lista se recorre otra con todas las peticiones HTTP interceptadas. Este algoritmo tiene una complejidad $O(n^2)$, optimizar este proceso reduciría el gasto de batería, que supone un aspecto prioritario en el desarrollo de aplicaciones móviles.

- **Añadir peticiones GET HTTPS al validador de perfiles:** Debido a que las peticiones HTTP se envían cifradas al servidor, no podemos extraer demasiada información acerca de ellas salvo la dirección IP de destino y el nombre de la red a la que van dirigidas. Una mejora futura podría construirse en torno a permitir que las validaciones de perfiles utilicen también esta escasa pero valiosa información.

- **Visor de ficheros PCAP:** Si bien la aplicación permite actualmente salvar sesiones de captura de paquetes en formato PCAP, no contiene ninguna funcionalidad que permita visualizar el contenido de este tipo de ficheros, tal y como se hace en numerosos programas de escritorio como por ejemplo *wireshark*. Esta funcionalidad permitiría ampliar las funciones de dSploit y lo convertiría un laboratorio portátil muy completo, ya que contaría con todas las herramientas necesarias para llevar a cabo un completo análisis de la red a la que conectemos el dispositivo.

7.4. Conclusiones Finales

Una vez concluida la elaboración de este proyecto, es momento de exponer las conclusiones extraídas de su desarrollo.

En primer lugar, a título personal, la realización de este proyecto me ha permitido poner en práctica conocimientos adquiridos en numerosas asignaturas. En el ámbito de los sistemas operativos, me ha obligado a utilizar diferentes entornos bajo diferentes plataformas, lo que ha ayudado a fijar conocimientos previos en esta materia y a desarrollar mis capacidades en lo que se refiere al uso de sistemas Unix. En el terreno de las redes, me ha ayudado a repasar los modelos arquitectónicos en los que se apoyan las redes modernas, como por ejemplo la propia Internet. Además, el desarrollo me ha permitido estudiar en profundidad protocolos como DNS, ARP o HTTP. En lo que se refiere a la seguridad de la información, ha sido el área de conocimiento que más he desarrollado con diferencia, ya que ha sido necesario estudiar numerosos protocolos, vulnerabilidades ataques y conceptos teóricos relaciones con el ámbito de la seguridad informática. Por último, también he adquirido nuevos conocimientos acerca de la gestión de proyectos y la ingeniería del software, sus métodos, estándares y técnicas.

La realización de este proceso también me ha ayudado a comprender mejor el desarrollo en de aplicaciones para el sistema operativo Android, y me ha permitido advertir la cantidad de posibilidades que tiene.

En otro sentido, se ha realizado una completa exposición de los riesgos del uso de redes inalámbricas de área local, analizando los principales ataques que pueden vulnerar la confidencialidad, la integridad y la disponibilidad de la información en estos entornos. La principal intención de esto ha sido la de concienciar a futuros lectores acerca de estos peligros y fomentar la puesta en práctica de políticas de seguridad y buenas practicas que ayuden a los usuarios a protegerse antes estos posibles ataques.

ANEXO I

Bibliografía y Recursos

- [1] *Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First time* disponible en *http://www.gartner.com/newsroom/id/2573415 consultado en el mes de agosto de 2013.*

- [2] *Android System Architecture* disponible en *http://en.wikipedia.org/wiki/File:Android-System-Architecture.svg consultado en el mes de septiembre de 2013.*

- [3] *Android Developer. Platform Versions.* disponible en *http://developer.android.com/about/dashboards/index.html consultado en el mes de septiembre de 2013.*

- [4] *Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains disponible en http://www.idc.com/getdoc.jsp?containerId=prUS24257413 consultado en el mes de agosto de 2013.*

- [5] *DroidSheep http://droidsheep.de/*

- [6]*FaceSniff http://faceniff.ponury.net/*

- [7] *WifiKill http://ponury.net/*

- [8] *dSploit http://dSploit.net/*

- [9] *Repositorio en Github de dSploit https://github.com/evilsocket/dSploit*

- [10] *Redes de Computadores, un enfoque descendente. James F. Kurose, Keith W. Ross. Capítulo 6 Redes móviles e inalámbricas. WiFi Redes Lan Inalámbricas 802.11.*

- [11] *Basic WEP encryption* disponible en *http://en.wikipedia.org/wiki/File:Wep-crypt-alt.svg consultado en el mes de agosto de 2013.*

