
UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

# Plataforma de análisis de aplicaciones móviles

Autor:

**D. Adrián Rojo Becerril**

Tutores:

**D. Álvaro Castellanos Andrés**

**D. Rubén Mateo Lorenzo**

Valladolid, 21 de Septiembre de 2020

| | |
|---|---|
| TÍTULO: | **Plataforma de análisis de aplicaciones móviles** |
| AUTOR: | **D. Adrián Rojo Becerril** |
| TUTOR: | **D. Álvaro Castellanos Andrés** |
| | **D. Rubén Mateo Lorenzo** |
| DEPARTAMENTO: | **Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática** |

**TRIBUNAL**

| | |
|---|---|
| PRESIDENTE: | **Dr. D. Evaristo Abril Domingo** |
| VOCAL: | **Dr. D. Rubén M. Lorenzo Toledo** |
| SECRETARIO: | **Dra. Dª. Patricia Fdez del Reguero** |
| SUPLENTE: | **Dr. D. Ramón Durán Barroso** |
| SUPLENTE: | **Dr. D. J. Carlos Aguado Manzano** |

| | |
|---|---|
| FECHA: | **18 de septiembre de 2020** |
| CALIFICACIÓN: | |

En colaboración con Orange España.

«*Per Aspera Ad Astra*»

**Séneca**

# Agradecimientos

Quiero agradecer a mis tutores, Álvaro y Rubén por su supervisión y valiosos consejos. A Miller, Eduardo y al resto de gente de Orange España por la oportunidad de realizar este proyecto.

A mi familia por todo el apoyo y la ayuda que me han dado durante estos cuatro años.

A Daniel, por su indispensable ayuda para afrontar la carrera.

Y por último agradecer los buenos momentos a mis amigos, tanto los que ya traía como a los que he conocido durante estos cuatro maravillosos años.

Gracias a todos, por todo.

## Palabras Clave

Ciberseguridad, análisis estático, análisis dinámico, Aplicaciones móviles, Android, iOS, Windows Phone, formato APK, MobSF, apklab.io, pila ELK, Elasticsearch, Logstash, Kibana.

## Resumen

En este proyecto se aborda cubrir la necesidad detectada en la mitigación de riesgos de seguridad en los usuarios como es el análisis desde el punto de vista de seguridad de las aplicaciones que instalan en los teléfonos inteligentes los usuarios. Este análisis requiere ser realizado desde dos aproximaciones diferentes:

- De manera estática: revisión del código fuente de la aplicación móvil y extracción de los componentes que la forman, identificación de estructuras sospechosas en el código, obtención de direcciones IP y web.
- De manera dinámica: monitorizando actividad de la aplicación durante su uso, siendo posible obtener los recursos a los que accede la aplicación, tráfico que esta genera.

Para ello se ha recopilado información del sistema operativo Android, sus aplicaciones y se han probado diversas herramientas de análisis de aplicaciones disponibles en el mercado, eligiendo para el desarrollo de la plataforma la herramienta **Mobile-Security-Framework (MobSF)** pues nos permite realizar el análisis desde las dos aproximaciones antes mencionadas. Para la realización del análisis dinámico es usado un dispositivo virtual con Android. Se documenta el manejo de la herramienta, así como el significado de los reportes generados.

Como factor diferencial, se usa el dispositivo móvil no virtual con el fin de obtener el tráfico generado por la aplicación durante el análisis dinámico, aportando una perspectiva más real al análisis. Para la obtención de este tráfico se crea un punto de acceso malicioso en el que se obtiene el tráfico del dispositivo de forma transparente para el dispositivo y el usuario que lo maneja.

Una vez obtenidos los reportes, son cargados en una base de datos basada en la tecnología de la pila de Elasticsearch, Logstash y Kibana con el fin de almacenarlos e interpretarlos. Buscando el encontrar una conexión con los resultados de los análisis obtenidos y la información del estado de la red de los que dispone Orange.

Para ilustrar el proyecto, varias aplicaciones móviles son analizadas por la herramienta MobSF, comentando los resultados obtenidos.

# Índice

# Índice de Figuras

# Índice de Tablas

# Glosario y acrónimos

| | |
|---|---|
| IoT | Internet de las Cosas |
| 5G | Quinta generación de tecnologías de telefonía móvil |
| GPS | Global Positioning System |
| IP | Internet Protocol |
| INCIBE | Instituto Nacional de Ciberseguridad |
| RAM | Random Access Memory |
| SSD | Solid State Disk |
| HDD | Hard Drive Disk |
| AP | Access Point |
| SSH | Secure SHell |
| .apk | Formato de aplicación en el sistema operativo Android |
| .xml | Extensible Markup Language |
| ROM | Read Only Memory |
| GSM | Global System for Mobile communications |
| 3G | Tercera generación de tecnologías de telefonía móvil |
| LTE | Long Term Evolution o Cuarta generación de tecnologías de telefonía móvil |
| API | Application Programming Interfaces |
| CVSS | Common Vulnerability Score System |
| CWE | Common Weakness Enumeration |
| OWASP | Open Web Application Security Project |
| HTTP | Hypertext Transfer Protocol |
| ELK | Elastichsearch, Loghstash y Kibana |

# Capítulo 1: Introducción

Con la llegada de los dispositivos inteligentes el mundo ha cambiado. Tus datos ya no solo están almacenados en tu ordenador en un lugar seguro como tu casa. Ahora siempre llevas contigo información que te identifica, datos personales, financieros en tu bolsillo, allá donde vas. Y esta tendencia va a continuar, favorecida por la introducción del Internet de las Cosas (IoT) así como el 5G con lo que aumentará de forma significativa el número de dispositivos conectados, así como la aparición de múltiples aplicaciones para su control. En el año 2020 hay 5.190 millones de dispositivos en el mundo. Actualmente cada dispositivo tiene una media de 40 aplicaciones instaladas, haciendo que haya múltiples formas de acceso a tu dispositivo. [1]

Este año 2020, ha sido excepcional, la pandemia de Covid-19 ha obligado a que gran cantidad de gente use sus dispositivos para desarrollar su actividad laboral, sus relaciones interpersonales, sus compras, sus estudios y así en todos los aspectos. En la Figura 1 podemos ver como los ciberataques a dispositivos móviles en esta primera mitad del año 2020 corresponden con el 18 % del total, cediendo la primera posición que habían ostentado en años anteriores. [2]

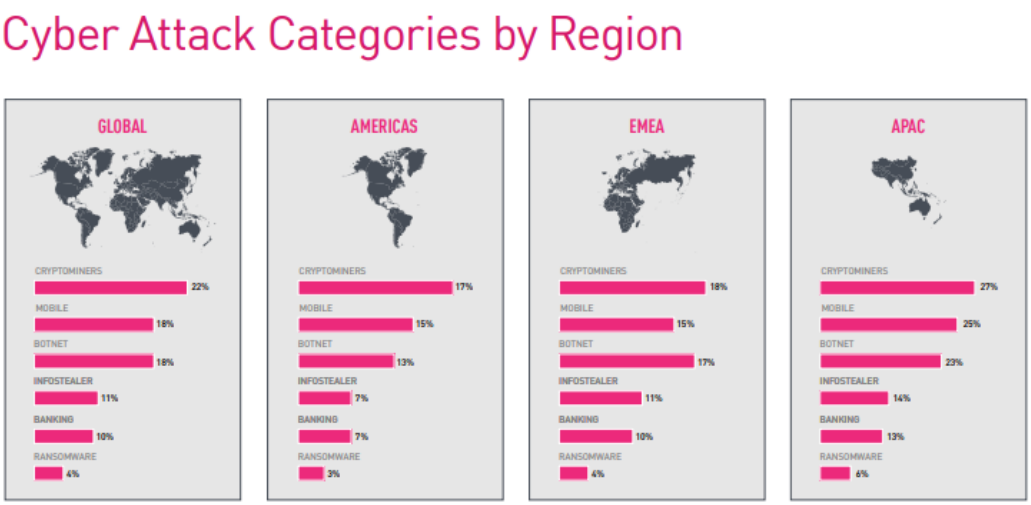



*Figura 1: Categorías de Ciberataques por región en la primera mitad de 2020 [2]*

Vamos a ver cómo han ido evolucionando estas aplicaciones a lo largo del tiempo. En un principio las aplicaciones para dispositivos móviles eran desarrolladas por las compañías que fabricaban los terminales. En esos tiempos, la documentación relativa a las aplicaciones era muy escasa. Esto se debía a el miedo de los fabricantes a permitir la introducción de terceros en su plataforma. Esas primeras aplicaciones eran similares a las aplicaciones de Contactos o Calendarios que encontramos en los teléfonos actuales. Otro ejemplo sería el famoso *Snake* de Nokia.

Cuando comenzaron a desarrollarse los dispositivos inteligentes (Smartphones) el desarrollo de aplicaciones experimentó un crecimiento no visto hasta la fecha. Conforme los dispositivos fueron aumentando sus prestaciones las aplicaciones fueron aprovechando estas mejoras ofreciendo nuevas funcionalidades que pronto fueron demandadas por los usuarios. Entre estas mejoras se encuentra el posicionamiento GPS, las cámaras, baterías…

Las aplicaciones desarrolladas por otras compañías aparecieron de la mano de Apple en 2008 con el anuncio de la primera aplicación de distribución de aplicaciones App Store, para el sistema operativo iOS. Google por su parte lanzó *Android Market* conocido ahora como *Play Store*.

En cada uno de los sistemas operativos las aplicaciones han sido desarrolladas usando diferentes lenguajes de programación. En iOS las aplicaciones usualmente son desarrolladas en el lenguaje *Objective-C* mientras que en Android el lenguaje principal es Java. Esto ha ocasionado que el mercado se encuentre dividido.[1]

Aunque existen otros sistemas operativos nos vamos a centrar en Android e IOS pues ambos suman el 99% del total. Esto lo podemos ver en la Figura 2.

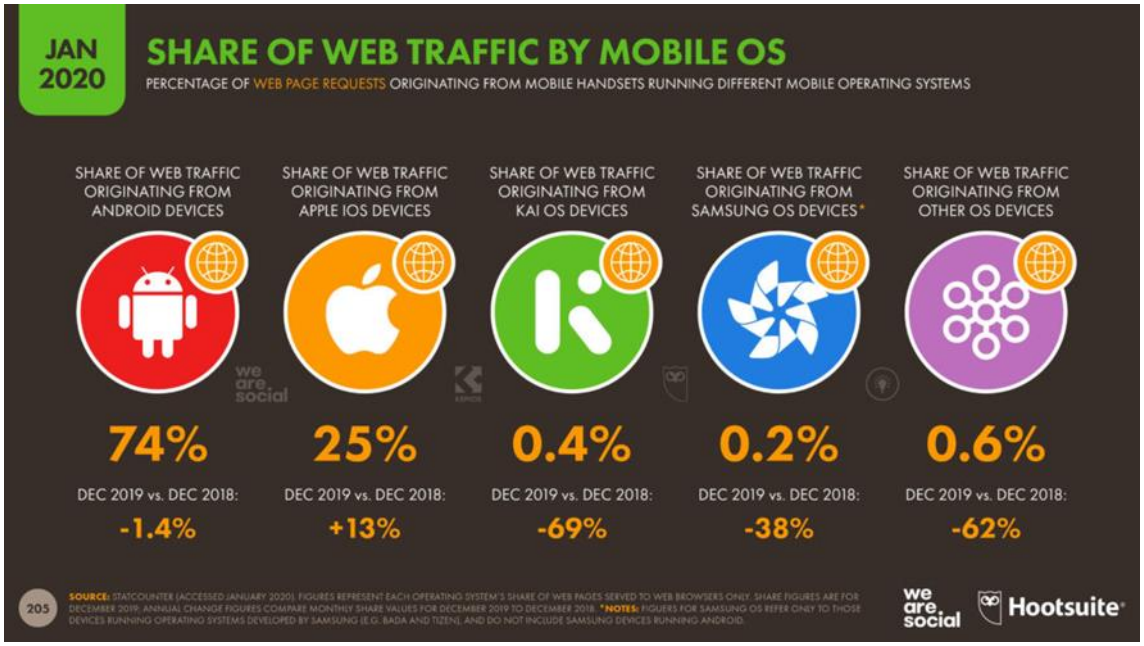



*Figura 2: Porcentaje de uso de los diferentes sistemas operativos en el mundo en diciembre de 2019 [3]*

Hay aplicaciones para todo tipo de cosas, combinando las aplicaciones disponibles em las aplicaciones de distribución de *Apple* y Android (*Apple Store* y *Play Store*) hay más de dos millones de aplicaciones que cubren aspectos como:

- Banca (*Santander*)
- Compras (*Amazon*)
- Redes Sociales (*Instagram*)
- Reproducción en directo (*Netflix*)
- Apuestas (*Betfair*)
- Mensajería instantánea (*WhatsApp*)
- Chat de voz (*Skype*)
- Correo Electrónico (*Gmail*)
- Transferencia de Ficheros (*Dropbox*)
- Juegos (*Pokemon Go*)
- Utilidades (Linterna)

Muchas de estas aplicaciones tienen funcionalidades basadas en páginas web, lo que ha permitido no alterar el lado de los servidores que reciben los datos.

En cuanto a la seguridad de estas aplicaciones nos encontramos con que las aplicaciones móviles están afectadas por una gran cantidad de vulnerabilidades, muchas de ellas heredadas de las presentes en páginas web y aplicaciones de escritorio. Pero sí que hay que destacar que las aplicaciones móviles se ven afectadas por algunas clases de ataques que son específicos de las aplicaciones móviles por el modo en el que las aplicaciones son usadas.

Los dispositivos móviles se encuentran expuestos de gran manera al hecho de entrar en una red insegura. Esta red puede ser Wi-Fi, celular, etc. Esto provoca que los datos deban ser encriptados al pasar por la red. Asimismo, se puede ver afectado por ataques de inyección.

Al tratarse de dispositivos móviles, aumenta la posibilidad de que el dispositivo sea robado o se pierda. Esto puede hacer que datos personales queden comprometidos.

En un dispositivo móvil posee gran cantidad de entradas de datos. No es extraño ver como hay aplicaciones que aceptan datos de las siguientes entradas:

- *Near Field Comunication* (NFC)
- *Bluetooth*
- Micrófono
- *Short Message Service* (SMS)
- *Universal Serial Bus* (USB)
- *Quick Response* (QR)

Es por esto por lo que nos encontramos ante un escenario en el que las vulnerabilidades presentes en las aplicaciones no son muy bien entendidas y la mayoría de las aplicaciones son en algún aspecto vulnerables. Además, el desarrollo de nuevas tecnologías como podrían ser el 5G y el IoT producirá una oleada de nuevos ciberataques pueden afectar a una gran cantidad de usuarios, así como a organizaciones. [1]

Con este fin, se plantea la necesidad de disponer de herramientas con las que realizar análisis de las aplicaciones, con lo que determinar si una aplicación es maliciosa (malware). El análisis de una aplicación presenta dos grandes vertientes que son las siguientes: [4]

**Análisis estático**: Durante la realización de este análisis se estudia el código fuente de la aplicación, obteniendo que primitivas, servicios, permisos… usa la aplicación. Seremos capaces de ver las claves presentes en el código, e identificar direcciones IP y direcciones web en el mismo. Es ventajoso respecto al análisis dinámico ya que se puede ver cómo actúa la aplicación en el caso de que sea maliciosa, pudiendo categorizarla como aplicación maliciosa de la forma adecuada.

Este análisis nos permite ver los permisos que se encuentran en la aplicación, viendo si estos son coherentes con el uso de la aplicación. También nos muestra aquellas estructuras de código que comparando con otras muestras de malware conocido podrían ser similares. Las direcciones IP que hay en el código de la aplicación, pudiendo analizar a donde pertenecen esas direcciones.

**Análisis Dinámico**: Para la realización de este análisis se ejecuta la aplicación móvil en un entorno controlado, que se denomina Sandbox. Durante el análisis podemos monitorizar los cambios que sufre la aplicación, los recursos a los que esta accede… En definitiva, nos permite analizar de forma adecuada su comportamiento.

Aunque no nos permite determinar de forma tan unívoca si una aplicación es maliciosa como el análisis estático, podemos realizar más pruebas, incorporar distintos datos de entrada con el fin de analizar de forma más profunda su comportamiento.

Es por eso que para este proyecto se van a tener en cuenta estos dos tipos de análisis, tratando de realizarlos de la forma más óptima posible.

## 1.1 Estructura del documento

El presente documento consta de quince capítulos los cuales podemos, dividir en varios bloques principales. En el primer bloque se realiza la identificación de la necesidad en el **capítulo primero** y durante cuánto tiempo se plantea y de qué modo se va a realizar el proyecto en los **capítulos segundo** y **tercero** respectivamente.

En el siguiente bloque que está compuesto por los **capítulos cuarto** y **quinto** se muestra el equipamiento usado en el proyecto, así como una comparación las diversas herramientas disponibles en el mercado que han sido probadas para la realización del proyecto.

El tercer bloque comprendido por los **capítulos sexto** y **séptimo** aborda el sistema operativo Android y su seguridad, el formato de las aplicaciones móviles en Android y la recolección y creación de una base de datos de malware para nuestra plataforma.

En el cuarto bloque se realiza un profundo comentario y guía de la herramienta de análisis de aplicaciones móviles MobSF y apklab.io, de la primera explicando su funcionamiento e interpretando los resultados que arroja ya que es la base de nuestra plataforma. Nos centraremos en explicar el análisis estático y dinámico de las aplicaciones, además de explicar cómo realizarlo. En el caso de apklab.io de la empresa Avast comentaos tiene unas funcionalidades similares a las de nuestra plataforma. Y realizamos una comparativa entre estas dos herramientas que hemos definido como muy similares, viendo los pros y contras de cada una de ellas. Este bloque se extiende por los **capítulos octavo, noveno y décimo**.

En el quinto bloque se presenta como se ha realizado el análisis dinámico sobre un dispositivo Android físico, a diferencia de lo realizado en los anteriores análisis dinámicos, que eran realizados sobre un dispositivo virtual. De este análisis obtenemos un tráfico que insertamos en una pila de Elasticsearch, Logstash y Kibana usada para presentar los resultados.

En el **capítulo décimo tercero** veremos una serie de casos de uso en los que probaremos a analizar varias aplicaciones móviles de Android analizando y comentando su comportamiento.

Por último, vemos las conclusiones del proyecto y sus futuras líneas de trabajo en los **capítulos décimo cuarto** y **décimo quinto**

# Capítulo 2: Fases del proyecto

Desde la empresa se me instó a elabora una propuesta de tiempos y objetivos que seguir durante la realización del proyecto. Esta práctica es muy habitual, ya que es de vital importancia dejar bien definidos al inicio del proyecto cuales son los objetivos de este, determinar los recursos que van a ser empleados y marcar objetivos de seguimiento.

La realización por mi parte de esta planificación con la supervisión de mis tutores previa a la realización del proyecto ha sido muy útil y me ha permitido ver como se aborda este tipo de proyectos desde el mundo laboral.

Asimismo, durante la realización del proyecto cumplir con los objetivos fijados previamente o adaptarlos según las situaciones del momento ha sido una valiosa lección.

Esto es lo que vamos a ver en este capítulo desgranando las fases del proyecto, declarando su inicio y su final, así como el número de horas destinadas a cada una de estas fases.

- Inicio del proyecto: En esta fase se planteó la necesidad de Orange de disponer de una plataforma centralizada donde realizar el análisis de aplicaciones móviles.
- Recopilación de conocimientos: En esta etapa se comenzó a estudiar las diversas alternativas. Se exploró el funcionamiento del sistema operativo Android y sus aplicaciones. (40 horas). Cabe destacar que esta fase se puede alargar de forma indefinida, por lo que se estipuló que llegase a ser de un 30% de la extensión del proyecto.
- Prueba de herramientas: Se realizó el despliegue de las diversas herramientas usadas estudiando los resultados que arrojaban. (30 horas)
- Análisis MobSF: Estudio en profundidad de la herramienta MobSF. E interconexión con los resultados de otras herramientas. (25 horas)
- Análisis apklab.io: Estudio en profundidad de la herramienta MobSF. (5 horas)
- Análisis de tráfico desde el terminal: Despliegue del punto de acceso malicioso (5 horas)
- Automatización de MobSF: Creación de scripts para el manejo de la plataforma. (20 horas)
- Procesado de. pcap: Introducción en la tecnología de la pila de Elasticsearch, Logstash y Kibana. Despliegue y creación de scripts de automatización. (30 horas)
- Redacción
- Final del proyecto: Obtención de una plataforma centralizada para realizar análisis estáticos y dinámicos de aplicaciones móviles.

# Capítulo 3: Metodología

En este proyecto se aborda cubrir la necesidad detectada en la mitigación de riesgos de seguridad en los usuarios como es el análisis desde el punto de vista de seguridad de las aplicaciones que instalan en los teléfonos inteligentes los usuarios. Este análisis requiere ser realizado desde 2 aproximaciones diferentes:

- De manera estática: revisión de código y componentes
- De manera dinámica: monitorizando actividad de la aplicación durante su uso

El proyecto se plantea en fases de manera que se vayan cubriendo hitos:

- Identificación de necesidades
    - Herramientas: Disponer de una plataforma con herramientas en las que realizar los análisis anteriormente descritos.
    - Documentación: Tener documentación relativa al manejo de la plataforma y los resultados que esta genera.
    - Hardware: Realizar un estudio de los requisitos de los que debe disponer el entorno en el que se desarrolla el proyecto con el fin de no sobredimensionarlo. Esto es tratado en el capítulo cuarto.
- Alcance
    - Herramientas usadas: Para comenzar la investigación se partió de una información del Instituto Nacional de Ciberseguridad (INCIBE) [5]. En ella se detallaban diversas utilidades para realizar el análisis de seguridad en aplicaciones móviles. Distinguiendo entre aquellas herramientas que son capaces de realizar un análisis estático, dinámico.
    - Equipos en los que reforzar la seguridad: Una vez implementadas las herramientas se procedió a reforzar la seguridad del equipo.
    - Aplicaciones interesantes para analizar, para ello se ha creado una base de datos de malware que podemos ver en el capítulo séptimo.
- Objetivos del proyecto
    - Automatizar análisis lo cual se ha hecho haciendo uso de los programas en Python que se pueden ver en los Anexos 1 y 2.
    - Obtener el tráfico del dispositivo móvil.
    - Cargar el tráfico obtenido en una pila de Elasticsearch, Logstash y Kibana con el fin de analizarlo.
- Plazos
    - El tiempo dedicado para cada fase lo hemos podido ver en el capítulo 2.
- Hitos
    - Recopilar información para el proyecto.
    - Probar herramientas mencionadas en el reporte del INCIBE con el fin de seleccionar
    - Establecer escenarios de pruebas haciendo uso de las aplicaciones maliciosas con las que hemos desarrollado la base de datos de malware que podemos ver en el capítulo séptimo.

- Desarrollo
    - Una vez seleccionadas las herramientas a integrar en la plataforma se realizaron análisis de aplicaciones móviles con ellas interpretando los resultados que los

análisis arrojaban. Posteriormente se realiza un análisis en profundidad de lo que representan los aspectos en los que se centra cada una de las categorías obtenidas en el análisis

- Conclusión y futuras líneas de trabajo
    - Comentar los resultados obtenidos tras la realización del proyecto, lo aprendido durante su realización y los próximos pasos a seguir en su mejora.

# Capítulo 4: Equipamiento usado en el proyecto

En este capítulo vamos a comentar el equipamiento usado para la realización del proyecto, sus principales características y requisitos necesarios.

El equipamiento usado ha sido el siguiente:

Un ordenador del siguiente modelo, HP EliteDesk 800 G1 SFF Black Desktop PC con las siguientes características:

- Intel Quad Core i5-4570 3.20GHz
- 8GB RAM
- 256GB SDD con Windows 10 Pro
- 256GB SDD con varias distribuciones de Unix instaladas.

Podemos verlo en la Figura 3.

*Figura 3: Equipo usado para el despliegue de la plataforma del proyecto*

Aunque a priori lo más indicado para desarrollar la plataforma era hacer uso de una máquina virtual se decidió usar un ordenador ya que la herramienta MobSF para realizar el análisis dinámico de las aplicaciones precisa de un dispositivo real.

Vamos a comentar un poco a que se debe la elección de los componentes de este equipo. En primer lugar, su procesador, es un procesador Intel i5 de la tercera generación. La potencia de la que dispone este tipo de procesadores es más que suficiente para su uso en este proyecto. Si es muy importante hacer notar que este **procesador** tiene la capacidad de ser **virtualizable**, requisito fundamental para la instalación de máquinas virtuales en el equipo. Esto es necesario

para desplegar nuestros dispositivos móviles virtuales. El uso de un procesador de mayor potencia no hubiera significado ningún cambio apreciable.

La memoria RAM de este equipo es de 8GB, memoria más que suficiente, aunque pudiera ser ampliable en un futuro ya que la placa base cuenta con más zócalos donde ubicar las memorias RAM.

Por último, vamos a detenernos en los discos duros del equipo, siendo ambos discos de estado sólido (SSD) de 256 GB. La elección de esta tecnología en detrimento de la tradicional unidad de disco duro (HDD) ha estado motivada por las velocidades de transferencias de la tecnología de los discos de estado sólido (SSD). En este caso se aprecia una mejora obvia del rendimiento del equipo.

Una tarjeta de red inalámbrica capaz de actuar como punto de acceso (AP) en una red inalámbrica WIFI. En este caso de la marca ALFA Network del modelo AWUS036NH que podemos ver en la Figura 4. Es importante que interfaz de red usada sea capaz de actuar como punto de acceso ya que no todas las interfaces de red inalámbricas lo son. Si no es capaz de realizar esto será imposible realizar un punto de acceso malicioso con ella. Esto supuso un problema ya que al principio no se disponía de esta antena sino de otra como la de la Figura 5 que no contaba entre sus funcionalidades con la de actuar como **punto de acceso (AP)**.


*Figura 4: Interfaz de red inalámbrica usada para el despliegue del punto de acceso malicioso. Es capaz de realizar la funcionalidad de punto de acceso.*


*Figura 5: Interfaz de red inalámbrica. No es capaz de realizar la funcionalidad de punto de acceso*

Aunque a priori lo más indicado para desarrollar la plataforma era hacer uso de una máquina virtual se decidió usar un ordenador ya que la herramienta MobSF para realizar el análisis dinámico de las aplicaciones precisa de un dispositivo real.

El sistema operativo que se encuentra instalado es **Linux Mint 19.3 (Tricia)** la versión más actualizada a la fecha de realización del proyecto. Es en esta distribución en la que se ha instalado los componentes de la plataforma.

En la otra partición del disco duro se encuentra instalado **Santoku Linux 0.5** siendo esta la versión más actualizada de esta distribución a la fecha de realización del proyecto.

A parte se procedió a la instalación del servicio **SSH** para tener un acceso seguro al equipo. Para reforzar la seguridad del equipo se restringió su acceso desde otras redes. Eso se modificando el cortafuegos con los siguientes comandos.

```
sudo iptables -I INPUT -p tcp --dport 22 -s 192.168.0.0/16 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 22 -j REJECT
```

Por otro lado, se ha usado un dispositivo móvil que se encontraba en el laboratorio del modelo **Samsung Galaxy S7** el cual cuenta con el sistema Android en su versión **6.0.1**. Lo podemos ver en la Figura 6


*Figura 6: Dispositivo móvil usado durante el proyecto*

Este terminal, tope de gama hace varios años ha sido ideal para realizar las pruebas ya que dispone de gran potencia.

# Capítulo 5: Comparativa de herramientas

En este capítulo vamos a realizar una explicación de las diversas herramientas empleadas en el proyecto, así como un análisis de sus ventajas y desventajas de cara a la realización del proyecto.

Las herramientas usadas se pueden desglosar en dos grandes bloques, es decir herramientas de análisis estático y herramientas de análisis dinámico.

En el **análisis estático** que es en el que se estudia el código fuente de la aplicación, obteniendo que primitivas, servicios, permisos… usa la aplicación. Seremos capaces de ver las claves presentes en el código, e identificar direcciones IP y direcciones web en el mismo. Es ventajoso respecto al análisis dinámico ya que se puede ver cómo actúa la aplicación en el caso de que sea maliciosa, pudiendo categorizarla como malware de la forma adecuada. Vemos las siguientes herramientas:

- **Distribución Santoku Linux:** En este caso la Distribución Santoku Linux aúna una gran cantidad de herramientas para el análisis de aplicaciones.
- **Androguard:** Esta herramienta nos permite realizar el análisis estático vía interfaz de comandos de una sola aplicación.
- **APKTool:** Al igual que el anterior nos permite realizar el análisis estático de una única aplicación, permitiendo profundizar en su estudio.
- **VirusTotal:** Esta herramienta en línea nos permite realizar el análisis estático usando más de 50 antivirus.
- **MobSF:** Esta herramienta permite almacenar en su base de datos gran cantidad de aplicaciones de las que realizar su análisis estático aportando además información muy útil sobre los resultados obtenidos.
- **apklab.io:** Similar a la anterior, esta herramienta en línea creada por Avast da la posibilidad de almacenar en su base de datos gran cantidad de aplicaciones de las que realizar su análisis estático

El **análisis dinámico** que consiste en ejecutar la aplicación móvil en un entorno controlado, que se denomina Sandbox. Durante el análisis podemos monitorizar los cambios que sufre la aplicación, los recursos a los que esta accede… En definitiva, nos permite analizar de forma adecuada su comportamiento… Nos encontramos las siguientes herramientas:

- **DroidBox:** Esta herramienta nos permite realizar el análisis dinámico de una sola aplicación.
- **apklab.io:** La herramienta creada por Avast nos permite realizar también un análisis dinámico de las aplicaciones móviles.
- **MobSF:** Usando esta herramienta somos capaces de realizar controlando el uso de la aplicación.

