Trabajo Fin de Máster:
Auditoría de seguridad de aplicaciones móviles

Alumno: Eva Rodriguez Sanchez
Máster en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Seguridad en la Internet de las Cosas

Consultor: Pau del Canto Rodrigo
Fecha: 04/06/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	Auditoría de seguridad de aplicaciones móviles
Nombre del autor:	*Eva Rodriguez Sanchez*
Nombre del consultor/a:	*Pau del Canto Rodrigo*
Nombre del PRA:	*Victor Garcia Font*
Fecha de entrega (mm/aaaa):	06/2019
Titulación:	Máster en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Área del Trabajo Final:	M1.848 - TFM-Seguridad en la Internet de las Cosas
Idioma del trabajo:	*Español*
Palabras clave	*Seguridad, aplicaciones móviles, auditoría*

Resumen del Trabajo:

El presente trabajo fin de máster busca proporcionar las pautas en el proceso de auditoría de seguridad de las aplicaciones móviles, de acuerdo con el reciente estándar de Verificación de Seguridad de Aplicaciones Móviles publicado por OWASP como un marco de requisitos de seguridad necesarios para diseñar, desarrollar y probar aplicaciones móviles seguras.

Para ello en primer lugar, se llevará a cabo un estudio de la situación actual respecto a las vulnerabilidades y vectores de amenazas más representativos en las aplicaciones móviles, puesto que el malware móvil ha experimentado en los últimos años un rápido crecimiento debido a que la mayoría de los dispositivos disponen de acceso a datos sensibles y demás información del usuario como la localización, contactos, imágenes, etc.

Se presentarán las distintas metodologías para la elaboración de auditorías de aplicaciones móviles y las herramientas de análisis disponibles actualmente y se procederá a proponer una metodología a seguir para la realización de dichas auditorías.

Finalmente se llevará a cabo el análisis de una aplicación Android, de acuerdo con las recomendaciones proporcionadas en los anteriores apartados para la realización de una correcta auditoría de seguridad y se presentarán los resultados obtenidos.

Abstract:

The current Master's Thesis aims to provide the guidelines in the process of the audits in security for mobile applications, according the recent published

standard of security verification of mobile applications, by OWASP, as a framework of security requisites for designing, developing and testing of secure mobile applications.

First, a study of the current situation regarding the more representative vulnerabilities and thread vectors will be carried out, as the mobile malware has experienced a high growth in the recent years, due to most of the mobile devices have access to sensitive data and other user information such as location, contacts, images, etc.

The different existing methodologies for mobile application audits will be presented, as well as the available analysis tools, and a selected methodology will be proposed for the execution of such audits.

Finally, according the provided guidelines in this document, the complete analysis of one selected Android application for its study will be performed, serving as an example of the execution of a right security audit. All the achieved results will be also presented at the end of this thesis as conclusion and future lines of work.

Índice de Contenidos

Lista de figuras

1. Plan de Trabajo

1.1 Contexto y justificación del Trabajo

De acuerdo con las estadísticas de 2018 recogidas en la web *We are social*, el número de usuarios de móvil el año pasado era de 5.135 billones con una penetración del 68%. De ellos, más de la mitad de los dispositivos usados son ya dispositivos inteligentes y de acuerdo con Nielsen, un smartphone promedio en EEUU tiene de media unas 41 aplicaciones descargadas por el usuario.

Si atendemos a los datos respecto al crecimiento de las descargas de apps, 197 billones en 2017 frente a los 149 billones en 2016, estaríamos hablando con una proyección similar, de 352 billones de descargas estimadas en 2021.

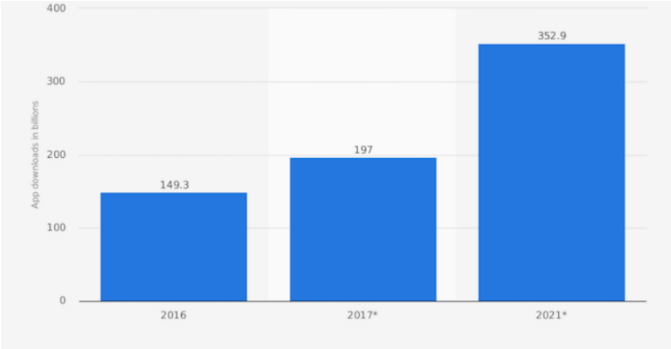


Imagen 1: Crecimiento de descargas de aplicaciones móviles

La mayoría de estas aplicaciones son descargadas desde sitios reputados como la Play Store de Google o su equivalente App Store en Apple, sin embargo, el malware móvil está experimentando un rápido crecimiento incluso en estas app supuestamente 'legítimas', debido a que la mayoría dispone de acceso a datos sensibles y demás información del usuario como las coordenadas de localización, lista de contactos, credenciales de usuario, archivos (imágenes, videos, documentos) o SMS entre otros.

Aunque son 3 los vectores potenciales a través de los cuales los hackers acceden a los datos sensibles de los móviles: las ya mencionadas aplicaciones (malware, spyware, adware), la red (Phishing, Man-In-The-Middle, etc.) y el dispositivo en sí mismo (explotación de vulnerabilidades del SO); en los últimos 2 años se ha experimentado un gran crecimiento en la filtración de datos sensibles a través de las aplicaciones, especialmente en SO de código abierto como Android, debido a que más del 59% de las apps envían datos fuera del dispositivo.

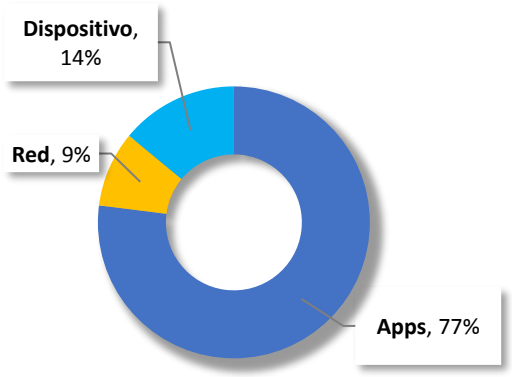


Imagen 2: Vectores de amenazas de seguridad en móviles

A pesar de ello, los usuarios a menudo ignoran las advertencias en relación con los permisos de acceso, las actualizaciones sugeridas y los avisos de contraseña ya que en general se carece de la necesaria conciencia de seguridad.

Ante esta situación, y especialmente en el ámbito empresarial donde los móviles también han adquirido una presencia muy relevante, parece evidente que los profesionales de la seguridad han de anticiparse a los riesgos y adaptarse al rápido crecimiento de las amenazas provenientes de las aplicaciones móviles, desde su desarrollo hasta el usuario final, llevando a cabo los controles necesarios para la adecuada y temprana detección de vulnerabilidades y posibles *exploits* que puedan comprometer la seguridad de datos en primer lugar y de sus negocios, por ende.

El presente trabajo busca servir de guía en el proceso de auditoría de seguridad de las aplicaciones móviles, de acuerdo con el reciente estándar de Verificación de Seguridad de Aplicaciones Móviles (MASVS) 1.0, publicado por OWASP como un marco de requisitos de seguridad necesarios para diseñar, desarrollar y probar aplicaciones móviles seguras.

Para ello en primer lugar, se llevará a cabo un estudio de la situación actual respecto a las vulnerabilidades y vectores de amenazas más representativos en las aplicaciones móviles, así como de los modelos de seguridad y los estándares de verificación de seguridad más reconocidos en la actualidad.

Se presentarán las distintas metodologías y herramientas disponibles para la elaboración de auditorías de aplicaciones móviles y se procederá a proponer una metodología a seguir para la realización de dichas auditorías.

Finalmente se llevará a cabo el análisis de una aplicación sobre la plataforma Android, que nos permitirá poner en práctica las pautas proporcionadas en los anteriores apartados para la realización de una correcta auditoría de seguridad de aplicaciones móviles.

1.2 Objetivos del Trabajo

El principal objetivo de este TFM será el de proporcionar una guía para el proceso de la auditoría de seguridad de las aplicaciones móviles aplicada al ejemplo de una aplicación real en Android.

Para su consecución, se buscará, además:

- Proporcionar información actual en relación con las principales vulnerabilidades de las aplicaciones móviles, detallando los vectores de amenazas existentes en la actualidad.
- Detallar los modelos de seguridad para aplicaciones móviles actuales y los estándares de verificación de seguridad más reconocidos y aceptados.
- Identificar las distintas metodologías existentes para el análisis de vulnerabilidades en aplicaciones móviles y detallar las particularidades de cada una.
- Proporcionar las bases del marco legal al que deben ceñirse los desarrolladores de aplicaciones móviles en relación con la protección de datos personales, financieros y de salud entre otros.
- Conocer las distintas herramientas disponibles actualmente para la realización de auditorías de seguridad de aplicaciones en función del sistema operativo.
- Proponer una metodología a seguir para la realización de auditorías de seguridad de aplicaciones móviles de acuerdo con los datos recopilados en el marco teórico del proyecto.
- Realizar el análisis práctico de un caso propuesta a modo de ejemplo que nos permita identificar las posibles vulnerabilidades de la aplicación de estudio y elaborar un informe de auditoría en función de los resultados obtenidos.
- Presentar a modo de conclusión los resultados alcanzados a lo largo del proyecto y las posibles líneas de investigación futuras.

1.3 Enfoque y método seguido

A lo largo de este TFM, se va a realizar una auditoría de seguridad de una aplicación móvil real sobre Android.

Para ello, el estándar elegido es el reciente estándar de Verificación de Seguridad de Aplicaciones Móviles (MASVS) 1.0, publicado por OWASP para proporcionar una línea de base para la verificación de seguridad en aplicaciones móviles.

Se tomarán por tanto como referencia los Requisitos de Seguridad MASVS para Android así como la "*Mobile security testing guide"* propuesta por OWASP y se llevarán a cabo tanto el análisis estático como el dinámico para la elaboración del informe final de conclusiones.

Para alcanzar este objetivo principal y demás objetivos enumerados en el apartado anterior, se ha previsto el siguiente desglose de tareas:

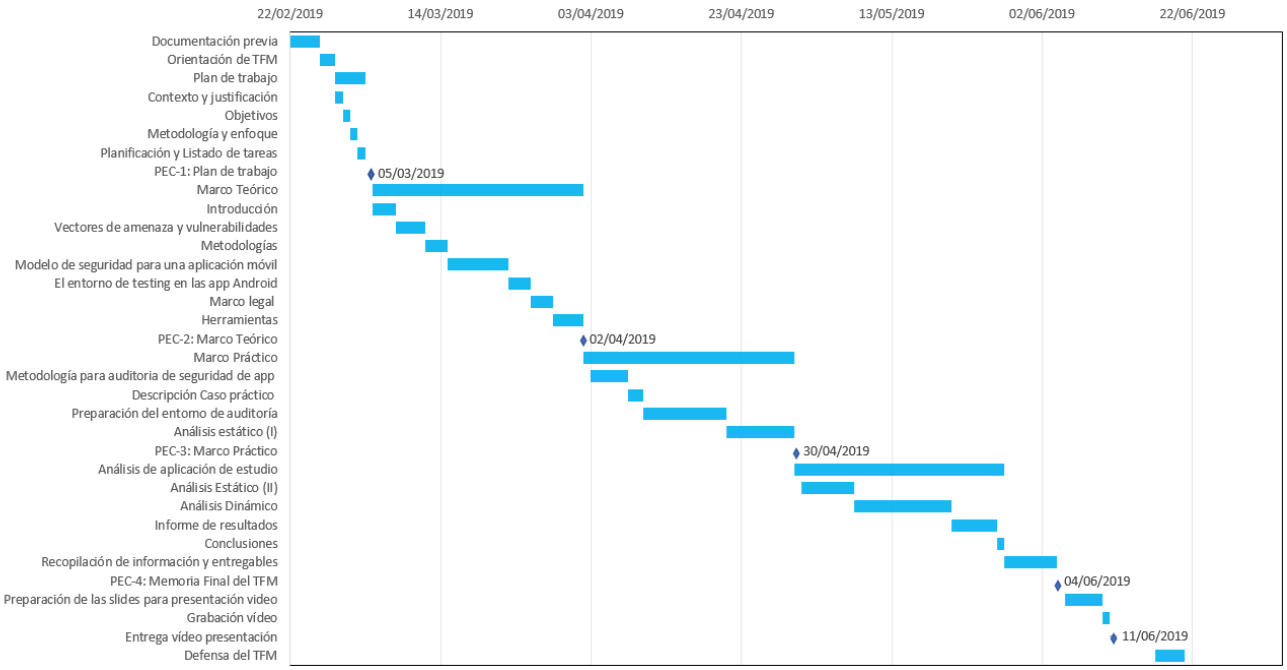
1. Documentación previa
2. Plan de trabajo
3. Introducción
4. Amenazas y vulnerabilidades en las aplicaciones móviles
5. Tipos de aplicaciones móviles
6. Metodologías de análisis de vulnerabilidades
7. Modelo de seguridad para una aplicación móvil
   a. MASVS, Estándar de Verificación de Seguridad de Aplicaciones Móviles de OWASP
   b. Áreas claves en la seguridad de aplicaciones móviles
8. El entorno de testing en las aplicaciones Android
9. Marco legal
10. Herramientas
11. Caso Práctico: Auditoría de seguridad de aplicación Android
    a. Metodología para auditoria de seguridad de app
    b. Descripción de la app de estudio
    c. Preparación del entorno de auditoría
    d. Análisis estático
    e. Análisis dinámico
    f. Informe de resultados

1.4 Planificación del Trabajo

El conjunto de tareas previstas a lo largo del TFM así como la planificación temporal de cada una de ellas se presentan a continuación en el siguiente diagrama:

Nombre de Tarea	Duración	Inicio	Fin
Trabajo Fin de Máster	**120**	**21/02/2019**	**21/06/2019**
Documentación previa	4	22/02/2019	26/02/2019
Orientación de TFM	2	26/02/2019	28/02/2019
Plan de trabajo	**4**	**28/02/2019**	**04/03/2019**
Contexto y justificación	1	28/02/2019	01/03/2019
Objetivos	1	01/03/2019	02/03/2019
Metodología y enfoque	1	02/03/2019	03/03/2019
Planificación y Listado de tareas	1	03/03/2019	04/03/2019
PEC-1: Plan de trabajo	**0**	**05/03/2019**	**05/03/2019**

Marco Teórico	**28**	**05/03/2019**	**02/04/2019**
Introducción	3	05/03/2019	08/03/2019
Vectores de amenaza y vulnerabilidades	4	08/03/2019	12/03/2019
Metodologías	3	12/03/2019	15/03/2019
Modelo de seguridad para una aplicación móvil	8	15/03/2019	23/03/2019
El entorno de testing en las app Android	3	23/03/2019	26/03/2019
Marco legal	3	26/03/2019	29/03/2019
Herramientas	4	29/03/2019	02/04/2019
PEC-2: Marco Teórico	**0**	**02/04/2019**	**02/04/2019**
Marco Práctico	**28**	**02/04/2019**	**30/04/2019**
Metodología para auditoria de seguridad de app	5	03/04/2019	08/04/2019
Descripción Caso práctico	2	08/04/2019	10/04/2019
Preparación del entorno de auditoría	11	10/04/2019	21/04/2019
Análisis estático (I)	9	21/04/2019	30/04/2019
PEC-3: Marco Práctico	**0**	**30/04/2019**	**30/04/2019**
Análisis de aplicación de estudio	**28**	**30/04/2019**	**28/05/2019**
Análisis Estático (II)	7	01/05/2019	08/05/2019
Análisis Dinámico	13	08/05/2019	21/05/2019
Informe de resultados	6	21/05/2019	27/05/2019
Conclusiones	1	27/05/2019	28/05/2019
Recopilación de información y entregables	7	28/05/2019	04/06/2019
PEC-4: Memoria Final del TFM	**0**	**04/06/2019**	**04/06/2019**
Preparación de slides para presentación	5	05/06/2019	10/06/2019
Grabación vídeo	1	10/06/2019	11/06/2019
Entrega vídeo presentación	**0**	**11/06/2019**	**11/06/2019**
Defensa del TFM	**4**	**17/06/2019**	**21/06/2019**



1.5 Breve sumario de productos obtenidos

Durante la elaboración del Trabajo Fin de Máster se prevén los siguientes entregables:

1. **PEC-1: Plan de trabajo**. Contendrá el objetivo y alcance del TFM, la descripción de la metodología a seguir, así como un listado de tareas para alcanzar los objetivos descritos y su planificación temporal.

2. **PEC-2: Marco Teórico**. La segunda entrega contendrá el resultado del análisis teórico del TFM incluyéndose información en relación con las amenazas y vulnerabilidades, las metodologías de análisis existentes, modelos de seguridad de las aplicaciones móviles y el marco legal, entre otras.

3. **PEC-3: Marco Práctico**. La tercera entrega tiene como objetivos describir la app de estudio, así como la preparación del entorno de auditoría para después incluir los primeros resultados procedentes del análisis estático de vulnerabilidades.

4. **PEC-4: Memoria final del TFM**. Esta última entrega incluirá la síntesis de todo el trabajo realizado durante el TFM, incluyendo los resultados completos de los análisis estáticos y dinámicos de la aplicación, las conclusiones extraídas y demás información relevante.

5. **Presentación en vídeo**

1.6 Breve descripción de los otros capítulos de la memoria

El trabajo fin de máster contendrá los siguientes capítulos dentro del marco teórico previsto:

1. Amenazas y vulnerabilidades en las aplicaciones móviles

Este capítulo repasará brevemente el concepto de amenaza de seguridad y vulnerabilidad de las aplicaciones móviles e incluirá la descripción de las principales amenazas de seguridad en la actualidad, de acuerdo con el listado Top 10 de riesgos móviles de OWASP Mobile Security Project.

2. Tipos de aplicaciones móviles

Las aplicaciones móviles están diseñadas para ser ejecutadas directamente en la plataforma para la que han sido diseñadas (Android e IOS actualmente aúnan el 90% de mercado), sobre un navegador web o mediante una mezcla de ambos. En este capítulo se definirán las características principales de cada tipo.

3. Metodologías de análisis de vulnerabilidades

A la hora de revisar la seguridad de las aplicaciones móviles, son dos las principales metodologías que se describirán en este apartado, con sus particularidades y herramientas disponibles: el análisis estático y el dinámico. Por un lado, el análisis estático nos permitirá realizar el análisis en reposo de la aplicación, esto es, centrándonos en su código fuente o binario de la aplicación.
Por otro, el análisis dinámico, llevará a cabo un estudio del comportamiento de la aplicación durante su ejecución, con el fin de identificar los posibles problemas.

4. Modelo de seguridad para una aplicación móvil

El presente trabajo pretende servir de guía en el proceso de auditoría de seguridad de aplicaciones móviles de acuerdo con el reciente estándar de Verificación de Seguridad de Aplicaciones Móviles (MASVS) versión 1.0, publicado por OWASP como un marco de requisitos de seguridad necesarios para diseñar, desarrollar y probar aplicaciones móviles seguras. En este capítulo se va a describir brevemente el estándar, los requerimientos de seguridad que contempla y cuáles son las áreas claves en la seguridad de aplicaciones móviles que considera imprescindibles.

5. Marco legal

Este apartado busca proporcionar brevemente la información necesaria en relación con el marco legal al que deben ceñirse los desarrolladores de aplicaciones móviles y cuáles son las normativas vigentes en lo que se refiere al tratamiento seguro de datos personales, los datos financieros y de pago, de salud, etc. según el ámbito de cada aplicación.

6. El entorno de auditoría en las aplicaciones Android

La aplicación elegida para llevar a cabo la auditoría de seguridad del presente trabajo se ejecuta en sobre la plataforma Android. Este apartado pretende definir las bases del entorno de pruebas necesarios para llevar a cabo un análisis de aplicaciones Android, además de proporcionar información básica en relación con la plataforma, la arquitectura de seguridad y demás información relevante para la preparación del entorno de auditoría.

7. Herramientas

Existen en la actualidad multitud de herramientas en el mercado que nos permiten llevar a cabo análisis de seguridad de las aplicaciones móviles en función de su sistema operativo. En este capítulo se describirán brevemente algunas de las herramientas más recomendadas para este tipo de análisis y se detallará la herramienta escogida para el análisis de la aplicación de estudio de este trabajo, y que será aquella que se adapte mejor a las características de ésta.

8. Caso Práctico: Auditoría de seguridad de aplicación Android

El caso práctico será el capítulo principal dentro del marco práctico del trabajo fin de máster, que contendrá principalmente, la siguiente información:

- Definición de la metodología empleada para la auditoria de seguridad
- Descripción detallada de la app de estudio elegida para servir de ejemplo práctico.
- Descripción del proceso de preparación del entorno de auditoría, emuladores y herramientas necesarias para alcanzar el objetivo de análisis previsto.
- Descripción del detalle del Análisis estático llevado a cabo sobre la aplicación.
- Descripción de detalle del Análisis dinámico realizado sobre la aplicación.
- Informe de resultados que contenga los resultados completos de los análisis estáticos y dinámicos ejecutados previamente sobre la aplicación.

9. Conclusiones

Finalmente, el capítulo de conclusiones incluirá el resumen de los resultados obtenidos de la auditoría de seguridad de la aplicación de estudio, así como las recomendaciones derivadas de los resultados de ésta.

2. Introducción

Las tendencias de crecimiento de descargas de aplicaciones móviles en los próximos años a nivel mundial auguran un crecimiento exponencial de las amenazas de seguridad y malware diseñados específicamente para los dispositivos móviles y en especial para las plataformas más extendidas Android e iOS.

Estudios recientes [5], como el llevado a cabo por *Arxan Technology* (proveedor líder de aplicaciones móviles en los sectores de salud y finanzas) han evidenciado que el 90 por ciento de las aplicaciones analizadas son vulnerables en, al menos, dos de los diez riesgos de seguridad publicados por OWASP dentro de su plan Mobile Web Security.

Estos datos revelan la necesidad de reforzar las medidas de seguridad a las que han de someterse las aplicaciones móviles y a pesar de ello, multitud de negocios hoy en día no destinan un porcentaje separado de sus presupuestos a la revisión de la seguridad de sus aplicaciones móviles.

La mayoría de las aplicaciones móviles en el mercado manipulan datos sensibles de los usuarios cumpliendo con las más recientes regulaciones en materia de privacidad, como el Reglamento General de Protección de Datos (GDPR), entrado en vigor en mayo de 2018:

- Datos de usuario: registros de llamadas, contactos, SMS, calendario, etc.
- Datos personales: nombre, correo electrónico, etc.
- Datos generales: proveedor de servicios, país, sistema operativo, etc.
- Identificadores: IMEI del dispositivo, credenciales, etc.

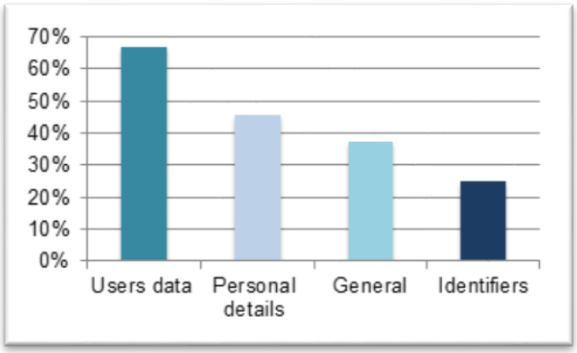


Imagen 3: Porcentaje de Datos sensibles manipulados, Pradeo 2018

Los usuarios de aplicaciones proporcionan toda serie de datos requeridos asumiendo que éstas son seguras y que los proveedores de aplicaciones llevan a cabo exhaustivos controles de seguridad para proteger sus aplicaciones, en especial

aquellas que manejan datos más sensibles, como las aplicaciones bancarias o de salud.

La mayoría de tales usuarios sin embargo no dudaría en cambiar de proveedor o dejaría de usar una aplicación de llegar a conocer que éstas no son los suficientemente seguras y los estudios recientes tasan en más de un 80% el porcentaje de usuarios que cambiarían a alternativas similares si éstas garantizaran mayores medidas de seguridad.

La creciente preocupación y concienciación de la sociedad en materia de seguridad implica cada vez más la necesidad de que los desarrolladores y proveedores de aplicaciones intensifiquen sus controles de seguridad no como una inversión en tecnología sino como una medida de fidelización de los consumidores basada en la confianza.

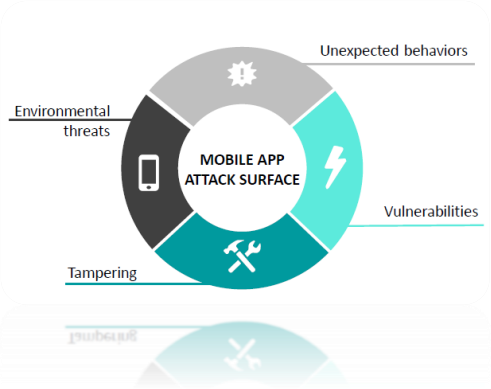
3. Amenazas y vulnerabilidades en las aplicaciones móviles

A la hora de realizar una auditoría de seguridad, es importante determinar los conceptos claves de vulnerabilidad y amenaza y como se relacionan entre sí, aunque no sea exclusivo del ámbito de la aplicación móvil.

- Una vulnerabilidad es el fallo o debilidad detectada en un sistema, en este caso, en una aplicación móvil, que pueda ponerlo en riesgo por llegar a comprometer su integridad, confidencialidad o la disponibilidad del servicio.
- Una amenaza, ya sea interna o externa, es toda acción llevada a cabo aprovechándose de una vulnerabilidad detectada en un sistema para atacar contra su seguridad.
- Un riesgo, será definido por tanto como la probabilidad de que una amenaza se materialice aprovechando una vulnerabilidad y produciendo un determinado daño o impacto en el sistema atacado.

En las apps móviles, las amenazas pueden ser clasificadas en dos categorías genéricas:

- Amenazas internas: representan agujeros de seguridad que pasan inadvertidos durante la fase de desarrollo de la aplicación, por ejemplo, al confiar en librerías de terceras partes que podrían incorporar comportamientos inesperados, como conexiones a servidores desconocidos, o vulnerabilidades.
- Amenazas externas: basadas en la habilidad de un hacker de acceder a la app, alterar el funcionamiento de ésta o derivarla hacia actividades maliciosas, en lo que se conoce como *tampering*. También se incluirían dentro de esta categoría los ataques que aprovechan las conexiones de red o las vulnerabilidades del SO, así como la publicación de malware de cualquier tipo.



Con el objetivo de concienciar acerca de la seguridad web a través de la identificación de algunos de los riesgos más críticos a los que se enfrentan las organizaciones, en 2003 nace el proyecto OWASP, *Open Web Application Security Project* o Proyecto Abierto de Seguridad de Aplicaciones Web, publicando una lista de referencia de las principales amenazas vigentes y actualizándola periódicamente.