- [12]*Weakness in the Key Scheduling Algorithm of RC4* disponible en *http://www.crypto.com/papers/others/rc4_ksaproc.pdf consultado en el mes de agosto de 2013.*

- [13] *Security of mobile Comunications. Noureddine Boudriga. Chapter 7. Wireless Local Area Netwoks Security.*

- [14] *LAN Switch Security: What Hackers Know About Your Switches. Eric Vyncke, Christopher Paggen. Chapter 6. Exploiting IPv4 ARP*

- [15] *Transmission Control Protocol* disponible en *http://commons.wikimedia.org/wiki/File:Tcp-handshake.svg*

- [16] *Android Developer. Managing the Activity Lifecycle* disponible en *http://developer.android.com/guide/components/activities.html*

- [17] *Waterfall model* disponible en *http://en.wikipedia.org/wiki/Waterfall_model*

- [18] *New Tricks For Defeating SSL In Patrice* disponible en *http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf* consultado en el mes de agosto de 2013.

- [19] *IP Security* disponible en *http://www.cisco.com/web/about/ac123/ac147/ac174/ac197/about_cisco_ipj_archive_article09186a00800c830b.html* consultado en el mes de septiembre de 2013.

- [20] *Legislación Española referente a delitos informáticos* disponible en *https://www.gdt.guardiacivil.es/webgdt/legislacion.php* consultado en el mes de septiembre de 2013.

- [21] *Hacking Exposed™ Wireless: Wireless Security Secrets & Solutions. Johnny Cache; Joshua Wright, Vincent Liu.*

- [22] *Dictionary of Information Security, Robert Slade.*

- [23]*Curso de seguridad en dispositivos móviles. Grado en Ingeniería Informática, especialidad en Ingeniería de Computadores. Universidad Carlos III de Madrid. Agustín Orfila Díaz-Pabón, Guillermo Suárez De Tangil-Rotaeche.*

- [24]*Repositorio GitHub del TFG https://github.com/alximw/DSPloit_TFG*

ANEXO II

dSploit

Manual De usuario

Para los módulos Semantic Extractor y DNSSpoofing

Índice

1. Sobre dSploit

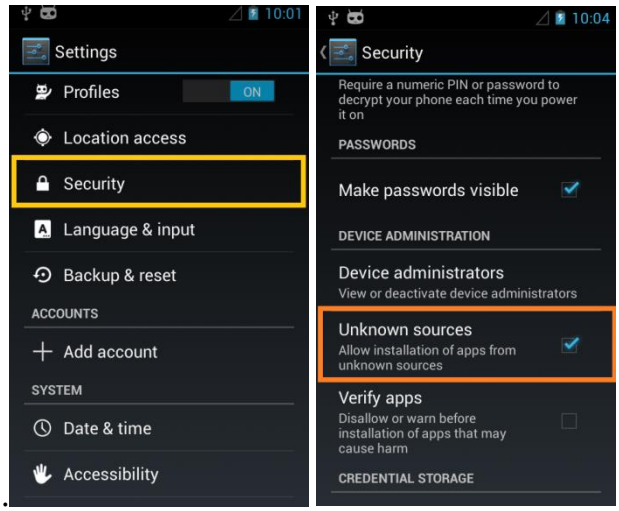
dSploit es un suite de herramientas para la plataforma móvil Android que permite llevar a cabo labores de análisis y *pentesting* sobre la red inalámbrica a la que el dispositivo esté conectado. Las herramientas incluidas en dSploit permiten realizar tareas tales como la identificación de sistemas operativos, detección de vulnerabilidades, el despliegue de un ataque *man in the middle*, *hijacking* de sesiones web, manipulación del tráfico en tiempo real etcétera. En general, permite realizar muchas de las tareas que pueden llevarse a cabo con un PC desde las principales distribuciones dedicadas a la seguridad, como por ejemplo Backtrack, con la ventaja añadida de la portabilidad que ofrece un dispositivo móvil.