Por último, se han realizado varias tablas en las que se resumen las características principales de cada una de las herramientas comparándolas con el resto.

## 5.1 Herramientas análisis estático

### *5.1.1 Distribución Santoku Linux*

En un primer momento se optó por la instalación de la distribución **Santoku Linux** la cual es mencionada como una distribución orientada al análisis forense y de seguridad para aplicaciones móviles. Dicha distribución se encuentra actualmente instalada en una de las particiones del disco duro añadido a la máquina, siendo posible su acceso durante el arranque del equipo.


*Figura 7: Programas de ciberseguridad preinstaladas en la distribución Santoku Linux*

Esta distribución contiene gran cantidad de programas útiles para el análisis de aplicaciones móviles, hecho por lo que se recomienda su despliegue en una máquina virtual ya que los programas que contiene no precisan de ser usados en un dispositivo real. En la Figura 7 se pueden ver como esta distribución alberga gran cantidad de utilidades relacionadas con el análisis de aplicaciones.

Sin embargo, el hecho de que la distribución Santoku Linux no haya sido la elegida para albergar la plataforma se debe al hecho de que dicha distribución se encuentra sin recibir actualizaciones desde hace varios años. Es por esto por lo que programas tan útiles como Python no puedan funcionar de forma correcta en esta distribución. Esto supuso un en los inicios del proyecto ya que imposibilitaba la instalación de varias herramientas.

### 5.1.2 Androguard

Posteriormente se probó **Androguard**, que es una herramienta desarrollada en Python por lo que puede usarse en múltiples plataformas y está orientada a realizar análisis estáticos de aplicaciones móviles.


*Figura 8: Logotipo de la herramienta Androguard.*

Esta aplicación permite desensamblar él .apk de una aplicación permitiendo ver su código siendo muy sencillo obtener información clave de la aplicación en base a sus servicios, actividades, receptores, permisos…

Esta herramienta es usada haciendo uso de la interfaz de línea de comando (CLI). Esto permite que sea una herramienta automatizable. Si bien la pega que podemos encontrar es que los resultados que arroja son muy escuetos comparados por ejemplo con los de MobSF. Es por tanto esta una herramienta que, si bien permite realizar el análisis de múltiples aplicaciones móviles simultáneamente, sus resultados no mejoran lo ofrecido por otras alternativas.

### *5.1.3 APKTool*

Herramientas como **APKTool** o **Dex2Jar** también aparecían en el reporte del **INCIBE** aunque estas herramientas están orientadas a realizar el análisis de una sola aplicación por lo que también fueron descartadas, aunque son herramientas muy útiles de cara a realizar un análisis más profundo de una aplicación en concreto y no está de más conocerlas.

### *5.1.4 Virustotal*

Una de las principales es **Virustotal**, en ella las aplicaciones son analizadas por los motores de más de 50 antivirus. Está basada en Androguard, pero añadiendo módulos que dan más información adicional al realizar los análisis. Esta herramienta haciendo uso de su API pública se encuentra actualmente interconectada con nuestra plataforma y siendo posible acceder al reporte generado en dicha herramienta de la aplicación que analicemos en nuestra plataforma.

### *5.1.5 MobSF*

La herramienta cuyos resultados han sido más satisfactorios ha sido con **Mobile-Security-Framework (MobSF)**. Esta herramienta automática permite la realización de análisis estáticos y

dinámicos de para aplicaciones móviles para iOS, Android y Windows. Es por eso por lo que nuestra plataforma se ha basado en esta herramienta. Esta herramienta a diferencia de las anteriormente mencionadas se encuentra muy actualizada, teniendo varias actualizaciones durante la realización del proyecto. Esto nos indica que actualmente es un proyecto en desarrollo que incluso mejorará.


*Figura 9: Logotipo de la herramienta MobSF.*

### *5.1.6 apklab.io*

Durante el desarrollo del proyecto, tuvimos la oportunidad de probar la herramienta online de la empresa Avast que es apklab.io que ofrece resultados similares, y que nos ha servido como espejo con el que comparar nuestros resultados.


*Figura 10: Logotipo de la herramienta apklab.io*

## 5.2 Análisis Dinámico

Para el análisis dinámico las posibles herramientas se redujeron drásticamente, encontrando pocas posibilidades entre las que elegir. Nos encontramos principalmente con tres herramientas.

### *5.2.1 DroidBox*

La primera de ellas **DroidBox**, es una herramienta basada en la interfaz de línea de comandos. Permite realizar el análisis dinámico de aplicaciones móviles. Presenta el inconveniente de que se encuentra muy desactualizada y sin visos de que la situación cambie.

### *5.2.2 apklab.io*

También la herramienta de Avast **apklab.io** es capaz de realizar un análisis dinámico, pero sin darte la posibilidad de manejar la propia aplicación, lo que deja el análisis menos completo.

### *5.2.3 MobSF*

Por otro lado, nos encontramos que la herramienta MobSF también realiza el análisis dinámico lo que también contribuyó a su elección como parte fundamental de la plataforma. Posteriormente veremos sus posibilidades.

Es por todo lo anteriormente mencionado que la herramienta MobSF es la elegida para desarrollar esta plataforma, por su gran versatilidad y los excelentes resultados que arrojan sus dos tipos de análisis (Estático y Dinámico).

El hecho de disponer en el laboratorio de un dispositivo móvil ha permitido la diferenciación del proyecto, respecto a la plataforma de Avast, pues ha posibilitado la incorporación al análisis de la aplicación en un entorno real. Esta diferencia es fundamental ya que numerosas aplicaciones que son malware, cuando son programadas tratan de detectar si se encuentra en un dispositivo virtual o real.

Al disponer de un dispositivo Android hemos optado por centrarnos más en este sistema operativo, que además es el usado por el 74% de los usuarios.

## 5.3 Tablas comparativas de las herramientas

En este apartado vamos a realizar un resumen de las características comentadas previamente de las que dispone cada herramienta y comparándolas con el resto.

En esta primera tabla mostraremos las herramientas usadas en el análisis estático.

| Análisis Estático | | | | | |
|---|---|---|---|---|---|
| Características\Herramientas | Androuguard | APKTool | Virustotal | MobSF | apklab.io |
| Actualizada | | | | | |
| Documentada | | | | | |
| Reportes automatizados | | | | | |
| Posibilidad de automatizar | | | | | |
| Información obtenida de los reportes extensa | | | | | |
| Herramienta en línea | | | | | |
| Multiplataforma | | | | | |
| Conexión con bases de datos externas | | | | | |
| Incorpora Base de Datos | | | | | |

*Tabla 1: Comparativas de herramientas para el análisis estático*

El verde indica si la herramienta tiene esa característica, mientras que el rojo indica si la herramienta no tiene esa característica.

En esta segunda tabla mostraremos las herramientas usadas en el análisis dinámico.

| Análisis Dinámico | | | |
|---|---|---|---|
| Características\Herramientas | DroidBox | MobSF | apklab.io |
| Actualizada | | | |
| Manejo de la aplicación durante el análisis | | | |
| Información obtenida de los reportes extensa | | | |

| | | | |
|---|---|---|---|
| Obtención de fichero con el tráfico | | | |
| Capturas de pantalla de la aplicación en el informe | | | |
| Integración del Logcat del dispositivo | | | |
| Ejecución del análisis con permisos privilegiados | | | |
| Presentación amigable de las direcciones a las que se envía o recibe tráfico | | | |
| Posibilidad de ejecutar scripts de prueba | | | |

*Tabla 2: Comparativas de herramientas para el análisis dinámico*

El verde indica si la herramienta tiene esa característica, mientras que el rojo indica si la herramienta no tiene esa característica.

# Capítulo 6: Android

Android es un sistema operativo originalmente pensado para dispositivos móviles. Fue desarrollado por Android Inc, los cuales lo vendieron a la empresa Google en 2005. El sistema operativo Android está basado en una versión modificada del **kernel de Linux 2.6**. Google y otros miembros de la Open Handset Alliance (OHA) colaboraron en el diseño, desarrollo y distribución de Android. Actualmente la entidad que se encarga del mantenimiento y los ciclos de desarrollo de Android es **Android Open Source Proyect (AOSP)**.

Vamos a ver a que nos referíamos cuando hemos dicho que el sistema operativo Android está basado en una versión modificada del kernel de Linux 2.6. Si lo comparamos con el kernel de Linux 2.6 original, nos vamos a encontrar que muchos controladores y librerías han sido añadidas o modificadas para que el sistema operativo sea capaz de maximizar su eficiencia en los dispositivos en los que se aloja el sistema operativo. Muchas de estas modificaciones vienen desarrolladas desde comunidades de código abierto.

La comunidad de Android ha desarrollado su propia librería de C (*Bionic*) y su propio motor en tiempo de ejecución para Java.

El objetivo de Android ha sido siempre optimizar los recursos disponibles en los dispositivos móviles. La mejor forma de describir a Android es común una solución en forma de pila que incorpora el sistema operativo, una lógica de intercambio de información entre aplicaciones y las propias aplicaciones. EL kernel modificado de Linux actúa como la capa que abstrae el hardware. [6]

## 6.1 Modelo de Seguridad de Android

El sistema operativo Android es usado por gran cantidad de fabricantes de dispositivos móviles, tanto de teléfonos como tablets. Debido a su naturaleza de código abierto también se puede encontrar en sistemas de entretenimiento, televisiones, e-books, relojes inteligentes, Android Auto…

Para entender cómo funciona la seguridad en un dispositivo Android debemos tener en cuenta como es la arquitectura multicapa en la que está basado el sistema operativo. La podemos ver en la Figura 11.

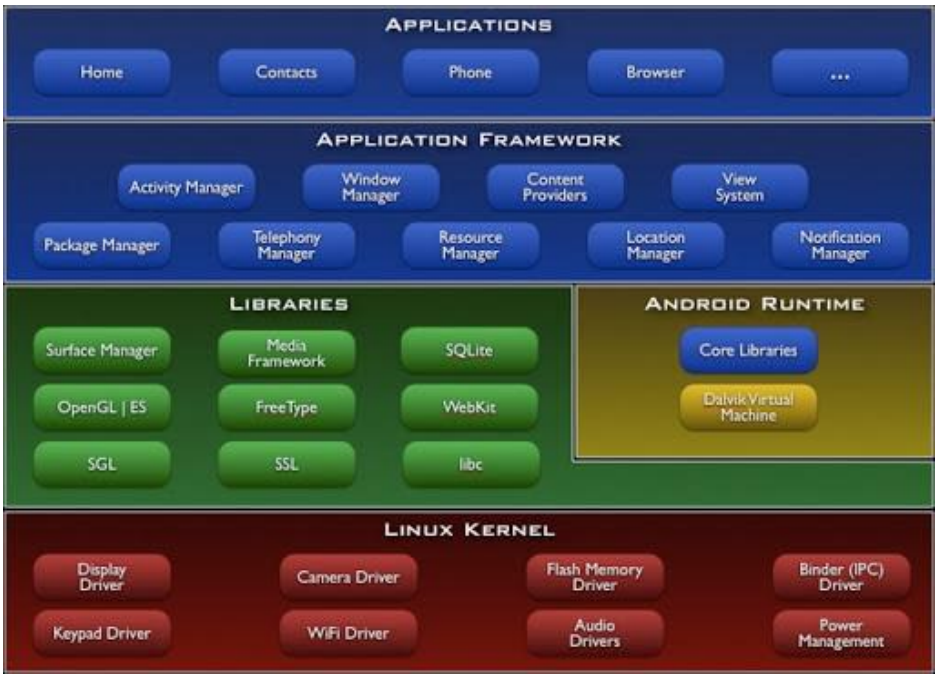



*Figura 11: Estructura de capas del sistema operativo Android*

Vamos a explicar cada campo de la Figura 11.

- **Aplicaciones:** La capa más alta del modelo, se compone de las aplicaciones instaladas en el dispositivo, tanto las que vienen por defecto como las que el usuario instala a posteriori.

- **Framework:** Es la capa que contiene las funcionalidades clave del sistema operativo de Android como el *Package Manager* que permite de instalar/borrar aplicaciones Android, *ActivityManager* que maneja el ciclo de vida de cada actividad de cada aplicación, etc.

- **Librerías**: Esta capa almacena el conjunto de librerías que utilizan los diferentes componentes del sistema operativo Android. Algunas de las más destacadas son las multimedia, el motor gráfico o el motor de base de datos SQLITE.

- **Android Runtime**: Esta capa está formado por dos componentes: las librerías del CORE y la máquina virtual Dalvik o su sucesora, ART. Todas las aplicaciones son ejecutadas en una máquina virtual con el fin de dotar de un entorno de seguridad al modelo del malware para dispositivos

- **Kernel:** Corresponde a una capa de abstracción entre el hardware y el software. Incluye servicios esenciales como la gestión de memoria o de procesos, o los drivers que permiten interactuar con la cámara, audio, Wi-Fi, etc... [7]

Una vez visto esto podemos ver como el sistema operativo Android está diseñado para prevenir ataques:

- Por ejemplo, el sistema operativo Android **limita a priori la instalación de software de terceros**, evitando así los ataques basados en la ingeniería social que buscan la instalación de aplicaciones maliciosas en el dispositivo.
- La principal forma de obtener una aplicación para el sistema operativo Android es desde *Play Store* el cual **limita las aplicaciones** que son ofertadas (no puede haber dos con el mismo nombre y logo) y eliminando además aquellas que contengan:

- Aplicaciones que incluyen contenido ilegal.
- Aplicaciones que facilitan juegos de apuestas reales.
- Aplicaciones que incluyen contenido de promoción del odio.
- Aplicaciones que incluyen pornografía.
- Aplicaciones que incluyen violencia real gratuita.

- El hecho de usar una **máquina virtual para la ejecución de cada aplicación** es con el fin de evitar que una aplicación pueda acceder a los datos de otras.
- El framework incorpora funcionalidades como el **sistema de permisos y herramientas criptográficas** para el cifrado de los contenidos de las aplicaciones.
- Las **memorias** internas y externas de los dispositivos móviles se encuentran **cifradas**.

Todo este sistema de prevención de ataques se puede representar como en la Figura 12.

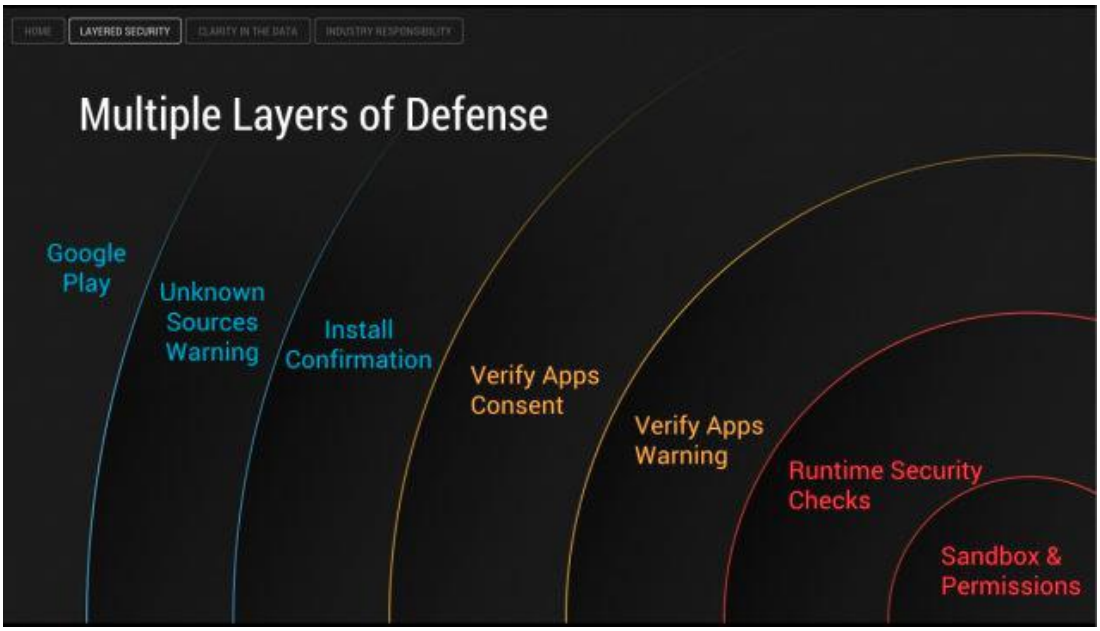



*Figura 12: Controles incorporados por Android para la protección de los usuarios. [7]*

Una vez mencionadas las fortalezas que presenta el sistema operativo no nos queda otra que mencionar también sus debilidades:

- Es habitual el uso de aplicaciones con **certificados autofirmados**. Eso supone un problema ya que no habría una autoridad certificadora detrás que valide el certificado.
- La aparición de **permisos personalizados** que podrían suponer un problema de privacidad.
- Controles poco exhaustivos en los repositorios de Play Store.

Es decir, nos encontramos ante un sistema operativo que busca limitar la instalación de aplicaciones móviles a las disponibles en la aplicación de distribución Play Store. Para poder hallarse allí las aplicaciones deben superar una serie de requisitos, aunque los controles en la aplicación de distribución no son todo lo exhaustivos que podrían ser, dejando la puerta abierta a la introducción de aplicaciones móviles maliciosas en Play Store si se camuflan lo suficiente.

## 6.2 Seguridad en el arranque de Android

Debemos mencionar también el sistema de **seguridad** que está presente también en el arranque de los dispositivos con el sistema operativo Android. Este proceso lo podemos ver en la Figura 13.

Desde que pulsamos el botón de encendido comienza este proceso, primero se ejecuta el código para el arranque del procesador (*Primary Boot Loader*), el cual se encuentra en un sitio de confianza ya que se encuentra en una memoria de solo lectura (ROM), con lo que *garantizamos su integridad*. Haciendo uso de la clave pública que se encuentra en esa misma memoria ROM es capaz de **verificar las diversas secuencias de arranque** secundarias que se encargan de encender subprocesos en el dispositivo tales como el sistema de radio para la conexión con GSM/3G/LTE, los sensores del dispositivo… El último de estos arranques secundarios también se encarga de verificar la integridad del *aboot* que es la secuencia de arranque del sistema operativo Android, e iniciar dicha secuencia de arranque. En este proceso se denomina *chain-of-trust* pues **cada acción es realizada tras comprobar la integridad de la anterior**.[7]

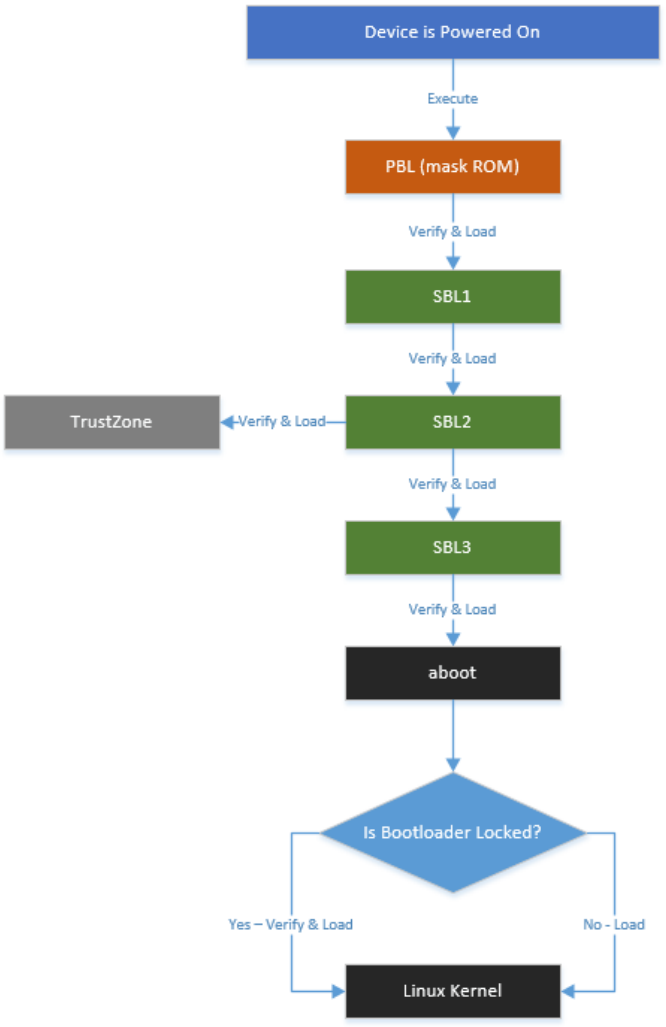



*Figura 13: Seguridad en el arranque [7]*

## 6.3 El Formato APK

Una vez visto como es el modelo de seguridad en Android, vamos a ver que forma el fichero .apk que es el que contiene la aplicación en el sistema operativo Android.
Las aplicaciones se encuentran en el formato *Application PacKage File (APK)* en el que son distribuidas e instaladas.[7]
Presentan la siguiente estructura, que podemos ver en la Figura 14.

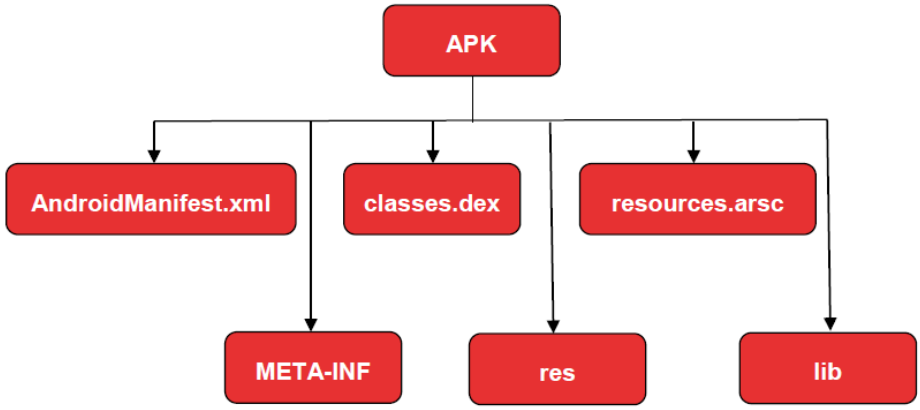



*Figura 14: Composición del formato APK.*

- **AndroidManifest.xml:** Fichero de configuración de la aplicación. En él se indican aspectos como el identificador único de la aplicación, los componentes de esta como serían las actividades, los receptores o sus permisos.
- **META-INF:** Directorio que guarda los datos referidos a la firma digital de la aplicación Android.
- **classes.dex:** Es el código de la aplicación compilado para que sea interpretado por la máquina virtual.
- **res:** Contiene las imágenes, texto, .xml usados por la aplicación.
- **resources.arsc:** Almacena compilados los recursos de la aplicación.
- **lib:** Contiene las librerías necesarias para la ejecución de la aplicación

Como se puede ver, cuando realizamos una instalación de una aplicación móvil de Android el archivo usado es del formato .apk. Por tanto, a priori no hay forma de inspeccionar los elementos anteriormente descritos, como podrían ser el código, los recursos… Nuestra plataforma permite la obtención de dichos componentes ya que realiza un desempaquetado de la aplicación.

Todos estos ficheros pueden ser obtenidos de diversas formas, siendo el principal y más recomendado la aplicación de distribución Play Store. Ya que las obtenidas en esa fuente han sido sometidas a un control, cosa que podría no ocurrir en las obtenidas por otros medios. Por tanto, no se recomienda su instalación salvo que se confíe plenamente en la fuente o se realice un análisis de esta, usando por ejemplo esta plataforma.

# Capítulo 7: Base de datos de aplicaciones Malware

Esta plataforma es capaz de analizar cualquier aplicación del sistema operativo Android e iOS, pero para la **realización de las pruebas y comprobaciones**, se ha probado con diversas aplicaciones del sistema operativo Android consideradas maliciosas, obtenidas de [8].

En dicha dirección se encuentra un repositorio de uso público en el que los usuarios cargan aplicaciones consideradas maliciosas, así como un enlace con la noticia de dicha aplicación explicando de qué forma es un riesgo de la seguridad.

Este repositorio se deja cargado en la máquina en la ruta: /Apps/Android-malware-master-20

Estas muestras nos permiten disponer de ejemplos de malware con los que comprobar si nuestro análisis es correcto. Además, quedan en el sistema para en un futuro poder realizar comparaciones de otras muestras de aplicaciones con estos ejemplos de aplicaciones maliciosas.

En la Figura 15 podemos ver una muestra.

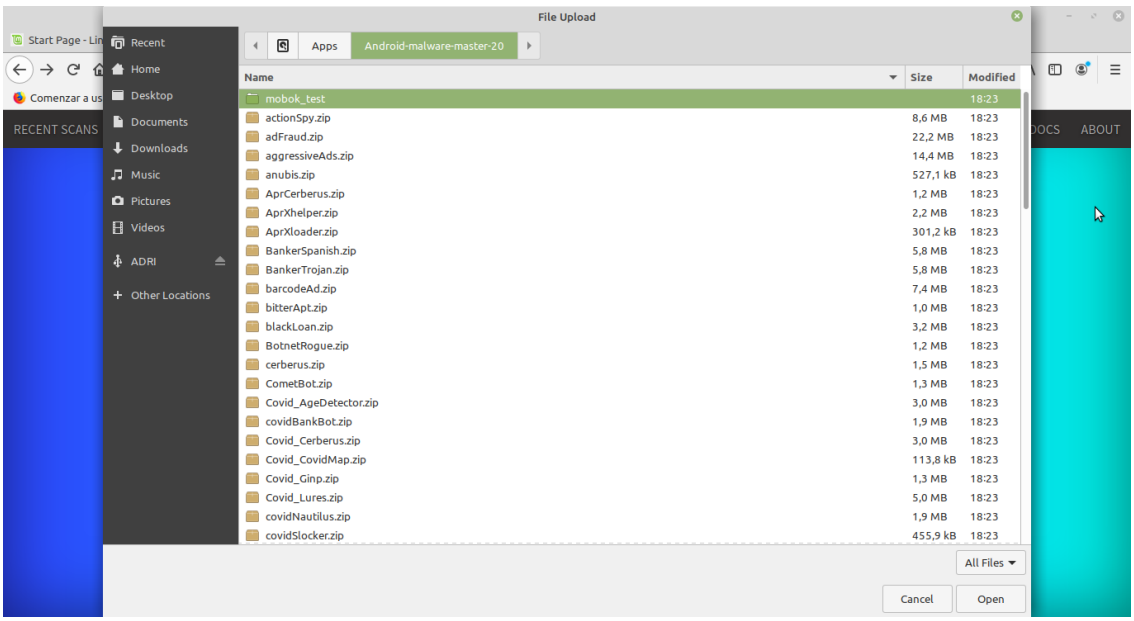



*Figura 15: Base de datos de aplicaciones malware en la plataforma.*

# Capítulo 8: MobSF

En este capítulo vamos a ver en profundad lo que los análisis que podemos realizar con **Mobile-Security-Framework (MobSF)** arrojan.

**Análisis Estático**: Durante la realización de este análisis se estudia el código fuente de la aplicación, obteniendo que primitivas, servicios, permisos… usa la aplicación. Seremos capaces de ver las claves presentes en el código, e identificar direcciones IP y direcciones web en el mismo. Es ventajoso respecto al análisis dinámico ya que se puede ver cómo actúa la aplicación en el caso de que sea maliciosa, pudiendo categorizarla como malware de la forma adecuada.

En estos primeros apartados vamos a ver como la herramienta nos presenta los diversos componentes presentes en una aplicación móvil.

- Actividades
- Servicios
- Receptores
- Proveedores
- Permisos

Los siguientes apartados comentan la información encontrada en el código fuente de la aplicación.

- Android Java API
- Análisis del manifiesto.
- Análisis del código
- APKiD
- Dominios y Direcciones en el código
- Cadenas

En este apartado del análisis se nos muestran todos los ficheros que componen el código fuente de la aplicación.

- Ficheros

**Análisis Dinámico**: Para la realización de este análisis se ejecuta la aplicación móvil en un entorno controlado, que se denomina *Sandbox*. Durante el análisis podemos monitorizar los cambios que sufre la aplicación, los recursos a los que esta accede… En definitiva, nos permite analizar de forma adecuada su comportamiento.

En este primer aparado se comenta como se configura el Sandbox en el que se realiza el análisis dinámico de la aplicación.

  - Genymotion

En estos siguientes apartados se comenta como se realiza el análisis dinámico y los diversos resultados que podemos obtener de él, así como modos de monitorización de la aplicación.

  - Menú análisis
  - Información y datos del dispositivo
  - API de la aplicación
  - Logcat
  - Eventos de comunicación entre procesos

- Dominios y Direcciones

MobSF es una herramienta automática permite la realización de análisis estáticos y dinámicos de para aplicaciones móviles para iOS, Android y Windows. Los datos de la herramienta son almacenados en el propio entorno, no haciendo uso de ningún servicio en línea.