Como parte de este proyecto, en 2014 surge el Mobile Security Project [3], con el objetivo de proporcionar los recursos adecuados a desarrolladores y equipos de seguridad para reforzar la seguridad, concretamente de las aplicaciones móviles. Para ello y de manera análoga a la lista de riesgos de aplicaciones web, se clasifican los riesgos de seguridad móvil y se proporcionan los controles enfocados a reducir tanto su impacto como la posible explotación.

La lista de riesgos de aplicaciones móviles más reciente, publicada en 2016, deriva de la especial atención prestada a la capa de aplicación, la infraestructura de los servidores con los que se comunican las aplicaciones móviles, la integración entre ellas, los servicios de autenticación remota y las características específicas de las plataformas cloud.

Así pues, el listado TOP 10 de los riesgos de seguridad móvil publicado por OWASP [4], es el incluido a continuación:

Imagen 4: OWASP Top 10 de los riesgos de seguridad móvil (2016)

M1: Uso inapropiado de la aplicación

Explotación: sencilla
Impacto: Severo

El Top1 es para el uso inapropiado o abuso de las características de la plataforma o de los controles de seguridad de ésta, incluyéndose en esta categoría los *intents* u objetos de acción de Android usados para solicitar una acción de otro componente de la aplicación, los permisos de la plataforma, el uso incorrecto del TouchID, el Keychain o cualquier otro control de seguridad que forme parte del SO.

Hay varias formas por las que una app puede sufrir este riesgo:

- Violación del cumplimiento de las guidelines de seguridad publicadas por las principales plataformas, Android, iOS y Windows Phone.
- Violación de las convenciones o prácticas comunes en las aplicaciones móviles
- Uso incorrecto no intencionado, por ejemplo, debido al uso incorrecto de la API.

M2: Almacenamiento inseguro de datos

Explotación: sencilla
Impacto: Severo

Esta categoría resulta de la combinación del M2 y M4 del Top 10 publicado en 2014: Almacenamiento inseguro de datos y Fuga de datos accidental. Su explotación podría llevarla a cabo un adversario que consigue un móvil perdido o robado para después desarrollar un malware u otra aplicación que se ejecute en el dispositivo y que actúe en nombre del adversario.

Las vulnerabilidades de almacenamiento de datos inseguros conllevan los siguientes riesgos empresariales para la organización propietaria de la aplicación: robo de identidad, fraudes, daño en su reputación, violación de la política externa, pérdida de material, etc.

M3: Comunicación insegura

Explotación: sencilla
Impacto: Severo

Los datos se intercambian comúnmente en modo cliente-servidor de manera que cuando la aplicación transmite sus datos, éstos atraviesan tanto la red del operador del dispositivo móvil como Internet. Han de contemplarse los siguientes agentes:

- una red Wi-Fi comprometida o monitorizada compartida por un adversario
- dispositivos de red (routers, celdas móviles, proxys, etc.)
- malware en el propio dispositivo móvil

Esta categoría incluiría los casos de pobre procedimiento de *handshaking*, versiones incorrectas de SSL, negociación pobre, comunicación en texto claro de datos sensibles, entre otros.

M4: Autenticación insegura

Explotación: sencilla
Impacto: Severo

Esta categoría abarca desde la autenticación del usuario final a la mala gestión de sesiones, que puede incluir, entre otros:

- fallo en la identificación del usuario cuando ha de ser requerido
- fallo en el mantenimiento de la identidad del usuario cuando es requerido
- debilidad en la gestión de sesiones.

El impacto de una pobre autenticación resultará como mínimo en daños en la reputación, robo de información o acceso no autorizado a los datos.

M5: Criptografía insuficiente

Explotación: sencilla
Impacto: Severo

Esta categoría incluye todos aquellos casos en los que a pesar de que la criptografía ha sido aplicada, no ha sido suficiente o aplicada correctamente y por tanto puede resultar en accesos no autorizados a información sensible en el dispositivo como violaciones de privacidad, robo de información, robo del código de la aplicación, de la propiedad intelectual, etc.

M6: Autorización insegura
Explotación: sencilla
Impacto: Severo

En este apartado se contemplan todos aquellos fallos acontecidos durante el proceso de autorización y se hace hincapié en la diferencia entre los posibles problemas detectados durante la autenticación del usuario, recogidos en el M4, y los propios de las decisiones de autorización llevados a cabo en el lado del cliente, como puede ser el proporcionar acceso a determinado recurso si la autenticación se ha realizado con éxito.

M7: Calidad del código del cliente
Explotación: difícil
Impacto: Moderada

Esta categoría es la evolución de la categoría de 2014: Decisiones de seguridad a través de entradas no confiables, en la que se consideraban aquellas entradas no confiables enviadas a métodos sensibles de la aplicación para que ésta realizase acciones no deseadas. La categoría en este caso se amplía especialmente a ataques típicos para explotar *memory leaks* y *buffer overflows* que puedan resultar en robo de información o de propiedad intelectual.

M8: Alteración del Código
Explotación: sencilla
Impacto: Severo

Una vez que una aplicación es descargada en el dispositivo, tanto el código como los datos pasan a residir en éste, por lo que, si un atacante es capaz de modificar el código o las API que utiliza, cambiar el contenido de la memoria dinámicamente o incluso modificar los datos de la aplicación, conseguirá alterar el uso original de la app en su propio beneficio.

Esta categoría incluye por tanto los casos de cambios en los paquetes binarios de la aplicación, la modificación de recursos locales, métodos o de la memoria dinámica, entre otros.

M9: Ingeniería inversa
Explotación: sencilla
Impacto: Moderada

Hoy en día son muchas las herramientas disponibles que permiten al atacante analizar el binario de una aplicación y determinar a partir de éste su código fuente, librerías, algoritmos y demás particularidades. Esta información puede ser usada posteriormente para explotar otras vulnerabilidades de la aplicación o bien para revelar información de sus servidores, constantes criptográficas o la propiedad intelectual.

M10: Funcionalidad externa
Explotación: sencilla
Impacto: Severo

Esta última categoría se centra en toda aquella información que pueda ser extraída de archivos de trazas, de configuración o de los propios binarios de la aplicación en los que el análisis del código fuente revele información dejada atrás accidentalmente durante el desarrollo o *debugging*.

Es habitual que las aplicaciones dispongan de funcionalidades ocultas o controles de desarrollo internos que no están destinados a ser liberados en los entornos de producción. De esta manera, si un atacante es capaz de acceder a tales capacidades podrá explotar fácilmente dicha funcionalidad para llevar a cabo un ataque desde su propio sistema.

4. Tipos de aplicaciones móviles

El término aplicación móvil o simplemente *app* en su abreviatura del inglés, se refiere a todo aquel programa autocontenido, diseñado para ser ejecutado en un dispositivo móvil y se estima que más del 80% del uso del móvil se emplea a través de tales aplicaciones.

El diseño de una app se puede enfocar de distintas maneras, ya sea para que sean ejecutadas directamente sobre una plataforma en concreto, sobre un navegador web o mediante una combinación de ambos, por lo que es importante conocer las particularidades de cada uno de estos enfoques como punto de partida para definir el enfoque de una auditoría de seguridad de una aplicación móvil.

4.1 La aplicación nativa

Las aplicaciones nativas son las desarrolladas específicamente para un sistema operativo móvil en concreto, como Android o iOS, a partir del Software Develoment Kit (SDK) que éstos proporcionan para su desarrollo.

Por el hecho de estar completamente integradas en un SO específico, estas aplicaciones presentan un funcionamiento muy estable y fluido y se adaptan perfectamente a las funcionalidades y componentes del dispositivo a la hora de acceder a las distintas utilidades y hardware de éste.

Una misma aplicación desarrollada para dos SO diferentes requerirá, por tanto, dos códigos de implementación diferentes, que en general será:

- SDK basado en los lenguajes de programación Java y Kotlin para aplicaciones nativas Android
- Lenguaje Objective-C o Swift para aplicaciones en iOS.
- .Net y más recientemente C++ y Javascript para aplicaciones en otros SO como Windows Phone (actualmente con una cuota de mercado muy residual)

La aplicación nativa presenta una serie de ventajas e inconvenientes, recogidas en la siguiente tabla:

Ventajas Apps nativas	Desventajas Apps nativas
1- Cada plataforma cuenta con su propia tienda de Aplicaciones, desde donde pueden ser descargadas.	1- Requiere mayor esfuerzo de desarrollo, en cuanto a tiempos, esfuerzo y coste.
2- Proporcionan mejor aprovechamiento de los recursos del dispositivo donde están instaladas (Memoria, CPU, Cámara, sensores, etc).	2- Es necesario desarrollar una App específica para cada tipo de Sistema Operativo.
3- Ofrecen mejor rendimiento, siendo más rápidas, más fluidas y proporcionando una mejor experiencia de usuario.	3- Han de pasar los filtros y controles de programación y calidad de cada Tienda de Aplicaciones para conseguir que sean publicadas.
4- Las Apps nativas son de mejor calidad por lo que ofrecen mejor experiencia de usuario.	
5- No requieren de una continua conexión a internet para su funcionamiento.	

Ha de tenerse en cuenta que lo que aparentemente se presenta como ventaja o inconveniente desde el punto de vista del usuario o desarrollador, puede no serlo desde el punto de vista de la auditoría de la seguridad y por tanto habrá que analizarlas de manera individualizada.

4.2 La aplicación web

Las aplicaciones web son básicamente páginas web optimizadas para trabajar en un dispositivo móvil de manera que aparentemente se vean como una app nativa. Tienen sin embargo una integración limitada con los distintos componentes del dispositivo ya que se ejecutan a través de una URL desde el navegador web de éste, con independencia del sistema operativo, y por tanto su rendimiento es peor en comparación con las apps nativas.

Este tipo de aplicaciones suelen desarrollarse en HTML5, como la mayor parte de las páginas web actuales y no siguen los principios de diseño específicos de las plataformas.

Podemos resumir las principales ventajas y desventajas de este tipo de aplicaciones, de nuevo en una tabla:

Ventajas Apps web	Desventajas Apps web
1- Las apps web son multiplataforma e independientes del SO del dispositivo	1- El rendimiento y la velocidad serán menores puesto que dependerán directamente del servidor web que aloje la *app*.
2- No requieren la aprobación de las tiendas de Aplicaciones para su publicación.	2- No pueden acceder a los recursos del dispositivo como cámara, sensores, calendario, etc.
3- Al ser un acceso vía Web, los usuarios siempre van a disfrutar de la última versión disponible.	3- Es necesario el acceso a internet para disponer de acceso a la *app*.
4- El proceso de desarrollo es más simple y económico que el resto de las alternativas.	4- Requieren mayores esfuerzos de marketing a la hora de su publicación al no disponer de la promoción de las Tiendas.

4.3 La aplicación híbrida

Las aplicaciones híbridas se ejecutan como una aplicación nativa, aunque sobre un navegador web embebido, llamado *webview*. Se suelen desarrollar en los lenguajes comunes de aplicaciones web, como HTML por lo que se pueden utilizar en las diferentes plataformas existentes; su interfaz de usuario por tanto será igualmente genérica y no estéticamente similar al del resto de aplicaciones de cada plataforma en concreto.

Estas aplicaciones además pueden acceder a las características hardware del dispositivo y presentan otra serie de pros y contras, reflejados en la tabla a continuación:

Ventajas Apps híbrida	Desventajas Apps híbrida
1- Son multiplataforma y pueden funcionar por tanto en cualquier SO.	1- La experiencia de usuario es peor en comparación con una *app* nativa.

2- Pueden publicarse en las Tiendas de Aplicaciones.	2- El aspecto visual de la app es genérico y por tanto no acorde con la estética de las diferentes plataformas.
3- Tienen acceso al hardware y demás recursos del dispositivo.	3- A pesar de poder acceder a los recursos, presentan un peor rendimiento.
4- El coste de desarrollo es menor comparado con una aplicación nativa.	4- No cuentan con las funcionalidades nativas del dispositivo.

En este trabajo fin de máster, la app de estudio elegida es una aplicación nativa para la plataforma Android por lo que la guía de auditoría estará especialmente enfocada a este tipo de aplicaciones en concreto. Sin embargo, las mismas técnicas y procedimientos podrían ser empleados para aplicaciones tanto web como híbridas.

5. Metodologías de análisis de vulnerabilidades

Se denomina análisis de vulnerabilidades al proceso de identificación de vulnerabilidades en una aplicación móvil. Dentro de este análisis, son dos los principales tipos que podemos diferenciar: el análisis estático y el dinámico, que idealmente se combinan con el objetivo de cubrir la mayor área posible de ataques sobre la aplicación.

5.1 Análisis estático

En líneas generales, el análisis estático o también denominado de código estático o SAST por sus siglas en inglés *Static Application Security Testing*, implica la revisión de la seguridad de una app a partir del código fuente de la misma, examinando los distintos componentes de ésta, ya sea de manera manual o automática.

Se trata de un análisis sencillo y aplicado comúnmente en las auditorías de código, que puede realizarse de dos maneras:

- análisis a partir del código fuente proporcionado por el equipo de desarrollo de la aplicación o en caso de ser código libre, descargado de su repositorio público.

- análisis a partir de la decompilación del código de la aplicación mediante el uso de ingeniería inversa o de herramientas de decompilación, si no se tiene acceso al código fuente directamente.

Aunque ambos enfoques son válidos desde el punto de vista de la auditoría, sin embargo, los mejores resultados se obtienen generalmente en los casos en los que se dispone de acceso total al código fuente de la aplicación.

Las revisiones de código manuales han de ser realizadas por expertos codificadores con la suficiente experiencia y conocimiento tanto en el lenguaje empleado como en los *frameworks* utilizados para la aplicación.

Estas revisiones, a pesar de ser procesos largos y a menudo tediosos para los expertos, destacan por ser buenas para la identificación de posibles vulnerabilidades en la lógica de la aplicación, violaciones de estándares y fallos en el diseño especialmente cuando el código es seguro pero lógicamente erróneo, escenarios que son difíciles de identificar con herramientas de análisis de código automáticas.

5.2 Análisis dinámico

El análisis dinámico, también conocido como DAST de *Dynamic Application Security Testing,* implica la evaluación de la app durante su ejecución para asegurar la calidad y seguridad de la aplicación durante este proceso en tiempo real**.** Se trata de un buen complemento al análisis estático pues proporciona información con relación a los recursos, las características de la app, los puntos de entrada, etc. desde el punto de vista del usuario.

El análisis dinámico se puedo llevar a cabo en la capa de plataforma como contra los servicios de *back-end* y las APIs, donde se analizan los patrones de las distintas solicitudes y respuestas de la aplicación móvil, y de esta manera comprobar si los mecanismos de seguridad aplicados proporcionan suficiente protección con los ataques más frecuentes: la divulgación de datos en tránsito, los problemas de autenticación y autorización y los errores de configuración del servidor, entre otros.

Para llevar a cabo este análisis, será necesario disponer de una herramienta analizadora de código dinámica, que puede ser:

- herramientas que podemos ejecutar en un pc conectado a un dispositivo móvil, ya sea Android o iOS, también llamadas emuladores, que nos permiten simular la ejecución real de la app
- herramientas que se ejecutan en el propio dispositivo móvil.

El análisis dinámico nos va a permitir detectar vulnerabilidades que serían difíciles de identificar si solo llevásemos a cabo un análisis estático. Por ejemplo, a través de un análisis estático podemos saber qué contraseña va a ser almacenada, mientras que

el dinámico leería la memoria y revelaría la contraseña almacenada en tiempo de ejecución.

Algunas aplicaciones de Android utilizan el método de ofuscación para prevenir que los posibles atacantes pudieran acceder al código de la app pero los análisis dinámicos revelarían la existencia de datos "*hardcoded*" enviado en las peticiones durante la ejecución, que no serían nunca detectados en un análisis estático exclusivamente.

En el apartado de herramientas veremos con más detalle las distintas herramientas disponibles en función del análisis y la plataforma a analizar.

6. Modelo de seguridad para una aplicación móvil

6.1 Estándar de Verificación de Seguridad de Aplicaciones Móviles de OWASP, MASVS

Con el objetivo de establecer un marco unificado de requisitos de seguridad para diseñar, desarrollar y probar aplicaciones móviles seguras en las plataformas iOS y Android, en 2016, el proyecto OWASP publica la primera versión del **Estándar de Verificación de Seguridad de Aplicaciones Móviles**, o MASVS.

Se trata de un estándar que busca proporcionar un nivel de confianza en la seguridad de las aplicaciones móviles, a emplearse con distintos objetivos:

- Métrica: proporcionando el estándar que permita comparar las aplicaciones entre sí por parte de desarrolladores o propietarios
- Guía: proporcionando una serie de pasos para la validación de las apps móviles en cualquiera de sus fases
- Línea de base: sirviendo de referencia para la verificación de seguridad de aplicaciones móviles.

MASVS se centra en la seguridad de la aplicación en el propio dispositivo móvil y otros aspectos como la comunicación de red entre ésta y otros puntos remotos, requerimientos generales del proceso de autenticación de usuarios y la gestión de sesiones; sin embargo, no incluye los posibles requerimientos específicos para servicios remotos, como, por ejemplo, los servicios web asociados a la aplicación.

6.1.1 Niveles de verificación

Tal y como se detalla en el [1] *Mobile Application Security Verification Standard*, MASVS define dos niveles estrictos de verificación de seguridad (L1 y L2), así como un conjunto de requisitos de resistencia a la ingeniería inversa (MASVS-R), en función

de si se desea alcanzar un nivel genérico de seguridad, la inclusión de medidas de defensa más profundas o la protección contra las amenazas del lado del cliente:

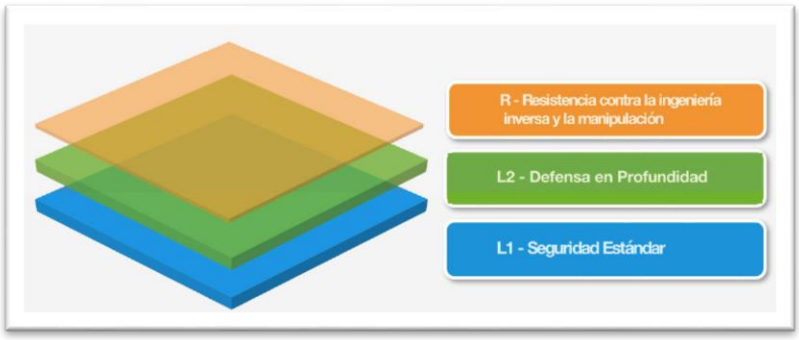


Imagen 5: Niveles de verificación

MASVS-L1: Nivel de Seguridad Estándar

Se trata del nivel que contiene los requerimientos genéricos de seguridad recomendados para todas las aplicaciones móviles, en cuanto a que proporciona las mejores prácticas relacionadas con la calidad del código, la adecuada gestión de datos sensibles y la interacción con su entorno.

Ejemplo de aplicación:

- todas las *apps*: como hemos visto, está enfocado a todas las aplicaciones móviles ya que el impacto de verificar los controles de seguridad sobre los costos de desarrollo y la experiencia del usuario es muy razonable.

MASVS-L2: Nivel de Defensa en Profundidad

El nivel L2 de MASVS está enfocado a todas aquellas aplicaciones que manejan datos altamente sensibles, pues proporciona controles de seguridad más avanzados que el nivel L1, necesarios cuando la seguridad ha de considerarse parte fundamental del diseño y arquitectura de la aplicación.

Ejemplos de aplicación:

- *Apps* de salud: en general este tipo de aplicaciones manejan información personal identificable que puede ser utilizada con distintos objetivos de fraude, desde el robo de identidad a pagos fraudulentos.

- *Apps* de banca móvil y/o sector financiero: estas aplicaciones tienen acceso a información de cuentas, tarjetas de crédito, etc. además de permitir realizar

operaciones bancarias, por lo que han de incorporar las suficientes medidas de seguridad para prevenir el fraude.

MASVS-R: Nivel de Resistencia contra la ingeniería inversa y la manipulación

El nivel MASVS-R cubre, además, todos los controles de seguridad adicionales que se han de aplicar cuando la prevención de las amenazas del lado del cliente es uno de los objetivos en el diseño de la app, de manera que se consiga prevenir la alteración o modificación de los datos, así como la ingeniería inversa para extraer datos sensibles o el propio código de la aplicación.

Así pues, este nivel aprovecha las técnicas de protección de software suficientemente fuertes y verificables, por lo que resultaría adecuado para aquellas aplicaciones que manejan datos altamente confidenciales, para proteger la propiedad intelectual o la manipulación de una aplicación.

Ejemplo de aplicación MASVS L1+R:

- Industria de los juegos: en especial aquellos de juegos en línea competitivos en los que resulta primordial garantizar medidas anti-engaño puesto que las trampas pueden conducir a un descontento general de los jugadores, afectando por tanto a la reputación de la app, o incluso provocar distintos tipos de fallos.

Ejemplo de aplicación MASVS L1+R:

- Aplicaciones de banca online: que permiten al usuario realizar operaciones como movimientos de fondos y por tanto es necesario evitar técnicas como las de inyección de código.

6.1.2 MASVS y la guía para la verificación de la seguridad de las aplicaciones

A la hora de aplicar los requerimientos de seguridad proporcionados por MASVS, OWASP también proporciona una guía de verificación que describe los procedimientos técnicos necesarios, la [2] *Mobile Security Testing Guide* (MSTG).

De esta manera, aunque los requerimientos del MASVS son más genéricos y a alto nivel, éste se ve complementado por la MSTG que proporciona recomendaciones detalladas y procedimientos de verificación para cada uno de los sistemas operativos móviles más extendidos, actualmente Android e iOS.

Tenemos por tanto a nuestra disposición, los siguientes documentos de OWASP:

- El estándar Mobile Application Security Verification Standard (MASVS), con los requisitos genéricos de seguridad y los distintos niveles de verificación que deberemos elegir convenientemente en el momento de llevar a cabo nuestra auditoría de seguridad de aplicación móvil.

- La guía Mobile Security Testing Guide (MSTG), que nos proporcionará durante todo el proceso de auditoría de seguridad, las instrucciones necesarias para la verificación de cada uno de los requerimientos de MASVS así como las mejores prácticas en seguridad para cada sistema operativo.

- El listado Mobile App Security Checklist que permitirá aplicar los requerimientos de MASVS a la evaluación práctica de la aplicación.

Se enumeran a continuación los objetivos de control sugeridos para MASVS L1 y L2, que como puede observarse, tratan de cubrir en gran parte el listado Top 10 de amenazas sobre aplicaciones móviles publicado por OWASP:

- **V1: Requisitos de Arquitectura, Diseño y Modelado de Amenazas:** Este conjunto de requerimientos buscan garantizar que la planificación de la arquitectura de la app haya considerado los principales aspectos de seguridad de todos los componentes.

- **V2: Requerimientos en el Almacenamiento de datos y la Privacidad:** este control aúna los requerimientos necesarios para garantizar la protección de datos sensibles, así como evitar la divulgación de éstos.

- **V3: Requerimientos de Criptografía**: El propósito de este control es asegurar que la criptografía se aplica según las mejores prácticas (uso de librerías conocidas, primitivas criptográficas apropiadas, etc.)

- **V4: Requerimientos de Autenticación y Manejo de Sesiones**: Define los requerimientos básicos de gestión de sesiones de usuario y cuentas, cuando no se realiza de manera remota.

- **V5: Requerimientos de Comunicación a través de la red**: El objetivo de este control es asegurar la confidencialidad e integridad de los datos intercambiados entre la aplicación y el servidor.

- **V6: Requerimientos de Interacción con la Plataforma**: Estos controles buscan comprobar el uso seguro de las APIs de la plataforma y demás comunicación entre aplicaciones.

- **V7: Requerimientos de Calidad de Código y Configuración del Compilador**: Estos controles buscan asegurar el cumplimiento durante el desarrollo de app, de las prácticas de seguridad básicas.

Además de los controles listados anteriormente, se sugiere añadir el siguiente control adicional para garantizar el nivel MASVR-R:

- **V8: Requerimientos de Resistencia ante la Ingeniería Inversa**: esta sección tiene como objetivo la evaluación de los riesgos que podrían ser causados por una manipulación no autorizada de la aplicación o la ingeniería inversa del código.

La lista completa de requerimientos por cada una de las mencionadas categorías se incluye en el [Anexo A].

7. Marco legal

El pasado 25 de mayo de 2018, entró en vigor la nueva normativa de la Unión Europea para proteger la privacidad de sus ciudadanos, así como para armonizar la legislación de los países miembros de la UE: la **RGPD o Reglamento General de Protección de Datos** (también conocido a nivel europeo con las siglas GDPR).

La normativa afecta a todas las empresas y usuarios para hacer partícipe a éstos y de forma activa, de cuánta información se tiene de ellos, para qué se va a utilizar, a quién se le puede ceder o quién es el responsable de su tratamiento.

Las aplicaciones móviles no son ninguna excepción y como el resto de los sistemas que procesan información, están obligadas a ceñirse a los estándares y normativas para asegurar la integridad y confidencialidad de los datos sensibles que manejan o de las transacciones realizadas.

Esta situación no es nueva pues ya en 2013, se aprobó un dictamen europeo para especificar claramente el marco jurídico que debe ser aplicado a las aplicaciones móviles, concretamente el artículo 29 sobre protección de datos, del Dictamen 02/2013 sobre las aplicaciones de los dispositivos inteligentes. Este dictamen incluía ya conceptos como la limitación de la finalidad para la que se solicitan o almacenan datos personales, el consentimiento previo del usuario para tratamiento de datos y demás derechos del interesado entre otros.

La entrada en vigor de la RGPD ha implicado cambios en las aplicaciones móviles existentes previas a la normativa y ha afectado directamente al desarrollo de las

nuevas apps pues, aunque éstas han hecho uso de datos personales desde siempre, ahora debe haber un propósito justificado para ello, ser adecuados a la normativa y disponer del consentimiento del usuario, que demanda actualmente más control y transparencia. De hecho, Las multas por no cumplir la regulación del GDPR pueden ascender al 4% de los ingresos anuales o de hasta los 20 millones de euros.

Son varios los aspectos claves que los desarrolladores y propietarios de apps han de tener en cuenta:

1. El Consentimiento

El consentimiento es una de las principales novedades de esta normativa que incluye al respecto en su artículo 11, que éste ha de ser libre, informado, específico e inequívoco.

Es decir, es necesario contar con un consentimiento explícito del usuario para para recoger, usar y ceder sus datos, que no se preste de manera condicionada, que indique claramente la finalidad o finalidades de la recogida de esos datos y que no deje lugar a dudas.