En las siguientes secciones se expondrá detalladamente el modo en el que el usuario debe operar la aplicación para obtener los mejores resultados. Comenzaremos con una breve explicación sobra la instalación de la aplicación en el terminal, para continuar con el funcionamiento de la función de recuperación de información semántica y finalizar exponiendo el funcionamiento del módulo de *DNS Spoofing*

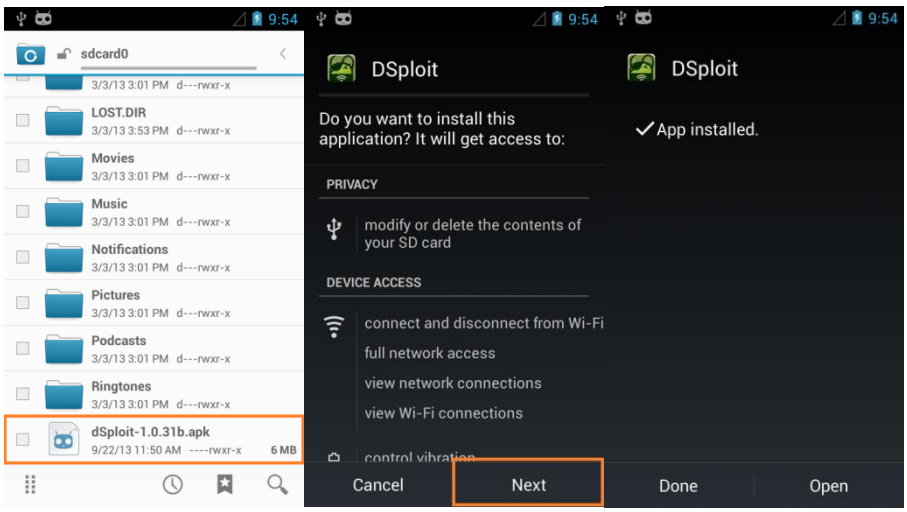
2. Instalación

Debido a las funcionalidades que ofrece dSploit, la aplicación no se encuentra disponible en la plataforma de distribución Google Play. Para conseguirla, es necesario descargar el fichero *apk* contendor de la *app* desde el repositorio en GitHub de su desarrollador. Para que dSploit pueda funcionar correctamente en el dispositivo, se debe contar con una instalación completa de BusyBox (disponible para descarga en Google Play) y además se debe tener acceso como root al mismo.

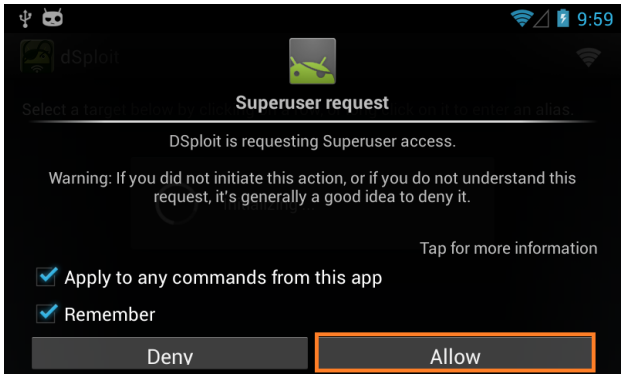
Una vez que se ha obtenido el fichero dSploit-1.0.31b.apk es necesario transferirlo al dispositivo. Como paso previo a la instalación es preciso configurar el sistema para permitir la instalación de aplicaciones de fuentes desconocidas (es decir, aplicaciones que no han sido descargadas de Google Play). Para ello, el usuario debe dirigirse al menú de ajustes, y en la lista seleccionar seguridad. En este menú se debe marcar la casilla “Fuentes desconocidas” o “Unknown sources”, dependiendo del idioma del dispositivo.



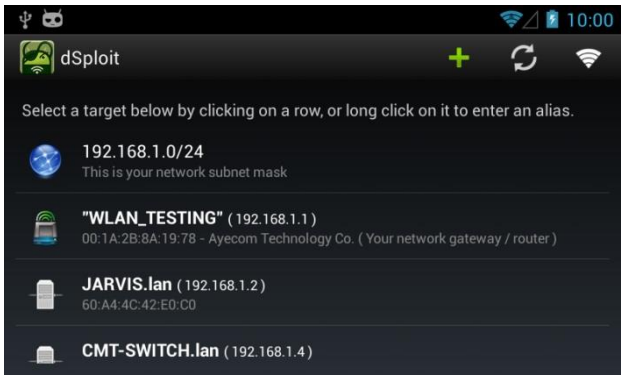
Una vez modificado este ajuste, ya puede comenzar la instalación. Para ello se localizará el fichero *apk*, copiado previamente, mediante el explorador de ficheros del sistema y se seleccionará para arrancar el proceso de instalación.