Esta herramienta se pude encontrar en [9] contando con una documentación muy desarrollada en cuanto a cómo realizar la instalación, tanto de la parte estática como de la parte dinámica.

Para iniciar la herramienta se debe ejecutar el comando *sudo ./run.sh* en el directorio **Mobile-Security-Framework-MobSF**. La salida de este comando la podemos ver en la Figura 15. Es importante destacar que aquí podemos obtener la *key* para poder manejar la API. Esta *key* va a ser siempre la misma en cada instalación y actúa como credencial en el acceso a los datos de MobSF.

Como podemos ver en la Figura 16 para esta prueba no se ha activado el terminar virtual para realizar el análisis dinámico, por lo que nos avisa que no será posible realizarlo en un primer momento.

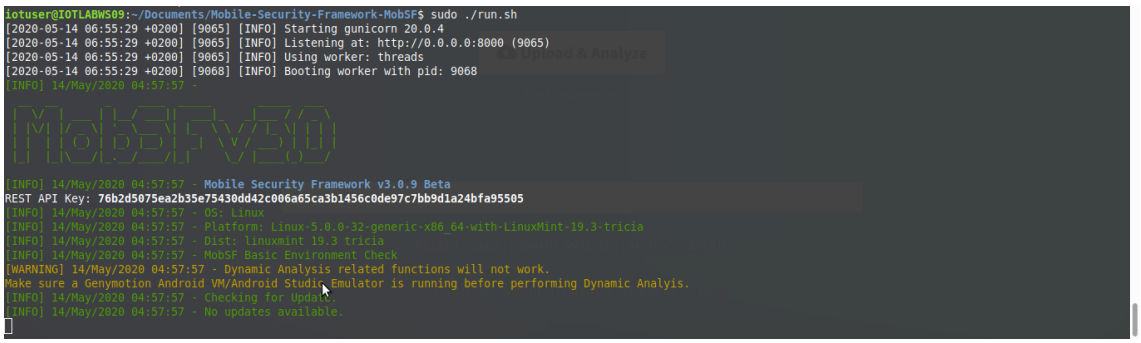



*Figura 16: Ejecución de MobSF sin tener iniciado el terminal virtual.*

En el caso de que haya disponible una nueva versión, se nos notifica en el arranque, como se puede ver en la Figura 17.

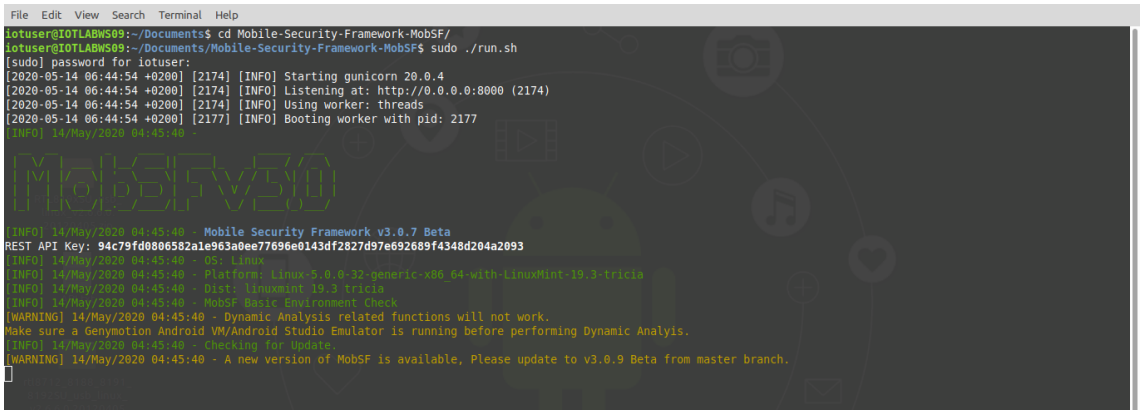



*Figura 17: Notificación de actualización en la herramienta MobSF*

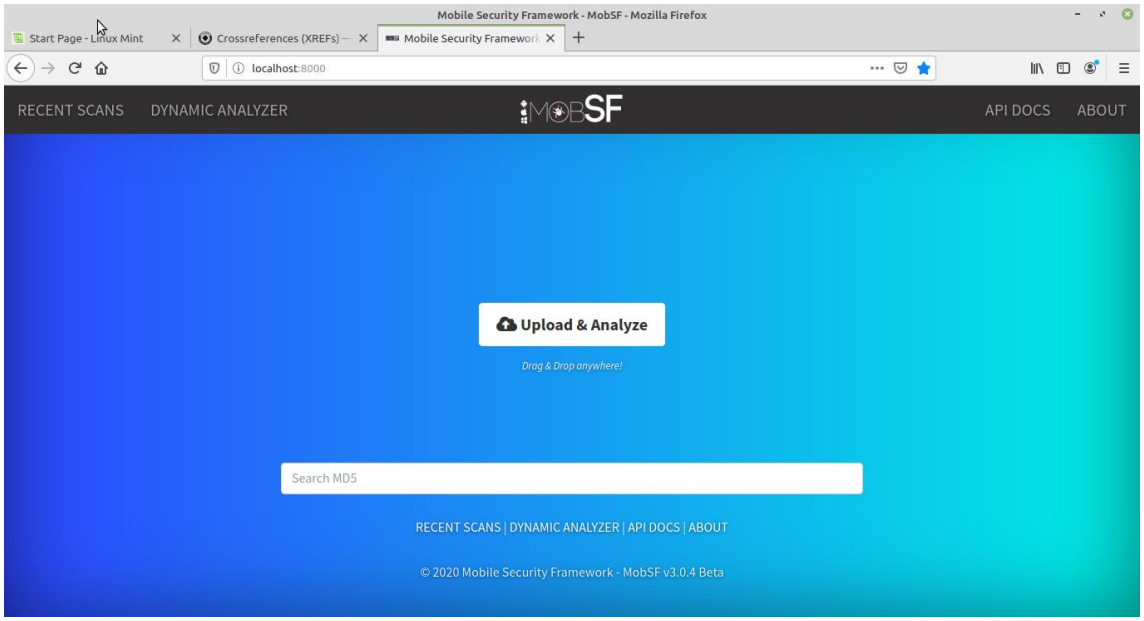



*Figura 18: Menú principal de la herramienta MobSF.*

Tras seguir los pasos de la guía de instalación en [9] obtenemos en el puerto **8000** de la máquina que contiene la herramienta lo que se muestra en la Figura 18:

También como se puede ver en la parte superior de la Figura 18 nos encontramos varias opciones entra las que destacan ***Recent Scans, Dynamic Analyzer y API DOCS***.

En la Figura 19 podemos ver lo que se obtiene en *Recent Scans* en la que se nos muestra que aplicaciones han sido analizadas, cuando se ha realizado el análisis, así como la versión de la aplicación que ha sido analizada, lo cual es muy útil de cara a analizar los posibles cambios entre versiones de una misma aplicación.

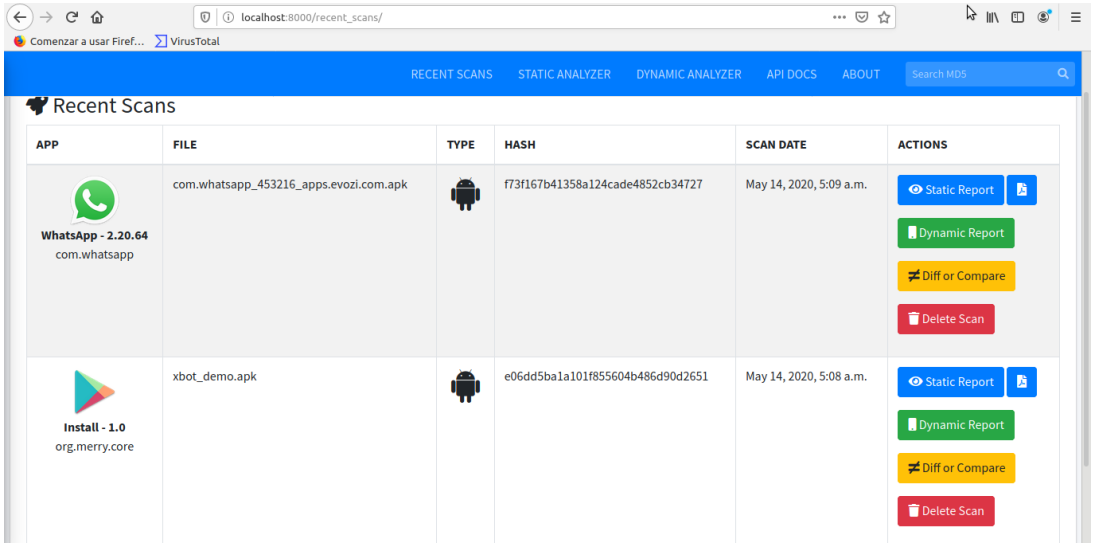



*Figura 19: Vista del menú Recent Scans de la herramienta MobSF*

En **API DOCS** nos encontramos la documentación referente a la API que nos permitirá automatizar la obtención de resultados. Dicha documentación pude consultarse en [9].

## 8.1 Análisis Estático

Durante la realización de este análisis se estudia el código fuente de la aplicación, obteniendo que primitivas, servicios, permisos… usa la aplicación. Seremos capaces de ver las claves presentes en el código, e identificar direcciones IP y direcciones web en el mismo. Es ventajoso respecto al análisis dinámico ya que se puede ver cómo actúa la aplicación en el caso de que sea maliciosa, pudiendo categorizarla como malware de la forma adecuada.

Para esta parte del análisis se ha usado el analizador estático de MobSF. En la Figura 20 podemos ver una versión simplificada de su arquitectura.

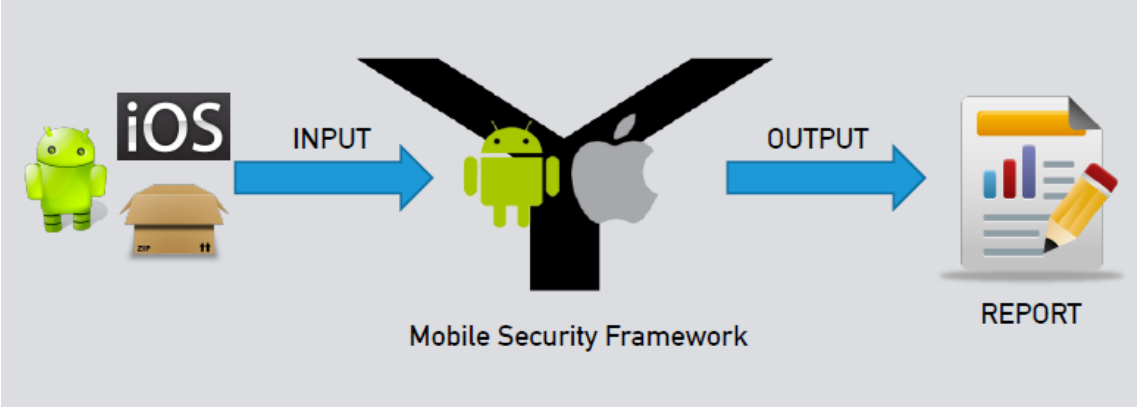



*Figura 20: Esquema simplificado del análisis estático en MobSF.*

Como podemos ver precisamos tener el fichero que contenga la aplicación. Ese fichero puede estar en formato .apk o incluso comprimido, requiriéndonos la contraseña en la interfaz de línea de comandos. Una vez cargado el fichero MobSF va a generar un reporte basándose en el análisis estático de nuestra aplicación.

Una vez explicado esto se ha procedido a cargar la aplicación de la forma indicada, aunque se puede hacer mediante la API que incorpora la propia herramienta MobSF. Haciendo uso de esta API se ha elaborado un script que realiza la carga de ficheros en el sistema y su posterior análisis, ya que no se trata de una única operación. Este script se puede ver en el Anexo 1.

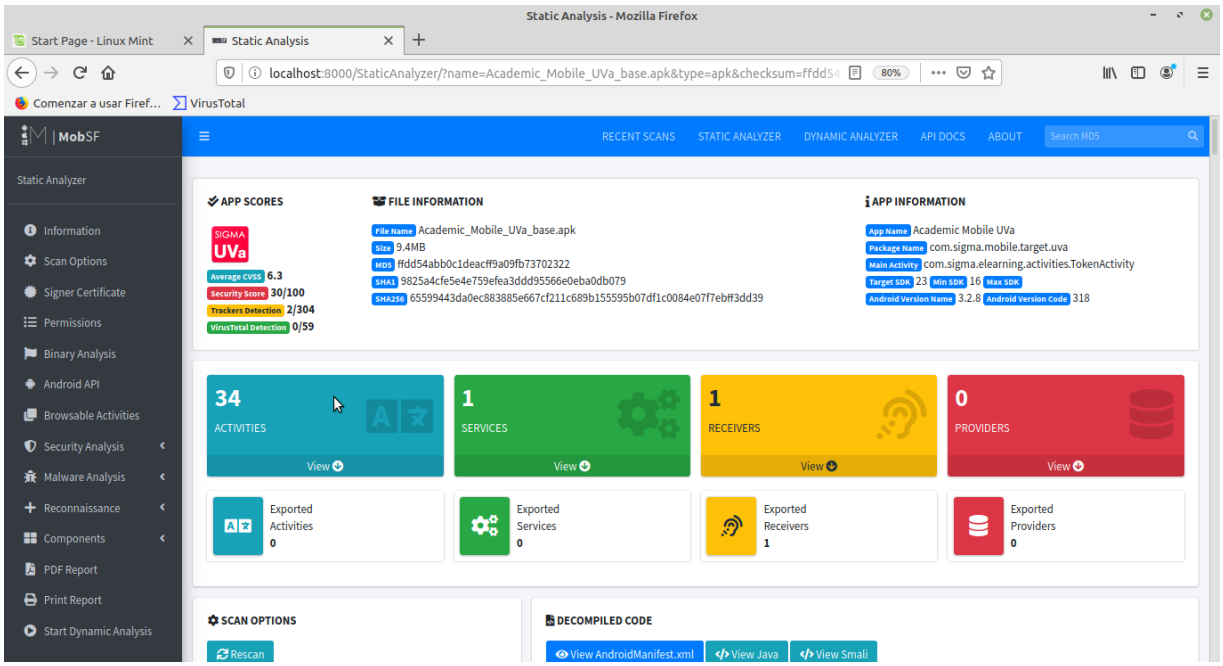



*Figura 21: Reporte del análisis estático de la aplicación Academic Mobile UVa.*

Una vez cargada la aplicación precisa un tiempo para realizar el análisis obteniendo un reporte como el que se muestra en la Figura 21.

En él podemos ver un resumen del análisis, otorgando valores de peligrosidad basados tanto en la propia aplicación como en el **Common Vulnerability Scoring System (CVSS)** que es un estándar de seguridad cuya valoración va del 0 al 10. Luego se nos precisa información sobre la aplicación, como serían su nombre, su versión y su logo. Es importante destacar que mientras que los parámetros que ayudar a calcular (CSVV) son obtenidos de la base de datos de la herramienta que se actualizan conforme lo hace MobSF la información obtenida de la aplicación es obtenida únicamente de la información contenida en él .apk, no siendo en ningún caso cotejada con ninguna fuente online. Durante la realización de pruebas se ha visto como el logo de algunas de las aplicaciones no han sido obtenidos, mostrando en su lugar lo que podríamos ver en la Figura 22.

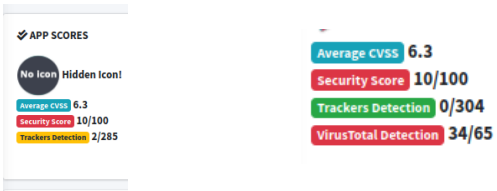



*Figura 22: Ejemplo de icono no encontrado y resultados del escaneo en Virustotal*

Como añadido y haciendo uso de la **API** pública que proporciona Virustotal se ha realizado una **interconexión con la base de datos de Virustotal**. Esto se ha realizado siguiendo los pasos indicados en [9]. Esto nos permite ver si nuestra aplicación ha sido analizada en Virustotal.

En la parte izquierda podemos ver un **menú de navegación** que nos permite acceder de forma rápida a cualquier parte del reporte. Esto lo podemos ver en la Figura 23. Vamos a seguir este menú en cada uno de sus apartados para ver que nos indica.

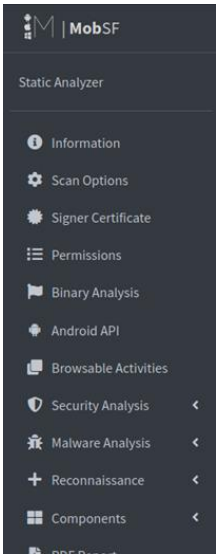



*Figura 23: Menú de navegación en el reporte del análisis estático de MobSF.*

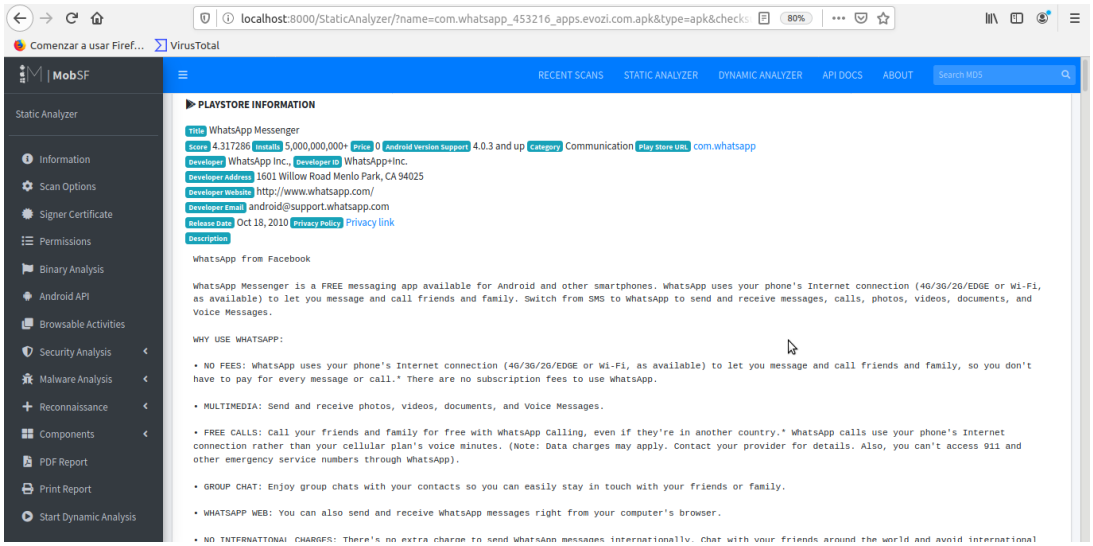



*Figura 24: Vista de la información que se muestra en la Play Store de la aplicación.*

En la Figura 24 podemos ver también la **información que es mostrada en la *Play Store***, la cual está contenida dentro del archivo .apk como se ha mencionado anteriormente.

Según nos desplazamos podemos ver un menú de navegación para acceder a las partes del reporte que más nos interesen como podemos ver en la Figura 25.

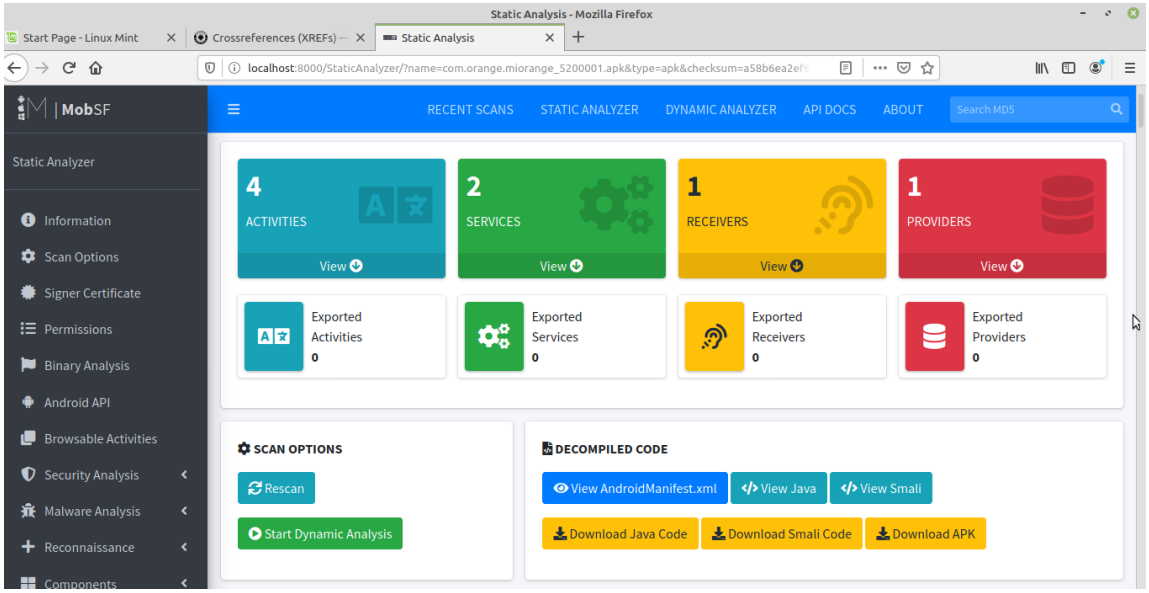



*Figura 25: Menú de opciones del reporte.*

Podemos ver 4 categorías principales que se corresponden con componentes que se encuentran en las aplicaciones de forma usual y procederemos a comentar su significado.

### 8.1.1 Actividades

Usar una aplicación en dispositivos móviles es muy distinto a usarla para la versión de escritorio. Esto se debe en un inicio a que no hay un lugar común en el que el usuario comienza su actividad en la aplicación. Podemos ejemplificar esto en que si el usuario abre una aplicación de correo electrónico desde el menú podría encontrarse una lista de correos mientras que si esa aplicación es abierta desde otra te encuentres una pantalla de la aplicación en la que puedes redactar un correo electrónico.

Es por eso por lo que la clase *Activity* nace para solventar este inconveniente**. Cuando una aplicación necesita de otra, la aplicación que realiza la llamada accede a una actividad de esa aplicación, no accediendo directamente a la aplicación**. La actividad es entonces el punto en el que una aplicación empieza a interaccionar con el usuario.

La actividad crea la ventana en la que la aplicación dibuja su interfaz de usuario. Generalmente, la ventana llena por completo la pantalla, pero es posible que sea más pequeña y esté sobre otras ventanas. Normalmente, una actividad implementa una de las pantallas en una aplicación. Como ejemplo, una actividad de una aplicación puede implementar una pantalla Opciones mientras otra que otra actividad implementa una pantalla Borrar foto.

La mayoría de las aplicaciones contienen múltiples pantallas, entonces incorporan varias actividades. En la mayoría de los casos, una actividad de la aplicación es declarada la actividad principal, siendo la primera pantalla que aparece cuando el usuario lanza la aplicación. Luego, cada actividad puede iniciar otras actividades a fin de realizar diferentes acciones.

Como ejemplo, volviendo al caso de una aplicación siempre de correo electrónico la actividad principal daría una lista de correos electrónicos. A partir de esta actividad se podrían iniciar nuevas actividades que nos proporcionarían pantallas nuevas para realizar diversas tareas como leer cada correo individualmente o redactarlos.

Las actividades trabajan conjuntamente para crear una experiencia de usuario coherente en cada una de las aplicaciones. Estas actividades se encuentran relacionadas entre sí generando pequeñas dependencias que permiten iniciar actividades que pertenecen a otras aplicaciones. Como ejemplo, el correo electrónico podría iniciar la acción de “Compartir actividad" de una app de redes sociales. [10]

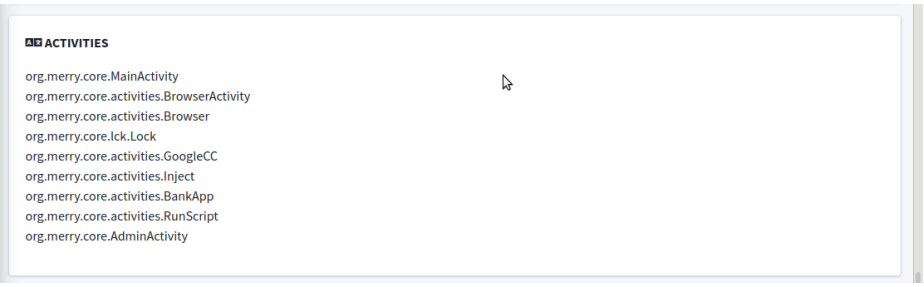



*Figura 26: Información de las actividades de una aplicación en el reporte.*

La información de las actividades debe quedar reflejada en el manifiesto de la aplicación.

### 8.1.2 Servicios

Los servicios son componentes de una aplicación orientados a realizar operaciones de larga ejecución en segundo plano. No proporcionan una interfaz de usuario. Estos servicios una vez iniciados pueden seguir ejecutándose en segundo plano, aunque el usuario haya abandonado la aplicación. Componentes de la aplicación pueden interactuar con el servicio e incluso realizar comunicación entre procesos (IPC). Como ejemplos de lo que es capaz de realizar un servicio en segundo plano tenemos manejar los cambios de red, abrir o cerrar archivos, interactuar con un proveedor de contenido…. [11]

Hay dos tipos diferentes de servicios:

**Primer plano**

Se denomina servicio en primer plano a aquel que durante su ejecución el usuario puede notar. Como ejemplo, nos trasladamos a una aplicación de reproducción de audio. En ella se usa un servicio en primer plano para reproducir una canción. Este tipo de servicios debe mostrar una notificación, y se seguirán ejecutando si el usuario deja de interactuar con la aplicación.

**Segundo plano**

Se denomina servicio en segundo plano a aquel que durante su ejecución el usuario no puede notar directamente. Como ejemplo, si una aplicación comprime su almacenamiento el usuario no es consciente de dicha compresión, por lo que suele ser un servicio en segundo plano.

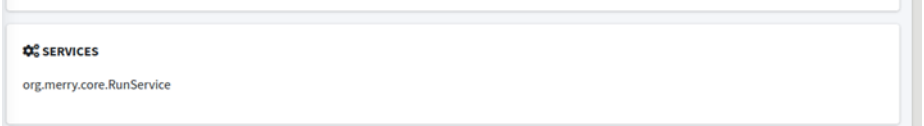



*Figura 27: Información de los servicios de una aplicación en el reporte.*

### 8.1.3 Receptores

En las aplicaciones de Android se pueden recibir o enviar mensajes de difusión amplia (Broadcast). Estos mensajes pueden tratados tanto por el Sistema Operativo Android como por

las distintas aplicaciones. Estos mensajes son enviados cuando tiene lugar un evento de interés. Como ejemplo el Sistema Operativo Android envía mensajes cuando el dispositivo se enciende o se inicia la recarga de su batería. Además, las aplicaciones pueden enviar mensajes personalizados a otras aplicaciones.

Las aplicaciones pueden registrarse para recibir mensajes específicos. Cuando se envía un mensaje de broadcast, el sistema redirige automáticamente los mensajes a las aplicaciones que indicaron querer recibir ese tipo de mensaje en particular.

Generalmente, los mensajes pueden usarse como un sistema de mensajería entre aplicaciones y fuera de la vista del usuario.

Es importante destacar que esto depende de que versión de Android nos encontremos, ya que en versiones posteriores a Android 7.0 estos mensajes cambian su contenido. [12]

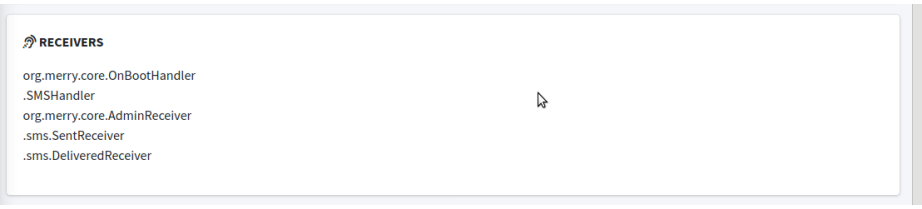



*Figura 28: Información de los receptores de una aplicación en el reporte.*

### 8.1.4 Proveedores

Los proveedores de contenido son herramientas que ayudan a una aplicación a gestionar el acceso a los datos que esa misma aplicación u otra almacena. Además, proporcionan un modo de compartir datos entre las aplicaciones. Los datos son encapsulados añadiendo mecanismos que ayudan a definir su seguridad. Los proveedores de contenido son la interfaz por defecto que relaciona los datos en un proceso con código que se está ejecutando en otro proceso. Estos proveedores de contenido pueden ser configurados para que las aplicaciones accedan de forma segura a los datos de esa aplicación. Esos datos pueden ser leídos y modificados, depende de la configuración, tal y como se ilustra en la Figura 29.

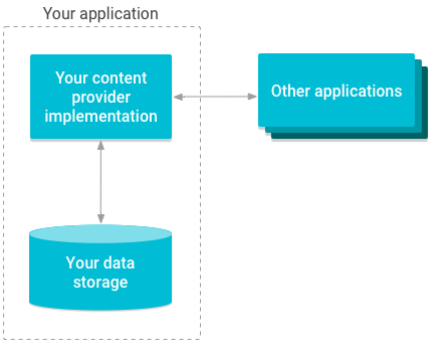



*Figura 29: Explicación del uso de un proveedor de contenido. [13]*

Los proveedores de contenido nos permiten tener un control detallado sobre los permisos de acceso a los datos de las aplicaciones.