La app móvil por su parte deberá incluir claramente la política de privacidad y avisos legales o en su defecto un enlace que dirija a una página que contenga esa información. La RGPD establece nuevas pautas para el tratamiento de datos de menores por lo que la información proporcionada ha de ser lo suficientemente comprensible incluso para ellos y además, ha de tenerse en cuenta que se prohíbe explícitamente marcar casillas de consentimiento por defecto.

2. Política de privacidad, finalidad y calidad de los datos

De acuerdo con esta nueva regulación, todas las apps deben establecer una política clara de privacidad que especifique en caso de solicitar datos personales a los usuarios, cuál es la finalidad o el plazo de conservación, además de ceñirse sólo a los estrictamente necesarios para prestar el servicio para el cual se ha creado la aplicación.

3. Nueva ampliación de derechos

A los ya reconocidos derechos de aceptación, rectificación, cancelación y oposición, la RGPD añade el derecho de supresión u olvido, por el que se permite a los usuarios solicitar que se elimine la información que se dispone de ellos.

La app debe incluir claramente en su política de privacidad, los derechos de los usuarios a solicitar el acceso, rectificación, supresión o limitación en el tratamiento de sus datos y puesto que éstos se recogen a través de la propia app, debe indicarse la dirección electrónica disponible para que pueda ejercer sus derechos.

4. Control de las brechas de seguridad

La nueva normativa establece en su artículo 33 la obligación de notificar cualquier violación sobre la seguridad de los datos personales en un plazo no mayor a 72 horas desde que se tiene conocimiento de ésta, a la autoridad de control competente. De igual manera deberá comunicarse al usuario el impacto sobre sus datos personales.

Aunque existen excepciones como en el caso de que los datos filtrados estuviesen cifrados, o no afectaran directamente a la intimidad de los usuarios, éstas no aplicarían si se trata de aplicaciones del sector bancario o financiero.

Por tanto, toda app estará obligada a notificar sobre una brecha de seguridad que afecte a los datos personales de sus usuarios, indicando el número de afectados, tipo de dato filtrado, consecuencias de tal filtración y cuáles son las medidas que se adoptan para atenuar tales consecuencias.

Se convierte así en requisito legal el concepto de privacidad por diseño, de manera que la protección tanto del usuario como de la privacidad de sus datos deben considerarse desde el inicio del proyecto y durante todo el ciclo de vida del mismo.

Existen además otras normativas que deberán de considerarse en función del área de la aplicación móvil. Así por ejemplo, las aplicaciones bancarias deberán de considerar la protección de los datos fiscales y de las transacciones bancarias, considerando las nuevas normativas como la PSD2 de regulación en los servicios de pagos digitales.

Otras aplicaciones en materia de salud que gestionen datos de salud de sus usuarios deberán ceñirse a normativas locales existentes en EU como HSCIC o CNIL entre otras.

8. El entorno de auditoría en las aplicaciones Android

La parte práctica del presente proyecto incluye el desarrollo de una auditoría de seguridad sobre una aplicación móvil nativa real en Android, tomándose como referencia los requisitos de Seguridad MASVS para Android, así como la "*Mobile security testing guide*" propuesta por OWASP.

Con el objetivo de proporcionar una guía completa en el proceso de la auditoría de seguridad de las aplicaciones móviles, se llevarán a cabo tanto el análisis estático como el dinámico, para lo cual será necesario como punto de partida la preparación de un entorno de auditoría de acuerdo con las distintas opciones disponibles y enumeradas a continuación, así como las particularidades que más se adecúen a la app analizada.

8.1 Básicos de la plataforma Android

➢ Estructura de las aplicaciones

El Android Package Kit, o APK como comúnmente es conocido, es el archivo que contiene tanto el código como recursos, activos y certificados necesarios para que la aplicación móvil funcione adecuadamente.

Se trata en esencia de un archivo comprimido que se emplea para distribuir e instalar los componentes empaquetados en la plataforma Android, tanto en dispositivos móviles como en otros dispositivos Android (*tablets*, ordenadores personales, etc.).

Como archivos comprimidos, podemos por tanto analizarlos con cualquiera de los softwares de compresión conocidos, para comprobar que tienen todos la misma estructura común que nos interesa conocer de cara a llevar a cabo una auditoría de aplicaciones:

- **AndroidManifest.xml**: ubicado en el directorio root del APK de la aplicación, es un fichero que describe la estructura de la app y contiene información general de ésta como el nombre y versión, versión API, configuración, componentes (actividades, servicios, broadcast receivers, etc.), permisos de acceso y demás información útil relativa a la app.

  Puede estar en formato binario xml que puede convertirse fácilmente en xml de texto claro mediante herramientas como AXMLPrinter2.

- **META-INF**: directorio que contiene los siguientes metadatos de la app:

  - MANIFEST.MF: archive que almacena los hashes de los distintos recursos de la app
  - CERT.RSA: certificado de la aplicación
  - CERT.SF: listado de los recursos y sus correspondientes SHA-1 en referencia al archivo MANIFEST.MF
- **assets**: archivo que contiene los recursos de la app tales como los archivos xml, JavaScript, imágenes, etc.
- **classes.dex:** conjunto de clases compiladas en formato DEX para su procesamiento por parte de la máquina virtual Dalvik o el proceso Android Runtime.
- **lib**: directorio que contiene el código compilado para los distintos procesadores como x86, ARM, MIPS, etc.
- resources.arsc: archivo que contiene los recursos pre-compilados como los ficheros xml.
- **res**: directorio que contiene los recursos no compilados en resources.arsc

➢ **Componentes de la aplicación móvil**

Toda aplicación móvil se compone de los siguientes componentes a alto nivel, proporcionados por el sistema operativo Android a través de APIs:

- *Activities*

Las actividades son los componentes principales de la interfaz gráfica de la aplicación de manera que habrá definida una actividad por cada una de las distintas ventanas de que disponga la aplicación. Todas las actividades, para poder invocarse, han de estar definidas en el archivo AndroidManifest.xml y están en cierta manera vinculadas. Para su ejecución, se almacenan en una pila tipo LIFO (*Last In – First Out*) que permite reanudar fácilmente la actividad anterior a partir de la actual.

Las actividades asimismo tienen su propio ciclo de vida y pueden encontrarse en cualquiera de los siguientes estados: activas, pausadas, detenidas o inactivas. Es el sistema operativo Android el encargado de gestionar estos estados.

- *Fragments*

Los fragmentos son componentes que funcionan dentro del ámbito de las actividades como una parte de la interfaz de usuario de la actividad, con el objetivo de facilitar el desarrollo en las apps multidispositivo (ya que proporcionan diseños más dinámicos y flexibles en la adaptación a diferentes tamaños de pantalla).

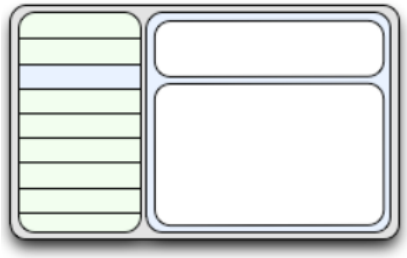


Imagen 6: Actividad con dos fragmentos definidos

Se trata de entidades independientes que incluyen todos los componentes necesarios pero que han de estar integradas dentro de las actividades. Tienen igualmente su propio ciclo de vida, pero ha de estar vinculado al de la actividad que los implementa, por ejemplo, si se pausa la actividad se pausarán necesariamente los fragmentos que contiene.

- *Services*

Los servicios son componentes del Sistema operativo Android que ejecutan tareas en segundo plano, tales como el procesamiento de datos, notificaciones, etc., sin interfaz gráfica. Se invocan a través de otros componentes, como las actividades, pero sin depender de éstas ya que aunque la actividad finalice, el servicio seguirá ejecutándose incluso si la app no está abierta. Al igual que las actividades, los servicios se declaran en el archivo AndroidManifest.xml.

- *Content providers*

Los proveedores de contenidos son los componentes que proporcionan un mecanismo estándar y eficiente para compartir datos entre las distintas aplicaciones, incluso las nativas.

Android emplea SQLite para almacenamiento de datos permanente de las aplicaciones y por defecto, las bases de datos definidas para una app solo son accesibles por ésta. Así, para que otras apps puedan acceder a su contenido bastará con definir explícitamente los proveedores de contenidos en el archivo *manifest* de la app que los comparte o bien en caso contrario, solo quedarán accesibles por la app que los creó.

A través de los proveedores de contenidos, además, cualquier aplicación que disponga de los permisos necesarios podrá modificar los datos de otras aplicaciones.

- *Intents*

Los objetos *Intent* son los elementos disponibles para comunicación entre los componentes de las aplicaciones o entre las distintas aplicaciones.

Android emplea estos objetos para enviar mensajes a todas las aplicaciones ante una llamada entrante o un SMS, por ejemplo; información relativa a la fuente de alimentación como batería baja; cambios en la red como pérdida de cobertura, etc.

Se pueden destacar tres casos de uso fundamentales de este tipo de objetos:

- Arrancar una actividad
- Iniciar un servicio
- Enviar un mensaje en modo *broadcast* o que será recibido por todas las aplicaciones

En cualquier caso, se trata de paquetes de información que contienen datos e información de interés para el componente que lo recibe como las instrucciones necesarias para realizar determinada acción, la acción a ejecutarse o la categoría del componente que maneja el *Intent*.

- *Broadcast receivers*

Los *Broadcast receivers* son los componentes que permiten a las aplicaciones recibir notificaciones procedentes de otras aplicaciones o del propio sistema operativo (por ejemplo, los de llamada o SMS entrante, batería baja, etc.) y reaccionar convenientemente ante éstas actualizando su interfaz, arrancando un servicio o actualizando algún contenido.

Estos componentes, al igual que las actividades y los servicios, han de estar declarados en el *manifest* de la aplicación para que pueda atender a los mensajes e incluso arrancarla de manera automática si se ha detectado un *Intent* relevante para ésta. De no estar declarados, la app no reaccionará a este tipo de mensajes.

➢ ***Sandbox* de aplicaciones**

La plataforma Android, basándose en la protección que proporciona Linux para identificar y aislar los recursos de las aplicaciones, asigna un ID de usuario único (UID) a cada aplicación para que todos sus procesos se ejecuten bajo éste y de esta manera cada app queda aislada del resto de las apps del sistema.

Con el objetivo de añadir una capa de seguridad, todas las apps se ejecutan de esta manera en lo que se denomina *Sandbox* de aplicaciones Android, que separa los datos de las apps y la ejecución del código del resto de apps del dispositivo, como se puede apreciar en el siguiente diagrama:

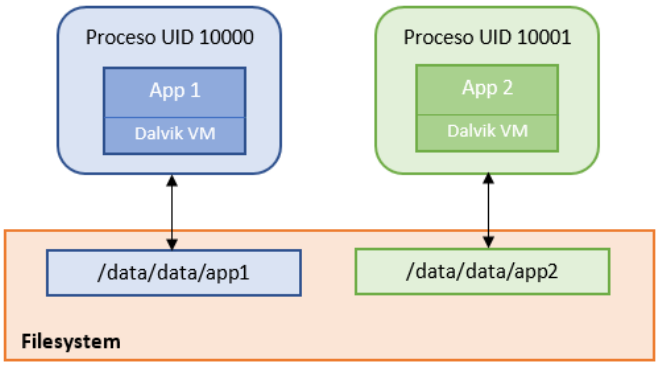


Imagen 7: Sandbox de aplicaciones

Todas las apps se instalan con el nombre del paquete, en la siguiente ruta: /data/data/[package-name]. Este directorio contiene todos los datos de la aplicación y por defecto, solo los procesos que se ejecuten dentro de su sandbox podrán acceder a estos, ya que los permisos de Linux solo autorizan al UID único de la app a acceder a ellos.

Las apps tampoco pueden interactuar con otras apps por defecto y tienen acceso limitado al sistema operativo. Así si una aplicación intenta acceder de forma maliciosa a los datos de otra, el sistema operativo los protegerá si ésta no dispone de los privilegios de acceso necesarios.

Existen varios métodos para hacer accesibles los datos de una app por otra en el caso de que sea necesario, como usar *content providers* o ubicaciones compartidas de almacenamiento externo. No es recomendable en absoluto establecer los datos de una app como accesibles totalmente pues se trata de una mala práctica de seguridad que puede dar lugar a fugas de información y vulnerabilidades. Es por eso que a partir de la versión Android 9, compartir archivos de esta manera está explícitamente deshabilitado para aplicaciones.

8.2 El entorno de pruebas.

A la hora de establecer un entorno de pruebas de auditoria, debemos tener en cuenta que disponemos de las siguientes posibilidades para realizar el análisis dinámico de nuestra aplicación móvil:

- **Usar un emulador**

Para poder recrear el funcionamiento de una app durante su ejecución, una posibilidad de la que se dispone es la de utilizar herramientas o máquinas virtuales que permiten crear un entorno de emulación para nuestro análisis dinámico.

Estas herramientas permiten emular cualquier dispositivo y versión de Android, de manera rápida y potente, facilitándonos el interactuar con la aplicación de la misma manera que se haría desde el propio dispositivo móvil. Además, proporcionan infinidad de funciones para las pruebas de aplicaciones.

Hay sin embargo algunas desventajas en el uso de estos emuladores: en general, las pruebas a través de estos son bastante más lentas llegando incluso a resultar tediosas. Asimismo, no todas las apps pueden analizarse de manera apropiada mediante emuladores, por ejemplo, si ésta funciona en una red móvil específica o usa conectividades como el NFC o el Bluetooth.

Algunas de las herramientas que la guía *Mobile Security Testing* recomienda son:

- **Mobile Security Framework:** MobSF es un entorno multiplataforma de análisis de seguridad de aplicaciones móviles cuyo detalle veremos a continuación en el apartado *Herramientas* del presente documento.

- **AppUse:** máquina virtual desarrollada por *AppSec Labs*, que proporciona una plataforma de pruebas de seguridad de aplicaciones móviles tanto Android como iOS y que incluye algunas herramientas exclusivas y scripts propietarios.

- **Android Studio:** software oficial de Google para el desarrollo y pruebas de las aplicaciones móviles. Entre sus múltiples herramientas incluye un emulador de Android para PCs con Windows, con la posibilidad de emular cualquier dispositivo y versión de Android. Entre sus ventajas, afirman iniciar las aplicaciones más rápido incluso que en el dispositivo real y facilitar la simulación de varias funciones de hardware, como la localización de GPS, la latencia de red y las funciones multitáctiles.

➢ **Usar un dispositivo real**

Disponer de un dispositivo móvil dedicado a las pruebas de auditoría es la alternativa que nos permite llevar a cabo un análisis dinámico de aplicaciones de manera más ágil y en un entorno más realista que el proporcionado por un emulador.

Si se opta por esta opción de probar con un dispositivo real Android, lo recomendado es hacer el *rooting* o rooteado del dispositivo, es decir, modificar el SO para ejecutar comandos como usuario root, tener absoluto control del sistema operativo y saltarse las restricciones del *sandboxing* de aplicaciones, un enfoque del desarrollo software y la administración de aplicaciones que limita los entornos en los que se puede ejecutar el código con el objetivo de mejorar la seguridad.

Se trata de una operación no exenta de inconvenientes pues puede significar la pérdida de la cobertura en algunos tipos de seguros o garantías, que algunas aplicaciones dejen de funcionar (incluso podría sufrir el bloqueo de descargas de

nuevas aplicaciones desde la tienda) o si éste no se realiza correctamente, el dispositivo puede quedar inservible o con riesgos de seguridad adicionales.

9. Herramientas

Para llevar a cabo el análisis de seguridad descrito en los apartados anteriores y de acuerdo con los enfoques estáticos y dinámicos para detección de vulnerabilidades, existen actualmente multitud de herramientas que simplifican y facilitan en gran medida esta tarea, ya que permiten manipular solicitudes y respuestas, decompilar la app, examinar el comportamiento de las apps en funcionamiento, etc.

Las herramientas pueden clasificarse según si sirven de marco de pruebas específico para una plataforma móvil en concreto, ya sea Android o iOS o para ambas; según si su objetivo es cubrir el análisis estático de código, se enfocan al análisis dinámico de la app durante su ejecución o si son capaces de llevar a cabo ambos análisis completos.

La elección de una herramienta u otra dependerá de las particularidades de la app a analizar y del entorno de pruebas disponible.

Se describen a continuación varias de las herramientas más populares para el análisis de seguridad de apps, con el objetivo de tener una completa visión de las posibilidades y funcionalidades que ofrece cada una de dichas herramientas, de modo que podamos realizar una comparativa y elegir aquella que mejor se adapte a nuestras necesidades o cumpla los requisitos de la app de estudio.

Las siguientes herramientas se presentan como entornos de análisis seguridad móvil para Android e iOS indistintamente:

9.1 Mobile Security Framework – MobSF

MobSF [24] es un entorno multiplataforma de análisis de seguridad de aplicaciones móviles con la capacidad de realizar tanto análisis estáticos como dinámicos de manera muy completa e intuitiva a través de su interfaz.

- **Análisis estático**

La herramienta puede ser utilizada para analizar ejecutables de Android (APK), iOS (IPA) o incluso Windows Mobile (APPX), así como código fuente empaquetado en formato ZIP. Desde su interfaz web, tenemos la posibilidad de subir la app a analizar y obtener la distinta información del archivo, como nombre, tamaño, hashes, nombre

del paquete, actividad principal, SDK de compilación, números de actividades, servicios, etc. como veremos en detalle a continuación.

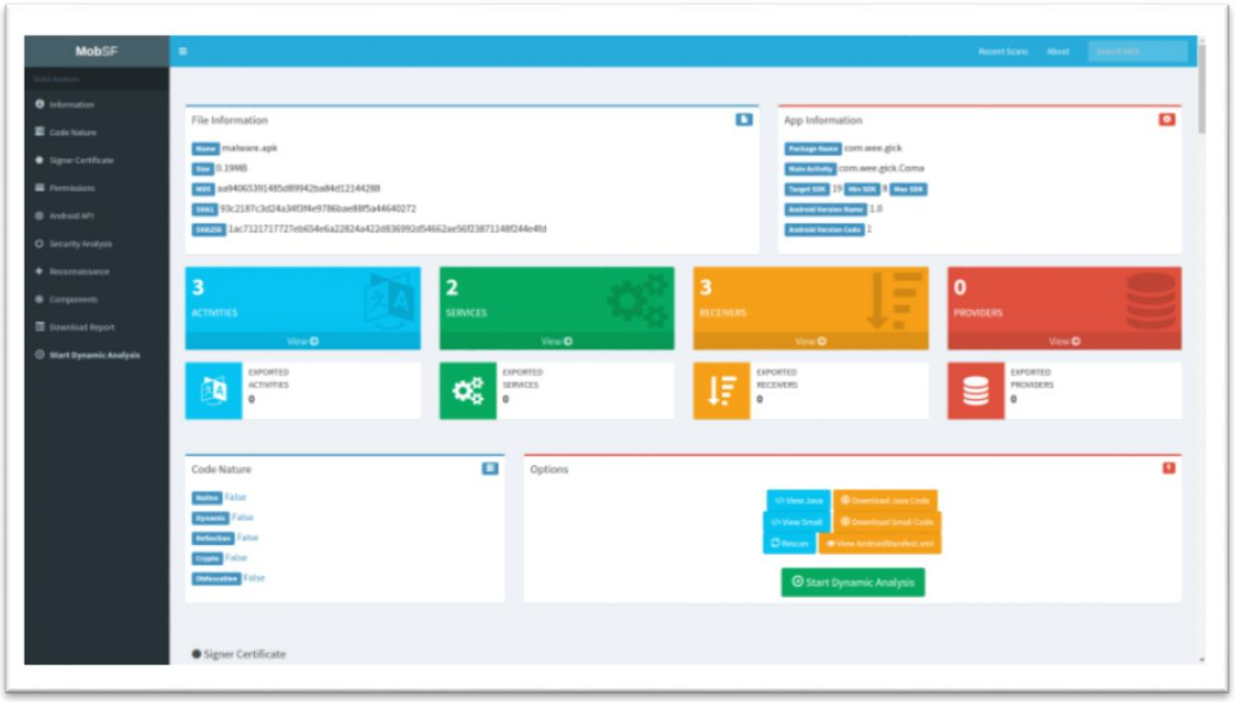


Imagen 8: Mobile Security Framework – Análisis estático

La herramienta también ofrece la posibilidad de visualizar o descargar el código decompilado de forma automática al subir el APK, IPA o APPX a la herramienta.

Los resultados del análisis estático se distribuyen en las siguientes secciones:

- **Información del archivo**. Resumen de las principales características de éste como el nombre, el tamaño, hashes, entre otros.
- **Información de la app**: proporciona información de ésta extraída del Android Manifest, como nombre del paquete y de la clase de la actividad principal, así como otros atributos relativos a la plataforma para la cual fue desarrollada.
- **Elementos vulnerables**: información de las actividades, servicios, proveedores de contenidos, etc. que podrían evidenciar posibles *exploits* de la app.
- **Naturaleza del código**: incluye información del código de la app, si posee funciones de cifrado, ofuscación, etc.
- **Análisis del código decompilado** y del listado de clases recuperadas, así como del archivo Manifest, a partir de la cual se podría ejecutar el análisis dinámico de la app.
- **Información del certificado** y de las posibles vulnerabilidades de éste.
- **Listado de permisos** declarados en el Manifest de la aplicación.

- **Android API**: listado de funcionalidades de la API del sistema que se acceden desde cada una de las clases que componen la aplicación.
- **Extras de seguridad**: proporciona también otras informaciones de utilidad en el análisis de seguridad como actividades, servicios, broadcast receivers y proveedores de contenidos especificados en la aplicación, strings encontradas en del código fuente, etc.

➢ **Análisis dinámico**

Este análisis puede ser configurado para ser ejecutado desde un dispositivo real, utilizando la máquina virtual que la herramienta proporciona o incluso una diferente de la que se disponga, según las necesidades de cada caso.

Una vez preparado el entorno, permite interactuar con la app de forma natural mientras va recopilando toda la información de datos y comunicaciones generadas durante la ejecución.

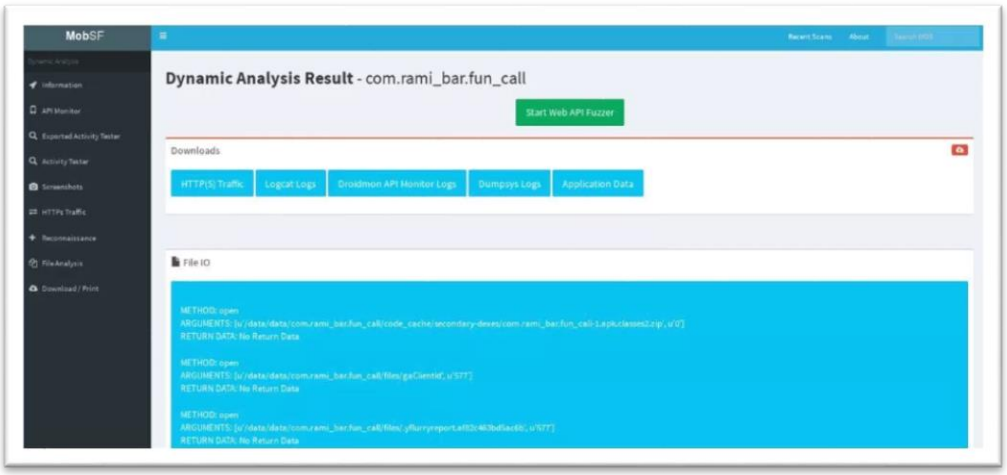


Imagen 9: Mobile Security Framework - Análisis dinámico

El informe de análisis dinámico contendrá información útil de la app como la enumerada a continuación:

- archivos accedidos por la app durante la ejecución, apertura de puertos y gestión de conexiones de red.
- información del dispositivo a la que accedió la app durante la ejecución (como IMEI, número de teléfono y operador, etc.)
- paquetes HTTP capturados durante la ejecución
- direcciones URL procesadas por la API de la plataforma y análisis de seguridad de cada dominio visitado.

- Otra información que podría ser relevante como bases de datos utilizadas por la app o cadenas que podrían relacionarse de direcciones de correo electrónico.

MobSF configurado para su análisis dinámico tiene sin embargo la limitación de no funcionar para todas las versiones de Android en el caso utilizarse la máquina virtual que la herramienta proporciona, siendo la configuración mediante un dispositivo real, rooteado, la única opción disponible en tal caso.

9.2 Inspeckage

Inspeckage [25] es un módulo instalado en Xposed Installer, un *framework* que permite instalar funciones extra por medio de módulos en los dispositivos rooteados. Inspeckage concretamente permite ver en tiempo real los eventos que ocurren en el teléfono y por tanto es de gran utilidad para realizar análisis dinámicos de aplicaciones.

Una vez instalado y activado, nos permite configurar para qué apps del dispositivo nos interesa activar su monitoreo, levantando un servidor en el terminal que puede ser accedido vía adb. Tras seleccionar la aplicación e iniciarla, comenzará el análisis dinámico de su funcionamiento y podremos acceder a los resultados desde el navegador de nuestro dispositivo en la dirección indicada.

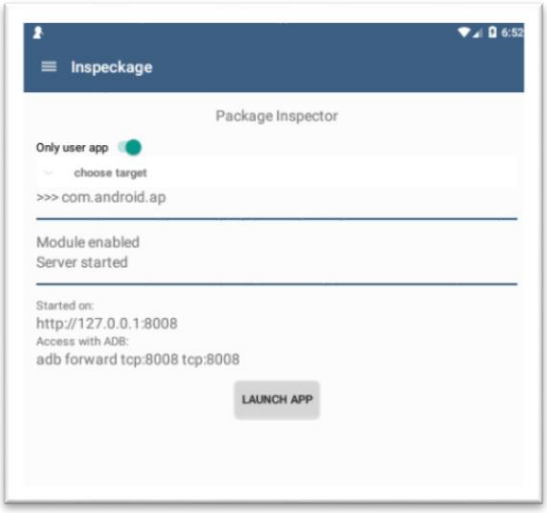


Imagen 10: Inspeckage activo para app seleccionada

En primer lugar, muestra información básica de la app analizada, como el nombre del paquete, localización de directorio de datos, atributos de *backups*, *Tree View* para ver gráficamente los archivos disponibles de la app y acceder a ellos, entre otras.

A continuación, se muestran una serie de pestañas con otra información útil:

- Preferencias compartidas con los cambios efectuados sobre las variables.
- Registro de actividad criptográfica
- Valores a los que se han aplicado funciones Hash
- Consultas SQLite ejecutadas por la aplicación
- Registro de HTTP
- *File system* o archivos con los cuales la aplicación ha interactuado
- Información varia sobre recursos accedidos por los *Content Providers*, etc.