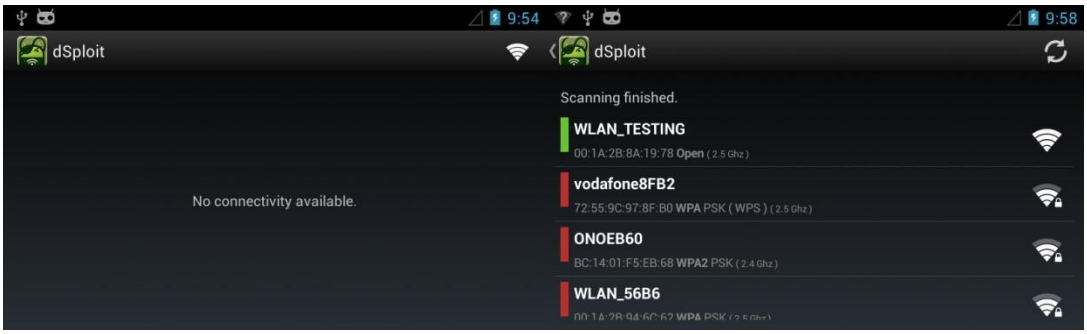
El proceso de instalación se demorará unos segundos. Al comienzo, dSploit solicitará permisos de superusuario.



Una vez finalizado, la aplicación se abrirá y durante unos instantes copiará las herramientas que contiene al almacenamiento del dispositivo. Cuando la pantalla de selección de objetivos se muestre, la aplicación está lista para su uso.



Si el dispositivo no se encuentra conectado a internet en el momento de la ejecución, la aplicación no mostrará la actividad de selección de objetivos, en su lugar mostrará una interfaz de selección de las redes inalámbricas detectadas. Cuando el dispositivo se conecte a una red inalámbrica, se mostrará la actividad de selección de objetivos.

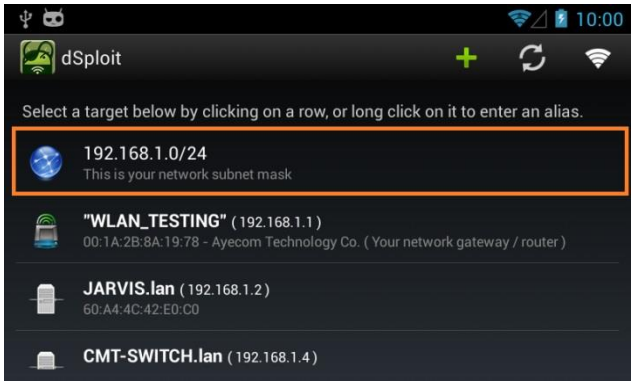


3. Uso de la función *Semantic Extractor*

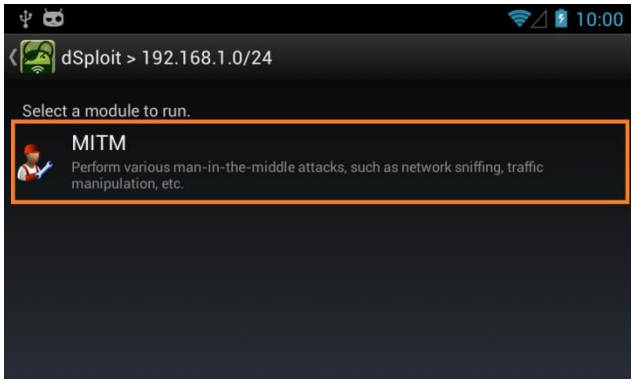
Esta función permite la captura en tiempo real de las peticiones HTTP y HTTPS que el objeto envía a la red, con el objetivo de conocer datos referentes al usuario como por ejemplo las páginas web que visita, el dispositivo que utiliza o el contenido de las cookies intercambiadas. Además permite la identificación de perfiles en base al tráfico web generado.