En este caso, la aplicación del malware Xbot no usa ningún proveedor. [13]

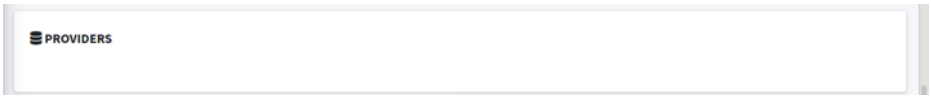



*Figura 30: Ejemplo del estado del reporte sí no hay registro de proveedores en la aplicación.*

Otra funcionalidad muy interesante es que la herramienta permite obtener el fichero *AndroidManifest.xml* así como el **código fuente de la aplicación**, tanto en Java como en Smali. Esto es muy interesante por si necesitamos realizar una búsqueda en profundidad ya que a priori el código de la aplicación en el formato .apk no es legible.

El fichero *AndroidManifest.xml* es importante porque es el fichero que configura la aplicación. Aparece reflejados valores como el identificador único, componentes y sus permisos. Luego se nos muestra el certificado, del que se indica la información relativa al mismo, tal y como su fecha de firma, el algoritmo usado para su cifrado. Esto lo podemos ver en la Figura 31.

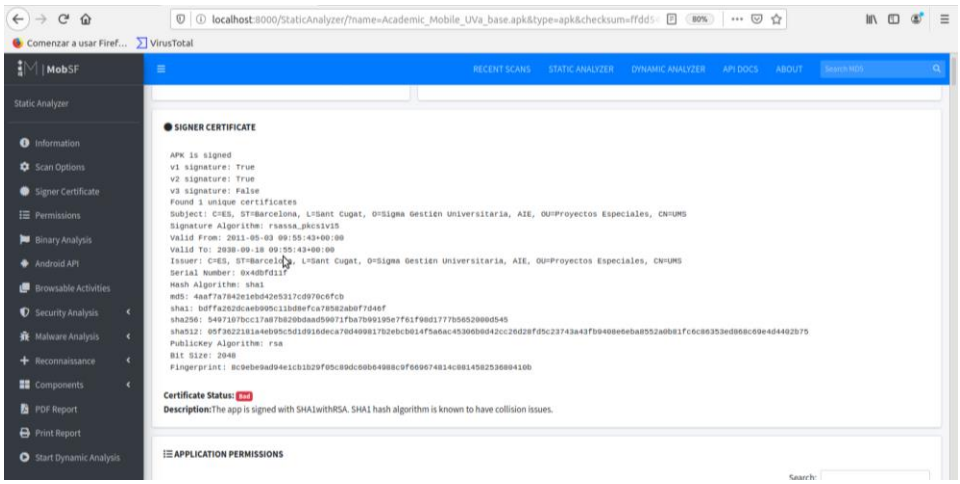



*Figura 31: MobSF muestra el certificado de la aplicación.*

### 8.1.5 Permisos

En el sistema operativo Android hay **privilegios independientes**, en el que cada aplicación se ejecuta con una identidad distinta (ID de usuario Linux y de grupo). Aparte se separan partes del sistema en distintas identidades, aislando las aplicaciones Android entre sí y el sistema operativo.

Pero es con los permisos cuando se ofrecen **funciones de seguridad más precisas**, aplicando restricciones a las operaciones que puede realizar un proceso en particular.

Los distintos permisos del sistema se dividen en varios niveles de protección. Los dos niveles más importantes son los permisos normales y los permisos peligrosos.

Los **permisos normales** están referidos a aquellos en los que la aplicación debe acceder a datos o recursos externos a su zona de pruebas, en los cuales hay un riego mínimo de vulnerar la privacidad del usuario o alterar el funcionamiento de otras aplicaciones en el dispositivo. Por

ejemplo, declarar en que huso horario debe usar la aplicación. Si la aplicación declara un permiso como normal el sistema operativo se lo otorga automáticamente.

En cambio, los **permisos peligrosos** son aquellos en los cuales la aplicación requiere datos o recursos externos en los que hay información privada,podría modificar esos datos o alterar el funcionamiento de otras aplicaciones. Por ejemplo, un permiso peligroso sería ser capaz de leer los contactos del usuario. Para la concesión de estos permisos, el sistema pide al usuario explicitamente su aprobación.[14]

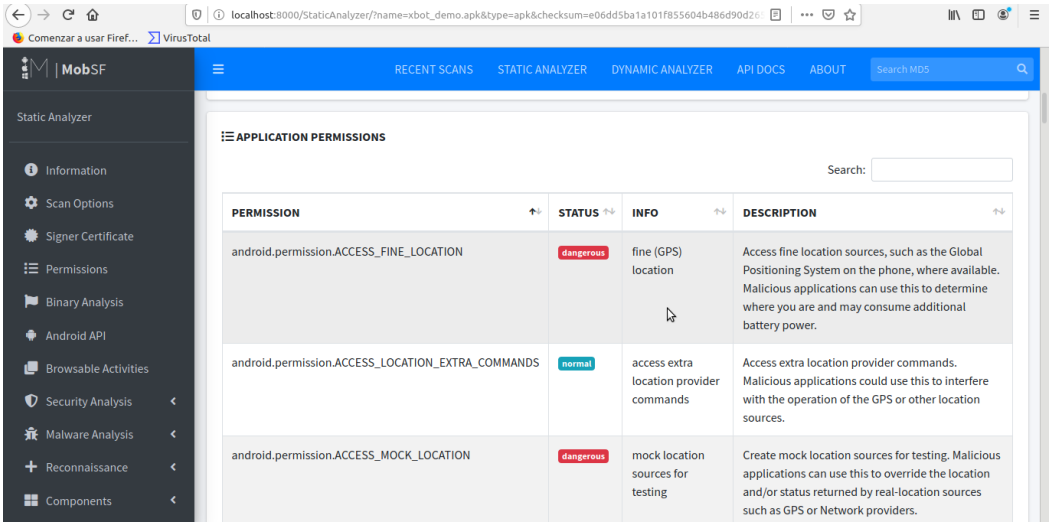



*Figura 32: Vista de los permisos de una aplicación en el reporte.*

### 8.1.6 Android Java API

En esta categoría se nos muestra que ficheros de la aplicación emplean la **interfaz de programación de aplicaciones (API) de Java** la cual es un conjunto de reglas (código) y especificaciones seguidos por las aplicaciones para realizar intercambios de comunicación entre ellas. Esto facilita la interacción entre distintas aplicaciones.

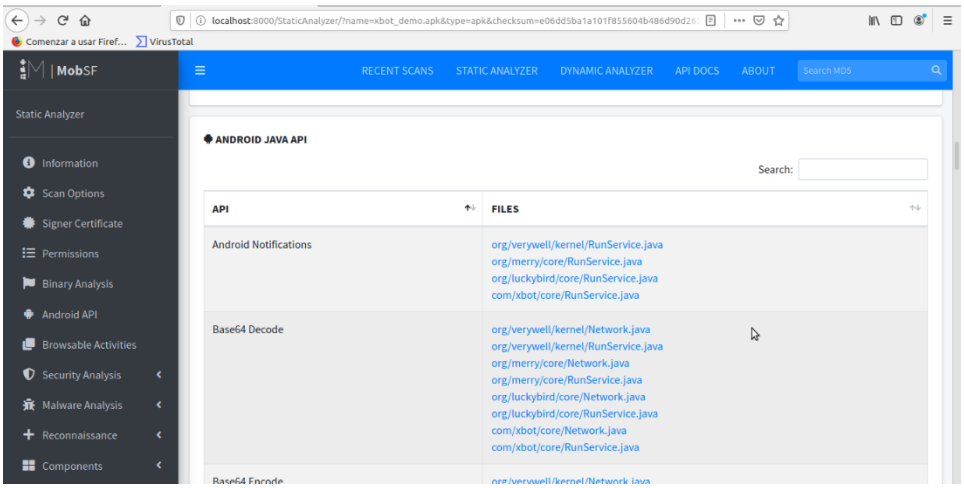



*Figura 33: Vista de la API entre Java y Android.*

### 8.1.7 Manifest Análisis

En este apartado se realiza una revisión de los valores que aparecen en el fichero *AndroidManifest.xml.*

Es decir, de cada uno de los comandos que aparecen el en el fichero *AndroidManifest.xml* se nos indica si es considerado una acción peligrosa y una breve descripción de esta.

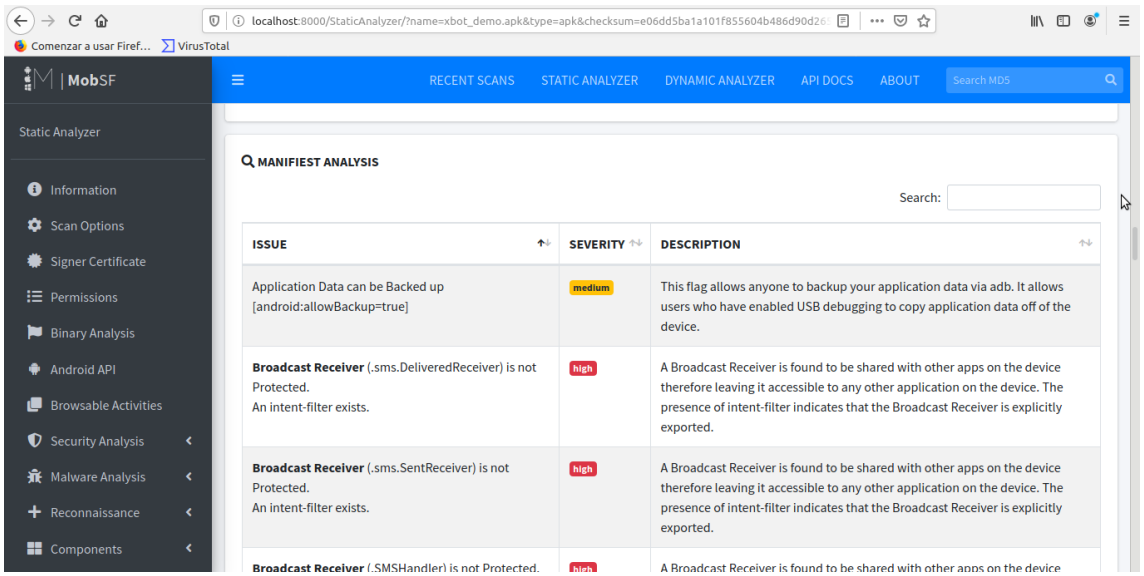



*Figura 34: Análisis de los elementos presentes en el AndroidManifest.xml.*

### 8.1.8 Análisis del código

En este apartado podemos ver como el código de la aplicación ha sido analizado, en busca de patrones de código predefinido. Lo podemos ver en la Figura 35.

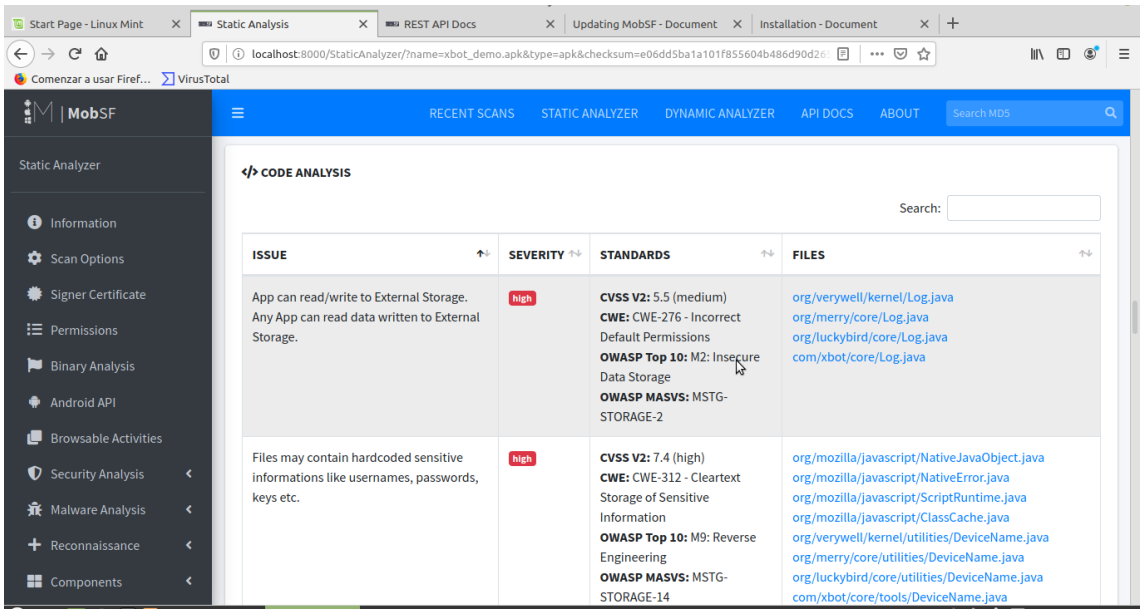



*Figura 35: Reporte analizando el código de la aplicación.*

Se nos informa de en qué ficheros ha sido detectado ese patrón de código, y su peligrosidad basada en los estándares.

El primer estándar con el que es comparado es el mencionado anteriormente como **Common Vulnerability Scoring System (CVSS)** que es un estándar de seguridad cuya valoración va del 0 al 10.

Posteriormente se menciona el **Common Weakness Enumeration (CWE)**. Este estándar es una lista de debilidades conocidas que presentan el software y el hardware. Este estándar cuenta a fecha de este documento con 891 casos. [15]

Se trata de una iniciativa comunitaria, extensamente documentada en la que se refleja de forma clara y concisa cada una de las vulnerabilidades.

Por último, nos encontramos el **OWASP Top 10**. Esta iniciativa sin ánimo de lucro fue creada por el grupo OWASP con la intención de ofrecer a los desarrolladores y equipos de seguridad los recursos necesarios para crear y mantener a las aplicaciones móviles seguras. Uno de sus proyectos principales el **OWASP Mobile Top Ten**. Este proyecto **identifica y categoriza los riesgos más críticos que pueden afectar a la seguridad de las aplicaciones móviles**. En la Figura 37 podemos ver una clasificación de estos 10 riesgos más peligrosos en las aplicaciones móviles.


*Figura 36: Las diez mayores amenazas en dispositivos móviles según OWASP.*

Procedemos a realizar una breve explicación de cada uno de ellos.[1]

**M1-Débiles controles en el lado del servidor:** Es considreado el riesgo más crítico, ya que un error en el lado de servidor va a provocar grandes consecuencias al resto de clientes. Incluye todos los riesgos presentes en los servicios web móviles, las configuraciones de los servidores y las páginas web tradicionales.

**M2-Almacenamiento de datos inseguro:** Se refiere a situaciones en las que la aplicación almacena datos sensibles en un formato de texto plano o un cifrado demasiado trivial. Esto puede provocar riesgos como robo de identidades, fraude…. Dentro de este riesgo se incluye tanto el acceso externo al dispositivo como el acceso usando malware al sistema de ficheros del dispositivo.

**M3-Protección insuficiente en la capa de transporte:** Este problema esta relacionado con la protección del tráfico en la red. Incluye aquellas situaciones en las que la información es enviada en texto plano, confiando en certificados autofirmados o usando mecanismos de cifrado obsoletos. En este caso se puede explotar esta vulnerabilidad sin tener acceso físico al dispositivo, únicamente valdría con estar en la misma red que el dispositivo móvil.

**M4-Filtrado de datos inintencionado:** Está relacionado con el hecho de que el desarrollador sin darse cuenta ubica información o datos sensibles en una localización de muy facil acceso por otras aplicaciones. Suele ser muy frecuente que los **registros de la actividad (logs)** incurran en este hecho.

**M5-Autorización y Autenticación débiles:** Este problema está relacionado con los fallos de autorización y autenticación en las aplicaciones además de los que pudieran ocurrir en el lado del servidor. La autenticación local es muy común en las aplicaciones que necesitan operar sin estar conectadas a la red. Si no se han seguido los controles de seguridad adecuados, existe la posibilidad de que dicha autenticación sea saltada obteniendo acceso a la aplicación.

**M6-Fallos de criptografía:** Está aceptado que toda aplicación que almacene datos en el dispositivo debería encriptarlos para mantener su confidencialidad. Pero ocurre que a veces el método de cifrado es débil dando una sensación de falsa seguridad. En el peor de los casos el atacante obtendrá los datos confidenciales. También se incluyen aquellos casos en los que se almacenan ficheros de claves privadas dentro de la aplicación,introducción dentro del código de claves estáticas y el uso de claves fácilmente obtenibles como sería el identificador de dispositivo de Android.

**M7-Inyección en el lado del cliente:** Esto ocurre cuando la aplicación acepta datos de entrada desde una fuente no confiable. Esto pude ser de forma interna como por ejemplo desde otra aplicación o de forma externa, por un componente del servidor. Un ejemplo podría ser una aplicación de redes sociales en la que se permite a los usuarios realizar publicaciones. Si un atacante fuera capaz de crear una publicación maliciosa y subirla al servidor esta sería propagada al resto de los usuarios de la aplicación móvil.

**M8-Decisiones de seguridad basadas en entradas inseguras:** Este riego está relacionado con el mecanismo de la comunicación entre procesos (IPC) .Como ejemplo podriamos ver una organización en la que sus múltiples aplicaciones comparte un mismo *backend.* Si el desarrollador ha decidido que en vez de que cada aplicación pida al usuario sus credenciales estas compartan una sesión haciendo uso de un token, en el caso de que la seguridad del mecanismo de la comunicación entre procesos no sea la adecuada nos encontraríamos ante un riesgo. Ya que una aplicación maliciosa podría pedir el token y comprometer la sesión del usuario.

**M9-Manejo inadecuado de la sesión:** El concepto de sesión es muy importante en las aplicaciones móviles. Es la forma en la que se puede mantener un estado para el usuario a través de protocolos que no admiten esta funcionalidad como sería HTTP. Pero la introducción de este concepto conlleva que numerosos riesgos salen a la luz como serían la exposición del token o la sobreescritura de parámetros.

**M10- Falta de protecciones binarias:** Esta vulnerabilidad esta referida a las protecciones que un desarrollador puede incorporar a su aplicación móvil. Normalmente sirven para ralentizar la actividad del atacante, en su intento de analizar, hacer ingeniería inversa o modificar el comportamiento de una aplicación.

### 8.1.9 APKiD

APKiD da información sobre cómo se empaquetó un fichero APK. Como se puede ver en la Figura 36 vemos que identifica parta de la estructura del código como **código Anti-VM**. Esto se debe a que en muchas ocasiones como hemos mencionado anteriormente muchas de las funcionalidades de una aplicación malware tratan de ser ocultadas. Haciendo uso de métodos como este, se consigue que si se realiza en análisis dinámico en un dispositivo virtual esa funcionalidad del malware no va a poder ser percibida.

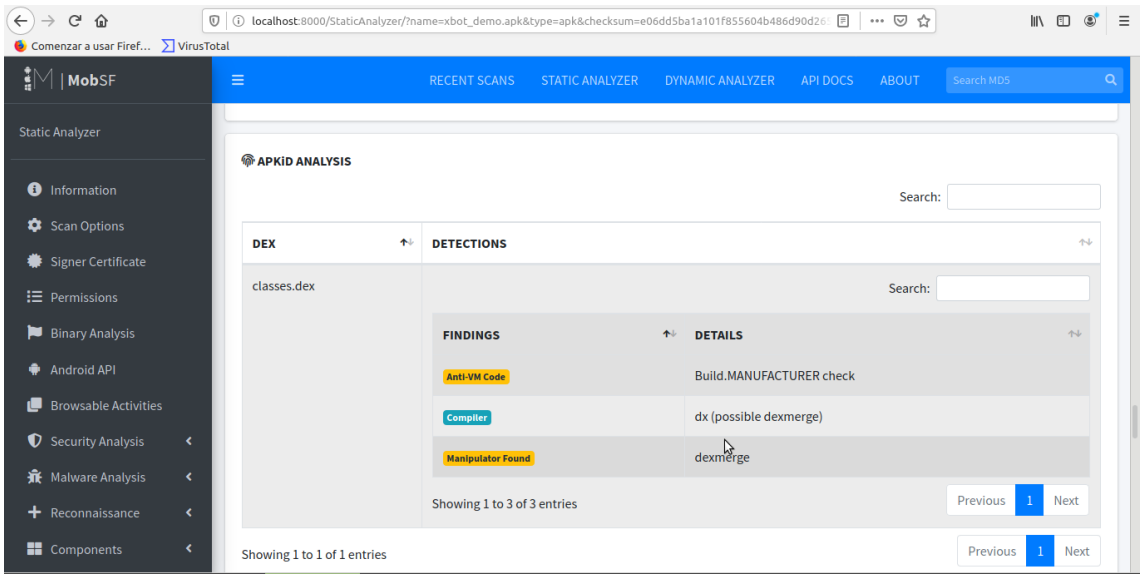



*Figura 37: Análisis del APKiD*

### 8.1.10 Dominios y Direcciones en el código

En la Figura 38 podemos ver algunas de las **direcciones IP** a las que se hace referencia en el código y su reputación en base a la base de datos de MobSF. MobSF incorpora además un módulo del Geolocalización, por lo que disponemos de la **información relativa a la ubicación de esas direcciones IP**. Por último, está enlazado con *Google Maps* permitiendo ver in situ dicha localización.

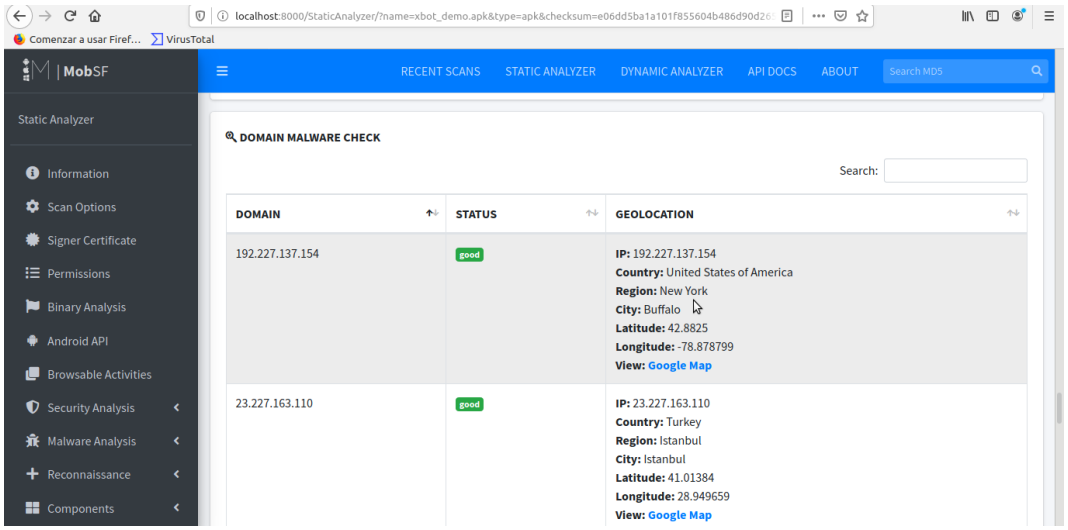



*Figura 38: Direcciones IP encontradas en el código.*

En la Figura 39 podemos ver las URL que se encuentran el código, así como en que fichero del código se encuentran.

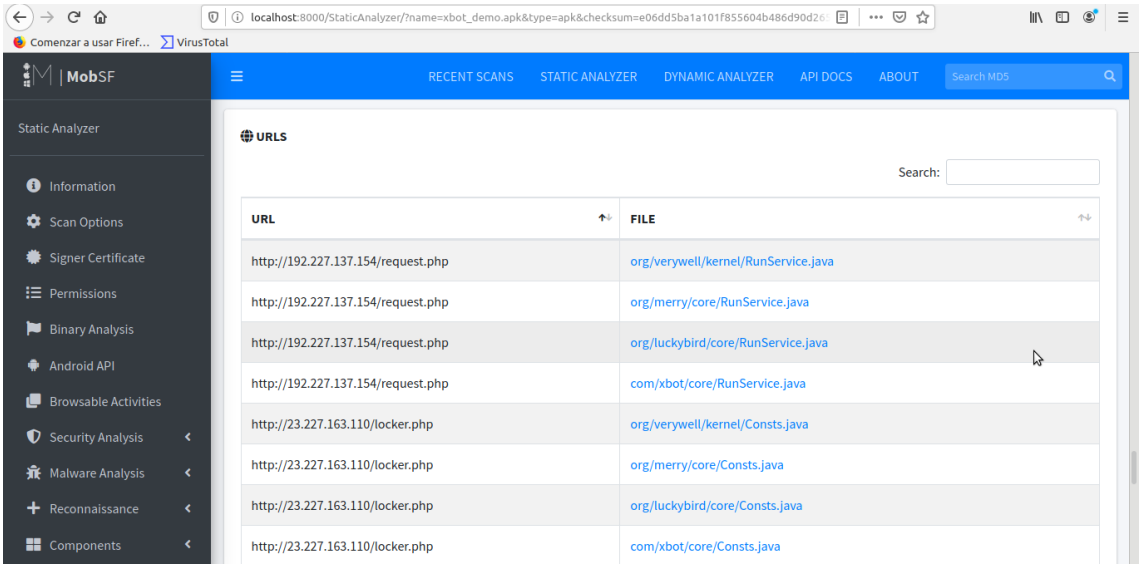



*Figura 39: Direcciones de páginas web encontradas en el código de la aplicación.*

Dichas direcciones web pueden apuntar a dominios o direcciones IP que tras una búsqueda en un navegador podríamos ver asociadas a noticias de aplicaciones maliciosas, lo cual es claramente un indicador de que nos encontramos ante una aplicación maliciosa.

### 8.1.11 Cadenas

En el apartado *String* podemos encontrar un cajón de sastre en el que se vuelcan todas las estructuras del código consideradas interesantes por la herramienta. Depende totalmente de la aplicación. En algunas aplicaciones se han observado gran cantidad de registros mientras que en otras la información es mínima.

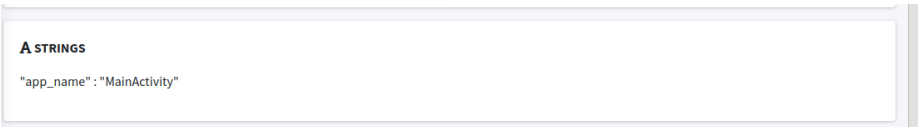



*Figura 40: Apartado en el que almacenan todas las cadenas identificadas en el código de la aplicación.*

### 8.1.12 Ficheros

En este apartado se muestran **los ficheros contenidos en él .apk**. Como se ha visto anteriormente el formato .apk contiene una distribución característica que vamos a recordar.

- ***AndroidManifest.xml***: Fichero de configuración de la aplicación. En él se indican aspectos como el identificador único de la aplicación, los componentes de esta como serían las actividades, los receptores o sus permisos.
- **META-INF:** Directorio que guarda los datos referidos a la firma digital de la aplicación Android.
- **classes.dex:** Es el código de la aplicación compilado para que sea interpretado por la máquina virtual.

- **res:** Contiene las imágenes, texto, .xml usados por la aplicación.
- **resources.arsc:** Almacena compilados los recursos de la aplicación.
- **lib:** Contiene las librerías necesarias para la ejecución de la aplicación

Tal y como podemos ver en la Figura 41 la herramienta ha desempaquetado el formato .apk mostrando los ficheros que componen la aplicación.

Podría también darse el caso de que algún fichero contuviera algún nombre sospechoso, pero esto es poco probable ya que el desarrollador del malware tiende a encubrir sus acciones.

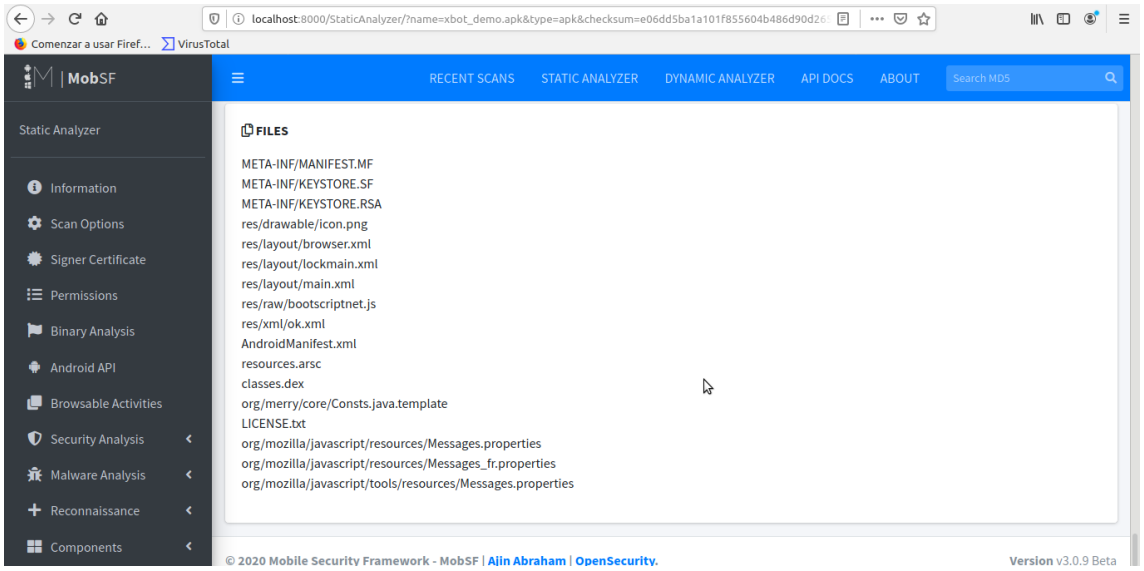



*Figura 41: Archivos que componen el formato APK.*

Por último, este reporte se puede obtener gracias a una gran utilidad de la herramienta es que permite la generación de este reporte tanto en .json como en PDF. Para la obtención del reporte en formato JSON se ha realizado un script que se pude ver en el Anexo 2.