9.3 Zed Attack Proxy de OWASP

Zed Attack Proxy (ZAP) [26] es una herramienta de código libre diseñada para monitorizar la seguridad y encontrar vulnerabilidades en aplicaciones. Forma parte del proyecto OWASP y es actualmente una de las aplicaciones más activas y utilizadas para auditorías de seguridad ya que presenta una serie de ventajas como ser gratuita, multi-plataforma y sencilla de instalar y utilizar tanto para desarrolladores web como para *testers* funcionales.

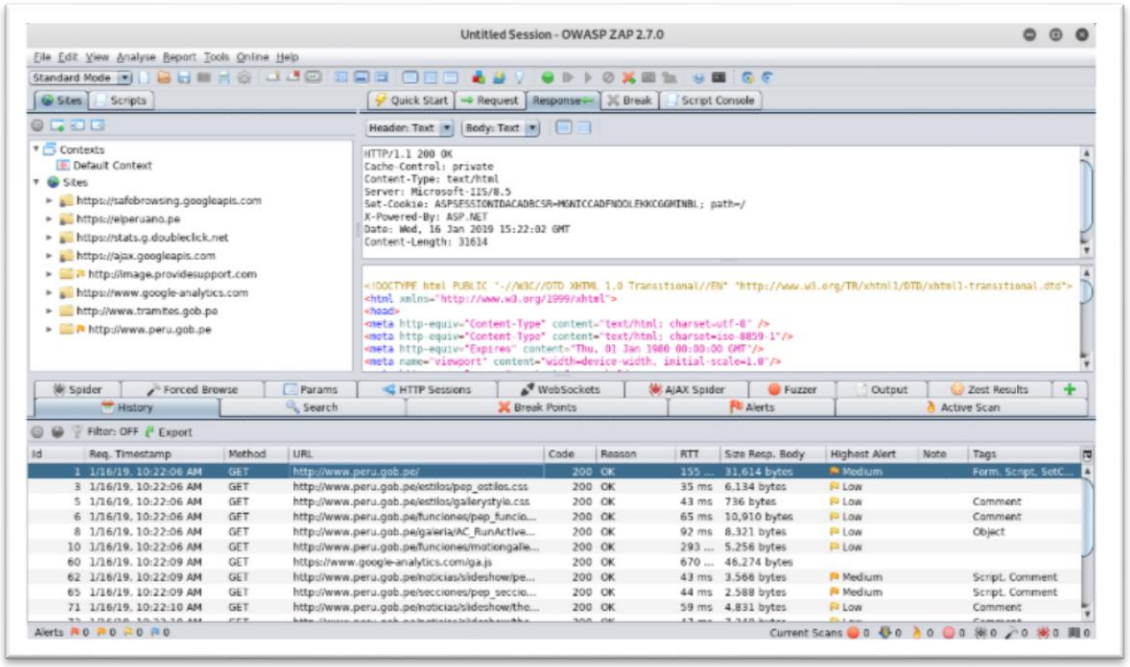


Imagen 11: OWASP ZAP para análisis de tráfico HTTP y detección de vulnerabilidades

Entre las funcionalidades que ofrece, se encuentran las siguientes:

- ZAP puede ser configurado como proxy o man-in-the middle para interceptar y analizar todas las peticiones y respuestas entre cliente y servidor, es decir, entre la app a analizar y su servidor.
- Permite realizar análisis de vulnerabilidades de dos maneras diferentes:

  - Escáner pasivo: se trata del análisis por defecto incorporado por ZAP para todo el flujo HTTP generado por y hacia la aplicación. Como su nombre indica, no modifica de ninguna forma los mensajes de petición y respuesta y su uso es totalmente seguro.
  - Escáner activo: se trata de un análisis en busca de potenciales vulnerabilidades por medio de ataques conocidos, utilizando técnicas como la de *Fuzzing*, en la que se envían datos inválidos o inesperados con el objetivo de detectar las vulnerabilidades. Este análisis ha de ejecutarse solo sobre aquellas aplicaciones para las cuales se disponga de permiso para realizar tareas de pententing.

- Además, permite lanzar ataques simultáneos, utilizar certificados SSL dinámicos, soporta el uso de tarjetas inteligentes, instalación de *plugins* para ampliar las funcionalidades de la herramienta, etc.

Similares a ZAP, podemos encontrar además otras herramientas como Burp Suite, Nikto, Acunetix, w3af, Arachni, Andiparos, HTTP Analyzer, etc.

9.4 Otras herramientas

El listado de herramientas específicas es amplio por lo que se recomienda acceder al Apéndice: Testing Tools de la *Mobile Security Testing Guide* [2] para encontrar una recopilación completa de aplicaciones actualizadas.

Existen además otras herramientas que facilitan el análisis estático manual de una aplicación móvil:

- Dev2jar, para obtener el código de las clases Java a partir del archivo .apk de la aplicación
- Java Decompiler GUI, interfaz gráfica que muestra el Código Java de todos los archivos .class que componen la aplicación.
- APK Extractor, aplicación que permite obtener el archivo .apk de una aplicación móvil para su análisis en caso de no disponerse de él.
- AXMLPrinter2, utilidad que nos permite convertir el archivo Manifest definido para cada aplicación móvil a un formato legible.
- Android Debug Bridge (ADB), herramienta que permite la conexión directa de un PC al dispositivo Android para, mediante comandos, instalar aplicaciones, abrirlas, verificar dispositivos, reiniciarlos, etc.

10. Caso Práctico

10.1 Metodología empleada para la auditoria de seguridad

En este capítulo se va a proceder a detallar la auditoría de seguridad realizada sobre una aplicación móvil de estudio proporcionada para tal efecto, de acuerdo con las metodologías y modelo de seguridad descritas en los apartados anteriores.

Con el objetivo de llevar a cabo una identificación más completa de las posibles vulnerabilidades, se llevará a cabo tanto el análisis estático como el dinámico de la app:

- El análisis estático cubrirá la revisión del código fuente de la aplicación y sus distintos componentes y se realizará tanto de manera manual como automática, a partir de la herramienta MobSF.
- El análisis dinámico cubrirá la evaluación de la aplicación durante su ejecución en tiempo real y se utilizará para ello un dispositivo móvil Android.

El estándar de verificación empleado será el descrito en el apartado 6 del presente documento, de acuerdo con el nivel MASVS-L1 correspondiente con el nivel de seguridad estándar, ya que la aplicación de estudio no maneja datos altamente sensibles que justifiquen la adopción del nivel de seguridad de Defensa en Profundidad. También queda fuera del alcance del presente proyecto el análisis correspondiente al nivel MASVS-R o nivel de resistencia contra la ingeniería inversa y la manipulación, recomendados para aquellas aplicaciones que manejan datos altamente confidenciales o para proteger la propiedad intelectual.

El nivel estándar por su parte contiene los requerimientos genéricos de seguridad recomendados para todas las aplicaciones móviles en general, y nos va a proporcionar, por tanto, las mejores prácticas relacionadas con la calidad del código y la adecuada gestión de datos sensibles.

Previamente a los análisis estático y dinámico se llevará a cabo un proceso de recolección de datos inicial de la app, teniendo en cuenta la funcionalidad de ésta, autenticación requerida, permisos solicitados durante la instalación, componentes hardware (por ej. GSP, Bluetooth), redes (móvil, Wifi) y otras aplicaciones, servicios o datos del dispositivo con los que interactúa (llamadas, contactos), así como la clase de operaciones permite realizar. Con este reconocimiento previo y manejo de la aplicación se busca identificar a priori, las áreas más críticas.

Al final del presente apartado se proporcionará un informe de resultados con los hallazgos más destacables procedentes de los distintos análisis realizados.

10.2 Recopilación de información de la aplicación de estudio.

La aplicación selecionada para llevar a cabo la auditoría de seguridad es una app de gestión de notarías que facilita una serie de servicios telemáticos para gestiones habituales en el ambito notarial como escrituras, expedientes, pólizas, legitimaciones, facturación, tramitación, tesorería, contabilidad, informes, etc.

Para su análisis, se dispone de su .APK el cual instalamos en un dispositivo Android físico rooteado. Al abrir la app, en primer lugar se presenta un formulario solicitando unas credenciales de acceso: Usuario y Contraseña. Dichas credenciales han de ser facilitadas previamente para el uso de la app ya que ésta no proporciona ninguna función de registro inicial:

Imagen 12: Pantalla de inicio de la app de estudio

Si no se introducen credenciales de acceso, las únicas opciones disponibles son las visibles en la parte baja de la pantalla principal para información de la app y los servicios incluidos, sección Noticias, y buscador de Notarios a partir de un formulario o sobre un mapa de localización.

Si la combinación Usuario/Contraseña no es adecuada, la app muestra el siguiente mensaje de error:

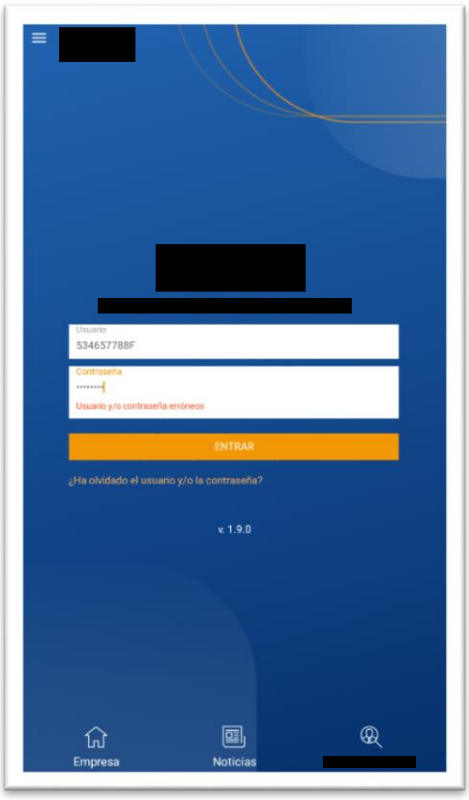


Imagen 13: Usuario y/o contraseña errónea

Si alguno de los dos campos requeridos para identificación no es completado, la app muestra el siguiente mensaje de error:

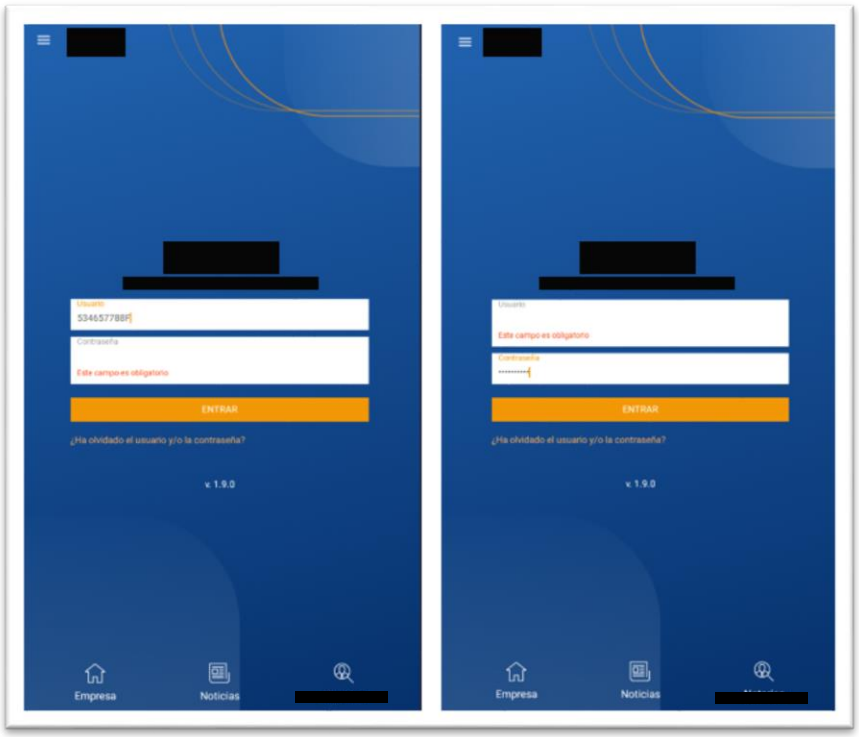


Imagen 14: Campos obligatorios de identificación

Finalmente, si introducimos las credenciales válidas, la app nos permite acceder a todas sus funcionalidades:

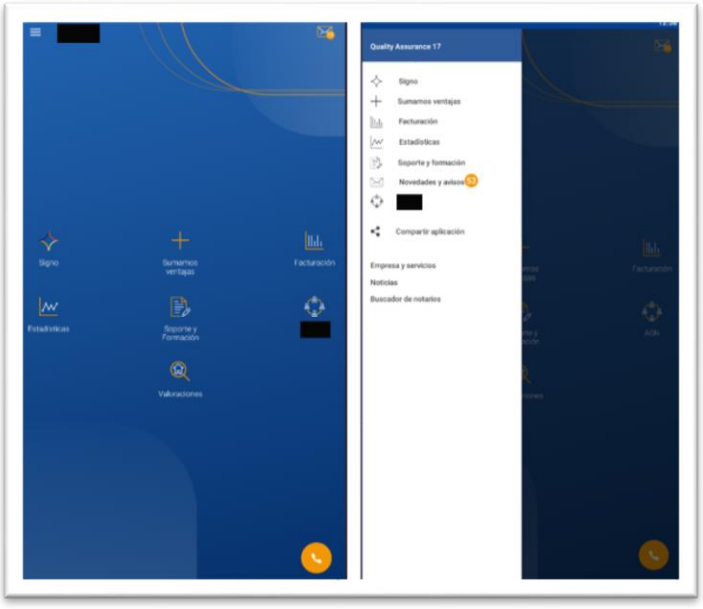


Imagen 15: Pantalla principal de la app

La aplicación solicita además permisos para interactuar con:

- Cámara
- Contactos
- Ubicación (para acceder a la localización precisa a través de GPS o red)
- Almacenamiento interno (lectura y modificación en la tarjeta SD)
- Acceso a la red móvil
- Acceso a redes Wifi.

10.3 Preparación del entorno de auditoría: herramientas necesarias.

Se definen a continuación cada una de las herramientas empleadas para la realización de los análisis estático y dinámico de la aplicación, así como la configuración requerida para cada una de ellas.

10.3.1 Dev2jar

Para poder obtener el código de las clases Java de la aplicación a partir del archivo .apk proporcionado, descargamos e instalamos dex2jar v2.0 desde [32].

Una vez instalado, se ejecuta el siguiente comando desde la ruta de descarga de dev2jar para cada archivo classes.dex disponible:

```
d2j-dex2jar.bat classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar
```

10.3.2 Java Decompiler-GUI

Usamos la herramienta JD-GUI v1.4.1, para visualizar de manera gráfica el código fuente de los archivos .class que conforman nuestra app, obtenidos tras ejecutar Dev2jar:

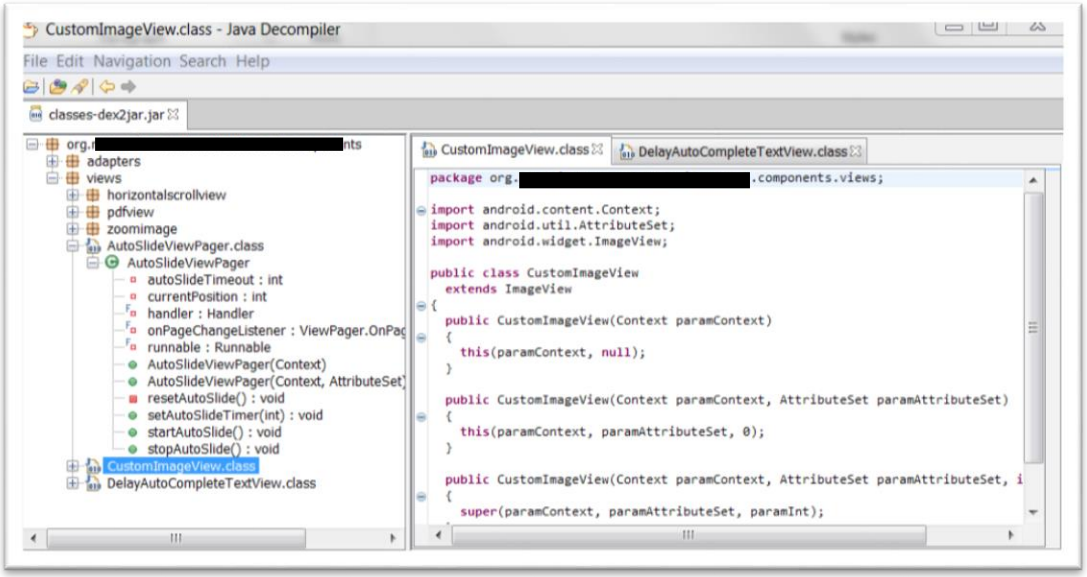


Imagen 16: Vista de la herramienta JD-GUI

10.3.3 AXMLPrinter2

Descargamos esta utilidad que nos permite convertir el archivo AndroidManifest.xml definido para nuestra app, a un formato legible, desde [32].

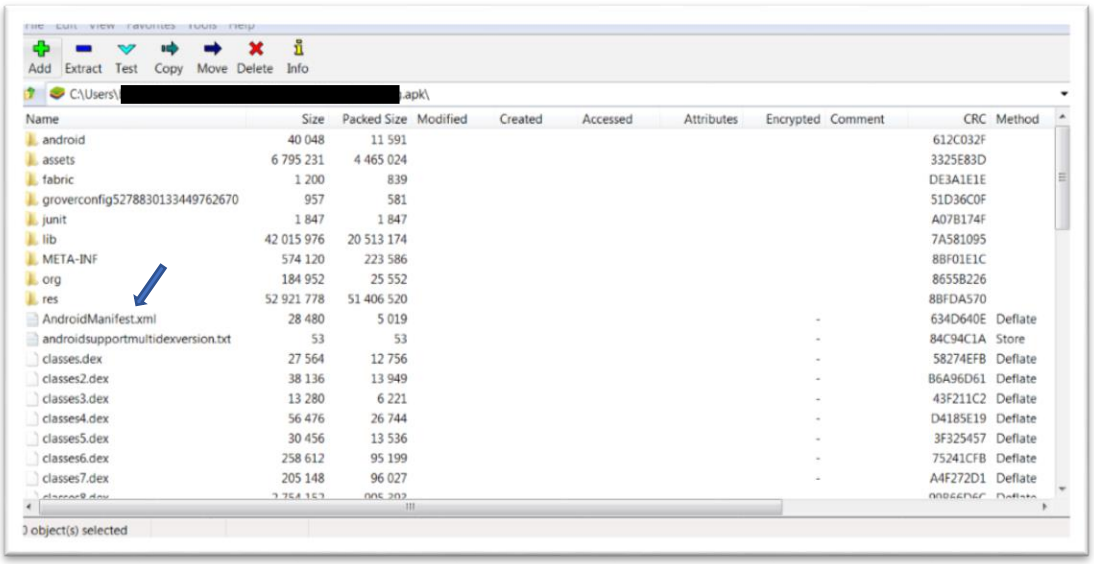


Imagen 17: Vista de las clases extraídas del .apk de la app

Ejecutamos el siguiente comando desde la ruta de descarga de AXMLPrinter2.jar y obtenemos la conversión del *AndroidManifest*:

```
java -jar AXMLPrinter2.jar [Ruta App]\AndroidManifest.xml >
[Ruta Destino]\AndroidManifest_decoded.xml
```

Una vez disponemos del *AndroidManifest* decodificado, verificamos varios de los requerimientos listados por MASVS durante el análisis estático manual de la aplicación como:

- Permisos: comprobamos los permisos definidos para la app y si coinciden con los solicitados por la app durante su instalación:

```
android:name="android.permission.INTERNET"
android:name="android.permission.ACCESS_FINE_LOCATION"
android:name="android.permission.ACCESS_NETWORK_STATE"
android:name="android.permission.WRITE_EXTERNAL_STORAGE"
android:name="android.permission.USE_FINGERPRINT"
android:name="android.permission.WRITE_CONTACTS"
android:name="android.permission.KILL_BACKGROUND_PROCESSES"
android:name="android.permission.CAMERA"
android:name="android.permission.FLASHLIGHT"
android:name="android.permission.VIBRATE"
```

Imagen 18: Permisos solicitados por la app

- Permisos: verificación de si la aplicación tiene los permisos adecuados y si el acceso a SMS, contactos y ubicación son realmente necesarios.
- Definición de permiso de acceso mediante huella dactilar.
- Valor del atributo *allow-backup* para permitir a través de ADB la realización o no de backups de la aplicación.
- Determinar si la app puede ser depurada o no por la existencia del flag *android:debuggable,* presente para versiones en desarrollo.
- Existencia de servicios, proveedores de contenidos, *intents*, etc. no protegidos.
- Definición de permisos personalizados de acceso a componentes expuestos para determinar si la información almacenada sensible queda expuesta por medio de mecanismos IPC.
- Detección del uso de la configuración de Seguridad de Red proporcionada por Android para versiones 7.0 y superiores.

- Detección de la definición de esquemas URL personalizados.
- Limitación a un nivel determinado de API, por medio del parámetro: *android:minSdkVersion*.

10.3.4 MobSF para análisis estático automatizado

La herramienta empleada para llevar a cabo el análisis estático automatizado de la aplicación es MobSF [24] [30], descrito en el apartado 9.1 Mobile Security Framework – MobSF.

Los requisitos y detalle de la instalación se describen en el *Anexo B. Instalación de MobSF* Análisis estático.

Esta herramienta nos permite obtener un análisis de la app de manera muy intuitiva: basta con arrastrar el APK de la aplicación a la ventana del navegador desde la que podemos acceder a la herramienta, una vez tenemos arrancado el servidor, para que de manera automática se genere el análisis cuyos resultados detallamos en el apartado *Detalle del Análisis estático de la app*.

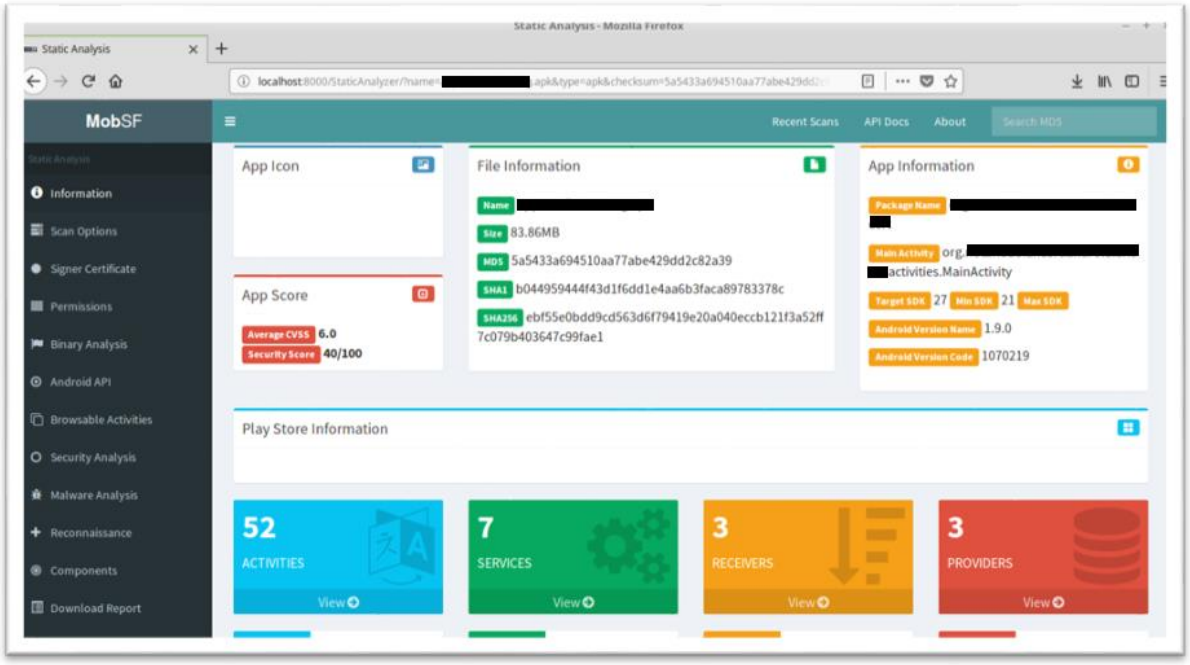


Imagen 19: Resultado del análisis estático de la app

El resultado del análisis también nos proporciona acceso desde su interfaz a las clases java de la aplicación, así como al archivo *manifest* de ésta.

10.3.5 Herramientas para análisis dinámico

- **Dispositivo Android**

Para el análisis dinámico de la aplicación se dispone de un dispositivo físico Android Bq Aquaris X Pro con versión Android 8.1.0 (Oreo, API 27) al que se le realizó un rooteo previo y donde se ha instalado el .APK proporcionado de la aplicación.

El dispositivo también tiene instalada la herramienta Xposed Installer versión 3.1.5.

- **MobSF**

Una de las herramientas escogidas para llevar a cabo el análisis dinámico automatizado de la aplicación es de nuevo MobSF [24] [30], descrito en el apartado 9.1 Mobile Security Framework – MobSF y que ha de configurarse de manera específica para tal análisis.

Los requisitos y detalle de la instalación para análisis dinámico se describen en el Anexo B. Instalación de MobSF, Análisis dinámico.

- **Módulo Inspeckage**

Se ejecutará un análisis mediante el módulo Inspeckage de Xposed Installer ya que también nos va a resultar de gran utilidad para ver en tiempo real los eventos que ocurren en el dispositivo.

Los requisitos y pasos seguidos para la instalación de Inspeckage se describen en el Anexo C. Instalación de Inspeckage.

- **OWASP ZAP Proxy**

Se complementará el estudio con el análisis del flujo de peticiones y respuestas HTTP de la aplicación a través de la herramienta ZAP (Zed Attack Proxy) proporcionada por OWASP.

Los pasos necesarios para su instalación y configuración como proxy se detallan en el Anexo D. Instalación de OWASP ZAP.

10.4 Detalle del Análisis estático automatizado de la app.

De acuerdo con el análisis estático automático de nuestra app de estudio ejecutado mediante MobSF, podemos destacar los siguientes resultados.

Se ha diseñado para ejecutarse en dispositivos con una versión de Android mínima de API 21 (Android 5.0) y objetiva de API 27 (Android 8.1). Está formada por 52 Actividades, 7 servicios de los cuales 2 están exportados, 3 receptores de los cuales 2 están asimismo exportados y 3 proveedores de contenidos.