Para utilizar esta función, se seguirán los siguientes pasos:

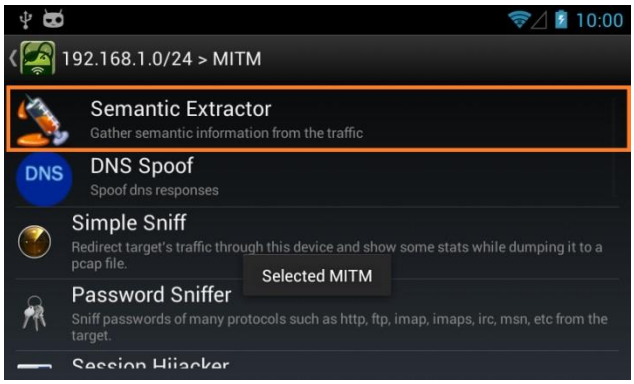
- 1. Dentro del menú de selección de objetivos se seleccionará la subred a la que el dispositivo esté conectado, en la mayoría de ocasiones esta opción aparece en la primera posición de la lista:



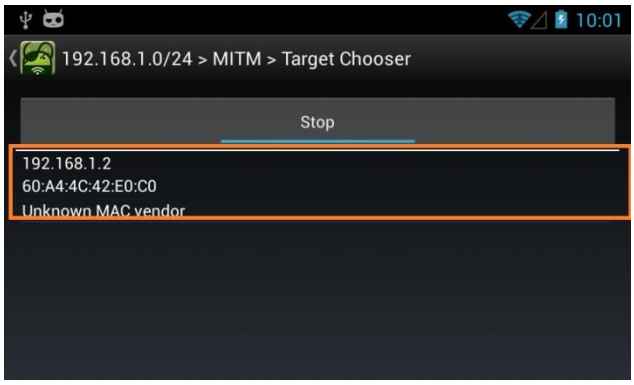
- 2. En la lista de opciones, se seleccionará la modalidad de ataques MITM (*man in the middle*):



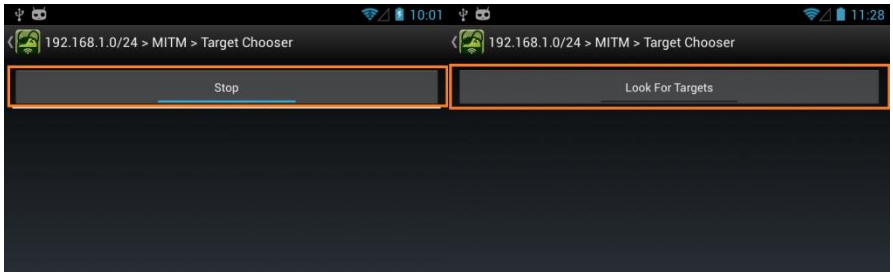
- 3. En la siguiente lista, se muestran las distintas herramientas de dSploit que utilizan la técnica MITM. Para Utilizar la función de recuperación de información semántica el usuario seleccionará la primera de las opciones "Semantic Extractor" :



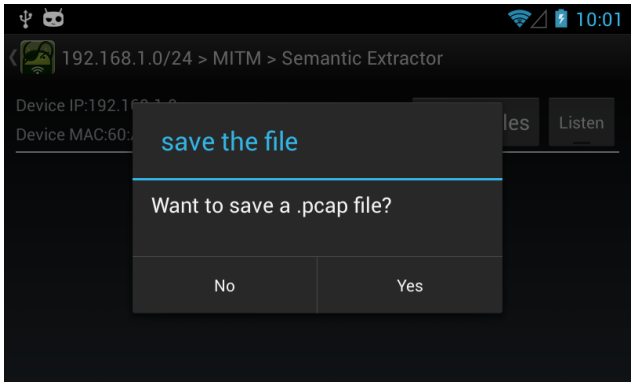
- 4. Una vez seleccionada la herramienta, se mostrará la pantalla de búsqueda de objetivos que actualmente estén intercambiando mensajes del protocolo HTTP o HTTPS. El proceso de búsqueda empieza en el momento en que se muestra la interfaz, los objetivos encontrados aparecerán en la lista:



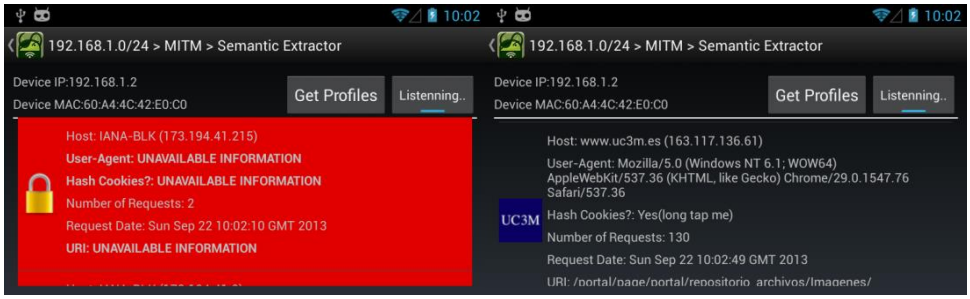
Para seleccionar uno de los objetivos encontrados, basta con pulsar sobre él. Si en algún momento se desea reiniciar la búsqueda, basta con pulsar sobre el botón “Stop” de la interfaz y pulsar de nuevo para iniciarla:



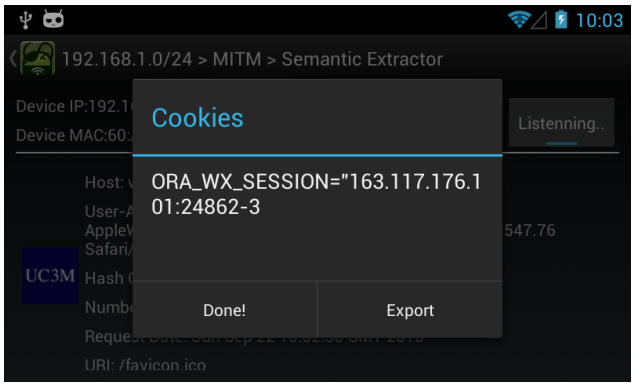
- 6. Una vez que se ha seleccionado el objetivo, se mostrará un diálogo, permitiendo al usuario generar un archivo .pcap con el contenido de los mensajes capturados:



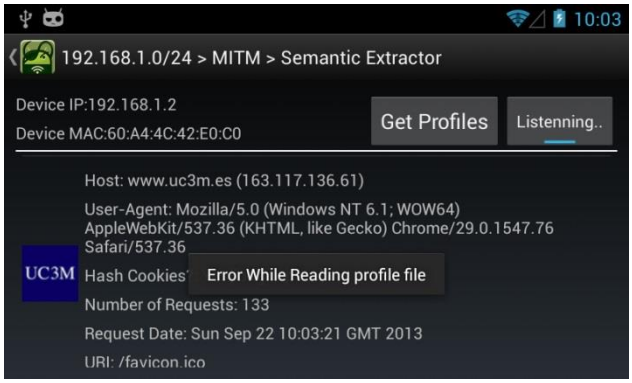
- 7. Cuando ya se ha elegido una de las dos opciones, se mostrará la interfaz en la que se listan los mensajes HTTP HTTPS capturados. Los mensajes sobre HTTPS se mostrarán sobre un fondo rojo, distinguiéndolos de mensajes HTTP convencionales:



Para conocer las cookies enviadas en una petición HTTP, basta con localizar un mensaje que las contenga y mantener pulsada la interfaz durante un segundo. Las cookies aparecerán en un diálogo y se ofrecerá al usuario la posibilidad de exportarlas al sistema de archivos:



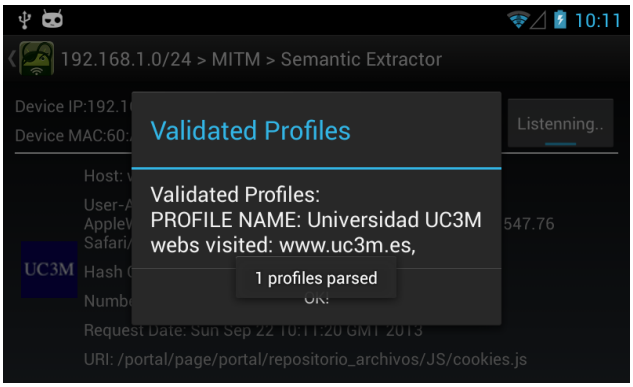
Para utilizar la función de identificación de perfiles, es necesario contar con un fichero profiles.xml localizado en la raíz del sistema de archivos del dispositivo. En caso contrario, al activarla mediante la pulsación del botón "Get Profiles", se mostrará el siguiente mensaje de error:



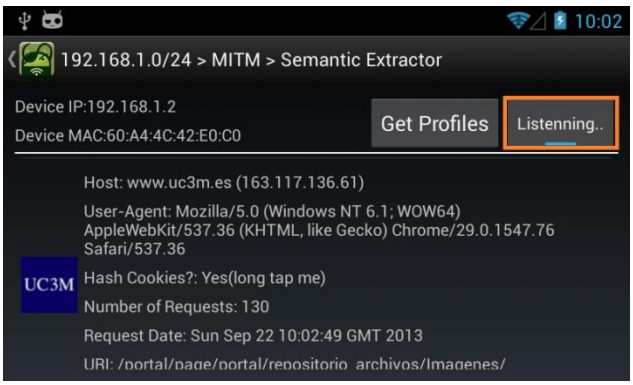
El fichero profiles.xml debe tener una estructura similar al que se muestra a continuación:

```
<profiles>
        <profile>
            <allow-or></allow-or>
            <name>Estudiante</name>
                <rule>
                    <host>www.uc3m.es</host>
                    <number>1</number>
                </rule>
        </profile>
</profiles>
```