Vamos a explicar en qué consiste el formato JSON. EL nombre proviene **de Notación de Objetos de JavaScript**. Se trata de un formato ligero para el intercambio de datos. En sencillo de entender y escribir por humanos, así como fácil de interpretar y generar para las máquinas. Está compuesto por:

- Una colección de pares de nombre/valor
- Una lista ordenada de valores.

Estas dos estructuras son universales, y casi aceptadas por todos los lenguajes de programación, de ahí su idoneidad.

## 8.2 Análisis Dinámico

Para la realización de este análisis se ejecuta la aplicación móvil en un entorno controlado, que se denomina **Sandbox**. Durante el análisis podemos monitorizar los cambios que sufre la aplicación, los recursos a los que esta accede… En definitiva, nos permite analizar de forma adecuada su comportamiento. Para entender cómo se realiza este análisis se adjunta un esquema simplificado en la Figura 42.

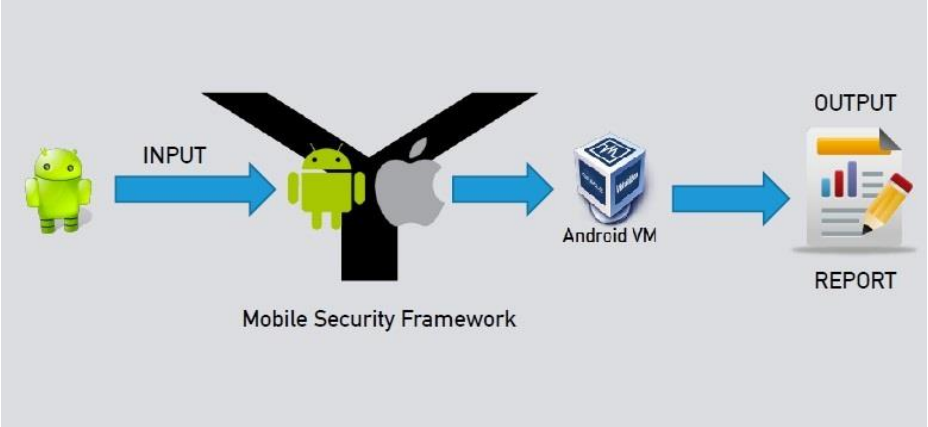



*Figura 42: Esquema del análisis dinámico en MobSF.*

Partiendo de nuestro fichero de la aplicación, este será cargado en MobSF. Después MobSF actuará sobre la máquina virtual Android que se está ejecutando (en nuestro caso será haciendo uso de Genymotion). Podremos navegar por la aplicación, mientras se realiza el análisis basado en la recopilación de los datos de la aplicación y los obtenidos de nuestro dispositivo virtual. Posteriormente se generará un reporte que nos permitirá ver los resultados.

Para realizar este análisis previamente debemos crear un terminal virtual en la misma máquina en la que se encuentra instalado MobSF. Para ello debemos usar Genymotion, que hace uso de *Oracle VM VirtualBox* por lo que debemos tenerlo instalado previamente.

Los pasos para la instalación de Genymotion pueden encontrarse en. Estos pasos son muy claros y concisos permitiendo una instalación sencilla del emulador de dispositivos virtuales de Android en nuestro equipo.

El análisis se realiza siguiendo la siguiente arquitectura, que podemos ver en la Figura 43.

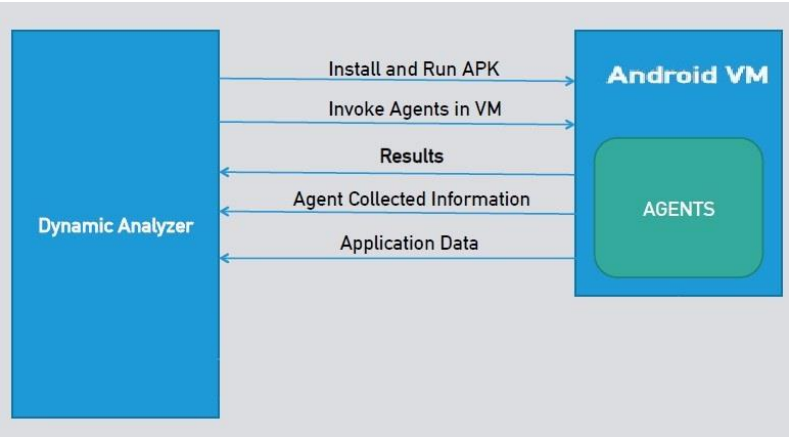



*Figura 43: Esquema detallado del análisis dinámico en MobSF.*

El procedimiento es siguiente. La herramienta MobSF al realizar el análisis dinámico instala y ejecuta la aplicación en el terminar virtual que tenemos encendido. Mientras realiza ese proceso, llama a agentes en el terminal virtual. Haciendo uso de la información proporcionada por esos agentes y los propios datos de la aplicación genera un informe siguiendo la estructura vista en la Figura 43.

### 8.2.1 Genymotion

En la Figura 44 podemos ver la ventana de inicio de Genymotion una vez hemos iniciado sesión.

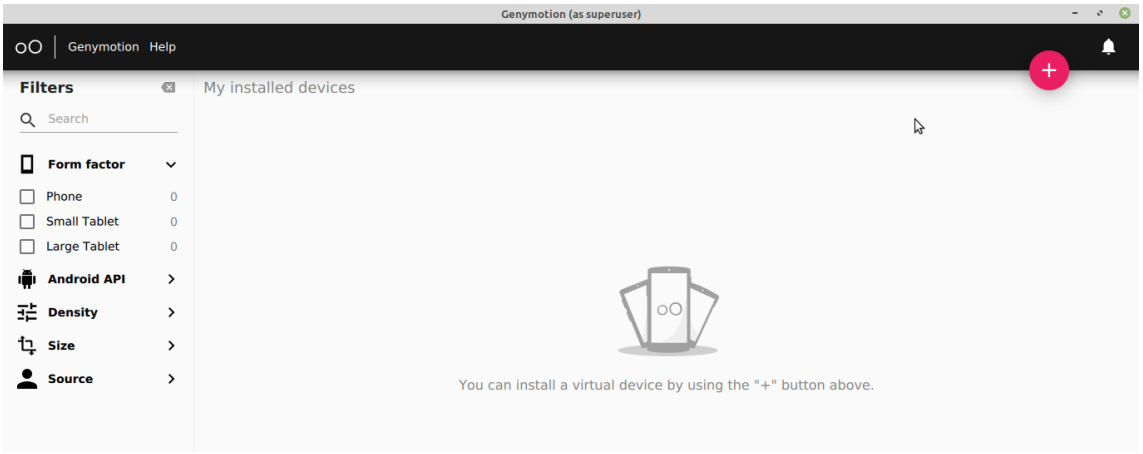



*Figura 44: Menú principal de Genymotion.*

Pulsando en + se nos despliega el siguiente menú que podemos ver en la Figura 45. Allí configuramos el dispositivo virtual que usaremos. Vemos que se puede realizar una búsqueda entre las opciones disponibles. Hay **dispositivos ya preconfigurados**, pero sin embargo podríamos crear el dispositivo a nuestra elección. Es decir, elegir el tipo de dispositivo, variar su versión de Android, su API que como ya henos comentado es importante para el manejo de muchas de las aplicaciones, el tamaño de su pantalla...

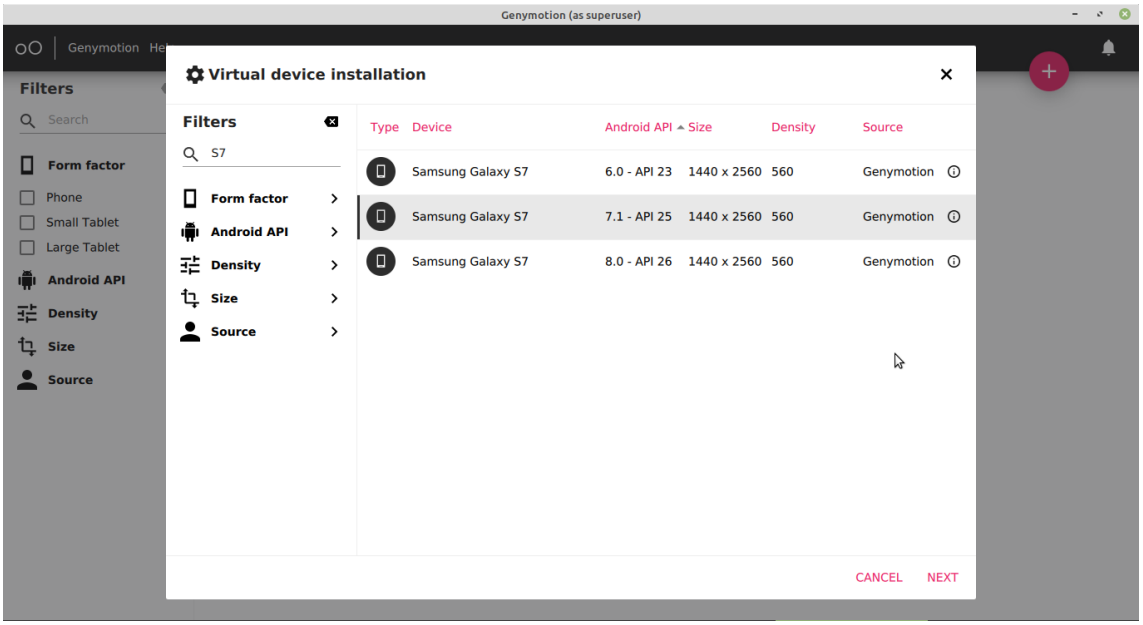



*Figura 45: Elección de las características del terminar virtual.*

Una vez creado, quedará en el programa cargado, desde donde podemos ver **las características que presenta nuestro terminal virtual.** A parte de ver estas características podremos iniciar, eliminar nuestro dispositivo. Aunque en la Figura 46 únicamente se puede ver un dispositivo, se pueden tener creados múltiples dispositivos.

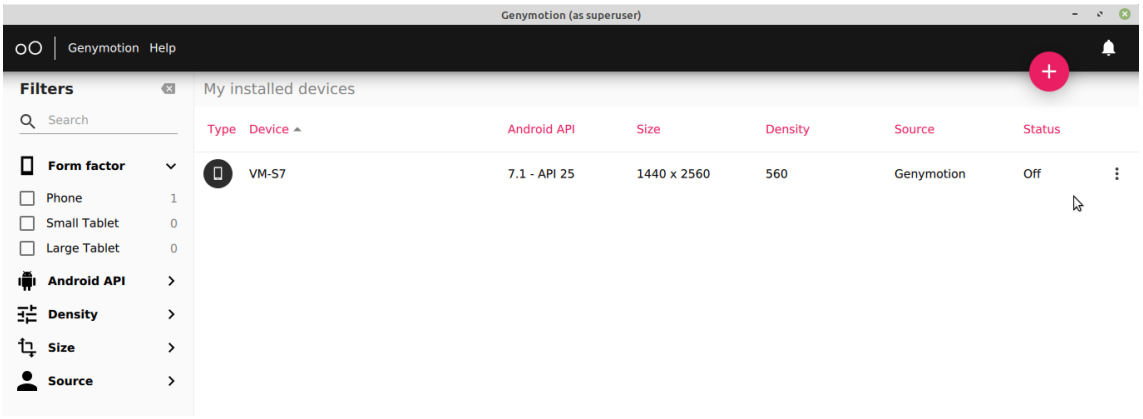



*Figura 46: Menú principal de Genymotion con un terminal ya creado.*

Cuando cambiamos su *Status* a On, el dispositivo se inicia tardando unos segundos en estar disponible como en la Figura 47.

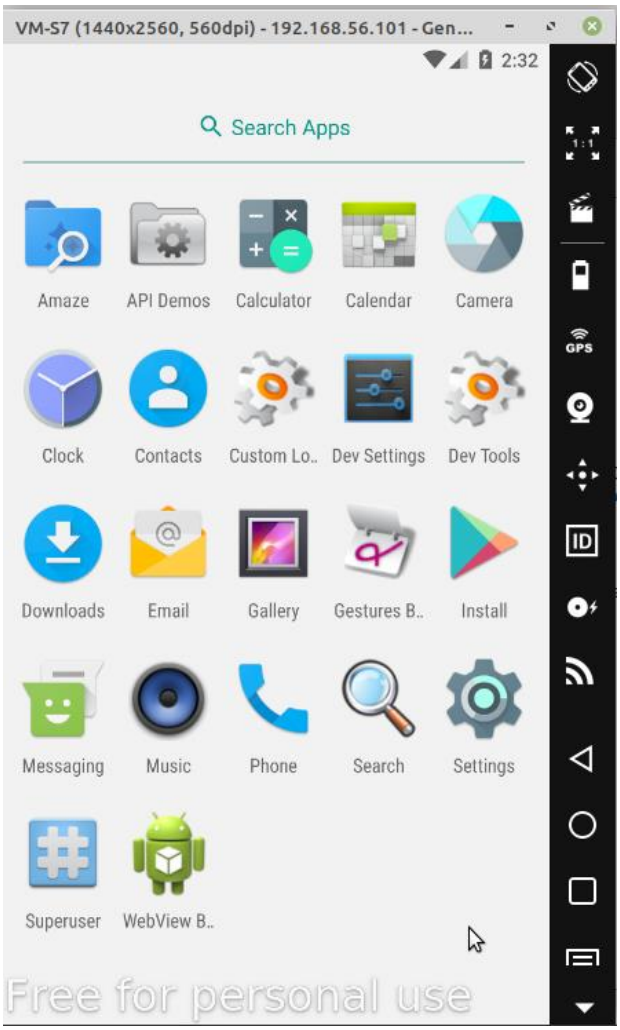



*Figura 47: Terminal virtual listo para ser usado.*

Una vez realizado esto, e iniciado el dispositivo estamos en condiciones de realizar el análisis dinámico con MobSF. No hace falta ninguna configuración más, pero si surgiera algún problema de interconexión en [9] se detalla cómo realizarla.

Ahora MobSF sí que nos muestra que el análisis dinámico está permitido, como podemos ver en la Figura 48.

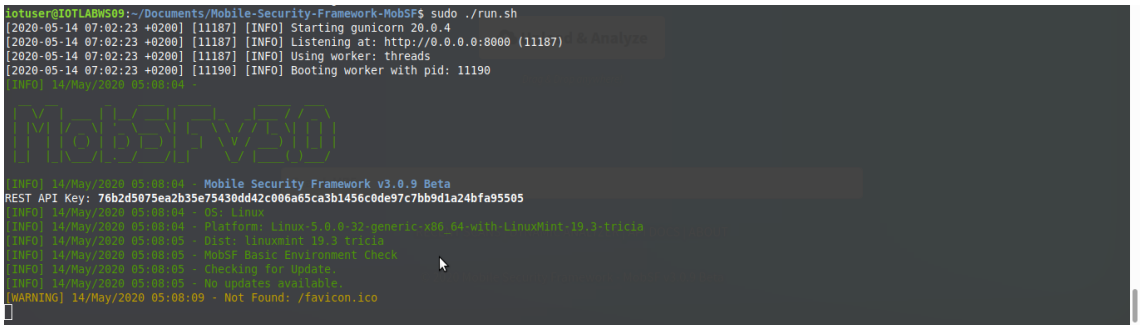



*Figura 48: MobSF nos indica que es capaz de realizar el análisis dinámico.*

Cuando realizamos el análisis dinámico en la CLI obtenemos los siguientes mensajes si se ha realizado de forma correcta.

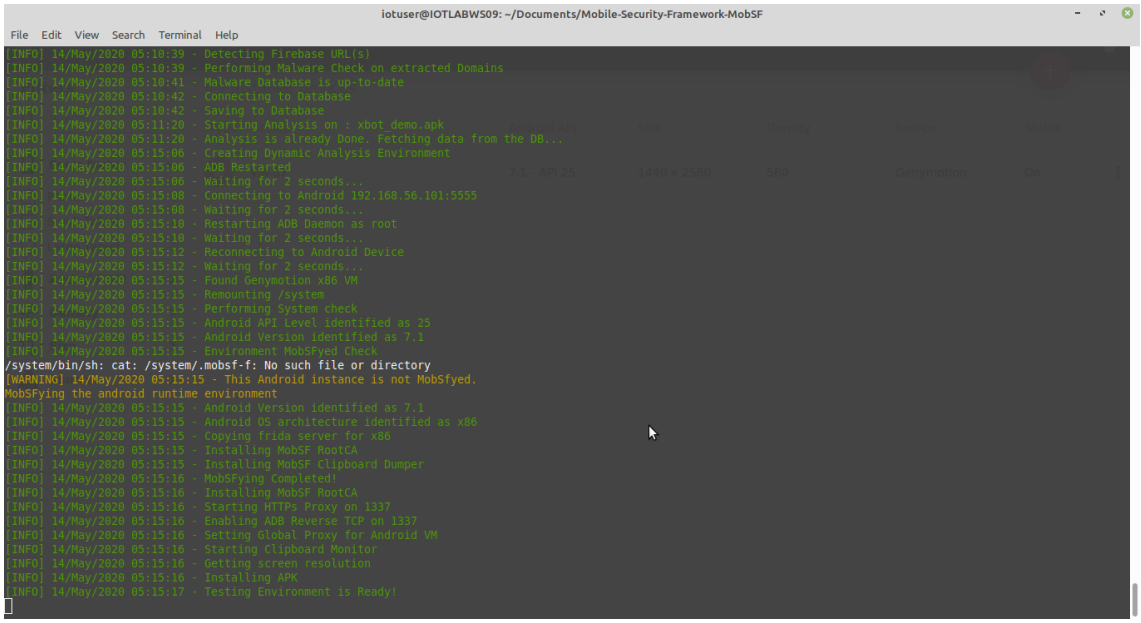



*Figura 49: Registros del inicio de forma correcta del análisis dinámico.*

### 8.2.2 Menú análisis

Si acedemos al análisis dinámico en la herramienta de una aplicación nos encontramos lo siguiente, que podemos ver en la Figura 50.

En él se detallan las diversas opciones que tenemos para realizar el análisis dinámico. Entre ellas destacan poder interactuar con el terminal virtual desde la herramienta, tener acceso privilegiado, monitorizar las actividades o mostrar en tiempo real el flujo del Logcat.

Logcat es una herramienta la cual vuelca un registro de mensajes del sistema, los errores del sistema y los mensajes que escribes desde una aplicación con la clase **Log**.

Podemos elegir mostrar la pantalla, tendremos este aspecto en la herramienta. Es importante destacar que aunque seamos capaces de ver el terminal en el navegador mientras realizamos el análisis dinámico, para controlar el dispositivo debemos usar la ventana de la herramienta Genymotion.

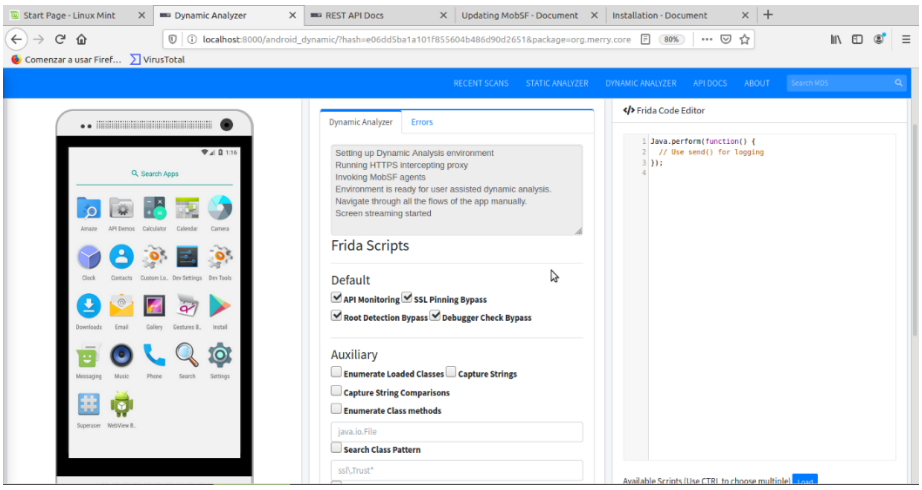



*Figura 50: Realizando el análisis dinámico en MobSF.*

Aquí podemos ver como una vez iniciado el análisis dinámico, nuestra aplicación aparece en el menú del dispositivo virtual. Esto pasa porque MobSF ha procedido a la instalación de la aplicación en el dispositivo virtual. Esto lo hace de forma automática, no necesitando transferir la aplicación al dispositivo virtual y realizar la instalación.

También como se puede ver en la Figura 49 que el análisis dinámico presenta una terminal en la que cargar análisis de Frida. Vamos a ver que es Frida y que implica esto.

Frida es una herramienta que permite inyectar fragmentos de código Javascript o de tu propia biblioteca en aplicaciones nativa de Windows, macOS, GNU/Linux, iOS, Android, and QNX. En este caso nos interesa en aplicaciones del sistema operativo Android. Todo esto nos permite realizar análisis mucho más exhaustivos del desempeño de la aplicación a analizar. [16]

También se pueden realizar capturas de pantalla de la aplicación en ejecución. Esas capturas de pantalla lego se pueden encontrar en el reporte en su correspondiente apartado.

Al iniciar la aplicación del malware XBOT, inicia detectando que se encuentra en un dispositivo virtual, por lo que no actua de forma normal como se esperaría de una aplicación lícita. Lo podemos ver en la Figura 51.

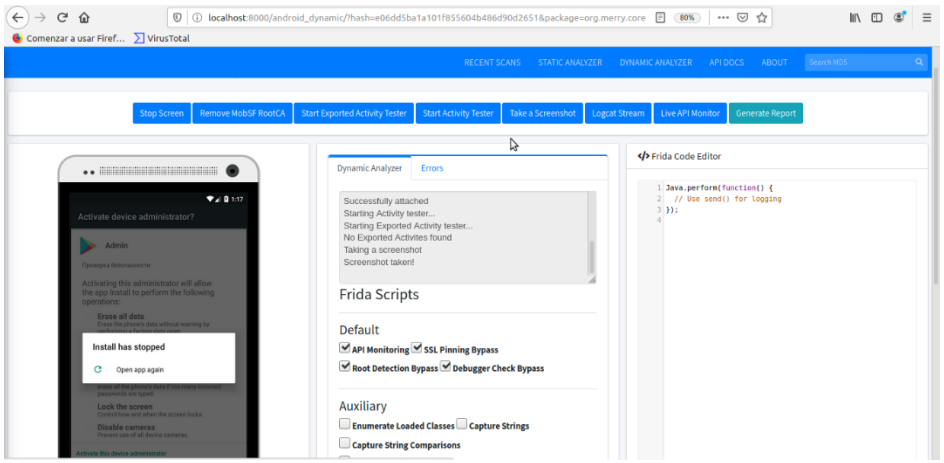
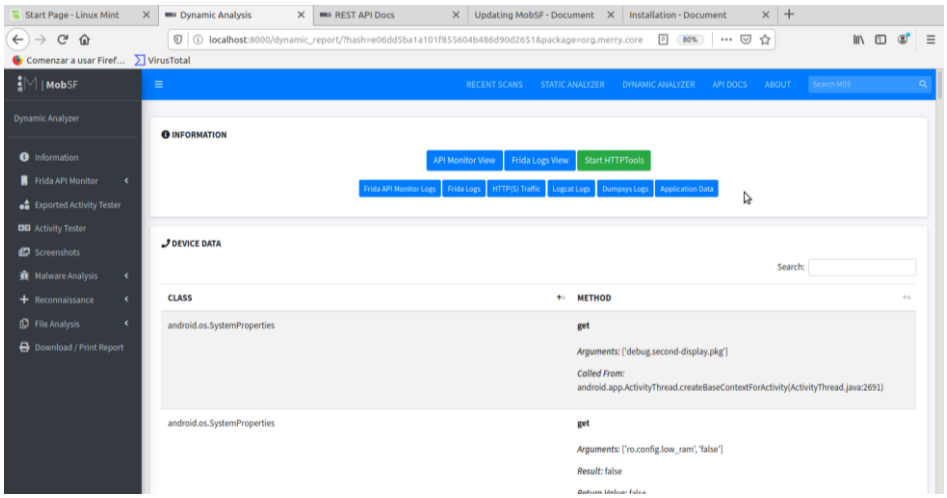



*Figura 51: La aplicación con el malware XBOT comportándose de forma distinta al ser un dispositivo virtual.*

Tras esto vamos a proceder al generar el reporte del análisis dinámico. El resultado lo podemos ver en la Figura 52. Es presentado al igual que el reporte estático con un menú de navegación en el lado izquierdo.



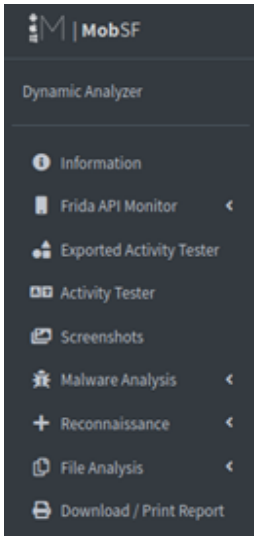




*Figura 52: Menú de navegación el reporte del análisis dinámico.*

Vamos a proceder a desplazarnos por el reporte generado.

### 8.2.3 Información y datos del dispositivo

En el reporte podemos ver los datos e información del dispositivo, como puede verse en las Figura 53. Así como los métodos usados para obtener la información, así como los argumentos pasados en los métodos. Por último, se ve que actividad ha realizado el método usado para obtener la información.

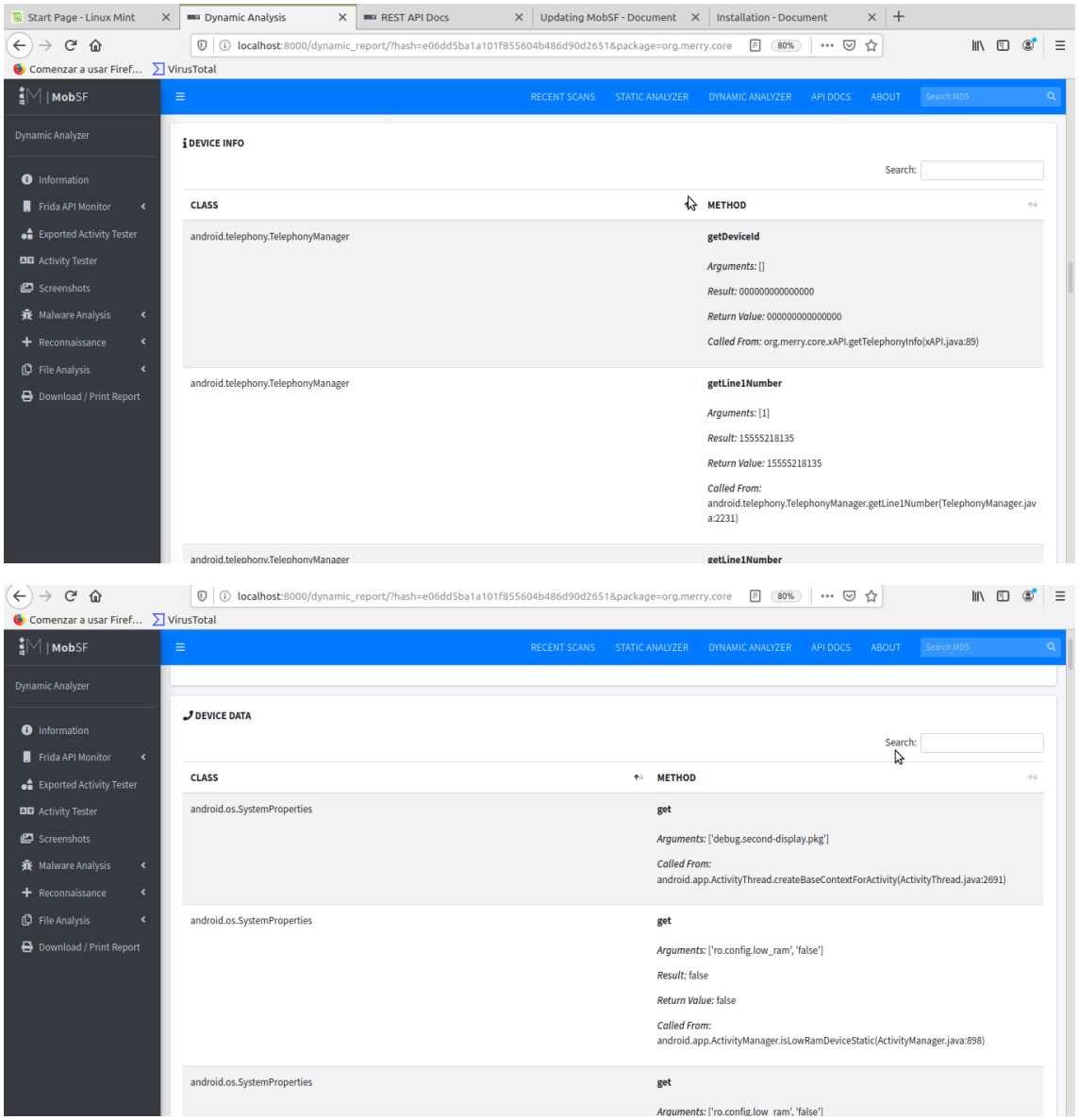



*Figura 53: Reporte de la información y datos del dispositivo obtenidos por la aplicación.*

### 8.2.4 API de la aplicación

Así como también ver un monitor de la API de la aplicación, mostrando los métodos de acceso del sistema operativo usado por la aplicación esto lo podemos ver en la Figura 54.