Imagen 20: Detalle del resultado del análisis estático de la app

- **PERMISOS**

Se detectan los siguientes permisos peligrosos:

- android.permission.WRITE_CONTACTS
- com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.CAMERA
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.INTERNET
- com.google.android.c2dm.permission.RECEIVE
- android.permission.ACCESS_FINE_LOCATION
- android.permission.WAKE_LOCK

- **ANALISIS DEL ARCHIVO MANIFEST**

Se detectan las siguientes vulnerabilidades en el archivo Manifest de la app:

Vulnerabilidad	Severidad
Debug habilitado para la app [android:debuggable=true]	**Alta**
Puede realizarse *back up* de los datos de la app [android:allowBackup=true]	**Media**

TaskAffinity activado para una Actividad: (SignatureEnrollActivity)	**Alta**
TaskAffinity activado para una Actividad: (org.xxxx.xxxx.android.xxxx.happ.scenes.signature.recover.SignatureRecoverActivity)	**Alta**
TaskAffinity activado para una Actividad: (org.xxxx.xxxx.android.xxxx.happ.scenes.signature.config.SignatureConfigChangePinActivity)	**Alta**
TaskAffinity activado para una Actividad: (org.xxxxx.xxxxx.android.xxxxxx.happ.scenes.signature.config.SignatureConfigCertificateRecoverActivity)	**Alta**
TaskAffinity activado para una Actividad: (org.xxxxx.xxxxx.android.xxxxx.happ.scenes.signature.config.SignatureConfigCertificateRecoverPinActivity)	**Alta**
Servicio no protegido (FirebaseMessagingService) [android:exported=true]	**Alta**
Receptor Broadcast protegido por un permiso cuyo nivel de protección ha de verificarse: (AppMeasurementInstallReferrerReceiver) Permiso: android.permission.INSTALL_PACKAGES [android:exported=true]	**Alta**
Receptor Broadcast protegido por un permiso cuyo nivel de protección ha de verificarse: (FirebaseInstanceIdReceiver) Permiso: com.google.android.c2dm.permission.SEND [android:exported=true]	**Alta**
Servicio no protegido (com.google.firebase.iid.FirebaseInstanceIdService) [android:exported=true]	**Alta**

- La depuración se encuentra habilitada para la aplicación lo cual permite acceder al asistente de depuración de depuración de clases.

- Se encuentra habilitada la posibilidad de realizar back up de los datos de la app lo cual permite realizar *backups* vía adb así como copiar los datos fuera dispositivo mediante depuración por USB.

- Se detectan varias actividades con *taskAffinity* activado, por lo que otras aplicaciones podrían leer los *Intents* enviados a Actividades pertenecientes a otra tarea.

- Se detectan 2 servicios no protegidos para ser compartidos con otras aplicaciones y por lo tanto, quedan accesible a cualquier otra aplicación en el dispositivo.

- Se detectan 2 receptores *broadcast* protegidos por un permiso que no está definido en la aplicación para ser compartidos con otras aplicaciones y por lo tanto, quedan accesible a cualquier otra aplicación en el dispositivo. Cualquier aplicación maliciosa podría solicitar y obtener el permiso para interactuar con el componente, por lo que se recomienda configurarlo a nivel firma para que solo las solicitudes firmadas con el mismo certificado puedan obtener el permiso.

➢ ANALISIS DE CÓDIGO

El análisis detecta los siguientes problemas en relación con algunos de los archivos .java que componen la app:

Problema	Severidad	CVSS	CWE
La app almacena en logs información sensible	Info	7.5	CWE-532
La app usa Java Hash Code, una función débil que no debería emplearse en implementaciones criptográficas de seguridad	High	4.3	CWE-327
La app emplea un generador de números aleatorios no seguro	High	7.5	CWE-330
Algunos archivos contienen información sensible *harcodeada* como nombres de usuario, *passwords*, claves, etc.	High	7.4	CWE-312
La app puede leer y escribir en el almacenamiento externo por lo que cualquier dato escrito podría ser asimismo leído por otras apps.	High	5.5	CWE-276
La app crea archivos temporales. La información sensible no debe ser conservada en tales archivos.	High	5.5	CWE-276
La app podría disponer de capacidades de detección de root.	Secure	0	
Revelación de direcciones IP	Warning	4.3	CWE-200

➢ ANALISIS DE ARCHIVOS

El análisis detecta una serie de archivos de certificados o claves *hardcodeadas* en la app. Para consultar la lista completa de archivos, el resultado detallado se adjunta en el Anexo E. Informes del presente documento.

- **PUNTUACIÓN DE LA APP**

No se detectan problemas en relación con el certificado (firmado utilizando el algoritmo SHA256 para el cual no se conocen hasta el momento colisiones), los emails o el chequeo de *malware*.

Con todo ello, la app recibe una puntuación media CVSS de **6.0** y de Seguridad de **40/100**.

El reporte completo en PDF obtenido desde la herramienta, con el resultado detallado del análisis estático de la aplicación se adjunta para su consulta en el Anexo E. Informes: PDF de resultados de análisis estático automatizado con MobSF.

10.5 Detalle del Análisis dinámico de la app.

10.5.1 Análisis automatizado mediante MobSF

A pesar de disponerse de un entorno de análisis dinámico configurado de acuerdo con lo especificado en el Anexo B. Instalación de MobSF, el análisis de la aplicación elegida no puede completarse.

- La máquina virtual proporcionada por MobSF para tal efecto no garantiza su funcionamiento para todas las aplicaciones por tratarse de una versión 4.2 de Android. Se comprueba que efectivamente, aunque sí funciona para otras, no realiza el análisis de nuestra app que se ha diseñado para ejecutarse en dispositivos con una versión de Android mínima de API 21 (Android 5.0).

- La solución de configuración de MobSF para realizar análisis dinámicos sobre dispositivos rooteados como el que disponemos, a pesar de solucionar aparentemente el problema de versiones mencionado anteriormente, tampoco nos permite finalizar con éxito el análisis por incompatibilidades en los distintos componentes instalados.

10.5.2 Análisis mediante Inspeckage

De acuerdo con lo especificado en el apartado 10.3.5 Herramientas para análisis dinámico, se realiza un análisis dinámico utilizando el módulo *Inspeckage* de la herramienta Xposed Installer.

Una vez arrancada la aplicación desde Inspeckage para iniciar el análisis, accedemos al navegador en el puerto 8000 para consultar el resultado en tiempo real de las distintas acciones realizadas sobre la app.

En primer lugar, obtenemos información general de la aplicación, donde se detalla su nombre, versión, nombre del paquete, UID, GID, la autorización para realizar backups, etc. como puede verse en la imagen a continuación:

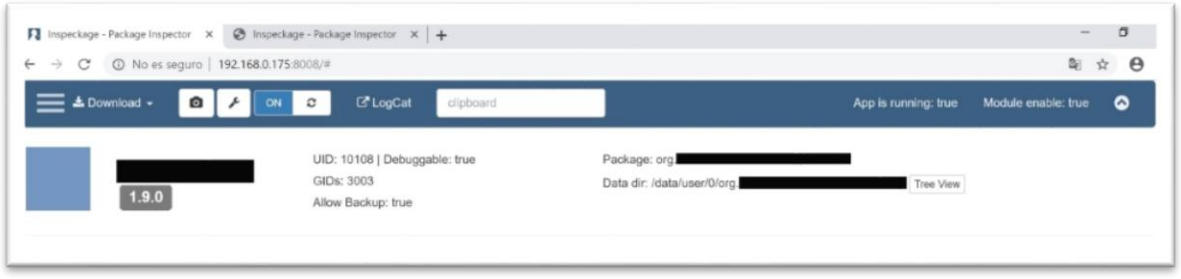


Imagen 21: Información general de la app mostrada por Inspeckage

También nos ofrece la posibilidad de acceder desde TreeView, al almacenamiento interno de la aplicación:

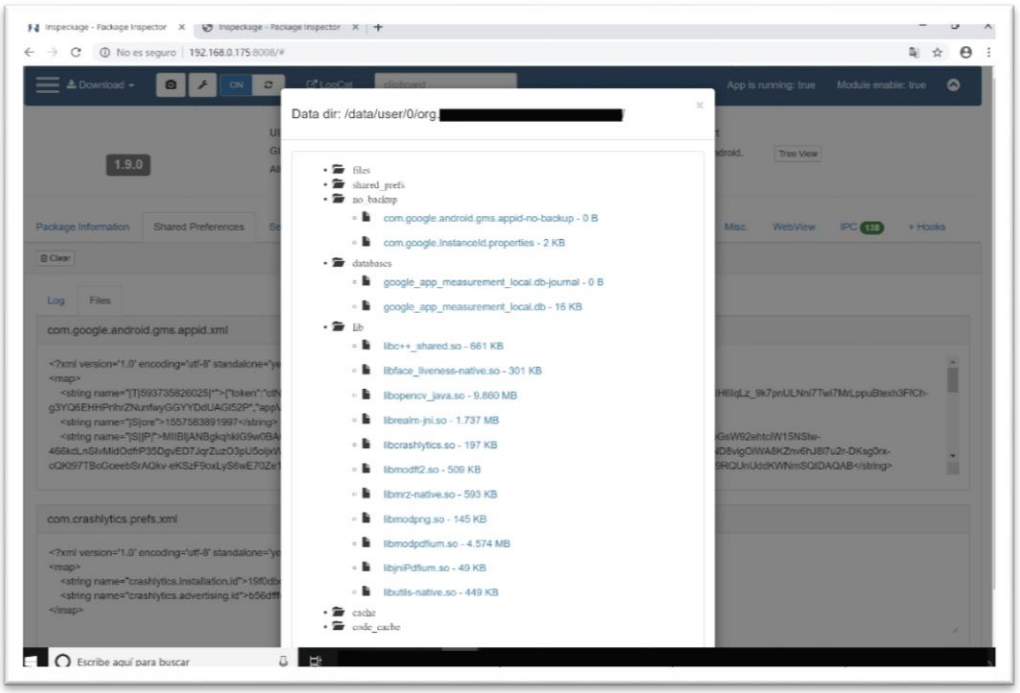


Imagen 22: Tree View de la app mostrado por Inspeckage

La primera pestaña del cuerpo del informe, *Package Information*, nos incluye la lista de componentes usados por la aplicación, en concreto:

- 1 actividad exportada
- 51 actividades no exportadas
- 3 Proveedores de contenidos no exportados
- 2 Servicios exportados
- 6 servicios no exportados
- 2 Broadcast receivers exportados
- 1 Broadcast receivers no exportado

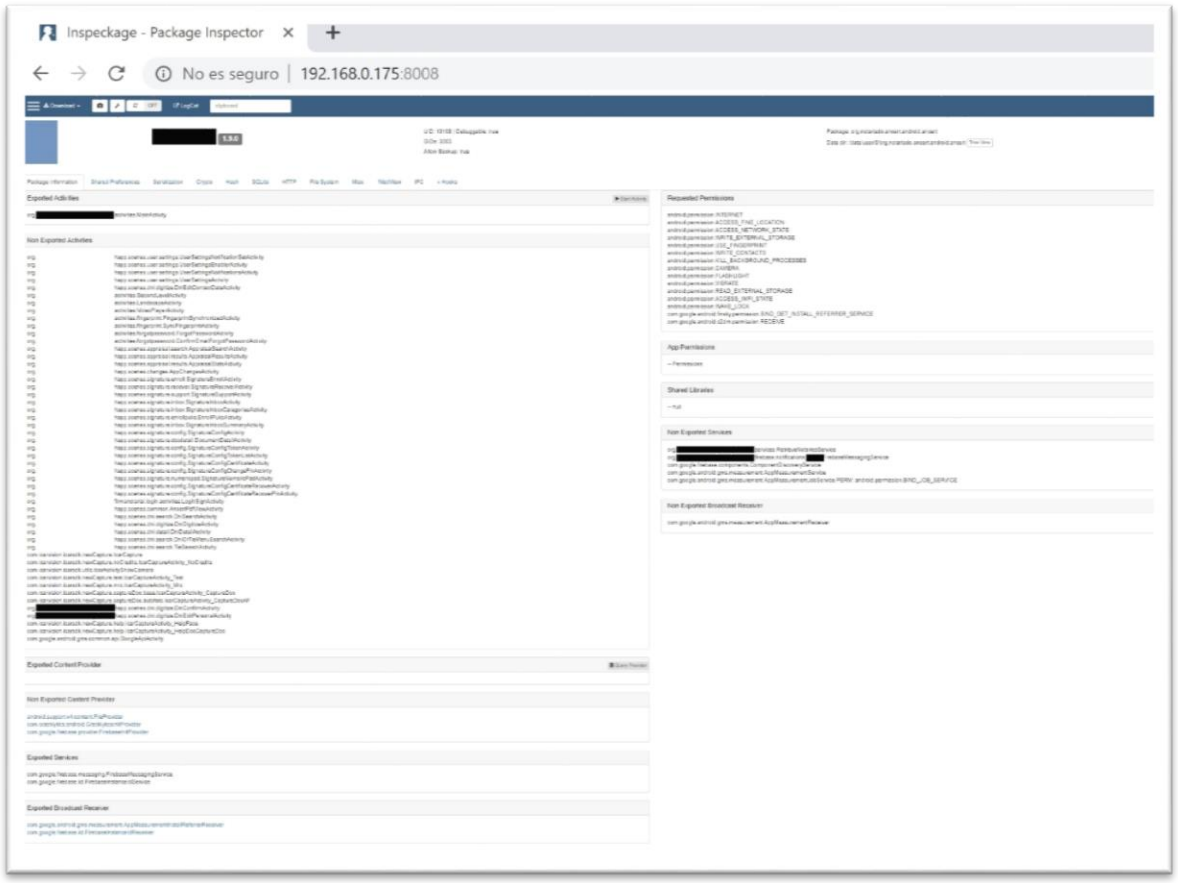


Imagen 23: Package Information mostrado por Inspeckage

En esta pestaña desde el apartado de Actividades exportadas, botón *Start Activity*, podemos además forzar el lanzamiento de algunas actividades a las cuales no tenemos acceso desde la app, como puede verse en las capturas a continuación:

A pesar de que muchas de ellas nos permiten realizar operaciones, se comprueba que ninguna llega a completarse ni nos permite realizar modificaciones indebidas.

A continuación, vemos que se va mostrando la siguiente información a medida que vamos navegando por las distintas opciones disponibles de la app:

- Shared Preferences:

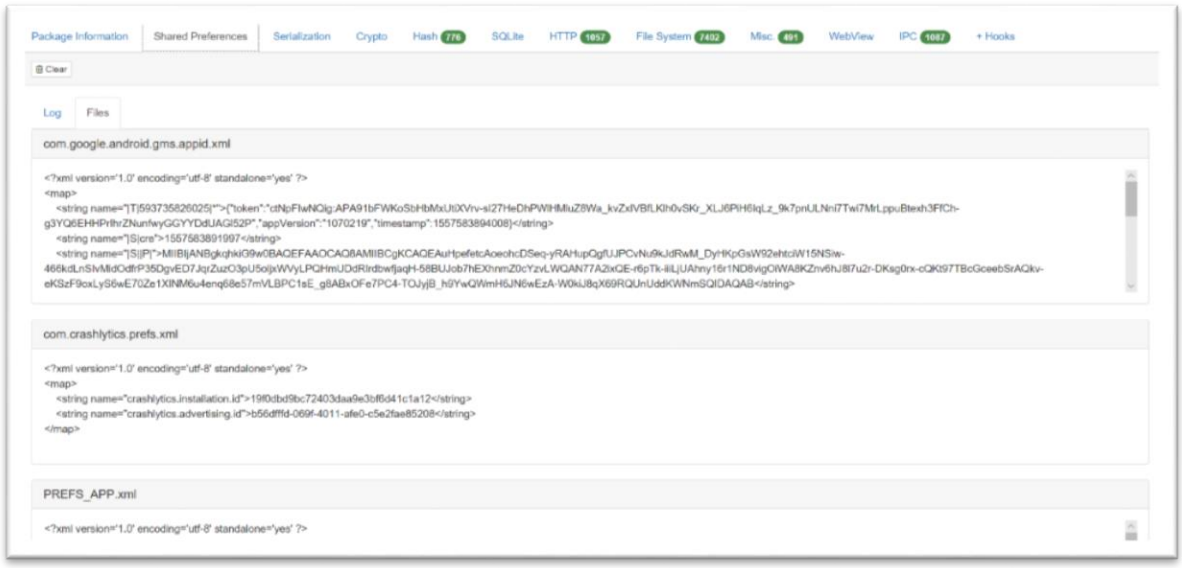


Imagen 24: Shared Preferences mostradas por Inspeckage

- Crypto: con el registro de la actividad criptográfica que ha realizado la aplicación, incluyéndose los algoritmos de cifrado, las claves y la información cifrada:

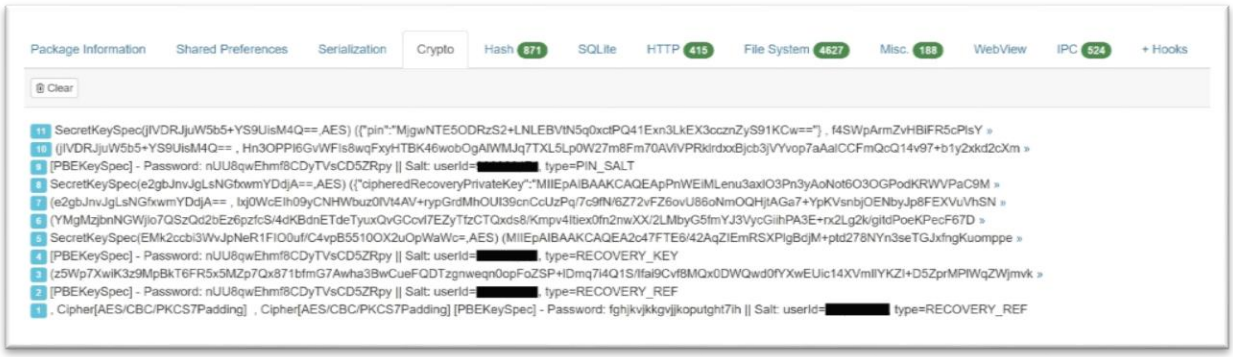


Imagen 25: Crypto mostrado por Inspeckage

- Hash: que incluye los valores a los cuales se les ha aplicado alguna función de hash, junto al tipo de función correspondiente:

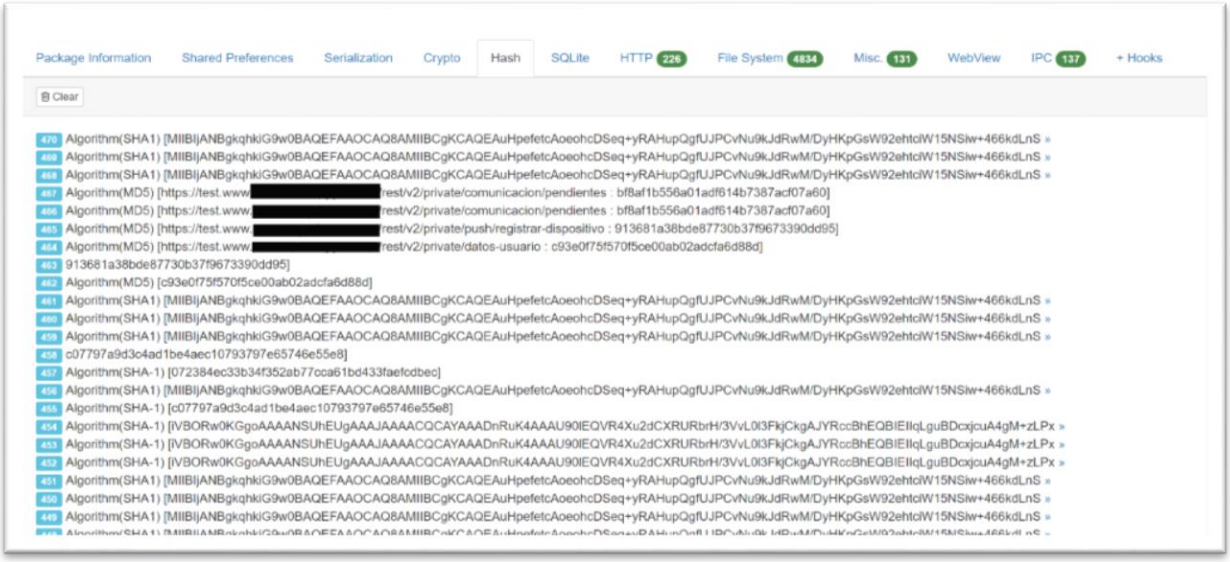


Imagen 26: Hash mostrado por Inspeckage

- SQL Lite: en esta sección Podemos ver si hay consultas SQLite ejecutadas por la aplicación.

- HTTP: que incluye las conexiones HTTP de la aplicación:

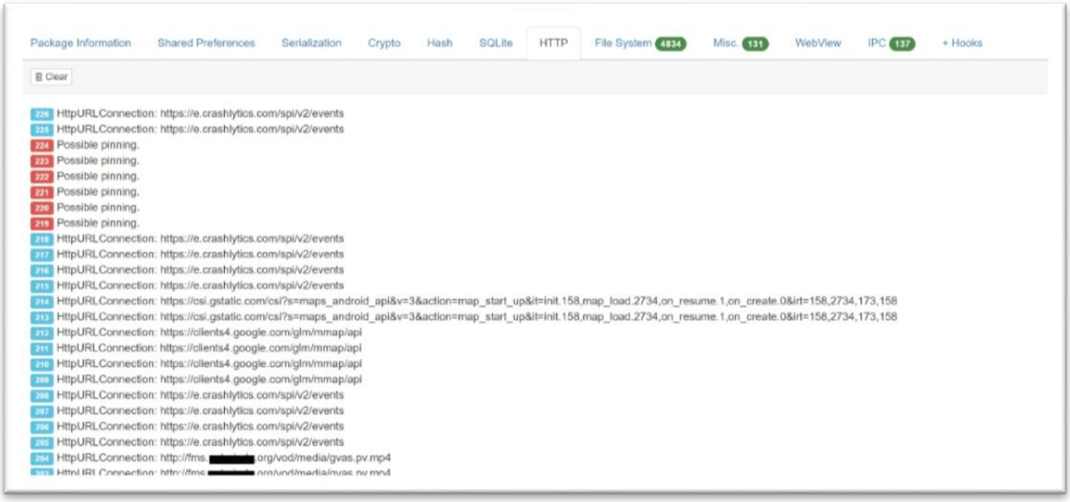


Imagen 27: Flujo HTTP mostrado por Inspeckage

- File System: listado de los archivos con los cuales la aplicación ha interactuado de alguna manera y que permite detectar si ésta crea nuevos archivos o los descarga desde alguna fuente no oficial.

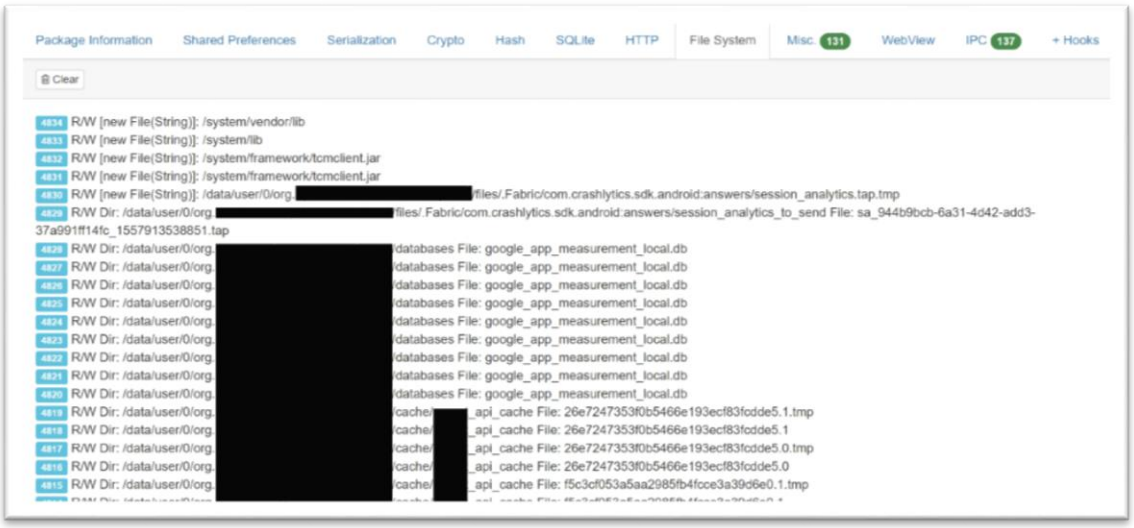


Imagen 28: Listado de FileSystem mostrado por Inspeckage

- Miscellanous: como portapapeles, URL.Parse(), etc.

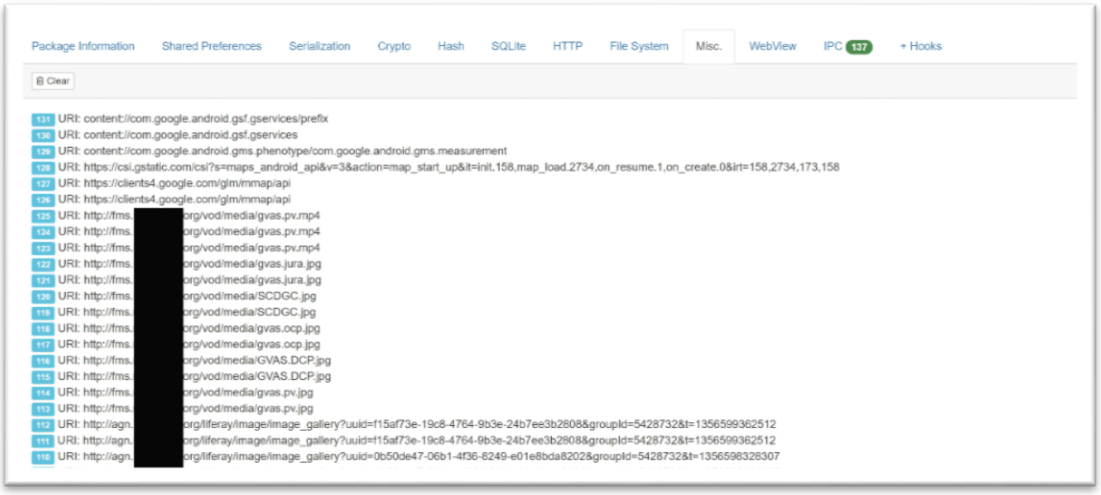


Imagen 29: Listado *Miscellaneous* mostrado por Inspeckage

- WebView: identifica si la aplicación usa WebView *addJavaScriptInterface* como puente para interactuar con Java por medio de JavaScript.