Siendo la etiqueta <allow-or> opcional. Si el fichero se encuentra en la ubicación especificada, al pulsar sobre el botón "Get Profiles" comenzará la búsqueda de perfiles. En caso de encontrar alguno que valide el tráfico capturado se mostrará el siguiente diálogo, indicando el nombre del perfil y las URLS visitadas:



Si se desea detener en algún momento la captura, bastará con pulsar sobre el botón "Listenning":

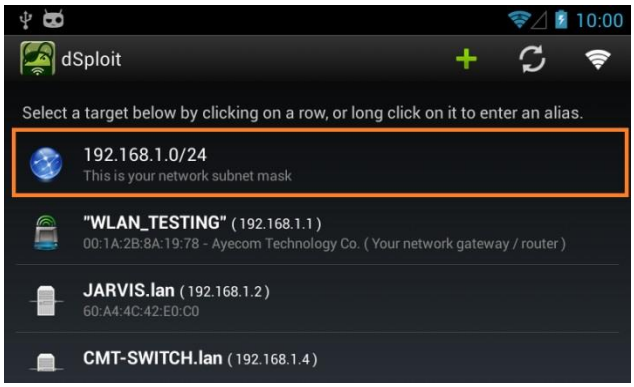


4. Uso de la función *DNSSpoof*

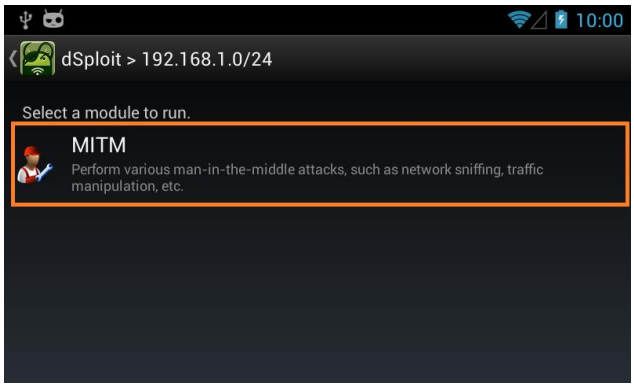
Esta función permite la modificación en tiempo real de las peticiones DNS que el objetivo envía a la red, con el objetivo de redirigir al objetivo a un dominio distinto del solicitado.

Para utilizar esta función, se seguirán los siguientes pasos:

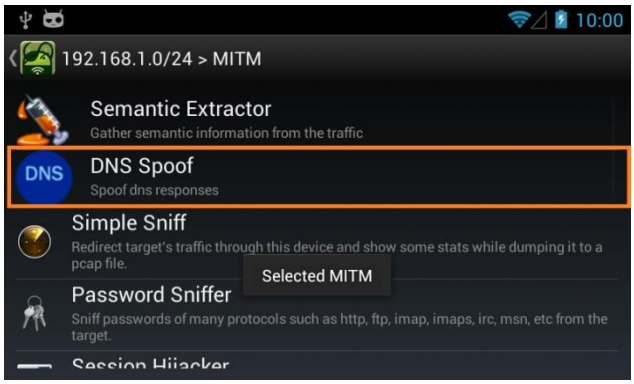
- 1. Dentro del menú de selección de objetivos se seleccionará la subred a la que el dispositivo esté conectado, en la mayoría de ocasiones esta opción aparece en la primera posición de la lista:



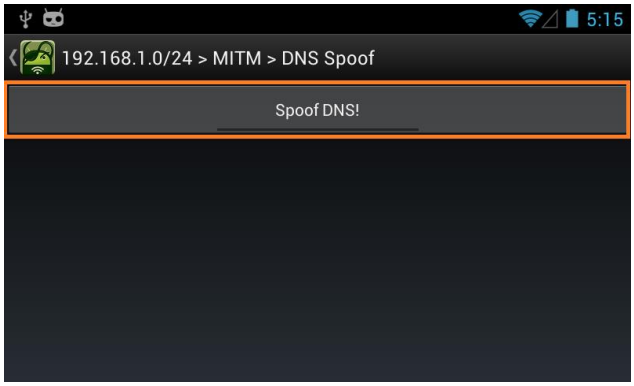
- 2. En la lista de opciones, se seleccionará la modalidad de ataques MITM (*man in the middle*):