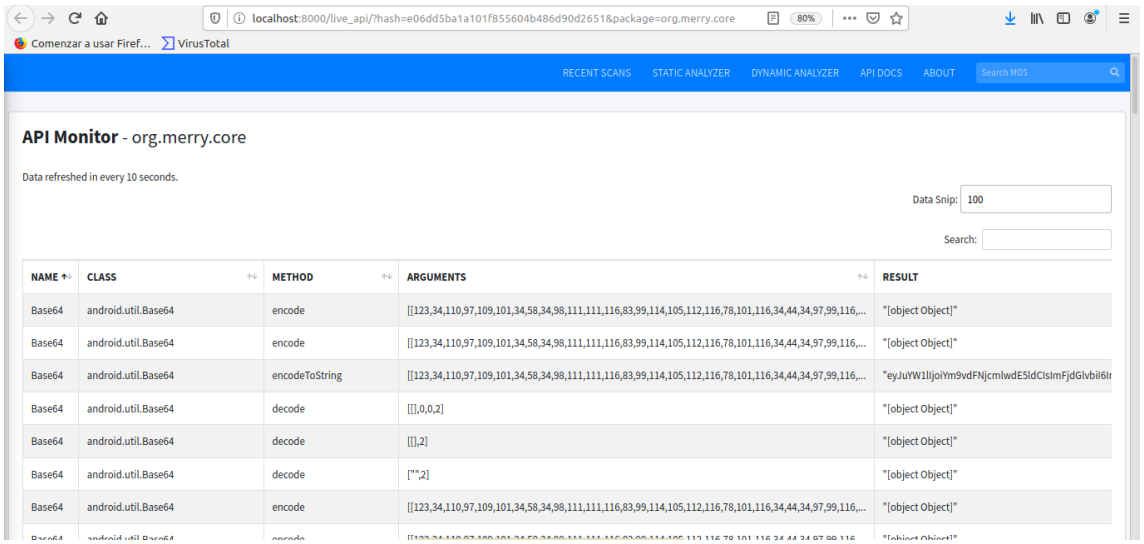



*Figura 54: Reporte de la monitorización de la API.*

### 8.2.5 Logcat

Podríamos ver también el Logcat generado por el sistema, lo que nos permitiría ahondar en nuestro análisis.

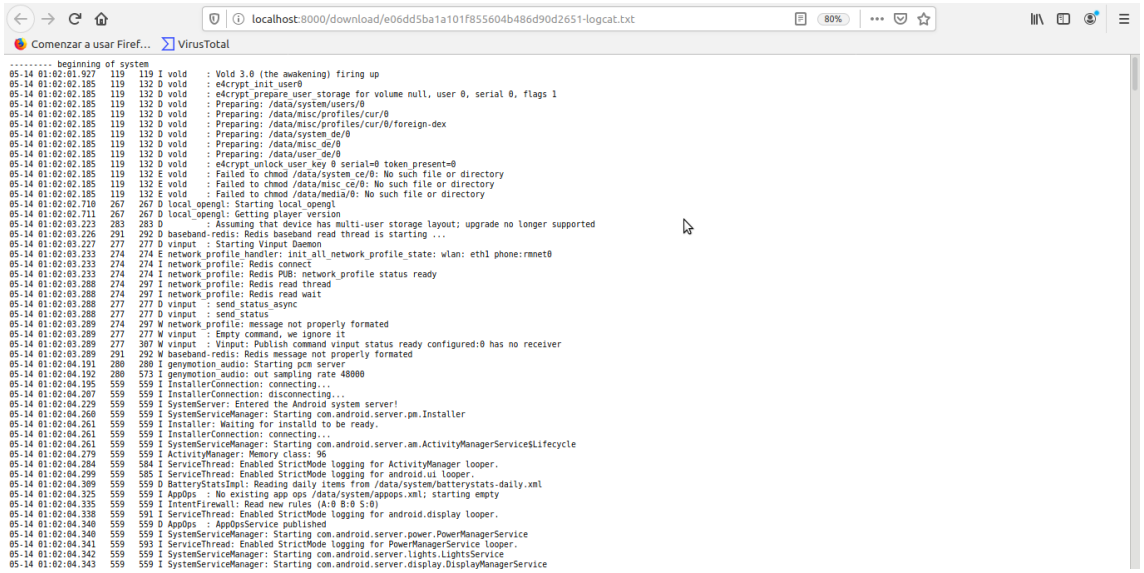



*Figura 55: Reporte del Logcat del dispositivo.*

El Logcat se obtiene en formato de archivo de texto plano, por lo que es muy sencillo de consultar y filtrar.

### 8.2.6 Eventos de comunicación entre procesos

El sistema operativo Android ofrece un mecanismo para comunicación entre procesos (IPC) que usa llamadas de procedimiento remoto (RPC). Vamos a explicar cómo ocurre. Esto sucede cuando un componente de la aplicación móvil llama a un método, pero este se ejecuta en otro proceso, es decir de forma remota. Los resultados son devueltos al emisor. Esto lo podemos ver en la Figura 56.

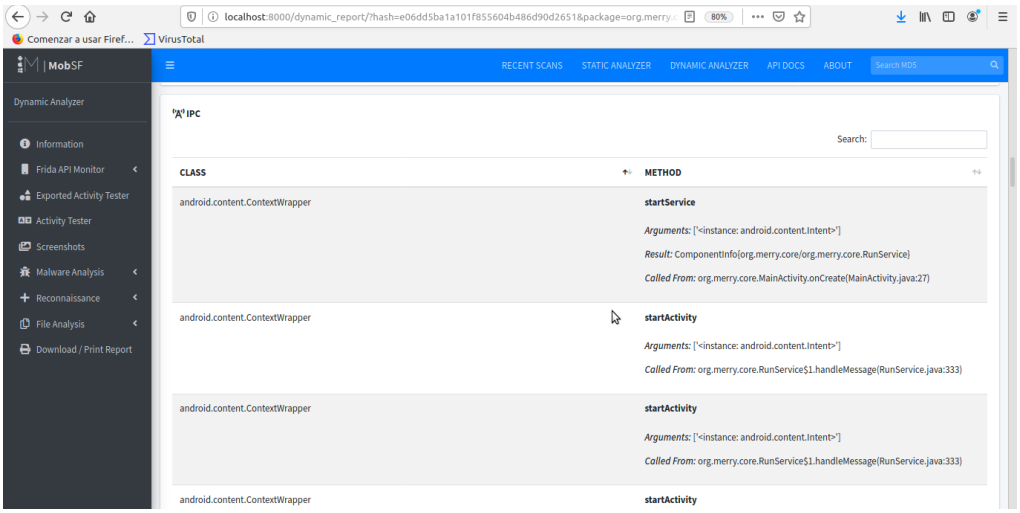



*Figura 56: Eventos de comunicación entre procesos iniciados por la aplicación a analizar.*

### 8.2.6 Dominios y Direcciones

Por último, al igual que en el reporte estático obtenemos también un registro de las direcciones IP y direcciones web empleadas por la aplicación en el caso de que las use.

# Capítulo 9: Apklab.io

Se ha analizado también la herramienta apklab.io desarrollada por Avast. Esta herramienta totalmente web, se pude encontrar en [17] muestra un aspecto inicial que podemos ver en la Figura 56. Al solicitar una licencia para esta herramienta, sus desarrolladores contactaron. Sin embargo, rehusaron compartir ningún tipo de información ni comentar como se realizaba exactamente el análisis dinámico.

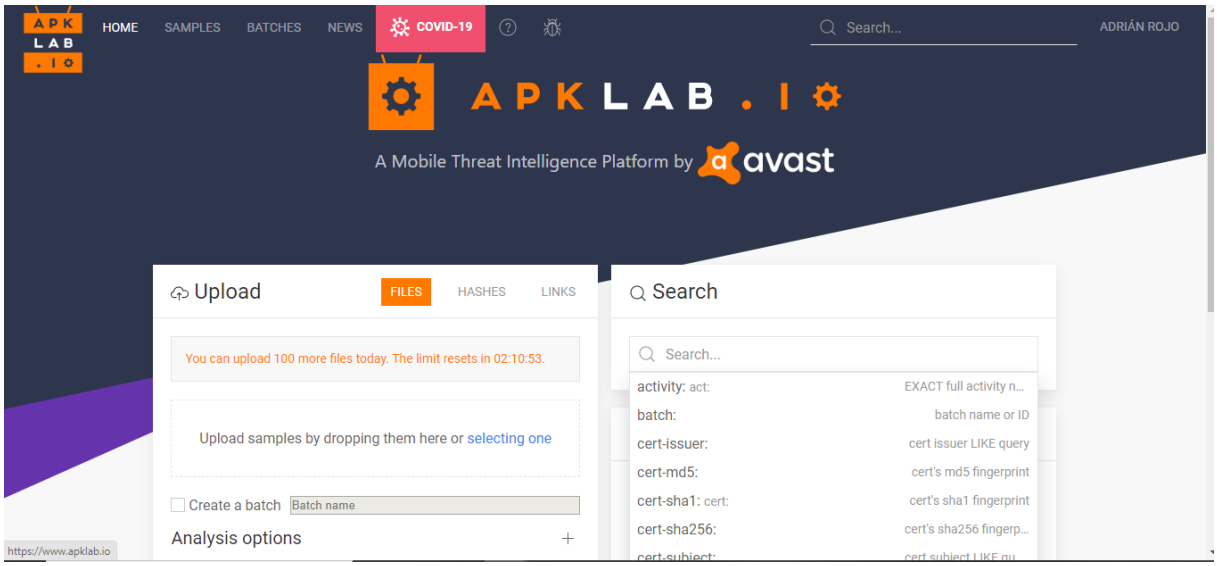



*Figura 57: Menú principal de apklab.io.*

Aquí se nos muestra la opción de cargar un fichero o de realizar una búsqueda sobre la base de datos que contiene la herramienta. En la Figura 58 podemos ver que opciones nos permite para la carga de ficheros:

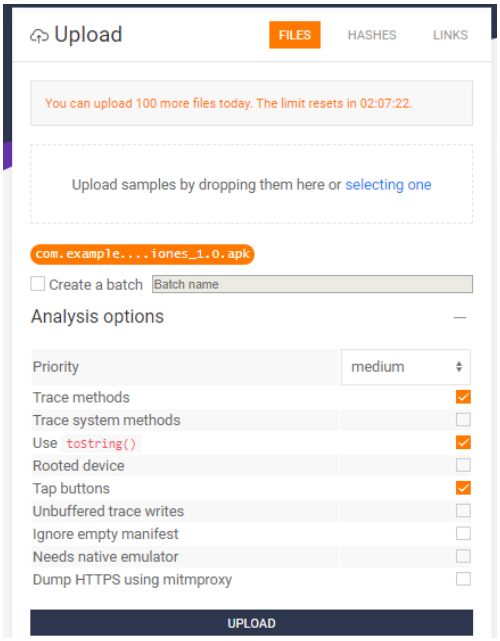



*Figura 58: Subida de ficheros en apklab.io*

Se nos permite crear un lote, ya que podemos subir a la vez varias aplicaciones, asignando un nombre lo que es útil de cara a una vez subida nuestra aplicación realizar una búsqueda y

encontrarla fácilmente. En cuanto a las opciones que se nos presentan podemos ver que se nos da la opción de trazar los métodos de la aplicación, del sistema, ejecutar la simulación en un dispositivo siendo usuarios privilegiados… Si mantenemos el cursor sobre la opción encontramos una descripción más detallada de la misma.

Una vez cargada la aplicación obtenemos un reporte como el que podemos ver en la Figura 59.

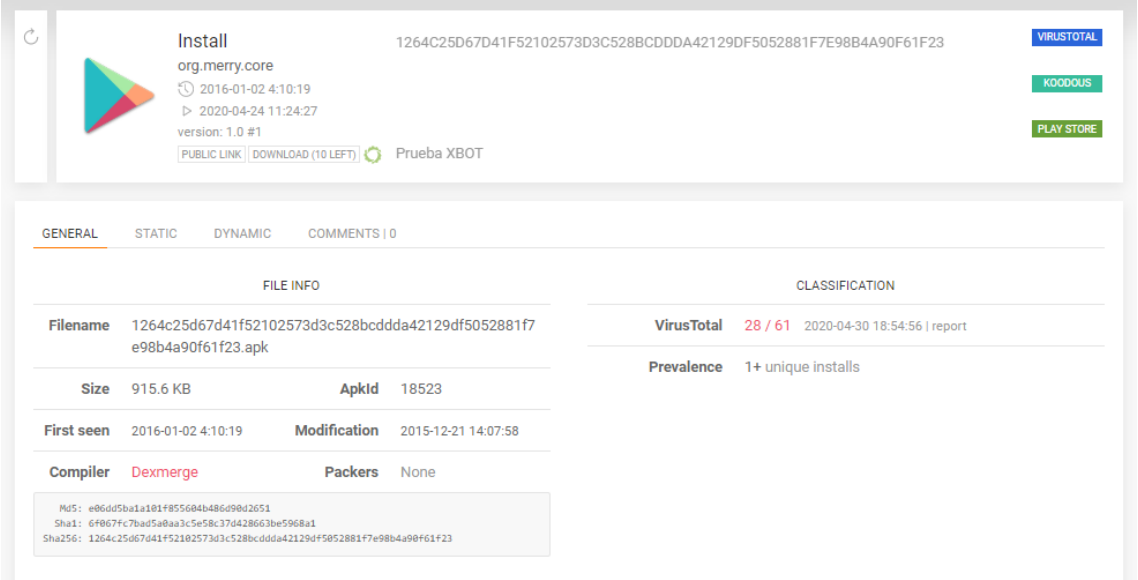



*Figura 59: Reporte de la aplicación analizada.*

En él se nos muestra una información similar a la obtenida mediante MobSF. En la Figura 60 podemos ver como se nos proporciona el fichero *AndroidManifest.xml*, la información de los certificados y un listado de las direcciones a las que se apunta.

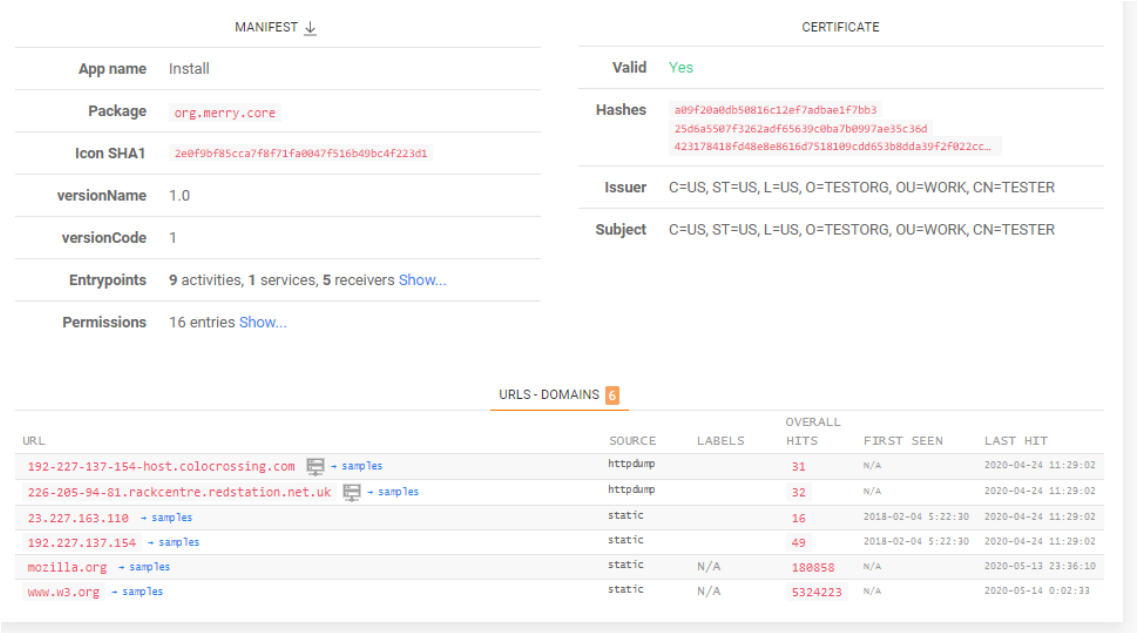



*Figura 60: Información general de la aplicación*

## 9.1 Análisis estático

Pasamos a ver entonces el reporte del análisis estático realizado por apklab.io. Aquí podemos ver la información de la aplicación como serían los receptores que contiene la aplicación.

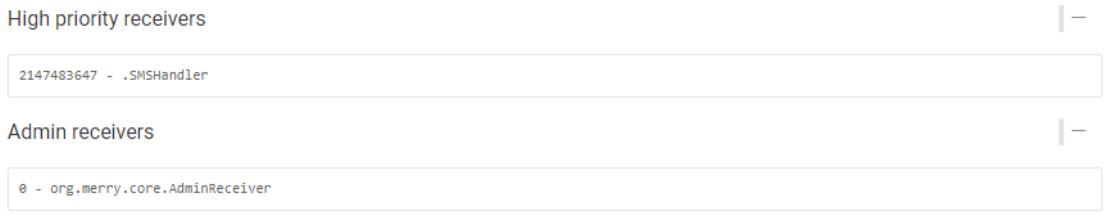



*Figura 61: Reporte de los receptores en apklab.io.*

En la Figura 62 podemos ver los permisos de los que hace uso esta aplicación, que se corresponden con mostrados por MobSF.

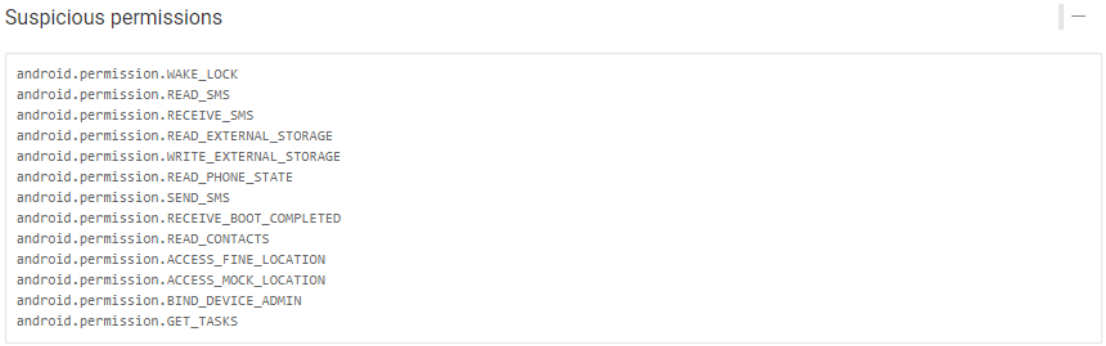



*Figura 62: Reporte de los permisos en apklab.io*

En la Figura 63 podemos ver aquellos fragmentos de código que se sirven de la API de Android para obtener información como podría ser el identificador del dispositivo, el número de teléfono…

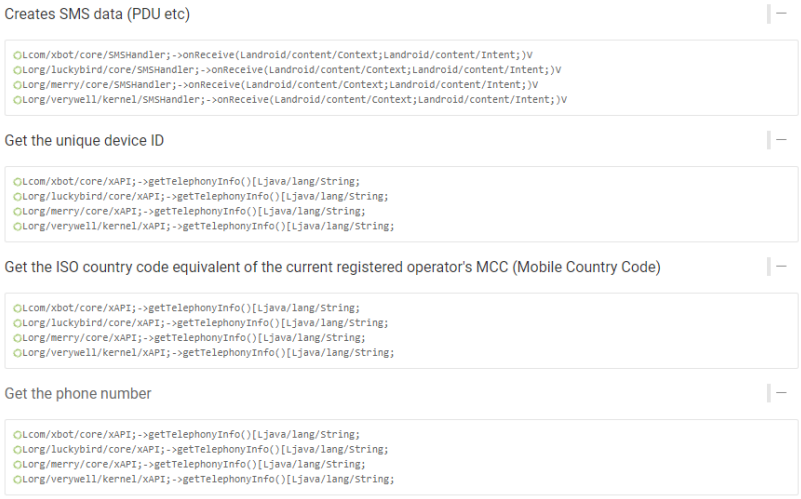



*Figura 63: Reporte de los usos de la API en apklab.io*

En la Figura 64 podemos ver también las direcciones IP y URL a las que apunta. Al igual que hemos mencionado con MobSF, muchas de las direcciones que aquí aparecen aun tratándose de una aplicación maliciosa no tienen por qué ser direcciones sospechosas. Ya que muchas son direcciones contenidas en la API.

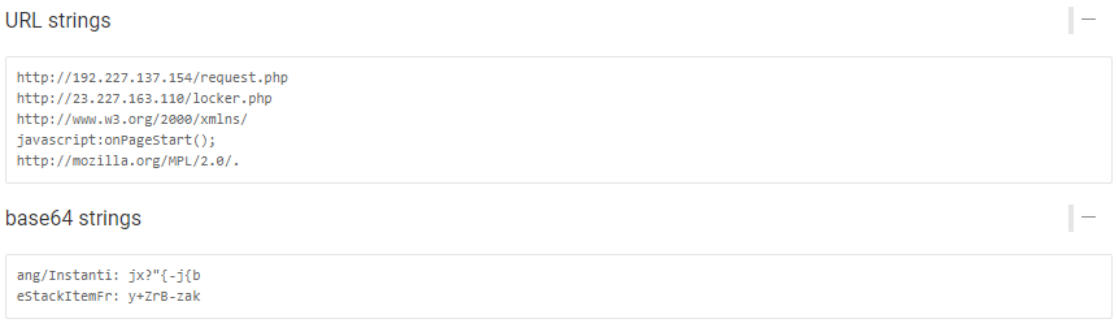



*Figura 64: Reporte de las direcciones IP encontradas en el código de la aplicación en apklab.io*

## 9.2 Análisis dinámico

La herramienta Apklab.io también permite realizar un análisis dinámico de las aplicaciones. Sin embargo, a diferencia del análisis estático los resultados que muestra esta herramienta son distintos a los de MobSF en cuanto al análisis dinámico. Un aspecto fundamental es que no podemos controlar las acciones que tienen lugar sobre la aplicación. En la Figura 65 podemos ver que muestra la visión general del análisis dinámico.

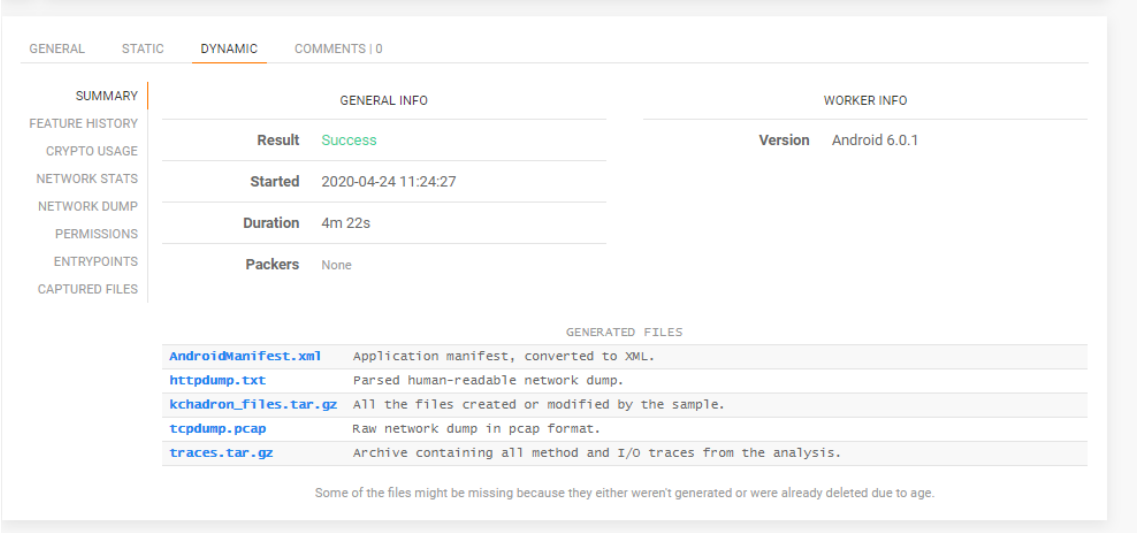



*Figura 65: Resumen del análisis dinámico de la aplicación en apklab.io.*

Como podemos ver en la parte inferior se nos da la posibilidad de obtener una traza del tráfico generado por la aplicación. Dicha traza se encuentra en formato .pcap. Por lo que se pude mostrar haciendo uso de Wireshark.

Podemos ver también un historial de las acciones, similar a lo que podíamos ver en el Logcat de MobSF.

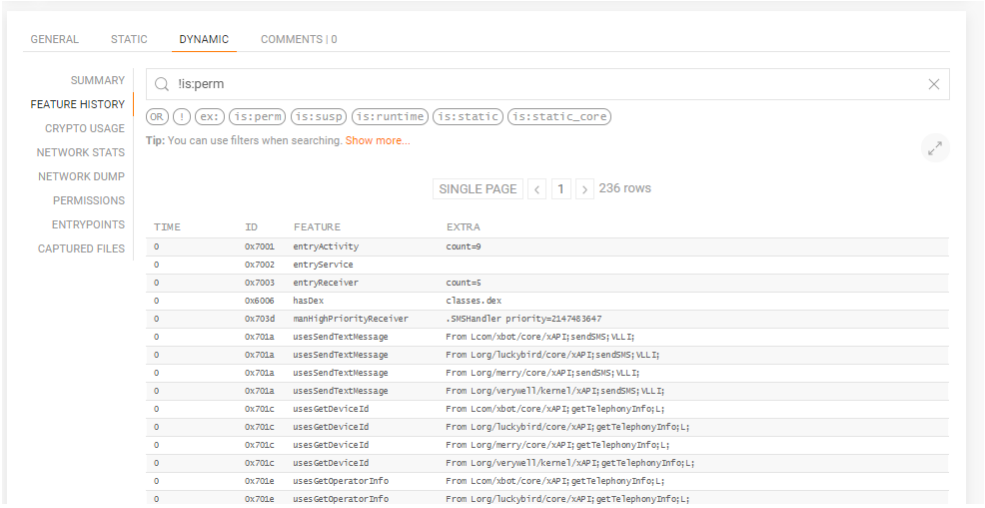



*Figura 66: Registro de las acciones llevadas a cabo por la aplicación.*

Se muestra las direcciones IP a las que la aplicación ha accedido, como podemos ver en la Figura 67. También podemos ver también en que países se encuentra ubicadas esas direcciones.

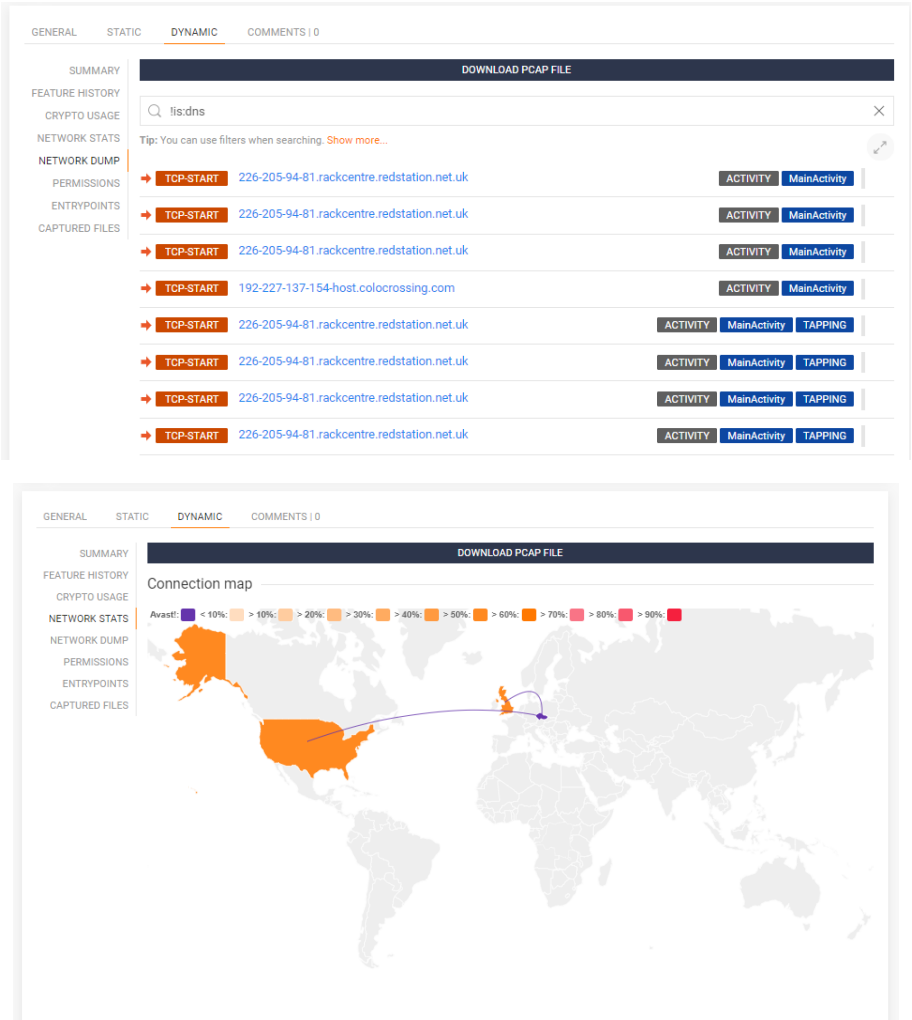



*Figura 67: Direcciones apuntadas por la aplicación y sus países de origen.*

Se muestra también de forma más detallada los distintos permisos, aunque a diferencia de MobSF no se da una descripción de estos, esto lo podemos ver en la Figura 68.