- IPC: listado de los *intents* generados por la aplicación:

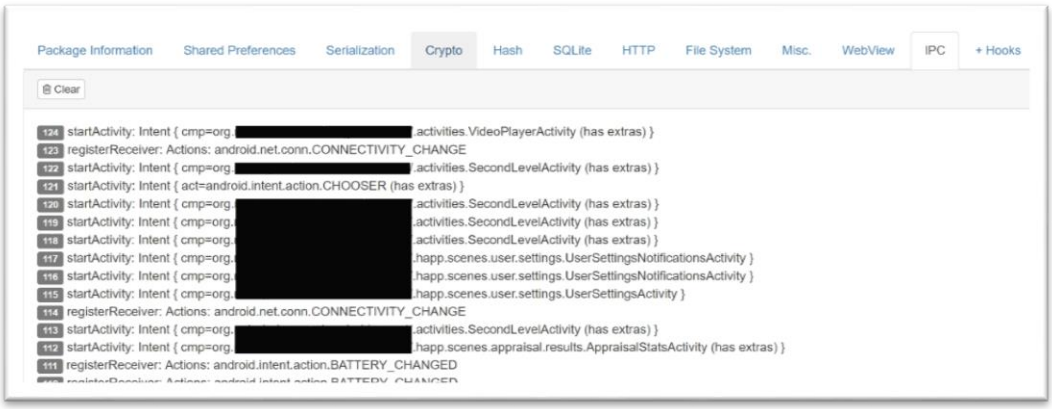


Imagen 30: Listado de IPC mostrado por Inspeckage

- Hooks: ofrece una interfaz gráfica para generar *hooks* personalizados indicando el método a intervenir, modificando los parámetros que recibe o modificando el valor devuelto por éste.

10.5.3 Análisis mediante OWASP ZAP

Tal y como se indica en el apartado 10.3.5 Herramientas para análisis dinámico, se realiza un análisis del flujo de peticiones y respuestas HTTP de la aplicación a través de la herramienta ZAP (Zed Attack Proxy) proporcionada por OWASP.

Del proceso de configuración de ZAP para analizar la aplicación se obtiene la siguiente información relevante:

- En primer lugar, nos encontramos con que la app emplea conexiones seguras HTTPS en todas sus comunicaciones por lo que tenemos que instalar en el dispositivo el *Certificado de confianza* generado por ZAP para poder visualizar dichas conexiones protegidas.

- En segundo lugar, comprobamos que la aplicación implementa el denominado *Certificate* o *SSL Pinning*, proceso en el cual se verifica no solo que el certificado enviado por el servidor sea válido, sino también que sea el certificado del servidor correcto para mejorar notablemente la seguridad.

  Debemos configurar algún método adicional que nos permita evadir el *SSL Pinning*, como el módulo *JustTrustMe* de *Xposed Installer* (también especificado en Anexo D):

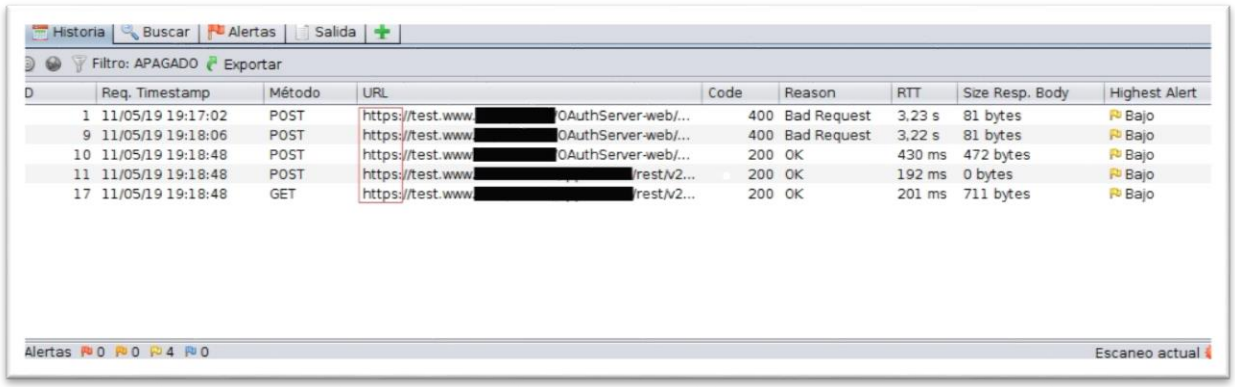


Imagen 31: Tráfico HTTPS generado por la aplicación

Una vez que tenemos ZAP configurado adecuadamente para que el dispositivo pueda utilizarlo como proxy, abrimos la app a analizar y comenzamos en primer lugar, por analizar el procedimiento de login:

1. Insertamos el usuario y la contraseña correctos proporcionados para realizar la auditoría, comprobando mediante ZAP que podemos visualizar en claro todos los datos enviados (aunque ocultados en la captura):

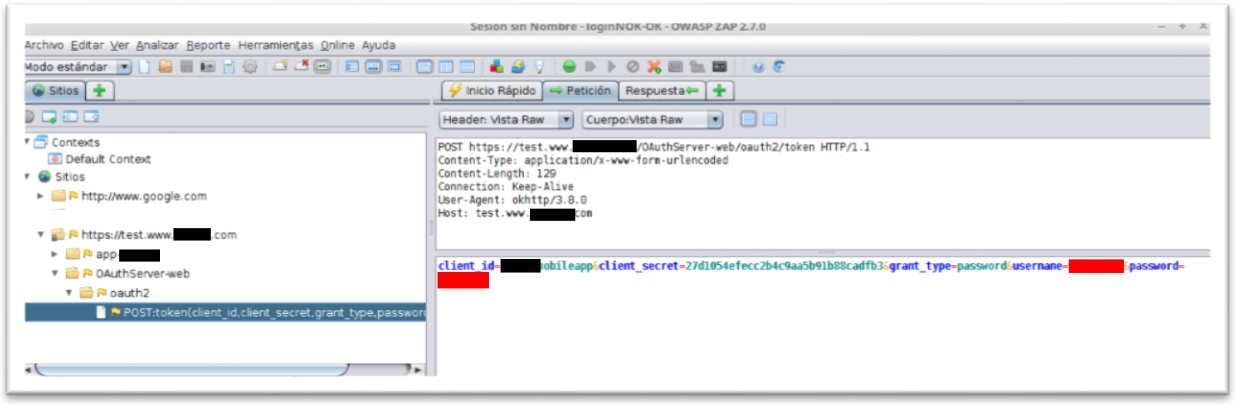


Imagen 32: Información de Login visualizada en ZAP

Cuya respuesta devuelve la siguiente información:

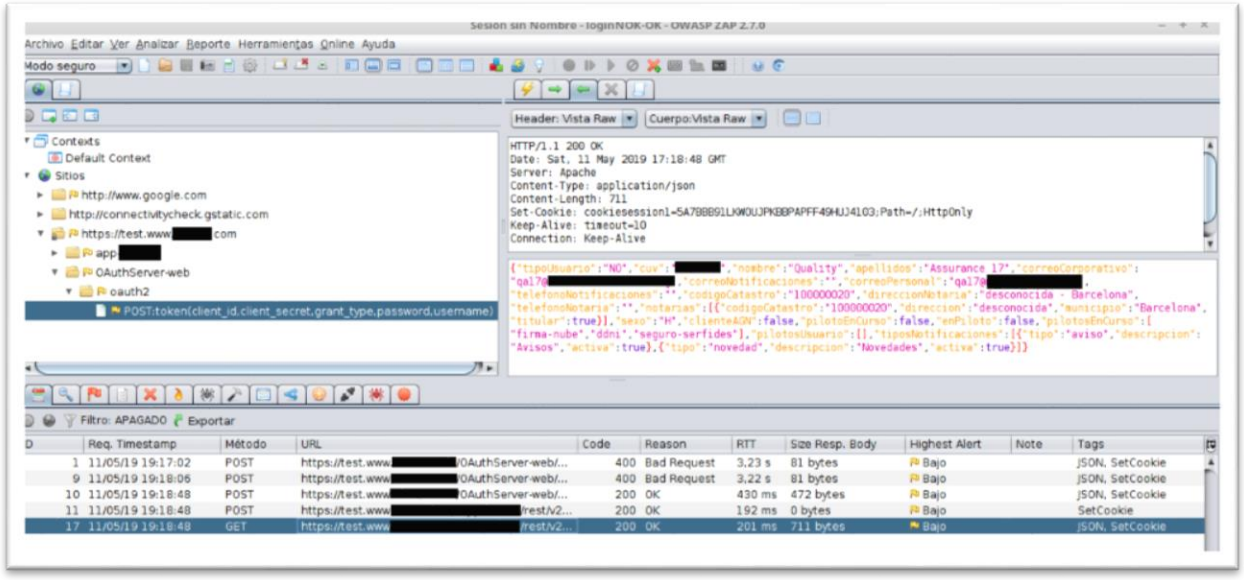


Imagen 33: Respuesta correcta de Login

2. Insertamos una combinación usuario/contraseña inválido para comprobar que no tenemos acceso a la aplicación y de nuevo mediante ZAP comprobamos que podemos visualizar perfectamente los datos:

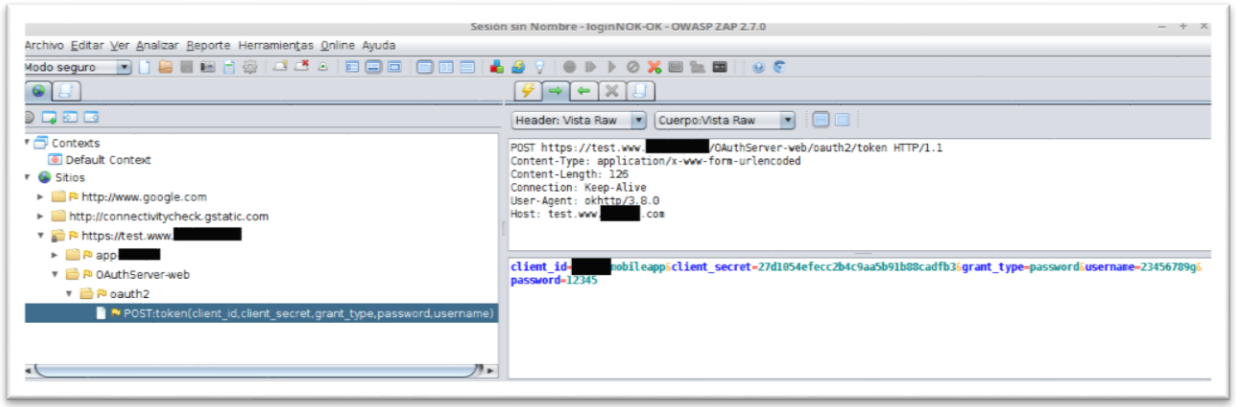


Imagen 34: Información de Login incorrecto visualizada en ZAP

Cuya respuesta devuelve la siguiente información:

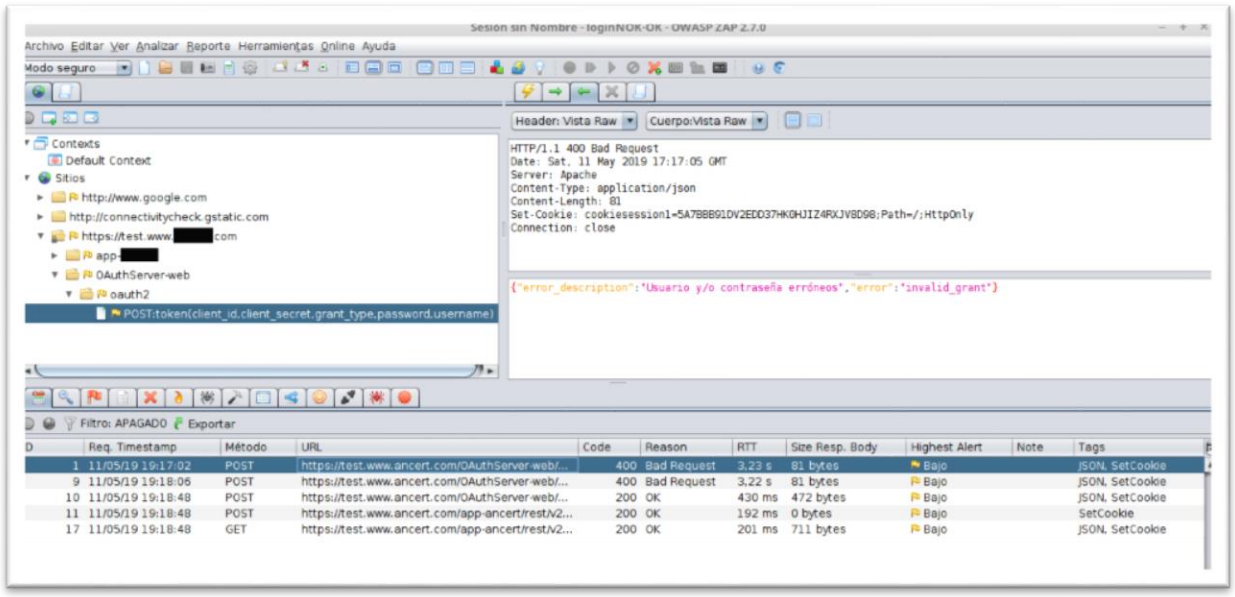
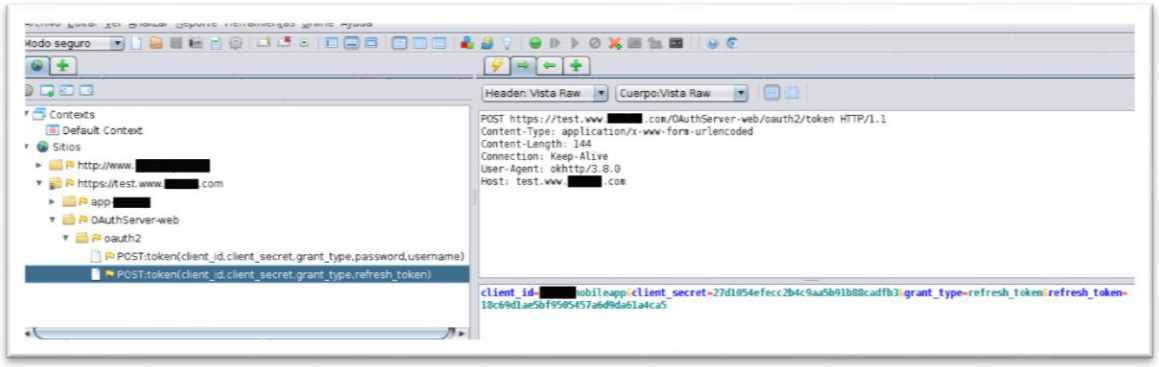


Imagen 35: Respuesta de Login incorrecto

3. Comprobamos que la app nos ofrece también la opción de acceder mediante el uso de la huella dactilar, el cual activamos. En este caso la información visualizada a través de ZAP es:



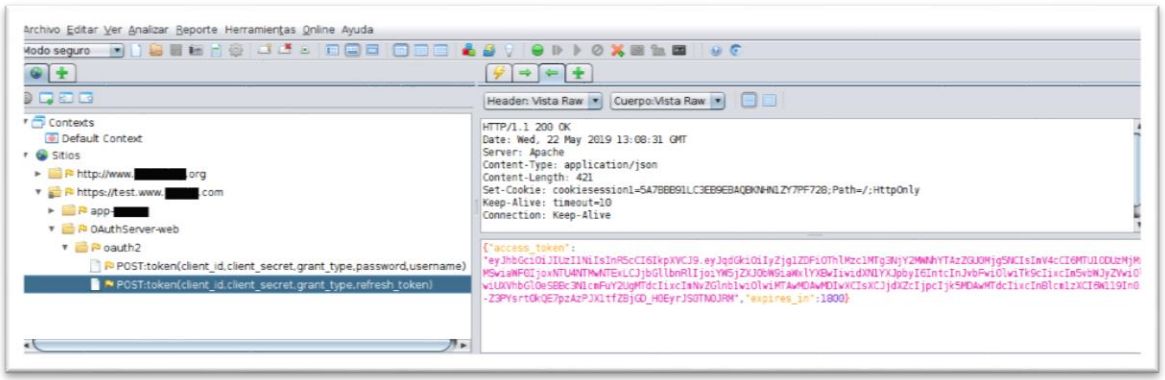


Imagen 36: Login correcto mediante huella dactilar

Se comprueba que una vez activada tal opción no nos deja acceder con una huella diferente y superado el límite de intentos, nos bloquea el acceso durante un periodo configurado.

4. Comprobamos que pasado el tiempo de inactividad configurado, para poder operar desde la app se nos solicitan de nuevo las credenciales de acceso. Comprobamos que el *access-token* de sesión proporcionado para este nuevo login es diferente al anterior:

Imagen 37: Access-token de login-1

Imagen 38: Access-token de login-2

Navegamos a través de las diferentes opciones de la aplicación comprobando que podemos tracear todas las comunicaciones mediante ZAP:

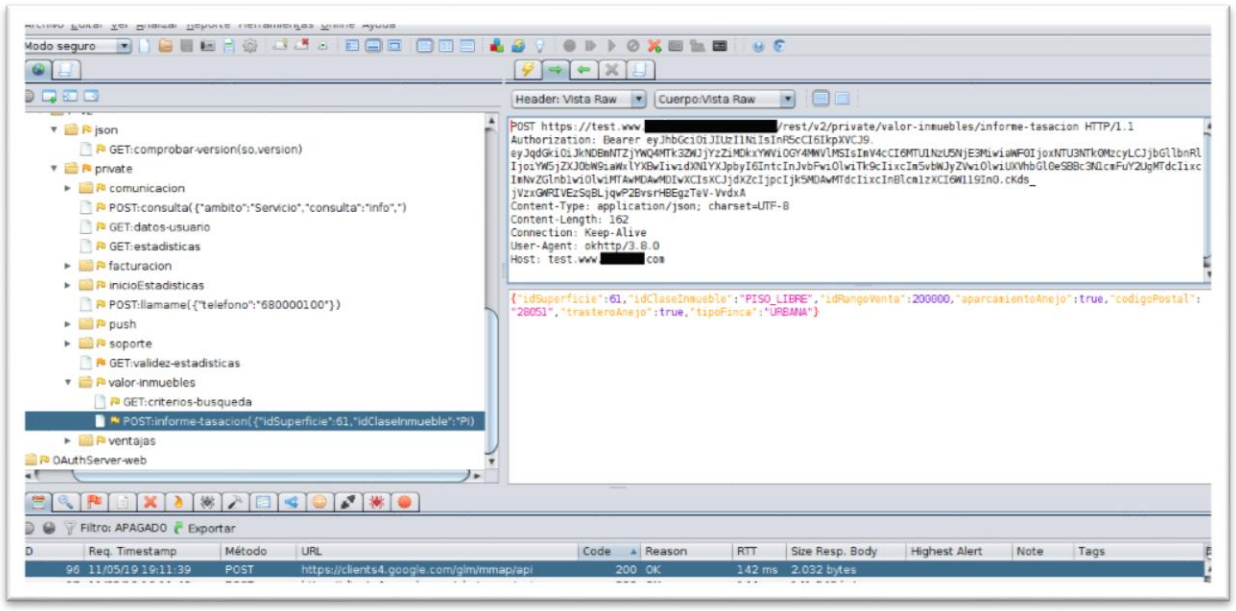


Imagen 39: Traceo del flujo HTTP de las distintas opciones de la aplicación

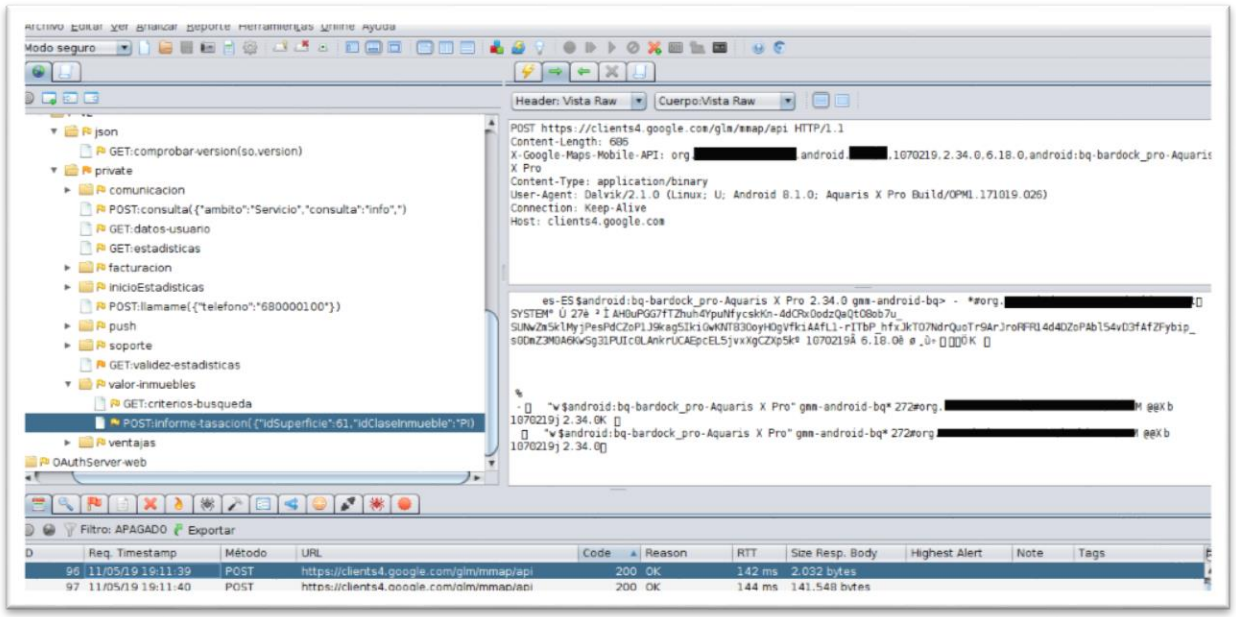


Imagen 40: Traceo del flujo HTTP de las distintas opciones de la aplicación (II)

Encontramos en claro todos los datos enviados a través de los formularios de solicitar información o de solicitar llamada:

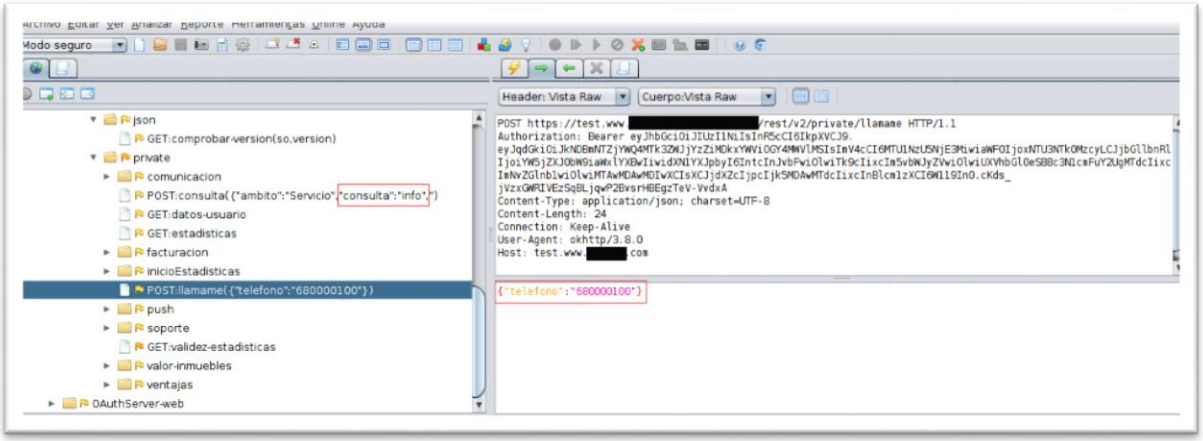


Imagen 41: Datos de formularios capturados por ZAP

ZAP nos proporciona asimismo un listado de alertas respecto a nuestra aplicación:

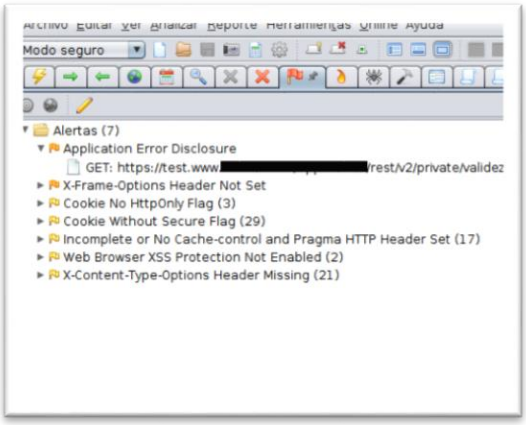


Imagen 42: Listado de Alertas mostrados por ZAP

Dos de ellas están clasificadas con riesgo medio:

- **Revelación de error de aplicación**:
  Evidencia: HTTP/1.1 *500 Internal Server Error*.
  La página contiene un mensaje de error que podría desvelar información sensible como la localización del archivo que provoca tal excepción no capturada, lo cual podría utilizarse para lanzar ataques contra la app.