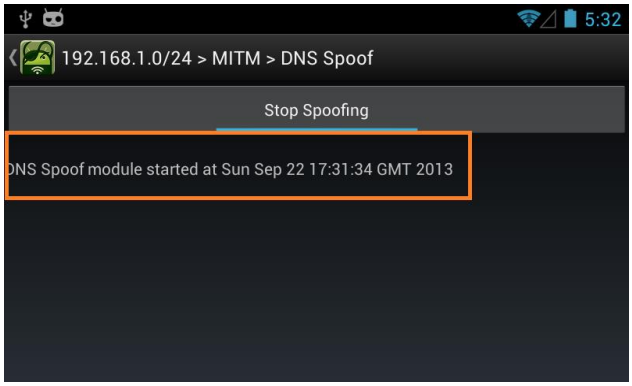
- 3. En la siguiente interfaz, se muestran las distintas herramientas de dSploit que utilizan la técnica MITM. Para utilizar la función de recuperación de información semántica el usuario seleccionará la primera de las opciones “DNS Spoof” :



- 4. Una vez seleccionada la herramienta, se mostrará la siguiente interfaz. Para comenzar el ataque de DNS Spoofing, el usuario deberá pulsar sobre el botón "Spoof DNS":



Cuando se inicie la herramienta, la interfaz mostrará un mensaje indicándolo:

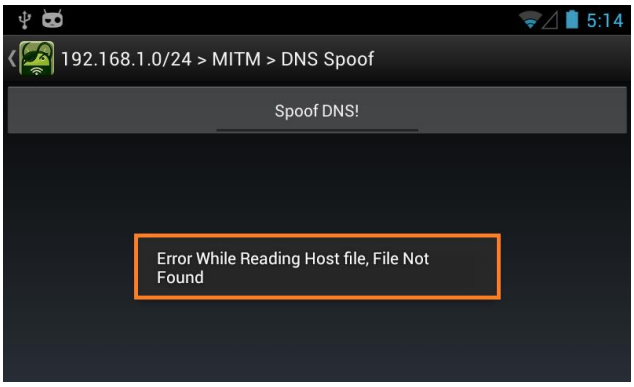


Una vez que la herramienta está ejecutándose, las consultas DNS se modificarán o reenviarán según corresponda.

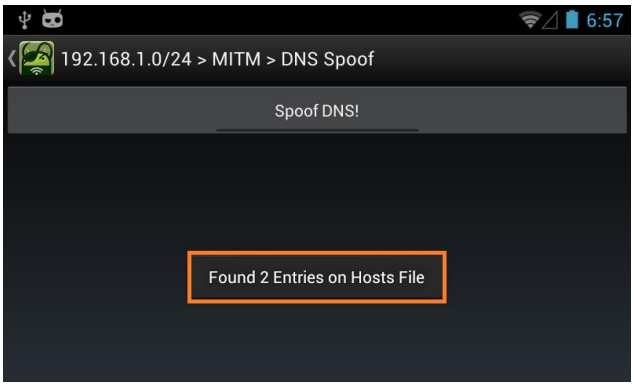
Para utilizar esta funcionalidad, es necesario contar con un fichero hosts en la raíz del sistema de ficheros. El fichero hosts almacena las equivalencias entre los dominios consultados y los dominios por los que serán sustituidos. Su estructura es la siguiente:

```
##domain     ##fake domain
yahoo.com    0377777706676f6f676c6502657300
```

En la primera columna, se especifican los dominios que se buscarán en las consultas capturadas y en la segunda columna se especifica en codificación hexadecimal, el dominio por el que se sustituirán en la consulta modificada. Si no se incluye un fichero de estas características, se mostrará el siguiente mensaje de error al iniciar la herramienta:



Si se ha localizado un fichero hosts correcto, se indicará el número de entradas encontradas en él:



Para detener la herramienta bastará con pulsar sobre el botón "Stop Spoofing". Se mostrará un mensaje indicando que el módulo se ha detenido.