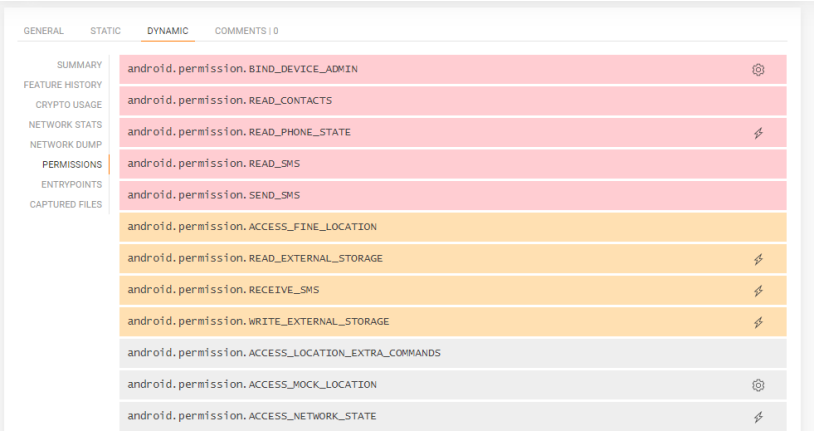



*Figura 68: Reporte de los permisos solicitados por la aplicación durante su análisis en apklab.io.*

Estos reportes los podemos obtener en un fichero en texto plano. Con lo cual se facilita su posible procesamiento en un futuro.

# Capítulo 10: Comparativa de las herramientas de análisis MobSF y apklab.io

Una vez vistos los dos análisis el proporcionado por MobSF y por apklab.io vamos a realizar un comentario sobre los mismos.

En cuanto al análisis estático, podemos ver que ambas herramientas muestran una información muy similar de los datos. Presentan una estructura de presentación muy similar, pero si algo hay que destacar de MobSF incorporan **gran cantidad de datos adicionales que ayudan a comprender el significado de los datos obtenidos en el análisis**.

La gran diferencia surge sin embargo en el análisis dinámico, en el que usando MobSF tenemos la gran ventaja de **poder tener el control de la aplicación mientras se realiza el análisis**, pudiendo controlar el dispositivo virtual de Android. Esto sin lugar a duda aumenta la versatilidad del análisis dinámico realizado en la plataforma desarrollada. En cambio, en apklab.io nos encontramos que el análisis dinámico es bastante opaco, en cuanto a la navegación por la aplicación. Y aunque, como se ha mencionado anteriormente se trató de conversar con los creadores no compartieron información.

También el hecho de poder incorporar rutinas de Frida en el análisis dinámico de MobSF posibilita realizar análisis exhaustivos de mayor calidad, controlando multitud de parámetros.

Como punto a favor, el hecho de que apklab.io sea una **herramienta web** posibilita el hecho de poder usarla en cualquier parte, a diferencia de la nuestra. Si bien es cierto que la decisión de que la máquina sea solo accesible desde una red local conocida ha sido tomada por nosotros. El hecho de que apklab.io pueda ser accedida por usuarios de todo el globo permite que la aplicación contenga millones de muestras de aplicaciones cosa que en el caso de nuestra herramienta no es así.

Otro aspecto diferencial es que apklab.io no tiene ninguna forma de extraer los reportes de una forma automatizada, cosa que sin embargo si permite MobSF haciendo uso del script del Anexo 2.

Por todo esto se considera que MobSF es la herramienta que más se adecua a las necesidades de Orange España y por ello ha sido más desarrollada en esta memoria del proyecto.

En la Tabla 3 podemos ver un resumen de las características que presenta cada una de las herramientas.

| Características\Herramientas | MobSF | Apklab.io |
|---|---|---|
| Accesible desde internet | | |
| Explicación detallada de los reportes | | |
| Obtención del tráfico en un fichero externo | | |
| Visualización de procedencia del tráfico más amigable | | |
| Base de datos de aplicaciones extensa | | |
| Control del dispositivo durante el análisis dinámico | | |
| Incorporación de scripts de Frida | | |

| | | |
|---|---|---|
| Extracción del análsis | | |
| Posibilidad de acceso a otras bases de datos | | |

*Tabla 3: Comparativa entre MobSF y apklab.io*

El verde indica si la herramienta tiene esa característica, mientras que el rojo indica si la herramienta no tiene esa característica.

# Capítulo 11: Análisis dinámico usando un dispositivo real

Como se ha mencionado anteriormente, es muy importante comentar el hecho de que algunos malware detectan si el dispositivo usado para la ejecución es virtual como sería el caso de realizarlo con Genymotion haciendo que algunas de las funcionalidades de la aplicación que podrían ser maliciosas no se ejecutasen. Es por tanto que hemos considerado muy adecuado hacer uso de un terminal real, sobre el que probar la aplicación.

El objetivo es capturar el tráfico generado por la aplicación en un terminal real. Para ello se van a realizar la captura de dos formas distintas:

- **Punto de acceso fraudulento:** Haciendo uso del terminal, y el equipo captando tráfico a través de la interfaz inalámbrica.
- *PCAP Remote:* Haciendo uso de una aplicación móvil en el propio dispositivo

Procedemos a desarrollar los modos de obtención del tráfico:

## 11.1 Punto de Acceso Fraudulento

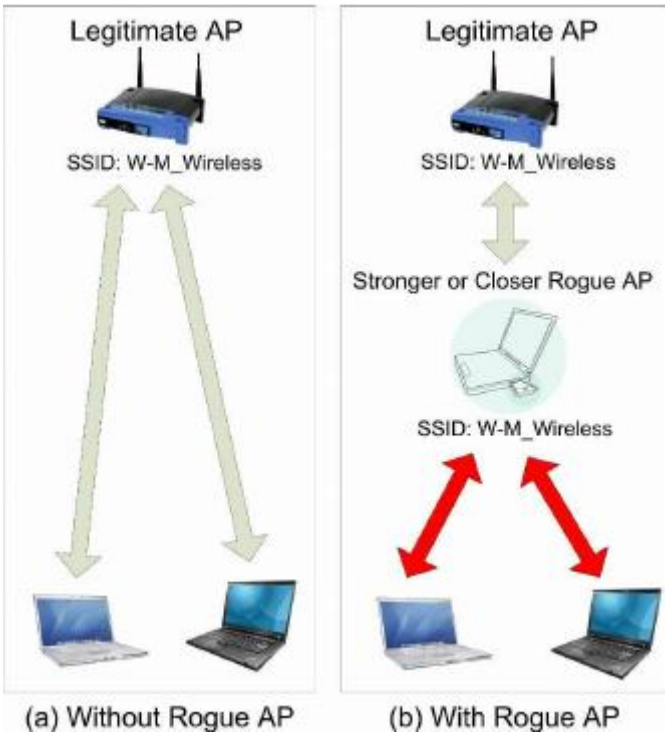



*Figura 69: Esquema de un punto de acceso malicioso.*

Este método consiste en montar un punto de acceso en la interfaz de red inalámbrica. Todo el tráfico que pase por esa interfaz será capturado. El esquema de este montaje lo podemos ver en la Figura 69

Levantaremos un punto de acceso y realizaremos la configuración del DHCP para que se pueda realizar la conexión de forma transparente en el terminal.

Para realizar esto debemos hacer uso de:

**hostapd:** Es un demonio que se encarga de levantar y manejar punto de acceso inalámbricos haciendo uso del driver nl80211 el cual no es soportado por todos los adaptadores.

**dnsmasq**: es un servidor de *Dynamic Host Configuration Protocol (DHCP)* muy ligero con la capacidad de redirección de DNS. Es rápido y fácil de modificar.

Para este método haremos uso de una máquina con la distribución **Linux Mint 19.3**. Necesitaremos una interfaz de red inalámbrica que permita su funcionamiento en el modo **Punto de Acceso (AP)** Esto lo podemos ver tras ejecutar el comando

```
iw list
```

Una vez realizado esto debemos tener instalado el software anteriormente mencionado. Pondremos nuestra interfaz inalámbrica en modo monitor y usaremos los programas con los ficheros de configuración correspondientes.

Debemos crear los dos ficheros de configuración que será usados para crear el punto de acceso.

El fichero *hostapd.conf* se puede ver a continuación y contiene la siguiente información:

```
interface=wlan0
driver=nl80211
ssid=TEST-TFG-APK
hw_mode=g
channel=1
macaddr_acl=0
ignore_broadcast_ssid=0
auth_algs=1
wpa=2
wpa_key_mgmt=WPA-PSK
rsn_pairwise=TKIP
wpa_passphrase=test1234
```

El fichero *dnsmasq.conf* se puede ver a continuación y contiene la siguiente información:

```
interface= wlan0
dhcp-range=192.168.1.26,192.168.1.30,255.255.255.0,12h
dhcp-option=3,192.168.1.1
dhcp-option=6,8.8.8.8
server=8.8.8.8
log-queries
log-dhcp
listen-address=127.0.0.1
```

Una vez creados los ficheros de configuración se debe poner dicha interfaz de red en modo monitor, ejecutando los siguientes comandos.

```
ifconfig wlan0 down

iwconfig wlan0 mode monitor

ifconfig wlan0 up
```

Después ejecutamos el programa **hostapd:**

```
hostapd hostapd.conf
```

Una vez realizado esto debemos cambiar la configuración de las rutas de nuestro dispositivo.

```
ifconfig wlan0 up 192.168.1.1 netmask 255.255.255.0

route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.1.1
```

Y ejecutar el programa **dnsmasq:**

```
dnsmasq -C dnsmasq.conf -d
```

Una vez hecho esto debemos configurar las *IPTABLES* de nuestro equipo, no olvidando que debemos disponer de una interfaz con acceso a internet, en este caso nuestra interfaz cableada `eth0`.

```
iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE

iptables --append FORWARD --in-interface wlan0 -j ACCEPT
```

Y por último debemos hacer que nuestro equipo sea capaz de encaminar por lo que debemos ejecutar.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Una vez tenemos nuestra interfaz inalámbrica actuando como un punto de acceso malicioso, no tendremos más que capturar nuestro tráfico en dicha interfaz, para ellos se ha realizado un script que podemos ver en el Anexo 3. En su ejecución se debe pasar el nombre de la interfaz, así como el tiempo durante el que se realizará la captura en horas. Como resultado obtendremos un fichero con extensión .pcap en cuyo nombre aparece la hora a la que inició la captura.

Hemos visto que el fichero que va a almacenar el tráfico que ha pasado por la interfaz inalámbrica se encuentra en formato PCAP. Vamos a ver que es ese formato de fichero.

PCAP viene de *Packet Capture library* que es una biblioteca que nos permite una interfaz de alto nivel en la que capturar los paquetes de un sistema. Todos los paquetes de una red, aunque estos estén destinados a otros destinatarios son accesibles gracias a este formato. También es capaz de almacenar los paquetes capturados en un fichero para su conservación al igual que podemos leerlos desde ese mismo fichero. Usando programas como Wireshark. Un ejemplo de esto lo podemos ver en la Figura 70.

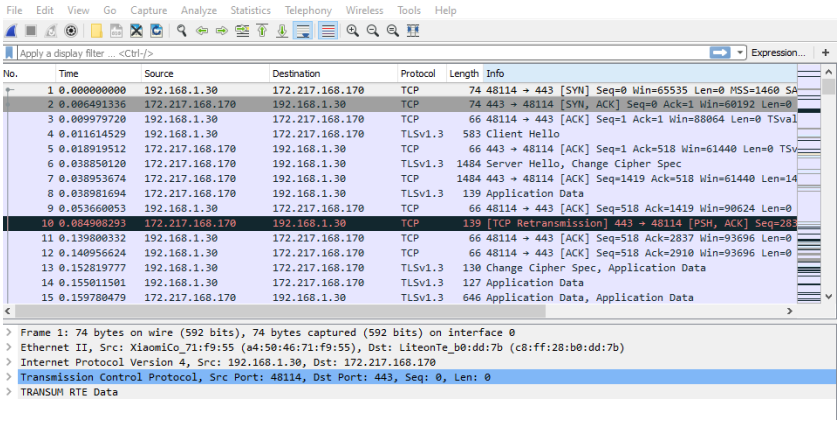



*Figura 70: Vista de un paquete capturado con el punto de acceso malicioso en Wireshark.*

## 11.2 PCAP Remote

Esta aplicación Android la podemos obtener en []. Esta aplicación que se encuentra en Play Store que nos permite obtener una traza del tráfico generado por el terminal, así como capturar el tráfico generado por una única aplicación. Una de las ventajas de esta aplicación es que no precisa tener el dispositivo como usuario privilegiado para realizar la captura.

En la Figura 71 podemos ver cómo es dicha aplicación

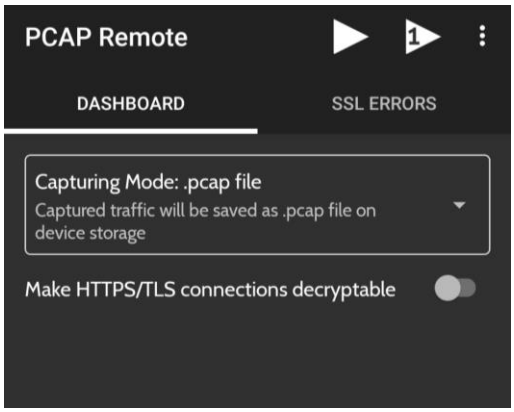



*Figura 71: Menú de la aplicación PCAP Remote*

Podemos elegir entre realizar la captura de todo el tráfico como sería en la Figura 72.

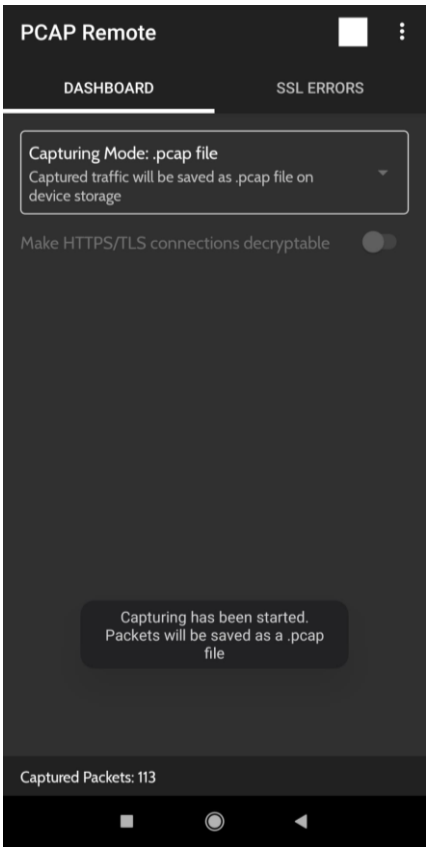



*Figura 72: Captura de todo el tráfico saliente del dispositivo.*

O eligiendo el tráfico generado por una aplicación específica.

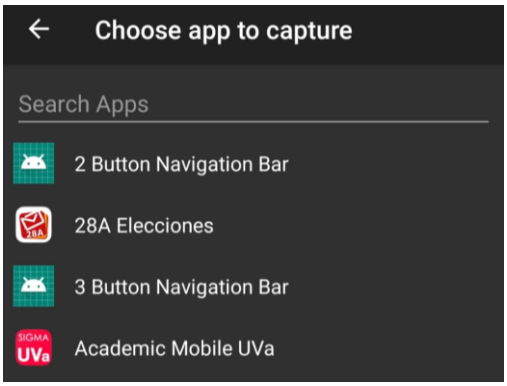



*Figura 73: Selección de la aplicación de la que capturar el tráfico.*

El fichero .pcap una vez generado necesita ser enviado desde el dispositivo móvil hasta el equipo que conforma la plataforma, para su posterior análisis.

# Capítulo 12: Análisis de tráfico

Una parte fundamental del **análisis dinámico que podemos realizar con nuestro dispositivo es el tráfico generado y recibido por el terminal**. Este tráfico es generado por las aplicaciones del dispositivo, así como por su sistema operativo, en este caso el Android.

Con el método del punto de acceso malicioso, podemos obtener en bruto todo el tráfico generado, mientras que haciendo uso de la aplicación PCAP Remote tenemos la posibilidad de obtener el tráfico en bruto y el tráfico únicamente de la aplicación de la que deseemos realizar el análisis.

El tráfico obtenido está en un fichero de formato .pcap que se podrá analizar el tráfico con herramientas como Wireshark.

En este caso, una vez obtenido el fichero.pcap se va a cargar en la **plataforma basada en Elastichsearch, Loghstash y Kibana (ELK)** para su posterior análisis.

Hemos elegido la herramienta conocida como ELK (Elastichsearch, Loghstash, Kibana) [19] para manejar estos datos filtrados y poder crear una base de datos y llegar al resultado final, que es mostrar los datos y estadísticas de estas capturas Estos tres programas forman una poderosa herramienta de procesado y tratamiento de datos, de forma que la función de cada una de las herramientas es la siguiente:

- **Logstash** se ocupa del procesado de datos del lado del servidor que ingiere datos de múltiples fuentes simultáneamente, los transforma y luego los envía a un "almacén" como Elasticsearch.
- **Elasticsearch** es un motor de búsqueda y análisis. Es capaz de operar con grandes cantidades de datos, los cuales te permite almacenar y realizar búsquedas sobre ellos. Realiza un indexado incremental permitiendo un mejor rendimiento ante la gran cantidad de datos presentes.
- **Kibana** permite visualizar datos con tablas y gráficos en Elasticsearch.

En la Figura 74 podemos ver la arquitectura de la pila de de Elastichsearch, Loghstash y Kibana.

Aunque la pila de Elastichsearch, Loghstash y Kibana está pensada para trabajar fundamentalmente con el formato JSON, debido a problemas con los valores obtenidos de las capturas se ha optado por usar el formato .csv.

Para realizar la conversión entre los ficheros con formato .pcap a .csv se ha usado el script del Anexo 4.

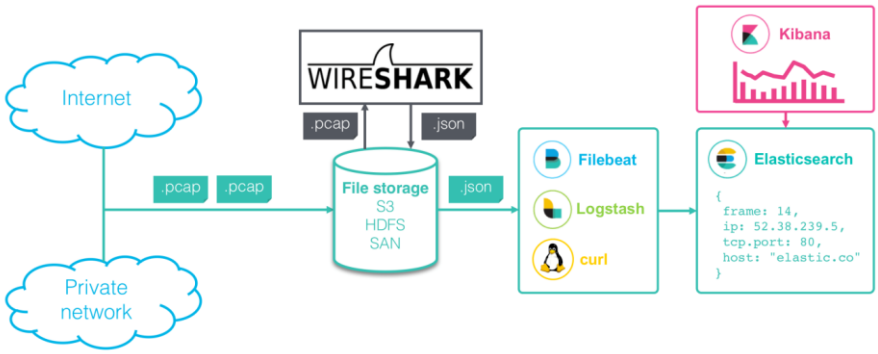



*Figura 74: Arquitectura de la pila ELK.*

## 12.1 Logstash

Logstash se encarga del procesado de datos. Para ello, hay que realizar un archivo de configuración .conf de Logstash para que el programa pueda indexar todos los campos que queramos y subirlos correctamente a Elasticsearch.[19]

Este archivo de configuración tiene 3 partes:

- INPUT: Se refiere a los archivos que vamos a utilizar como entrada, es decir, los archivos de datos que queramos procesar
- FILTER: Este campo se utiliza como filtro. Aquí se le indicara al programa si queremos borrar algún campo, si queremos añadir más campos, si queremos que un campo sea tipo Date o si queremos que se un número. Por defecto, todos los campos son cadenas "*string*".
- OUTPUT: Se refiere a donde vamos a enviar los datos. En nuestro caso, será Elasticsearch. Además, en este campo se le indicara un nombre de índice para poder diferenciar los datos una vez subidos. IMPORTANTE: El nombre de estos índices, no puede estar en mayúsculas, porque Kibana no lo reconocerá y dará error en la subida de datos.

Este fichero puede verse en el Anexo 5.

Una vez realizado el archivo de configuración se procederá a ejecutar el programa Logstash y ejecutar el archivo de configuración que hemos creado. Para ello, se ejecutará en modo superusuario con la siguiente línea de comando:

**sudo /usr/share/logstash/bin/logstash –f "ruta del archivo de configuración"**

## 12.2 Kibana

Kibana es una interfaz web de código abierto para Elasticsearch. Puedes acceder a ella desde tu navegador. Es una herramienta excelente para visualizar datos en gráficas, mapas y tablas.

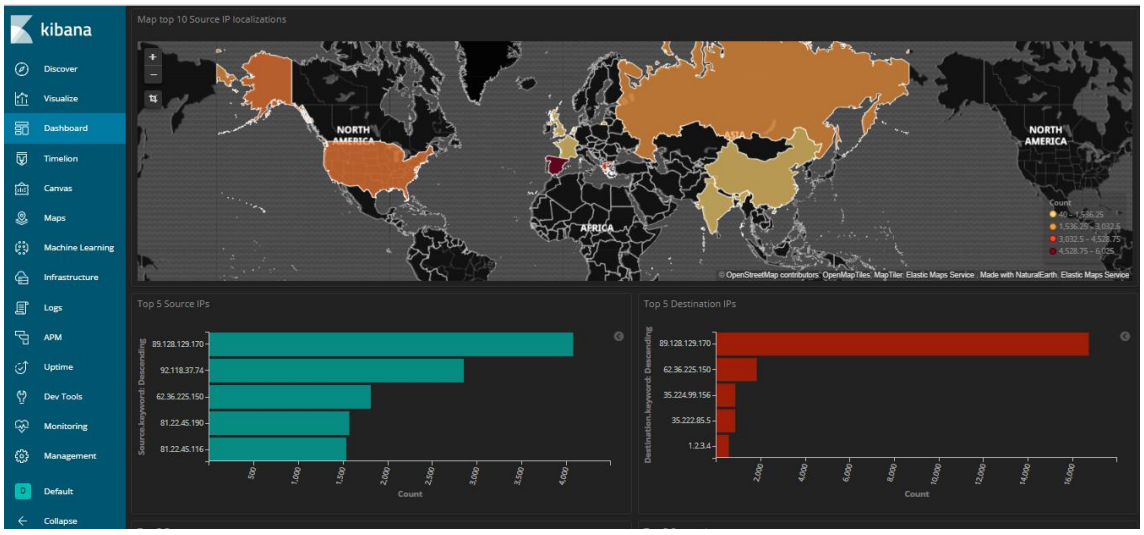



*Figura 75: Visualización de los datos de tráfico obtenidos en la captura de una aplicación.*

Como podemos ver en la Figura 75 podemos ver los datos en un formato mucho más agradable, se pueden ver los países de origen de las diez direcciones IP más accedidas por el dispositivo. Así como cuales son las 5 direcciones IP de las que proviene la mayor cantidad de tráfico.

# Capítulo 13: Casos de uso

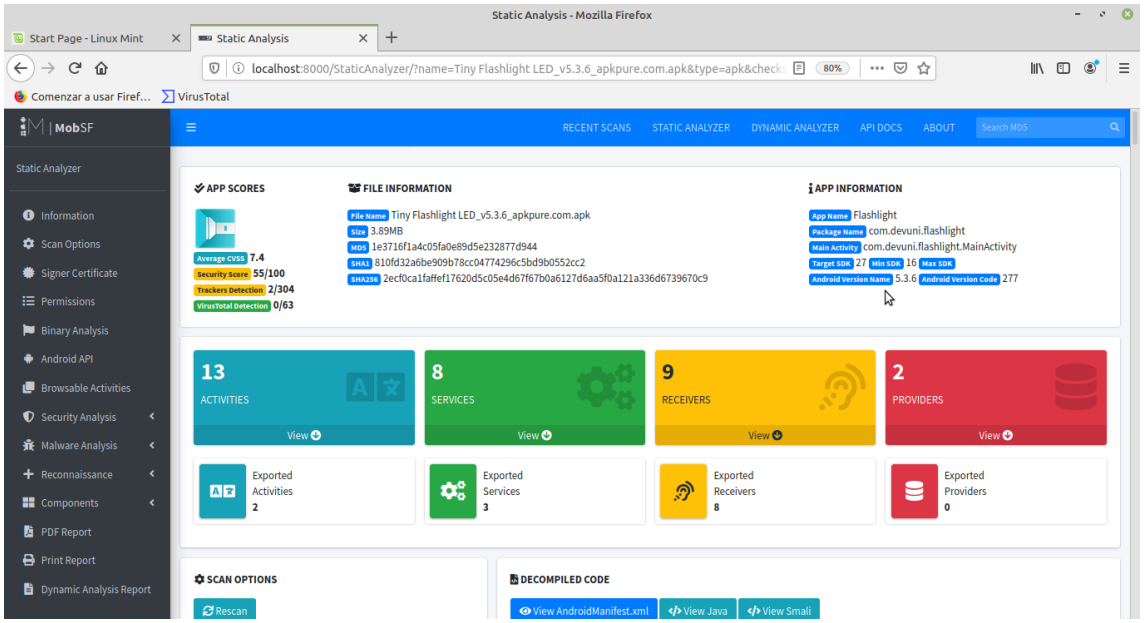
En este apartado vamos a ver el comportamiento de la plataforma ante varias aplicaciones y analizar sus resultados.

## 13.1 Caso de Uso 1: Aplicación con permisos Sospechosos

En este caso vamos a ver el análisis de las dos aplicaciones de linternas Flashlight y Tiny Flashlight que podemos encontrar en la aplicación de distribución de Play Store. El objetivo de esta aplicación es muy simple. Se nos mostrará un botón en la pantalla del dispositivo que tras pulsarlo activará de forma permanente el flash de nuestro dispositivo hasta volvamos a actuar sobre el botón.

Es por tanto que lo lógico para esta aplicación sería limitarse a actuar sobre la cámara y en el caso de que contenga anuncios, cosa muy común en este tipo de aplicaciones necesite tener acceso a internet.

En la Figura 76 se ve el vistazo general de Tiny Flashlight junto a sus permisos.