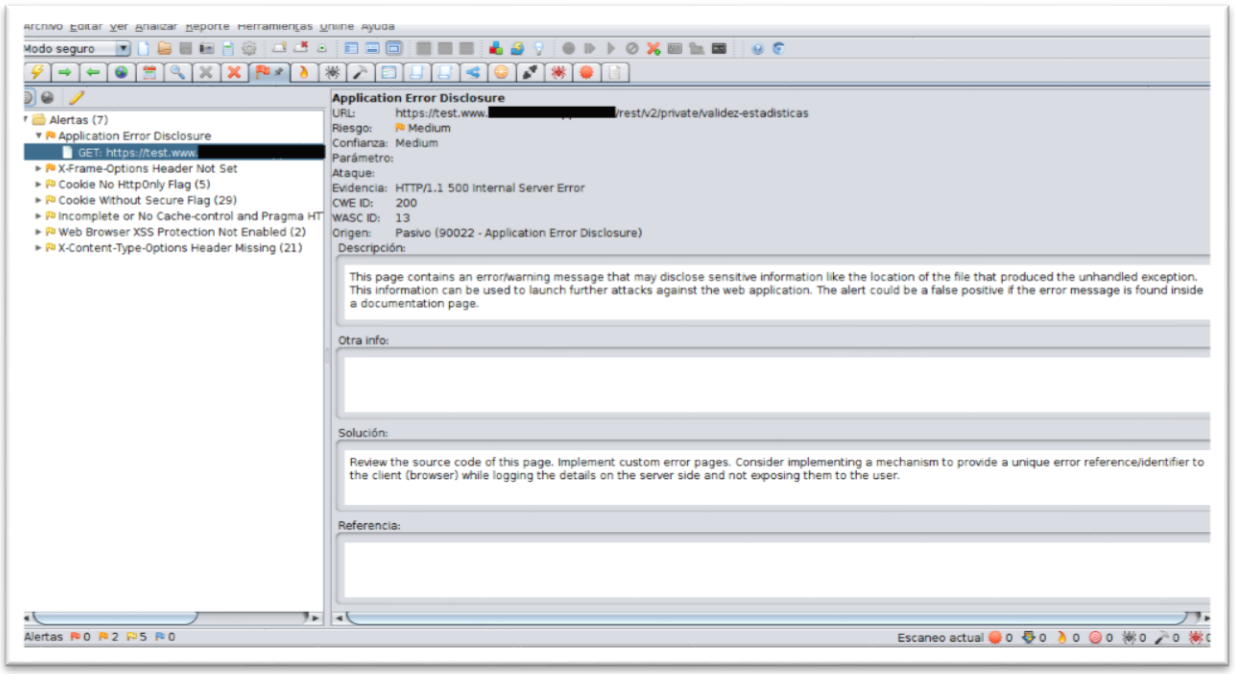


Imagen 43: Alerta de revelación de Error de Aplicación

- **Cabecera X-Frame-Options no establecida:**
  Se detecta que la cabera X-Frame-Options no está incluida en la respuesta HTTP que proteja contra ataques de tipo '*ClickJacking*'

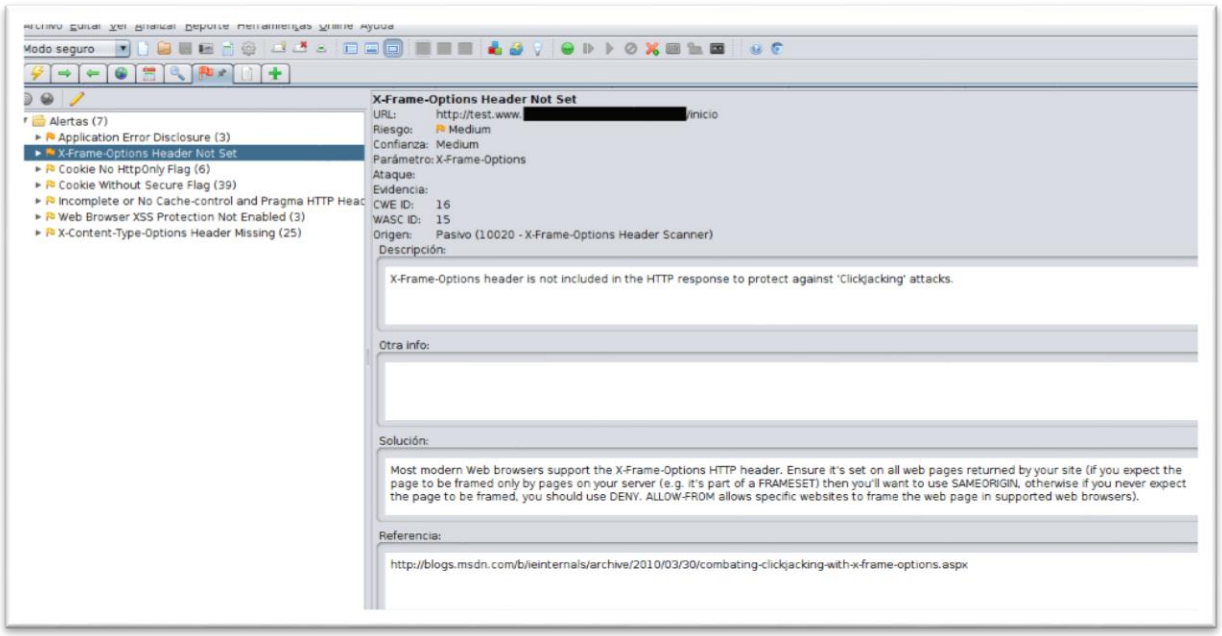


Imagen 44: Alerta de Cabecera X-Frame-Options no establecida

y cinco con riesgo bajo:

- **Cookie sin flag HttpOnly:**
  Evidencia: Set-Cookie: awelocale-soporte
  Se detecta cookie sin flag HttpOnly, lo que implica que ésta puede ser accedida a través de JavaScript. Si se ejecutara un script malicioso en la página, la cookie sería accesible y podría trasmitirse a otra página.

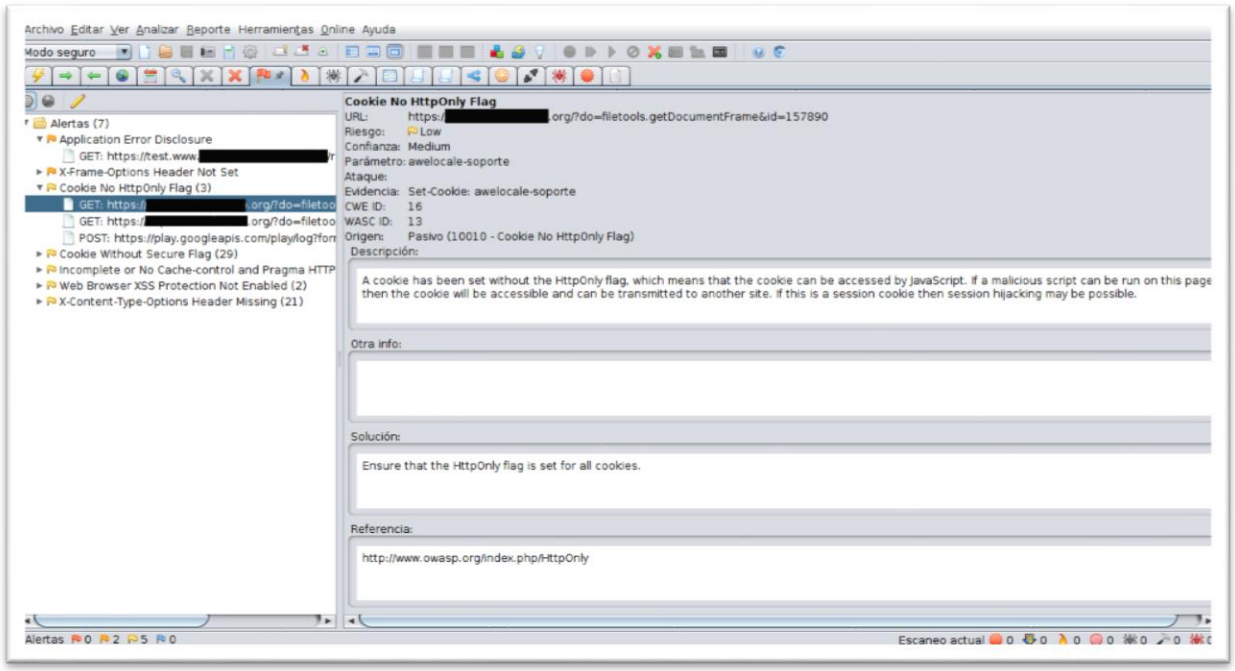


Imagen 45: Alerta de Cookie No HttpOnly Flag

- **Cookie sin Secure Flag:**
  Evidencia: Set-Cookie: cookiesession1
  Se detecta cookie sin *secure flag*, lo que implica que puede ser accedida en conexiones no encriptadas. Tanto si una cookie contiene información sensible como si es un token de sesión, ésta debe ser traspasada usando un canal encriptado.

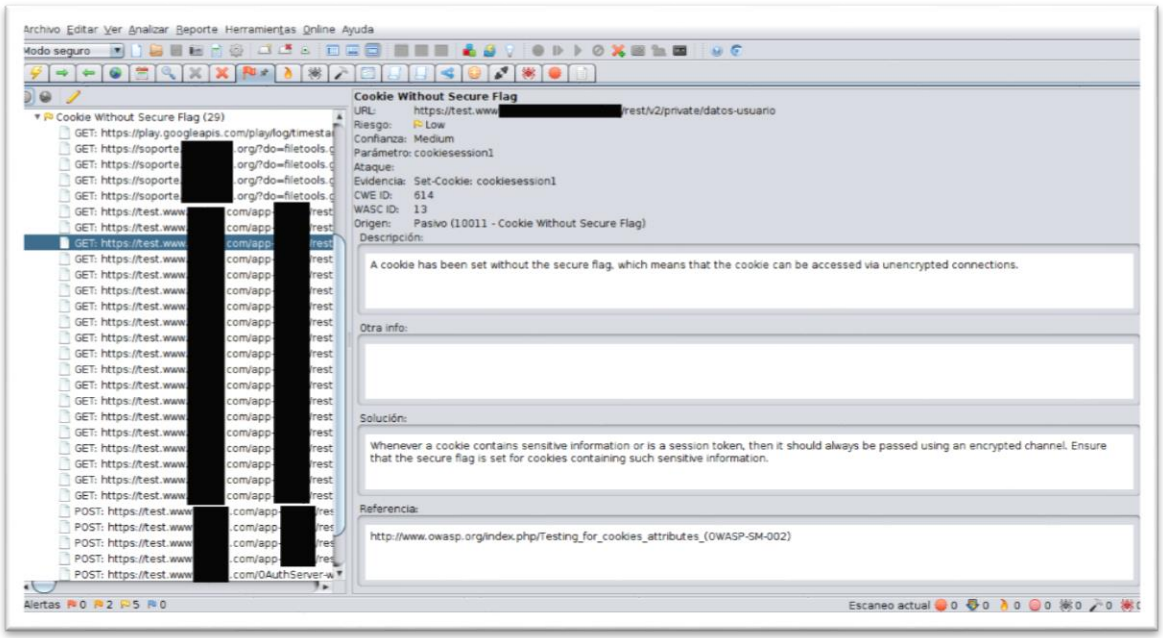


Imagen 46: Alerta de Cookie sin Secure Flag

- **Ausencia de cabecera o cabecera incompleta Cache-control y Pragma HTTP:**
  Se detecta que el control de caché y la cabecera pragma HTTP no están establecidos correctamente permitiendo al navegador y los proxies a contenido de la caché.

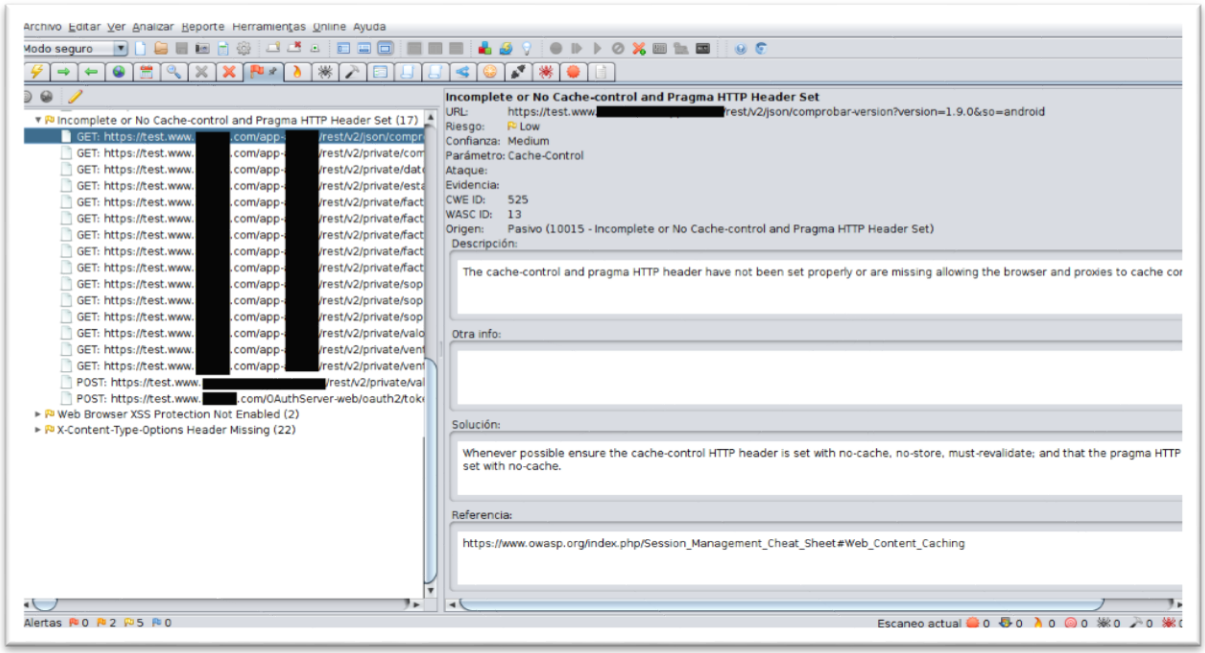


Imagen 47: Alerta de ausencia de cabecera o cabecera incompleta No Cache-control y Pragma HTTP

- **Protección XSS de navegador web no habilitada:**
  La protección XSS del navegador web no está habilitada o ha sido deshabilitada por la cabecera HTTP respuesta “X-XSS-Protection”

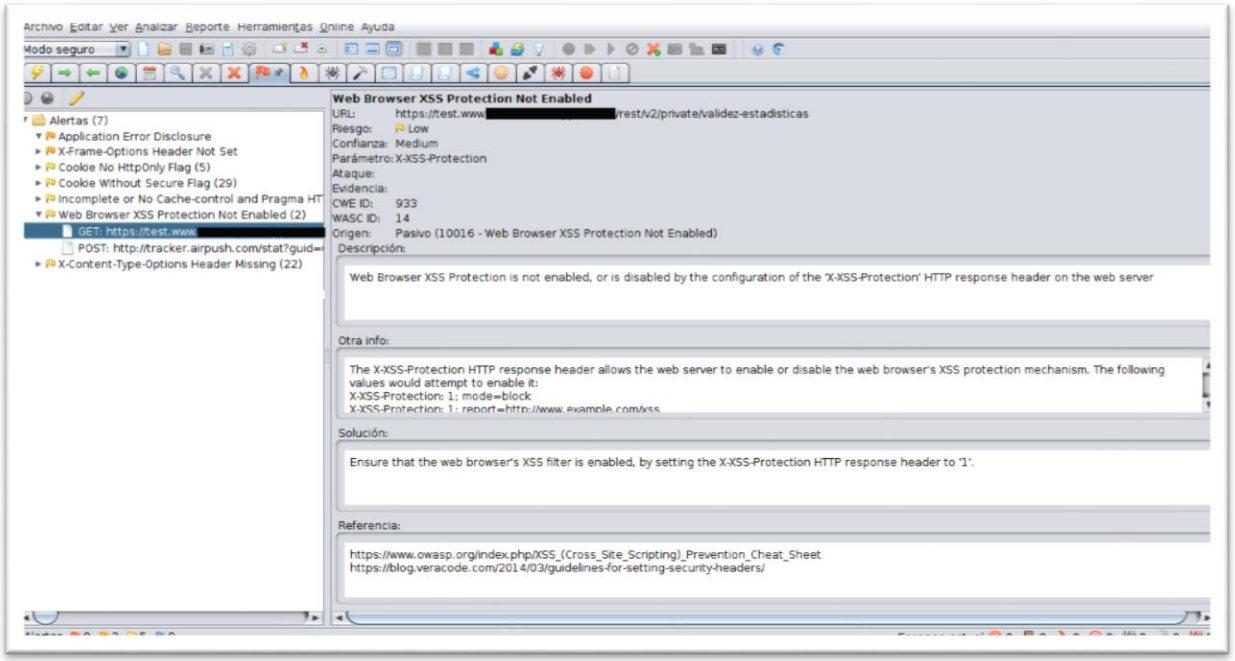


Imagen 48: Alerta de protección XSS de navegador web no habilitada

- **Ausencia de cabecera X-Content-Type-Options:**
  Se detecta que la cabecera *X-Content-Type-Options* no tiene valor ‘*nosniff*’. Esto permitiría a versiones antiguas de IE y Chrome realizar MIME-Sniffing en el cuerpo de la respuesta, lo que podría causar que éste se interpretara como un tipo de contenido diferente al declarado. Este error aplica a las páginas de error (401, 403, 500, etc.) que son las que generalmente se pueden ver afectadas por problemas de inyección.

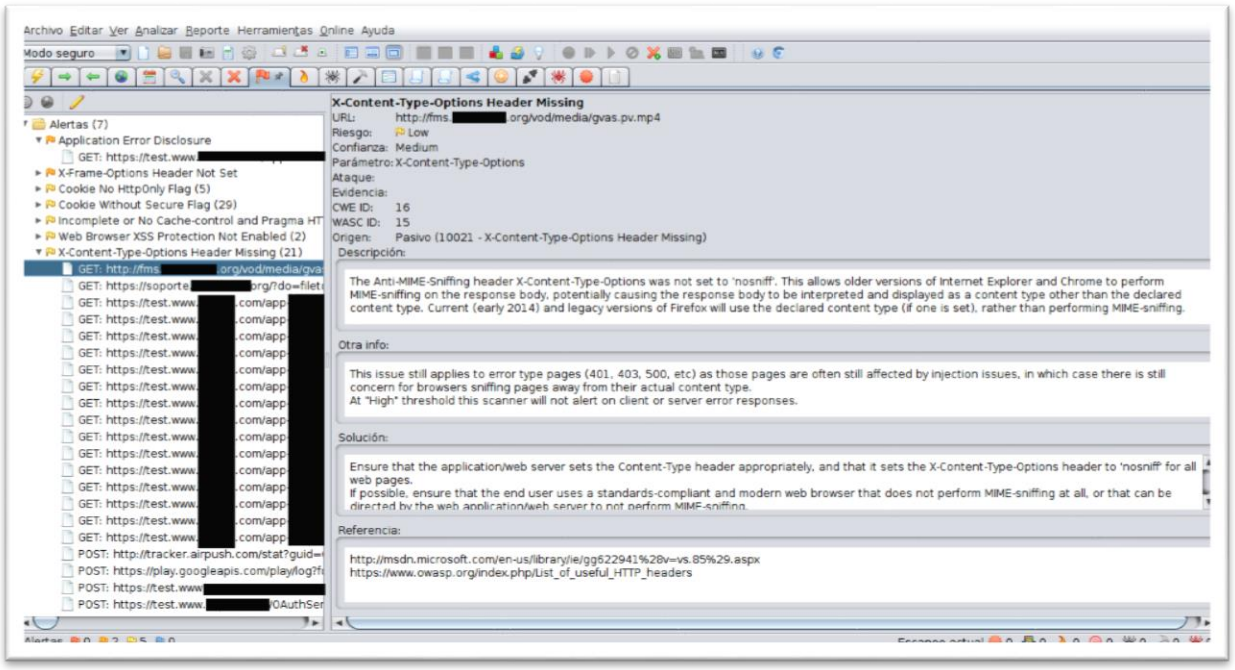


Imagen 49: Alerta de Ausencia de cabecera X-Content-Type-Options

10.6 Informe de resultados.

Los resultados obtenidos a partir de la ejecución del análisis estático tanto manual como automatizado, y dinámico, se presentan a continuación repartidos en los principales puntos de control propuestos por MASVS de OWASP y detallados en la guía *Mobile Security Testing* [2].

El detalle de los reportes generados a partir del análisis estático automatizado de MobSF, el análisis dinámico de Inspeckage y ZAP, así como las sesiones de análisis guardadas de este último se encuentran adjuntos en el Anexo E. Informes, para complementar el informe de resultados a continuación incluido.

10.6.1 Verificación de requerimientos V1: Arquitectura, Diseño y Modelado de Amenazas

Se comprueba para la app que todos los componentes están identificados y son necesarios para el correcto funcionamiento de ésta.

A partir de los resultados obtenidos de los análisis estáticos y dinámicos, se asume que se definió una arquitectura de alto nivel para la app y los servicios y se comprueba que se han incluido los controles de seguridad necesarios para éstos.

Se asume asimismo que la información considerada sensible en el contexto de la app está claramente identificada y se aplican las medidas adecuadas de criptografía.

10.6.2 Verificación de requerimientos V2: Almacenamiento de datos y la Privacidad

Se comprueba el archivo AndroidManifest.xml para determinar si los datos sensibles se envían a terceras partes ya que de acuerdo con las recomendaciones, los permisos definidos para acceder a SMS, contactos y ubicación han de ser comprobados específicamente. En la app de estudio se comprueba que se han definido los permisos de:

- WRITE_CONTACTS
- ACCESS_FINE_LOCATION

Se recomienda por tanto que se revisen los posibles cambios en código posteriores a que las librerías se hayan añadido al proyecto, si están obsoletas o si son estrictamente necesarias. Todos los datos enviados a terceras partes han de estar anonimizados.

Se comprueba en el archivo *manifest* que el atributo *allowBackup* tiene valor “true” por lo que el backup de la aplicación podría realizarse por medio de ADB (Android

Debug Bridge). Se recomienda que el valor de este atributo sea expresamente false ya que por defecto éste se encuentra habilitado. En caso de mantenerlo habilitado es importante revisar que la app no guarde ningún tipo de información sensible que pueda ser recuperada del backup.

A partir del análisis dinámico se comprueban los siguientes aspectos:

1. Se ha comprobado que al pinchar sobre los campos que requieren información sensible (por ejemplo, la contraseña) no se sugiere ninguna cadena y por tanto la caché del teclado ha sido desactivada para dichos campos.
2. Se ha comprobado que la información sensible como, por ejemplo, la contraseña, se reemplaza con asteriscos para evitar una posible filtración de datos a la interfaz de usuario.
3. A partir de la configuración de un proxy local para interceptar el tráfico entre el cliente y el servidor mediante la herramienta ZAP de OWASP, se ha comprobado que se pueden encontrar en claro los campos enviados en cada solicitud, incluida la contraseña de usuario.

10.6.3 Verificación de requerimientos V3: Criptografía

Se comprueba el uso de las clases e interfaces más frecuentes en las primitivas criptográficas: Cipher, Mac, MessageDigest, Signature, Key, PrivateKey, PublicKey, SecretKey, paquetes java.security.* y javax.crypto.*.

De las comprobaciones del análisis estático manual y automático mediante MobSF, se puede destacar lo siguiente:

1. Se ha comprobado que no se emplea SHA1PRNG, actualmente en desuso por no ser criptográficamente seguro.
2. Se utilizan los algoritmos recomendados de integridad (SHA-256), firma digital y establecimiento de claves (RSA).
3. El análisis estático automático de MobSF detecta archivos con información sensible (nombres de usuario, *passwords*, claves, etc.) *hardcodeadas*. El detalle de las clases afectadas se incluye en el reporte completo en Anexo E del presente documento para su revisión.
4. El análisis estático automático de MobSF detecta el uso de un Generador de Números aleatorios inseguro en una serie de clases, cuyo detalle puede consultarse en el reporte completo incluido en el Anexo E del presente documento. De acuerdo con las mejores prácticas propuestas, la criptografía requiere el uso del *pseudo random number generation* (PRNG) ya que las clases Java Standard no proporcionan el suficiente nivel de aleatoriedad. En general, se aconseja el uso de SecureRandom.

   Se recomienda en este punto revisar las clases incluidas en el reporte de MobSF.

5. El análisis estático automático de MobSF detecta asimismo que la app utiliza Java Hash Code en las clases listadas en detalle en el reporte incluido en el Anexo E. Se trata de una función hash considerada débil cuyo uso no está recomendado en implementaciones de criptografía segura. Se recomienda en este punto revisar las clases incluidas en el reporte de MobSF.

10.6.4 Verificación de requerimientos V4: Autenticación y Manejo de Sesiones

El análisis de tráfico con la herramienta ZAP revela las contraseñas usadas para la autenticación en texto plano, sin cifrar, lo cual la hace vulnerables frente ataques de MiTM.

Sí se comprueba, sin embargo, a partir del análisis dinámico que:

1. La aplicación provee un mecanismo de autenticación usuario/contraseña (o huella) en un servidor remoto, aceptable para aplicaciones no-críticas de acuerdo con el Nivel 1 de MASVS, que no exige para este nivel el segundo factor de autenticación.
2. Se comprueba además que los IDs de sesión solo se intercambian sobre conexiones seguras (HTTPS), no se detecta que la app los guarde en ningún almacenamiento permanente y no son adivinables fácilmente.
3. Se comprueba que el mecanismo de concesión de tokens combinado de *access-token y refresh-token* lleva configurado un tiempo de expiración razonable, con el objetivo de limitar la probabilidad de secuestros de sesión.
4. Cuando el usuario cierra la sesión se termina la sesión también en el servidor, y se comprueba que para un nuevo login, la ID de sesión y el token de acceso son diferentes.
5. Cuando la sesión se cierra por inactividad (*session timeout*), se termina la sesión también en el servidor y se comprueba que para un nuevo login, la ID de sesión y el token de acceso son diferentes.
6. El servidor implementa mecanismos cuando se supera el máximo número de intentos de las credenciales de autenticación, tanto de contraseña (3) como de acceso por huella (5). El acceso permanece bloqueado durante un determinado tiempo y se muestra el mensaje apropiado por pantalla.

10.6.5 Verificación de requerimientos V5: Comunicación a través de la red

Se comprueba que el flag *android:debuggable* está habilitado en el archivo manifest de la aplicación. Se ignora este aspecto ya que la versión proporcionada para el estudio es una versión en actual desarrollo.

Se destaca la necesidad de que el flag se desactive en el momento de liberar la app, ya que la implementación del framework interno Apache Cordova de WebView ignorará los errores TLS del método onReceivedSslError mientras este flag permanezca activado.

Por otro lado, a partir del análisis dinámico:

1. Se ha configurado un proxy local para interceptar el tráfico entre el cliente y el servidor mediante la herramienta ZAP de OWASP donde se comprobó en primer lugar, que éste era incapaz de interceptar el tráfico HTTPS sin disponer del certificado raíz, y en segundo lugar, que la app implementa SSL Pinning para lo cual hubo de hacer uso de otras herramientas adicionales que permitieran saltar dicho mecanismo.
2. Respecto a la verificación de certificados se comprobó que la aplicación acepta los certificados auto-firmados, como el generado desde la herramienta ZAP para la monitorización de tráfico HTTPS.

10.6.6 Verificación de requerimientos V6: Interacción con la Plataforma

Se comprueban los permisos definidos para la app en AndroidManifest.xml para eliminar los que realmente no son necesarios.

1. Se comprueba que el permiso INTERNET (`<uses-permission android:name="android.permission.INTERNET" />`) se ha definido para nuestra app, necesario para que una actividad cargue una página web en WebView.
2. Se comprueba además que de entre los permisos considerados como potencialmente peligrosos [2], se han definido, además: WRITE_CONTACTS, CAMERA, ACCESS_FINE_LOCATION y WRITE_EXTERNAL_STORAGE.

Se subscribe por tanto la recomendación de que el desarrollador debe verificar si la aplicación tiene los permisos adecuados cada vez que se realiza una acción que requiera ese permiso, ya que un usuario puede revocar el derecho de una aplicación a usar un permiso peligroso.

No se detecta en AndroidManifest.xml la definición de ningún permiso personalizado de acceso a componentes expuestos (no se detecta el uso de la etiqueta requerida para tal caso: *android:protectionLevel*)

10.6.7 Verificación de requerimientos V7: Calidad de Código y Configuración del Compilador

1. Se comprueba que el flag *android:debuggable* está habilitado en el archivo manifest de la aplicación. Se ignora este aspecto ya que la versión proporcionada para el estudio es una versión en actual desarrollo, pero sí se destaca la

necesidad de que el flag se desactive en el momento de liberar la app (*non-debuggable*) junto con las configuraciones apropiadas para el mismo.