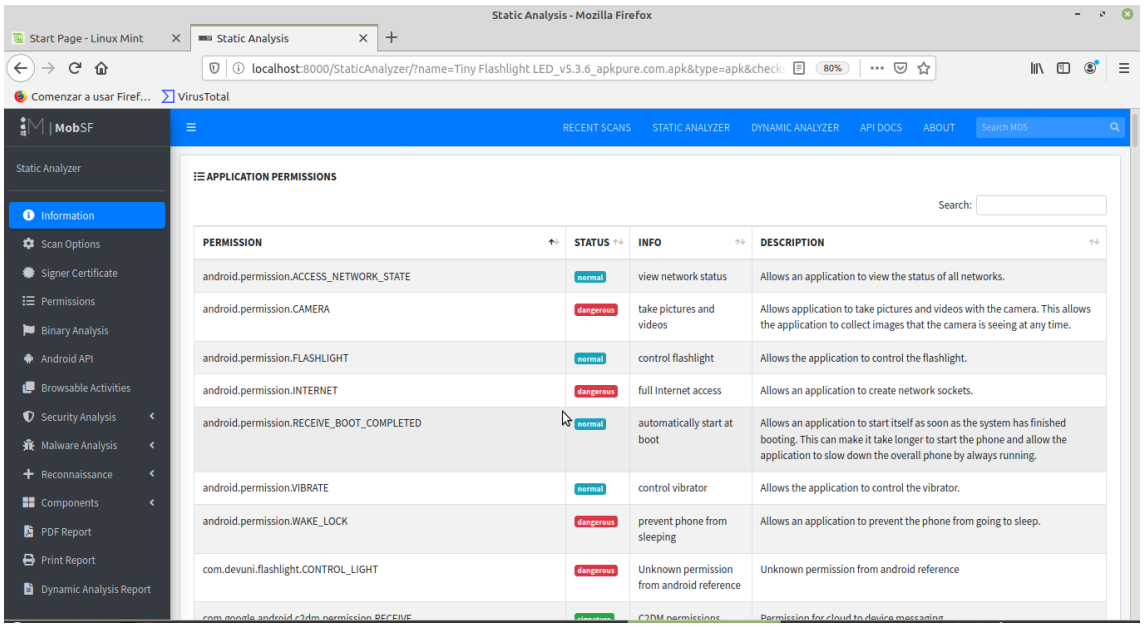
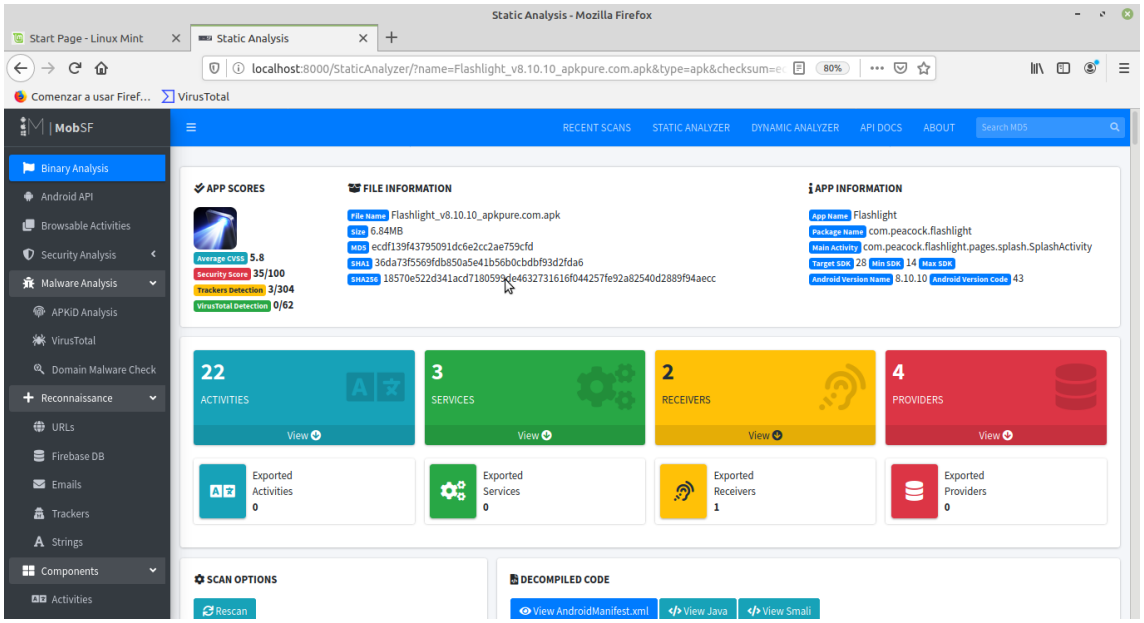



*Figura 76: Información general y permisos de la aplicación Tiny Flashlight.*

Podemos ver que efectivamente esta aplicación tiene los permisos esperados para una aplicación de su tipo. Aunque sea molesto e inadecuado el permiso referido al acceso a Internet.

Y en la Figura 77 se ve el vistazo general Flashlight junto a sus permisos.



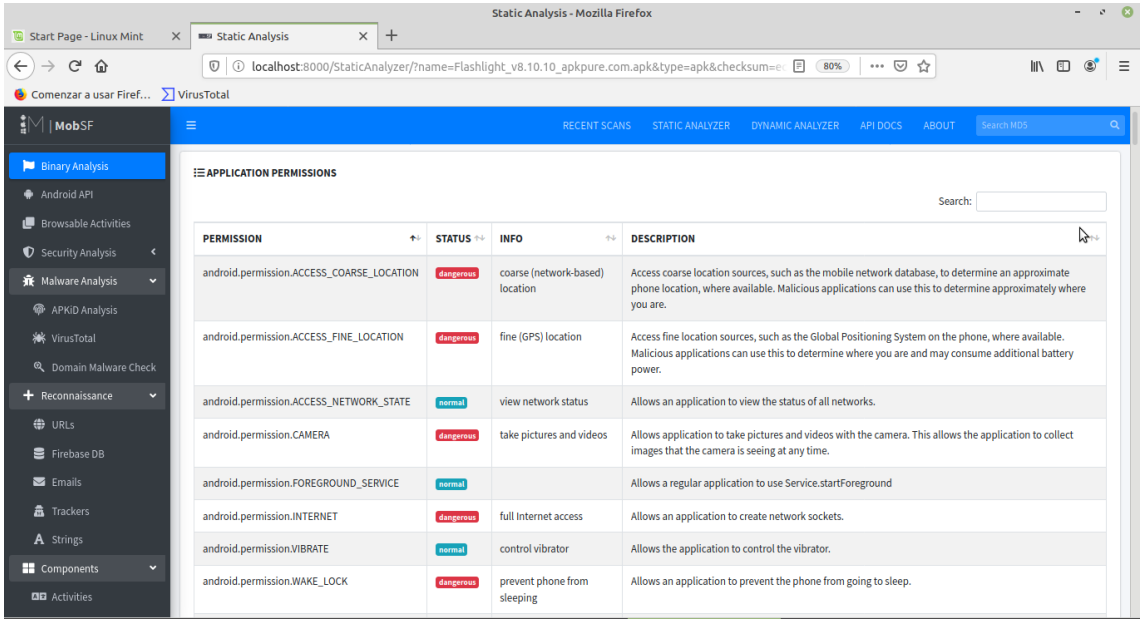



*Figura 77: información general y permisos de la aplicación Flashlight.*

En este otro caso sin embargo la situación es muy distinta. Podemos apreciar que en esta aplicación el tipo de permisos no es como el de la anterior. Tiene los permisos esperados comentados anteriormente, cámara e internet, pero añade dos permisos nuevos. Los dos nuevos permisos añadidos están relacionados con la localización. El primero de ellos es un permiso que trata de estimar tu localización haciendo uso de la red móvil. El segundo se trata de un permiso que hace uso del sistema de posicionamiento global (GPS) para detectar tu ubicación.

Es por tanto que nos encontramos con una aplicación que sin ninguna necesidad trata de estimar tu localización, lo cual vulnera tu privacidad y además ocasiona un aumento en el gasto de la batería del dispositivo móvil al tratar de establecer contacto el sistema de posicionamiento global.

Con este ejemplo se ha tratado de ilustrar que dos aplicaciones aparentemente inofensivas y simples, de muy fácil acceso pues se encuentran en la aplicación de distribución Play Store, presentan permisos innecesarios. En el caso de Flashlight va más allá vulnerando tu privacidad y alterando las posibilidades de uso de tu terminal.

### 13.2 Caso de Uso 2: Aplicación maliciosa con direcciones IP sospechosas

En este otro caso, vamos a analizar una aplicación que sabemos que es una aplicación maliciosa, en busca de obtener a partir del análisis en la plataforma datos que demuestren que efectivamente nos encontramos ante una aplicación maliciosa.

Al realizar el análisis estático de la misma, podemos ir al apartado en el que se muestran las direcciones IP encontradas lo cual podemos ver en la Figura 78.

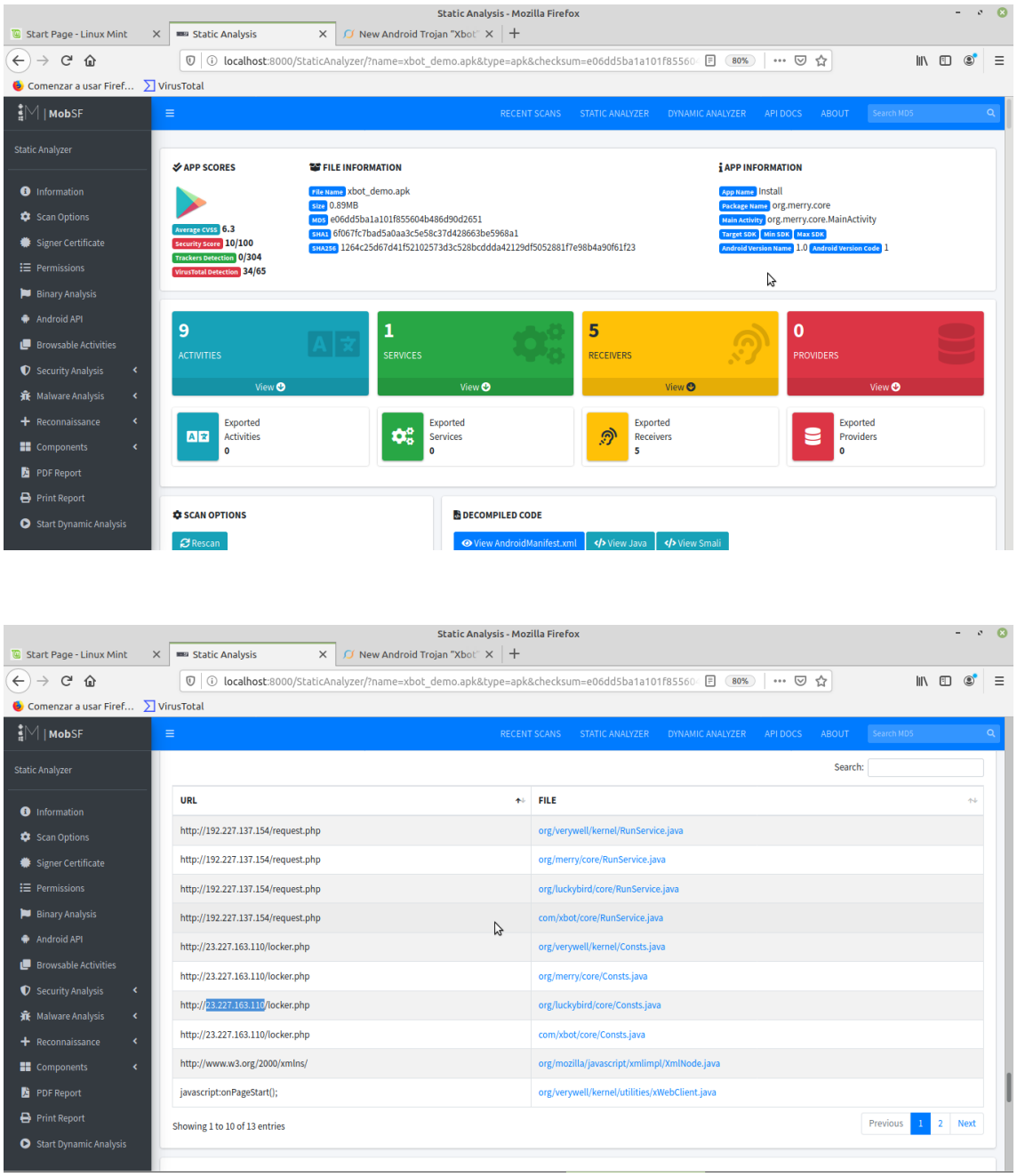






*Figura 78: Aplicación maliciosa las direcciones IP a las que apunta.*

Realizando una búsqueda de esas direcciones encontramos que una de ellas la correspondiente a 23.227.163.110 al realizar una búsqueda la encontramos en una noticia de malware. [20]

Con este ejemplo hemos demostrado que se puede realizar la detección de malware usando esta técnica.

# Capítulo 14: Conclusiones

La realización de este proyecto ha supuesto un auténtico reto, más aún debido a las excepcionales circunstancias acontecidas este año 2020, poniendo a prueba las capacidades adquiridas durante los últimos cuatro años. La organización ha sido un aspecto fundamental, y gradezco a mis tutores el haber insistido en ello.

Para el desarrollo del proyecto ha sido necesario un gran esfuerzo de aprendizaje respecto al tema de las aplicaciones móviles que, aunque están a la orden del día no contaba más que con conocimientos generales sobre ellas.

Encontrar una necesidad y tratar de solucionarla haciendo uso de las herramientas disponibles ha sido muy enriquecedor. Probar múltiples herramientas alternativas con la intención de ver si estas cumplían las necesidades del proyecto o sin embargo eran más adecuadas para otro tipo de escenario, me ha permitido explorar una gran cantidad de soluciones.

Una vez realizada la selección de las herramientas sobre las que se enfocaría nuestro proyecto ha sido muy interesante realizar su despliegue e integración en el equipo destinado a ello. Así como, ver los resultados que estas arrojaban al realizar los análisis.

De todas las herramientas probadas las que más destaca sin ninguna duda es MobSF, la cual ha sido elegida para formar parte de la base de nuestra herramienta. Su versatilidad, el hecho de permitir su interconexión con otras herramientas y la posibilidad de extraer los reportes de la aplicación móvil analizada son características ideales para nuestro proyecto. Es por ello por lo que se ha comentado de forma extensa lo que dicha herramienta es capaz de obtener en sus análisis.

El uso de Genymotion para realizar la emulación de dispositivos virtuales Android permite la creación de múltiples dispositivos variando sus características, lo cual es extremadamente útil a la hora de realizar los análisis.

Gracias a la corta curva de aprendizaje del lenguaje de Programación Python y sus infinitas posibilidades ha sido posible la creación de varios scripts para automatizar la gestión de la plataforma.

Se ha comentado también la herramienta apklab.io ya que es una herramienta muy similar a MobSF ya durante la realización del proyecto ha servido como comparativa con los resultados que se iban obteniendo en la plataforma.

Ha sido una suerte disponer de un dispositivo móvil real con el que poder realizar todo tipo de pruebas, durante la ejecución del proyecto. La creación de un punto de acceso malicioso ha supuesto un gran aprendizaje y me ha recordado la necesidad de intentar cumplir lo máximo posible con todas las pautas de ciberseguridad, ya que es relativamente muy sencillo no cumplirlas y cometer un error que exponga nuestros datos personales.

También ha sido muy interesante trabajar con una tecnología tan actual como es una pila de Elasticsearch, Logstash y Kibana del que he descubierto sus grandes virtudes, como su gran flexibilidad.

Por último, se ha incorporado a la memoria del proyecto varios análisis de aplicaciones disponibles en Play Store para ver como la plataforma es capaz de detectar sus características,

realizando un análisis de estas. Además, se ha probado una muestra de malware obtenido de nuestra base de datos con el fin de ejemplificar su detección.

Como conclusión, las aplicaciones maliciosas han incrementado su número de forma notable en los últimos años tal y como se ha visto en el proyecto. La seguridad en el sistema operativo Android permite tal y como se ve en el capítulo sexto no ser tan vulnerables a ellas, pero las vulnerabilidades siguen existiendo. Es por eso que se plantea necesario disponer de una plataforma como la que ha sido desarrollada en este proyecto con la que analizar y probar las aplicaciones móviles, antes la instalación en nuestros dispositivos evitando que una posible aplicación maliciosa nos pueda ocasionar problemas.

## 14.1 Objetivos conseguidos

En la siguiente lista se van a detallar los objetivos conseguidos durante la realización del proyecto:

- Planificación con hitos de un proyecto y presentación periódica de avances.
- Instalación y despliegue de la herramienta MobSF en el equipo, para la realización de análisis dinámico y estático.
- Despliegue de un emulador de Android funcional en el equipo usando Genymotion.
- Interconexión de la herramienta MobSF con Virustotal.
- Creación de scripts para el manejo de la aplicación y su automatización.
- Desarrollo de un punto de acceso malicioso usando una interfaz inalámbrica de red.
- Interconexión con una pila de Elasticsearch, Logstash y Kibana para ver el tráfico capturado.

# Capítulo 15: Futuras líneas de trabajo

En este capítulo, vamos a explorar futuras líneas de trabajo para el proyecto desarrollado.

Debiendo partir de la base del proyecto que tenemos, es decir nuestro equipo con la herramienta MobSF instalada, los emuladores de dispositivos Android, la tarjeta de red inalámbrica actuando como punto de acceso malicioso y la pila de Elasticsearch, Logstash y Kibana se plantean las siguientes opciones.

## 15.1 Análisis automático de los reportes generados en MobSF

Partiendo de los reportes generados en .json estos podrían ser cargados en una pila de Elasticsearch, Logstash y Kibana para realizar un análisis automático de ellos. Al estar la pila enfocada a trabajar con el formato JSON por defecto se podría integrar de forma óptima.

Se realizaría declarando perfiles de aplicación, es decir, si un tipo de aplicación móvil debe contener ciertos permisos, actividades, receptores…. Si al realizar el análisis, el reporte de la aplicación no se corresponde con lo esperado o aceptable la aplicación sería sospechosa de ser una aplicación maliciosa.

Lo mismo ocurriría con las direcciones IP presentes en el reporte, que serían comparadas con una base de datos como la de [20]. Si alguna de las direcciones IP a las que apunta la aplicación coincidiera con las almacenadas la aplicación sería sospechosa de ser una aplicación maliciosa.

## 15.2 Creación de una celda de telefonía falsa

Para continuar haciendo uso del dispositivo físico con el que contamos, una opción que se plantea es levantar una estación base de telefonía falsa. Es decir que no pertenezca a la red de los operadores y de la que se pueda capturar todo el tráfico que pase por ella. Esto nos permitiría explorar el comportamiento de las aplicaciones en un entorno más real, ya que los dispositivos móviles no se encuentran siempre en una red Wi-Fi.

Al disponer del control de esa estación base se podría capturar todo el tráfico que pase por ella, incluyendo llamadas, SMS…

Esto por supuesto sería invisible tanto para el usuario como para las aplicaciones móviles, por lo que no cabría la posibilidad de que las aplicaciones varían el comportamiento real. De esta forma el análisis sería más exhaustivo.

## 15.3 Clarificar el tráfico capturado

Esto es uno de los retos más significativos, ya que, aunque el tráfico de las aplicaciones ha sido capturado, este tráfico se encuentra cifrado. Una de las diez vulnerabilidades más peligrosas para las aplicaciones móviles explicadas anteriormente estaba referida a las comunicaciones inseguras. Es por tanto que toda aplicación debería cifrar su tráfico haciendo que si este tráfico es obtenido de alguna manera los datos no puedan ser desencriptados.

Esto haciendo uso de herramientas como Burp [21] podría ser solventado.

### 15.4 Desarrollo de scripts de Frida

Otro aspecto que se podría realizar es la creación de scripts genéricos de pruebas de penetración haciendo uso de Frida con el fin de evaluar las aplicaciones. Aprovechando que el análisis dinámico permite la incorporación de esos scripts, se podrían desarrollar algunos de ellos con el fin de encontrar vulnerabilidades. Ya que además se permite la prueba simultánea de varios scripts de ese tipo.

### 15.5 Mejora de la transferencia de ficheros del dispositivo a la plataforma

Aunque a priori parece no ser un gran problema tener que usar el dispositivo conectado a la plataforma por el cable USB, realizar esto de forma inalámbrica podría ser más ventajoso.

# Anexo 1. Carga y escaneo de aplicaciones

```
#!/usr/bin/env python
# -- coding: utf-8 --
# -------------------------------------------------------------------------
-----
# Name:     json_report
# Purpose:  Does an analysis of an appplication to the platform
# Author:   Adrian
#
# Created:     27/07/2020
# Licence:     <your licence>
# -------------------------------------------------------------------------
-----

'''
Help Script to analyze an application.

'''
__author__ = "Adrian Rojo"

import argparse
import logging
import os
import urllib.error
import urllib.parse
import urllib.request

import requests

logger = logging.getLogger(__name__)


def is_server_up(url):
    try:
        urllib.request.urlopen(url, timeout=5)
        return True
    except urllib.error.URLError:
        pass
    return False


def start_scan(directory, server_url, apikey, rescan='0'):
    print('\nLooking for Android/iOS/'
          'Windows binaries or source code in : ' + directory)
    logger.info('Uploading to MobSF Server')
    uploaded = []
    mimes = {
        '.apk': 'application/octet-stream',
```

```
        '.ipa': 'application/octet-stream',
        '.appx': 'application/octet-stream',
        '.zip': 'application/zip',
    }
    for filename in os.listdir(directory):
        fpath = os.path.join(directory, filename)
        _, ext = os.path.splitext(fpath)
        if ext in mimes:
            files = {'file': (filename, open(fpath, 'rb'),
                              mimes[ext], {'Expires': '0'})}
            response = requests.post(
                server_url + '/api/v1/upload',
                files=files,
                headers={'AUTHORIZATION': apikey})
            if response.status_code == 200 and 'hash' in response.json():
                logger.info('[OK] Upload OK: %s', filename)
                uploaded.append(response.json())
            else:
                logger.error('Performing Upload: %s', filename)

    logger.info('Running Static Analysis')
    for upl in uploaded:
        logger.info('Started Static Analysis on: %s', upl['file_name'])
        if rescan == '1':
            upl['re_scan'] = 1
        response = requests.post(
            server_url + '/api/v1/scan',
            data=upl,
            headers={'AUTHORIZATION': apikey})
        if response.status_code == 200:
            logger.info('[OK] Static Analysis Complete: %s', upl['file_na
me'])
        else:
            logger.error('Performing Static Analysis: %s', upl['file_name
'])


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-d', '--directory',
                        help='Path to the directory that contains '
                             'mobile app binary/zipped source code')
    parser.add_argument(
        '-s', '--ipport', help='IP address and Port number '
                               'of a running MobSF Server. '
                               '(ex: 127.0.0.1:8000)')
    parser.add_argument(
        '-k', '--apikey', help='MobSF REST API Key')
    parser.add_argument(
```

```
        '-r', '--rescan', help='Run a fresh scan. '
                                'Value can be 1 or 0 (Default: 0)')
    args = parser.parse_args()

    if args.directory and args.ipport and args.apikey:
        server = args.ipport
        directory = args.directory
        server_url = 'http://' + server
        apikey = args.apikey
        rescan = args.rescan
        if not is_server_up(server_url):
            print('MobSF REST API Server is not running at ' + server_url
)
            print('Exiting!')
            exit(0)
        # MobSF is running, start scan
        start_scan(directory, server_url, apikey, rescan)
    else:
        parser.print_help()
```

# Anexo 2: Generación del reporte JSON

```
#!/usr/bin/env python3
# -- coding: utf-8 --

# -------------------------------------------------------------------------
-----
# Name:     json_report
# Purpose:  Gets a JSON report of an appplication form the platform
# Author:   Adrian
#
# Created:     7/07/2020
# Licence:     <your licence>
# -------------------------------------------------------------------------
-----

'''
Help Script to get the JSON report of an application.
Requires the autorization code that is defined when the system is up.
And the hash of the application
'''
__author__ = "Adrian Rojo"

import subprocess
import os
import sys
import requests

def usage():
    print(" %s must use 2 argument the key from MobSF and the hash of the
 app" % (os.path.basename(sys.argv[0])))

try:
    key=sys.argv[1]             # Take argument key, can be passed from s
cripts
    hash_apps=sys.argv[2]       # Take argument hash of the application c
an be passed from scripts

except:
    usage()
    sys.exit(1)

headers = {
    'Authorization': key,
}

data = {
  'hash': hash_apps
}
```

```
response = requests.post('http://localhost:8000/api/v1/report_json', head
ers=headers, data=data)
```

# Anexo 3: Código de captura

```
#!/usr/bin/env python3
# -- coding: utf-8 --

# -------------------------------------------------------------------------
-----
# Name:     01esniferPC
# Purpose:  Start the capture .pcap of trafic on an interface.
# Author:   Adrian
#
# Created:    6/05/2020
# Licence:     <your licence>
# -------------------------------------------------------------------------
-----

'''
Help Script to start the capture if .pcap file daily. You could introduce
 the time in hours to capture.
'''
__author__ = "Adrian Rojo"

#Añado librerias
import subprocess
import shlex
import os
import sys
#Añado el modulo necesario
from datetime import datetime
def usage():
    #Need  special software loaded
    print(" %s must use four arguments \n type_input Device \n number_lin
e 01 \n interface enx086d41e63a86 \
    \n time file (hours) 24 " % (os.path.basename(sys.argv[0])))

try:
    type_input=sys.argv[1]       # Take argument type input, can be passe
d from scripts
    number_line=sys.argv[2]     # Take argument number of line  can be pa
ssed from scripts
    interface=sys.argv[3]        # Take argument of interface
    time_capture=sys.argv[4]        # Take argument of path of the file /
home/iotuser/CPE-SNIFFER/CPE-
except:
    usage()
    sys.exit(1)

#Genero la fecha en el formato deseado
HOY = datetime.now()
```

```
FORMATO = "%Y-%m-%d_%Hh"
FECHA = HOY.strftime(FORMATO)

#Genero el archivo .cap con el nombre deseado
FILE = '/SNIFFER/'+type_input+number_line+'/'+type_input+number_line+'_'+
FECHA+'.cap'
time_tshark = (time_capture*3600)-40
#Empieza la captura mediante el comando subprocess que permite
#ejecutar comandos en la consola
COMANDOS1 = "tshark -a duration:"+time_tshark + "-i "+interface+ " -
w "+FILE+" -F pcap"
#COMANDOS2 = shlex.split(COMANDOS1)
CAPTURA = subprocess.call(shlex.split(COMANDOS1))
```

# Anexo 4: Conversión de pcap a csv

```
#!/bin/bash

set -x

####!/usr/bin/env bash
#PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames:/usr/local/games


#this script converts all the .pcap file in the folder just above the "sc
ripts" folder to csv. It filters all the arp fields before conversion


DEST=/SNIFFER/csv
for file in `ls /SNIFFER/*.pcap`; do
    filename=${file##*/}
    base=${filename%.cap}
    extension=${filename##*.}
#    echo "filename: $filename"
 #   echo "file: $file"
  #  echo "extension: $extension"
   # echo "base: $base"
    if [ ! -f "$DEST/$base.csv" ]; then
    #echo "$DEST/$base.csv"
        tshark -Y 'not arp' -r $file -T fields -e frame.number -
e frame.time -e ip.src -e ip.dst -e _ws.col.Protocol -e tcp.srcport -
e _ws.col.Length -e tcp.dstport -e _ws.col.Info -E header=y -
E separator=";" -E quote=d -E occurrence=f > $DEST/$base.csv
    fi
done

echo "Conversions completed"
```

# Anexo 5: Fichero configuración Logstash

```
input {
  file {
    path => "/SNIFFER/*.csv"
    start_position => "beginning"
    sincedb_path => "/dev/null"
  }
}
filter {
  csv {
      separator => ";"
      columns => ["No.", "Date", "Source", "Destination", "Protocol", "SR
C Port", "Length", "Dest Port", "Info"]
  }

  mutate {convert => {"Length" => "integer"}}

  mutate {
    add_field => {
      "timestamp" => "%{Date}"
    }
  }

  date {
    match => ["timestamp", "MMM dd, yyyy HH:mm:ss.SSSSSSSSS 'CET'", "MMM
dd, yyyy HH:mm:ss.SSSSSSSSS 'CEST'", "MMM  dd, yyyy HH:mm:ss.SSSSSSSSS 'C
ET'", "MMM  dd, yyyy HH:mm:ss.SSSSSSSSS 'CEST'"]
    target => "Date"
  }

  mutate {convert => ["frame.len", "integer"]}

   mutate {
     remove_field => [ "message" ]
     remove_field => [ "@version" ]
     remove_field => [ "timestamp" ]
     remove_field => [ "@timestamp" ]
    }

  geoip {
      source => "Source"
  }

}

output {
   elasticsearch {
     action => "index"
```

```
    hosts => ["http://127.0.0.1:9200"]
    index => "apps"
    workers => 1
    user => "elastic"
    password => "changeme"
  }
  stdout {codec => rubydebug}
}
```

# Referencias


[1] D. Chell, S. Colley, T. Erasmus, O. Whitehouse, "Mobile App Hackers Handbook", WILEY, USA, 2017

[2] https://research.checkpoint.com/2020/cyber-attack-trends-2020-mid-year-report/, última visita: agosto 2020

[3] https://yiminshum.com/mobile-movil-app-2020/, última visita: agosto 2020

[4] K. A. Monnappa, "Learning Malware Analysis", PACKTUB, USA, 2018

[5] https://www.incibe-cert.es/blog/utilidades-para-analizar-apks, última visita: agosto 2020

[6] K. Chinetha, J. Daphney Joann, A. Shalini, "An Evolution of Android Operating System and Its Version", International Journal of Engineering and Applied Sciences (IJEAS) ISSN: 2394-3661, Volume-2, Issue-2, February 2015

[7] A. M. Retenaga, "Situación del Malware para Android", INCIBE, ESPAÑA, 2015

[8] https://github.com/sk3ptre/AndroidMalware_2020, última visita: agosto 2020

[9] https://github.com/MobSF/Mobile-Security-Framework-MobSF, última visita: agosto 2020

[10] https://developer.android.com/guide/components/activities/intro-activities, última visita: agosto 2020

[11] https://developer.android.com/guide/components/services, última visita: agosto 2020

[12] https://developer.android.com/guide/components/broadcasts, última visita: agosto 2020

[13] https://developer.android.com/guide/topics/providers/content-providers, última visita: agosto 2020

[14] https://developer.android.com/guide/topics/permissions/overview, última visita: agosto 2020

[15] https://cwe.mitre.org/data/index.html, última visita: agosto 2020

[16] https://frida.re/docs/home/, última visita: agosto 2020

[17] https://www.apklab.io/, última visita: agosto 2020

[18] https://play.google.com/store/apps/details?id=com.egorovandreyrm.pcapremote&hl=es , última visita: agosto 2020

[19] V. Sharma, "Beginning Elastic Stack", APRESS, USA, 2016

[20] https://unit42.paloaltonetworks.com/new-android-trojan-xbot-phishes-credit-cards-

and-bank-accounts-encrypts-devices-for-ransom/, última visita: agosto 2020

[21] https://www.dshield.org/ipsascii.html, última visita: agosto 2020

[22] https://portswigger.net/burp, última visita: agosto 2020