2. La aplicación está firmada y provista con un certificado válido. El algoritmo empleado es SHA256withRSA, recomendado para firmas digitales para garantizar la integridad. La revisión del certificado llevado a cabo de manera automática con MobSF le otorga un Estado de certificado con valor Bueno.

3. El análisis estático automático realizado con MobSF revela que la app contiene logs que almacenan información sensible. De acuerdo con las recomendaciones, la app no debe realizar ningún tipo de log de errores o mensajes de debug por lo que, considerando que se ha analizado una versión de desarrollo de la app, se aconseja revisar este punto antes de liberar la app.

4. La herramienta ZAP configurada para el análisis dinámico de la aplicación nos muestra una serie de alertas que debemos revisar en este punto:

   - **Revelación de error de aplicación**: se detecta para algunas operaciones de la app, páginas que devuelven un error *500 Internal Server Error* que podría desvelar información sensible, por lo que se sugiere la revisión del código fuente de éstas, ya que tal información podría utilizarse para lanzar ataques contra la app.
   - **Cabecera X-Frame-Options no establecida:** se recomienda asegurar que todas las respuestas devueltas por la app establezcan tal cabecera.
   - **Cookies sin flag HttpOnly o sin flag Secure:** Se recomienda revisar que todas las cookies se configuren con flag HttpOnly, para evitar que puedan ser accedidas a través de JavaScript así como con flag Secure, ya que tanto si contienen información sensible como si es un token de sesión, ésta debe ser traspasada usando un canal encriptado.
   - **Protección XSS de navegador web no habilitada:** se recomienda revisar que el filtro XSS del navegador web esté habilitado mediante la cabecera HTTP respuesta “X-XSS-Protection” con valor “1”.
   - **Ausencia de cabecera o cabecera incompleta Cache-control y Pragma HTTP:** en la medida de lo posible, se recomienda asegurar que la cabecera HTTP cache-control está configurada como *no-cache*, *no-store*, *must-revalidate;* y pragma HTTP como no-cache.
   - **Ausencia de cabecera X-Content-Type-Options:** se recomienda revisar que la aplicación establece la cabecera Content-Type apropiada y que X-Content-Type está configurado como *’nosniff’* para todas las páginas.

En lo referido a las opciones que provocan un error 500 Internal Server error, se comprueba que la app muestra un error por pantalla informando al usuario de que ha ocurrido un error que imposibilita continuar con la operación solicitada, pero no revela por pantalla en ningún caso, información sensible o interna de la app y permite al usuario continuar con el uso normal de ésta.

11. Conclusiones

Las auditorias de seguridad de aplicaciones móviles son hoy en día una práctica necesaria y recomendada para garantizar que se cumplen los requerimientos básicos de seguridad ante la amenaza creciente de malware enfocado a este tipo de dispositivos.

Las características particulares de los dispositivos móviles y sus aplicaciones se han tenido en cuenta en la elaboración de la guía propuesta por OWASP para *testing* de seguridad móvil en concreto, resultando una muy completa referencia para llevar a cabo auditorías de seguridad ya que proporciona, además, un listado de buenas prácticas para que el auditor verifique los Requisitos de Seguridad MASVS de acuerdo con 3 niveles de seguridad.

En el presente trabajo se ha llevado a cabo un estudio de las posibilidades de auditoría de seguridad de las aplicaciones móviles, las metodologías existentes y los modelos de seguridad aceptados en la actualidad como estándar para su realización, así como de algunas herramientas disponibles para su ejecución.

Teniendo en cuenta las pautas proporcionadas por OWASP para seguridad móvil como metodología elegida de referencia, se ha ejecutado un análisis estático tanto manual como automatizado de una aplicación móvil seleccionada previamente, así como un análisis dinámico manual, de acuerdo con el nivel L1 propuesto por MASVS.

El reporte de resultados incluido contempla una revisión de requerimientos propuestos por MASVS para el nivel de verificación elegido, de manera que se han cubierto todos los objetivos de control sugeridos. Sin embargo, llevar a cabo un análisis pormenorizado de todos los puntos enumerados en los requerimientos de acuerdo con las pautas sugeridas en la guía OWASP de testing de seguridad, requiere de unos plazos mayores a los inicialmente planteados en el presente proyecto.

La aplicación cumple en general con la mayoría de los requisitos de Seguridad MASVS L1 para Android, pero los distintos análisis sí revelan algunos detalles importantes que deberían tenerse en cuenta antes de liberar la app:

- el análisis de tráfico con la herramienta ZAP reveló las contraseñas usadas para la autenticación del usuario en texto plano, sin cifrar, lo cual la hace vulnerables frente ataques de MiTM.
- esta misma herramienta también detectó, para algunas páginas que devuelven errores del tipo: *500 Internal Server Error* que éstas podrían desvelar información sensible o ser utilizadas para lanzar ataques contra la app. También se muestran otras alertas como ausencia de algunas cabeceras HTTP o cabeceras incompletas.

- el análisis estático por su parte evidenció algunos valores no recomendables en la configuración del archivo *manifest*, como el flag *android:debuggable y* atributo *allowBackup* con valor “true”.
- el análisis estático automatizado también detectó el uso de un Generador de Números aleatorios inseguro así como funciones hash no recomendadas en implementaciones de criptografía segura.

En el momento de seleccionar las herramientas que facilitasen o apoyasen los distintos análisis ejecutados sobre la app, se ha podido comprobar que la oferta de aplicaciones que permiten llevar a cabo análisis automatizados estáticos y dinámicos es amplia, pero en muchos casos han quedado obsoletas, funcionando solo para versiones de Android antiguas y prácticamente en desuso, siendo incompatibles con las nuevas versiones.

Como ejemplo, una de las herramientas planificadas inicialmente para llevar a cabo el análisis dinámico automatizado de la app fue MobSF – pues también se configuró para su uso en el análisis estático -, no pudo finalmente ser configurada para la app seleccionada: los diversos intentos fallidos de configurar la herramienta (ni mediante la máquina virtual proporcionada por MobSF para tal efecto, por tratarse de una versión demasiado anticuada, ni con la conexión a dispositivos físicos rooteados en versiones actuales) unidos a la limitación de tiempo influyeron directamente en la decisión de ejecutar finalmente un análisis dinámico a partir de otras herramientas.

Para finalizar, se propondrían las siguientes líneas de trabajo futuro en relación con la auditoría de seguridad sobre aplicaciones móviles:

- Ya que el nivel elegido para realizar la auditoría de acuerdo con MASVS fue L1 basándonos en la aplicación elegida para el estudio, podría ser interesante ampliar el alcance de la auditoría a niveles L2 y R para otras aplicaciones que por su nivel de tratamiento de datos sensibles o realización de operaciones así lo requirieran.
- El actual análisis dinámico podría complementarse con otras técnicas de búsquedas de vulnerabilidades como la de *Fuzzing*, para comprobar la configuración de seguridad ante datos inválidos o inesperados.
- Ya que las fuentes oficiales consultadas afirman que MobSF es compatible con versiones actuales de Android, una futura línea de investigación podría centrarse en trabajar en la configuración de un entorno para la automatización de análisis dinámicos mediante la mencionada herramienta, vinculándola a dispositivos rooteados o emuladores, pues no hay prácticamente bibliografía actualizada al respecto.

12. Glosario

ADB	Android Debug Bridge
API	Application Programming Interface
APK	Android Application Package
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
GDPR	Reglamento General de Protección de Datos
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IPC	Inter-Process Communication
MASVS	Mobile Application Security Verification Standard
MiTM	Man in The Middle
MobSF	Mobile Security Framework
MSTG	Mobile Security Testing Guide
OWASP	Open Web Application Security Project
SDK	Software Development Kit
SO	Sistema Operativo
URL	Uniform Resource Locator
ZAP	Zed Attack Proxy

13. Bibliografía

[1] Mobile Application Security Verification Standard v1.0. OWASP.
[2] Mobile Security Testing Guide. OWASP. https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide

[3] Proyecto de Seguridad Móvil de OWASP. https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

[4] Top 10 Riesgos Móviles de OWASP. https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10

[5] https://universoabierto.org/2017/05/13/informe-anual-de-estado-de-seguridad-de-aplicaciones-percepcion-vs-realidad/
[6] https://arpentechnologies.com/es/blog/aplicaciones-movil/mejores-practicas-de-seguridad-al-desarrollar-una-aplicacion-movil/
[7] https://blog.pradeo.com/research-current-state-mobile-application-security
[8] https://blog.pradeo.com/banking-institution-prevents-fraud-rasp
[9] https://mobile-security.gitbook.io/mobile-security-testing-guide/overview/0x04a-mobile-app-taxonomy
[10] https://www.yeeply.com/blog/tipos-de-app-y-para-que-sirven/
[11] https://appyourself.net/es/blog/tipos-de-aplicaciones-moviles/
[12] https://www.seguridadparatodos.es/2013/02/OWASP-Parte1MetodologiaAppMobile.html
[13] https://www.seguridadparatodos.es/2013/02/OWASP-Parte2MetodologiaAppMobile.html
[14] https://www.seguridadparatodos.es/2013/04/OWASP-Parte3MetodologiaAppMobile.html
[15] https://www.newgenapps.com/blog/10-biggest-risks-to-mobile-apps-security
[16] https://www.yeeply.com/blog/gdpr-apps-moviles-nuevas-regulaciones/
[17] https://www.deustoformacion.com/blog/desarrollo-apps/afecta-rgpd-desarrollo-app
[18] http://auroralabs.es/blog/cumple-mi-app-con-el-rgpd/?lang=en
[19] https://sontusdatos.org/wp-content/uploads/2013/04/ce-dictamen-02-2013-aplicaciones-dispositivos-inteligentes.pdf
[20] https://www.owasp.org/index.php/Static_Code_Analysis
[21] https://academiaandroid.com/componentes-aplicacion-android/
[22] http://academiaandroid.com/activity-y-fragments/
[23] https://source.android.com/security/app-sandbox
[24] https://www.welivesecurity.com/la-es/2016/12/19/analizar-apk-con-mobsf/
[25] https://www.welivesecurity.com/la-es/2017/10/20/analizar-apps-android-inspeckage/
[26] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
[27] https://www.securityartwork.es/2016/03/14/22035/
[28] https://www.csirtcv.gva.es/es/paginas/herramientas-para-la-seguridad-en-dispositivos-m%C3%B3viles.html
[29] https://hub.packtpub.com/auditing-mobile-applications/

[30] https://github.com/MobSF/Mobile-Security-Framework-MobSF/wiki/1.-Documentation
[31] https://hydrasky.com/malware-analysis/mobile-security-framework-mobsf-configuration/
[32] https://github.com/flyfei/ApkDecompile/tree/master/Tools
[33] https://github.com/MobSF/Mobile-Security-Framework-MobSF/wiki/11.-Configuring-Dynamic-Analyzer-with-MobSF-Android-4.4.2-x86-VirtualBox-VM
[34] https://github.com/zaproxy/zaproxy/wiki/Downloads
[35] https://www.welivesecurity.com/la-es/2016/03/03/como-interceptar-trafico-https-android/

14. Anexo A. Requisitos de seguridad - Android.

ID	Verificación detallada de requerimientos	Nivel 1	Nivel 2
V1	**Arquitectura, Diseño y Modelado de Amenazas**		
1.1	Todos los componentes se encuentran identificados y asegurar que son necesarios.	✓	✓
1.2	Los controles de seguridad nunca se aplican sólo en el lado del cliente, sino que también en los respectivos servidores remotos.	✓	✓
1.3	Se definió una arquitectura de alto nivel para la aplicación y los servicios y se incluyeron controles de seguridad en la misma.	✓	✓
1.4	Se identificó claramente la información considerada sensible en el contexto de la aplicación móvil.	✓	✓
1.5	Todos los componentes de la aplicación están definidos en términos de la lógica de negocio o las funciones de seguridad que proveen.		✓
1.6	Se realizó un modelado de amenazas para la aplicación móvil y los servicios en el que se definieron las mismas y sus contramedidas.		✓
1.7	La implementación de los controles de seguridad se encuentra centralizada.		✓
1.8	Existe una política explícita para el manejo de las claves criptográficas (si se usan) y se refuerza su ciclo de vida. Idealmente siguiendo un estándar del manejo de claves como el NIST SP 800-57.		✓
1.9	Existe un mecanismo para imponer las actualizaciones de la aplicación móvil.		✓
1.10	Se realizan tareas de seguridad en todo el ciclo de vida de la aplicación.		✓
V2	**Almacenamiento de datos y la Privacidad**		
2.1	Las funcionalidades de almacenamiento de credenciales del sistema son utilizadas para almacenar la información sensible, como credenciales del usuario y claves criptográficas.	✓	✓
2.2	No se escribe información sensible en los registros de la aplicación.	✓	✓
2.3	No se comparte información sensible con servicios externos salvo que sea una necesidad de la arquitectura.	✓	✓
2.4	Se desactiva el caché del teclado en los campos de texto donde se maneja información sensible.	✓	✓
2.5	Se desactiva el portapapeles en los campos de texto donde se maneja información sensible.	✓	✓
2.6	No se expone información sensible mediante mecanismos entre procesos (IPC).	✓	✓
2.7	No se expone información sensible como contraseñas y números de tarjetas de crédito a través de la interfaz o capturas de pantalla.	✓	✓
2.8	No se incluye información sensible en los respaldos generados por el sistema operativo.		✓
2.9	La aplicación remueve la información sensible de la vista cuando la aplicación pasa a un segundo plano.		✓
2.10	La aplicación no conserva la información sensible en memoria más de lo necesario y la memoria es limpiada luego de su uso.		✓
2.11	La aplicación obliga a que exista una política mínima de seguridad en el dispositivo, como que el usuario deba configurar un código de acceso.		✓
2.12	La aplicación educa al usuario acerca de los tipos de información personal que procesa y de las mejores prácticas en seguridad que el usuario debería seguir al utilizar la aplicación.		✓
V3	**Criptografía**		
3.1	La aplicación no depende de únicamente de criptografía simétrica con "claves a fuego".	✓	✓
3.2	La aplicación utiliza implementaciones de criptografía probadas.	✓	✓
3.3	La aplicación utiliza primitivas de seguridad que son apropiadas para el caso particular y su configuración y sus parámetros siguen las mejores prácticas de la industria.	✓	✓
3.4	La aplicación no utiliza protocolos o algoritmos criptográficos que son considerados deprecados para aspectos de seguridad.	✓	✓
3.5	La aplicación no reutiliza una misma clave criptográfica para varios propósitos.	✓	✓
3.6	Los valores aleatorios son generados utilizando un generador de números suficientemente aleatorios.	✓	✓
V4	**Autenticación y Manejo de Sesiones**		
4.1	Si la aplicación provee acceso a un servicio remoto, un mecanismo aceptable de autenticación como usuario y contraseña es realizado en el servidor remoto.	✓	✓
4.2	Si se utiliza la gestión de sesión por estado, el servidor remoto usa tokens de acceso aleatorios para autenticar los pedidos del cliente sin requerir el envío de las credenciales del usuario en cada uno.	✓	✓
4.3	Si se utiliza la autenticación basada en tokens sin estado, el servidor proporciona un token que se ha firmado utilizando un algoritmo seguro.	✓	✓
4.4	Cuando el usuario cierra sesión se termina la sesión también en el servidor.		
4.5	Existe una política de contraseñas y es aplicada en el servidor.	✓	✓
4.6	El servidor implementa mecanismos, cuando credenciales de autenticación son ingresadas una cantidad excesiva de veces.	✓	✓
4.7	Las sesiones y los tokens de acceso expiran luego de un tiempo predefinido de inactividad.		✓
4.8	La autenticación biométrica, si hay, no está atada a un evento (usando una API que simplemente retorna "true" o "false"). Sino que está basado en el desbloqueo del keychain (iOS) o un keystore (Android).		✓
4.9	Existe un mecanismo de segundo factor de autenticación (2FA) en el servidor y es aplicado consistentemente.		✓
4.10	Para realizar transacciones o acciones que manejan información sensible se requiere una re-autenticación.		✓
4.11	La aplicación informa al usuario acerca de los accesos a su cuenta. El usuario es capaz de ver una lista de los dispositivos conectados y bloquear el acceso desde ciertos dispositivos.		✓
V5	**Comunicación a través de la red**		
5.1	La información es enviada cifrada utilizando TLS. El canal seguro es usado consistentemente en la aplicación.	✓	✓
5.2	Las configuraciones del protocolo TLS siguen las mejores prácticas o tan cerca posible mientras que el sistema operativo del dispositivo lo permite.	✓	✓
5.3	La aplicación verifica el certificado X.509 del servidor al establecer el canal seguro y solo se aceptan certificados firmados por una CA válida.	✓	✓
5.4	La aplicación utiliza su propio almacén de certificados o realiza una fijación del certificado o la clave pública del servidor y no establece una conexión con servidores que ofrecen otros certificados o clave por más que estén firmados por una CA confiable.		✓
5.5	La aplicación no depende de un único canal de comunicaciones inseguro (email o SMS) para operaciones críticas como registros o recuperación de cuentas.		✓
5.6	La aplicación sólo depende de bibliotecas de conectividad y seguridad actualizadas.		✓

V6	Interacción con la Plataforma		
6.1	La aplicación requiere la mínima cantidad de permisos.	✓	✓
6.2	Toda entrada del usuario y fuentes externas es validada y si es necesario sanitizada. Esto incluye información recibida por la UI, y mecanismo IPC como los intents, URLs y fuentes de la red.	✓	✓
6.3	La aplicación no exporta funcionalidades sensibles vía esquemas de URL, salvo que dichos mecanismos estén debidamente protegidos.	✓	✓
6.4	protegidos.	✓	✓
6.5	JavaScript se encuentra deshabilitado en los WebViews salvo que sea necesario.	✓	✓
6.6	Los WebViews se encuentran configurados para permitir el mínimo de los manejadores (idealmente, solo https). Manejadores peligrosos como file, tel y app-id se encuentran deshabilitados.	✓	✓
6.7.	Si objetos nativos son expuestos en WebViews, verificar que solo se cargan JavaScripts contenidos del paquete de la aplicación.	✓	✓
6.8	Serialización de objetos, si se realiza, se implementa utilizando API seguras.	✓	✓
V7	**Calidad de Código y Configuración del Compilador**		
7.1	La aplicación es firmada y provista con un certificado válido.	✓	✓
7.2	La aplicación fue liberada en modo release y con las configuraciones apropiadas para el mismo (ej. non-debuggable).	✓	✓
7.3	Los símbolos de debug fueron removidos de los binarios nativos.	✓	✓
7.4	La aplicación no contiene código de prueba y no realiza log de errores o mensajes de debug.	✓	✓
7.5	Todos los componentes de terceros se encuentran identificados y revisados por vulnerabilidades conocidas.	✓	✓
7.6	La aplicación captura y maneja debidamente las posibles excepciones.	✓	✓
7.7	Los controles de seguridad deniegan el acceso por defecto.	✓	✓
7.8	En código no administrado, la memoria es pedida, usada y liberada de manera correcta.	✓	✓
7.9	Funcionalidades de seguridad gratuitas se encuentran activadas.	✓	✓

V8	Resistencia ante la Ingeniería Inversa	R
	Impedir el Análisis Dinámico y la Manipulación	
8.1	La aplicación detecta y responde a la presencia de un dispositivo rooteado, ya sea alertando al usuario o finalizando la ejecución de la aplicación.	✓
8.2	La aplicación previene el debugging o detecta y responde al debugging de la aplicación. Se deben cubrir todos los protocolos.	✓
8.3	La aplicación detecta y responde a modificaciones de ejecutables y datos críticos de la propia aplicación.	✓
8.4	La aplicación detecta la presencia de las herramientas de ingeniería reversa o frameworks mas utilizados.	✓
8.5	La aplicación detecta y responde al ser ejecutada en un emulador.	✓
8.6	La aplicación detecta y responde ante modificaciones de código o datos en su propio espacio de memoria.	✓
8.7	La aplicación implementa múltiples mecanismos de detección para los puntos del 8.1 al 8.6. Nótese que a mayor cantidad y diversidad de mecanismos usados, mayor la resistencia.	✓
8.8	Los mecanismos de detección disparan distintos tipos de respuestas, incluyendo respuestas retardadas y silenciosas.	✓
8.9	La ofuscación es aplicada a las defensas del programa, lo que a su vez impide la des-ofuscación mediante el análisis dinámico.	✓
	Atadura del Dispositivo	
8.10	La aplicación implementa un “enlace al dispositivo” utilizando una huella del dispositivo derivado de varias propiedades únicas al mismo.	✓
	Impedir la comprensión	
8.11	Todos los archivos ejecutables y bibliotecas correspondientes a la aplicación se encuentran cifrados, o bien los segmentos importantes de código se encuentran cifrados o empaquetados. De este modo el análisis estático trivial no revela código importante o datos.	✓
8.12	Si el objetivo de la ofuscación es proteger el código propietario, se utiliza un esquema de ofuscación que es apropiado tanto para la tarea particular como robusto contra los métodos de des-ofuscación manual y automatizada, considerando la investigación publicada actualmente. La eficacia del esquema de confusión debe verificarse mediante pruebas manuales. Tenga en cuenta que las características de aislamiento basadas en hardware son preferibles a la ofuscación siempre que sea posible.	✓

15. Anexo B. Instalación de MobSF

15.1 Análisis estático

15.1.1 REQUISITOS

Para llevar a cabo la instalación de la herramienta MobSF para análisis estáticos, seguimos las instrucciones proporcionadas en la documentación oficial de la página de GitHub/MobSF [30], para su instalación directamente sobre nuestro SO.

Se dispone para ello del siguiente entorno:

- SO Linux-4.10.0-38-generic-x86_64-with-LinuxMint-18.3-sylvia.
- Python 3.5
- JDK 1.7

15.1.2 DESCARGA

Descargamos la última versión de MobSF, en nuestro caso la v1.0.7 Beta, desde la siguiente dirección: https://github.com/MobSF/Mobile-Security-Framework-MobSF.

15.1.3 INSTALACIÓN

1. Copiamos y descomprimimos el MobSF descargado:

   /home/[username]/MobSF

2. A continuación, instalamos Python, mediante el siguiente comando:
   $sudo apt-get -y install python-pip

3. Finalmente, instalamos MobSF Python usando pip, desde la carpeta generada por el fichero descomprimido:
   $pip install -r requirements.txt

4. Si deseamos obtener el reporte de los resultados en versión PDF, debemos además instalar el binario wkhtmltopdf:
   $sudo apt-get -y install wkhtmltopdf

15.1.4 EJECUCIÓN

Desde la carpeta de MobSF, ejecutamos el servidor con el siguiente comando:

$sudo python3 manage.py runserver

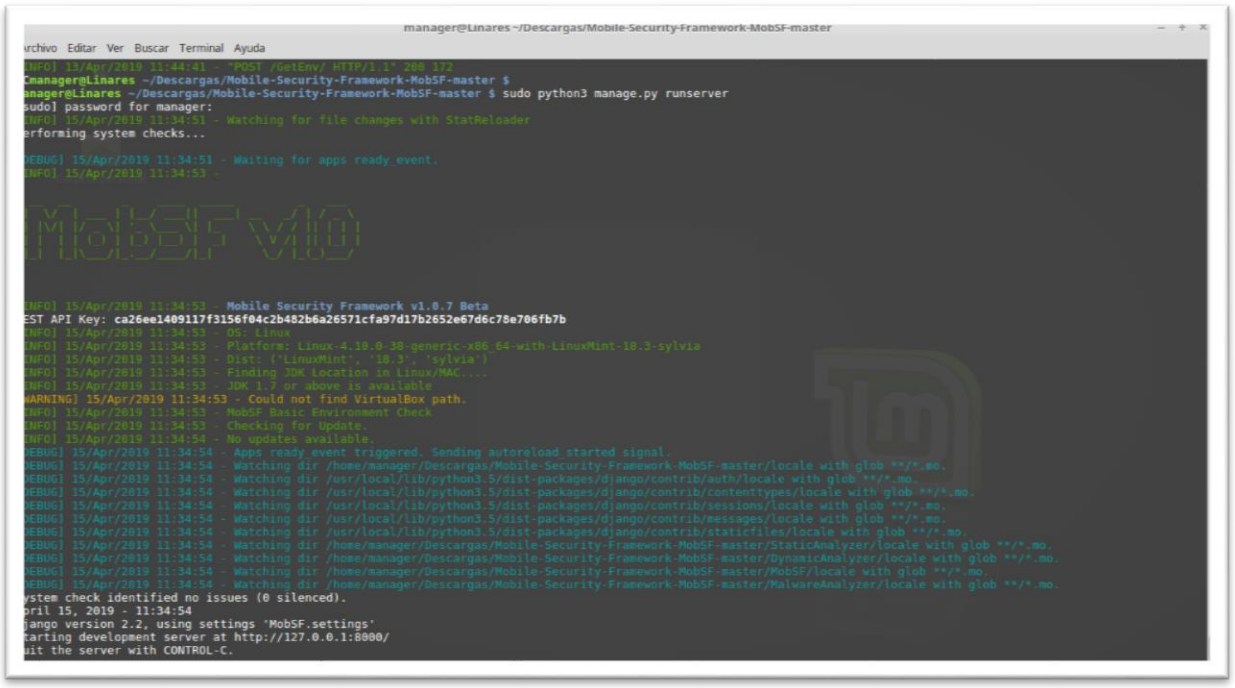


Imagen 50: Ejecución desde consola del servidor MobSF

Comprobamos que podemos acceder desde el navegador a la dirección: http://localhost:8000/ para acceder a la interfaz web de MobSF:

Imagen 51: Pantalla de inicio de MobSF

15.2 Análisis dinámico

15.2.1 REQUISITOS

Para completar la instalación de la herramienta MobSF para análisis dinámicos, seguimos las instrucciones proporcionadas en la documentación oficial de la página de GitHub/MobSF [33].

Se dispone para ello del siguiente entorno:

- Oracle VirtualBox v 5.1.38.
- Descargamos la máquina virtual para MobSF Android x86 4.4.2 (v0.3)

15.2.2 INSTALACIÓN Y CONFIGURACIÓN

1. Abrimos la máquina virtual desde VirtualBox, importando el archivo descargado MobSF_VM_X.X.ova.

2. Accedemos a la configuración de la máquina virtual para configurar los ajustes de Red, concretamente dos adaptadores de red:

- Adaptador 1: Lo configuramos como Sólo-anfitrión o Host-only y seleccionamos *vboxnet0* del desplegable de opciones.

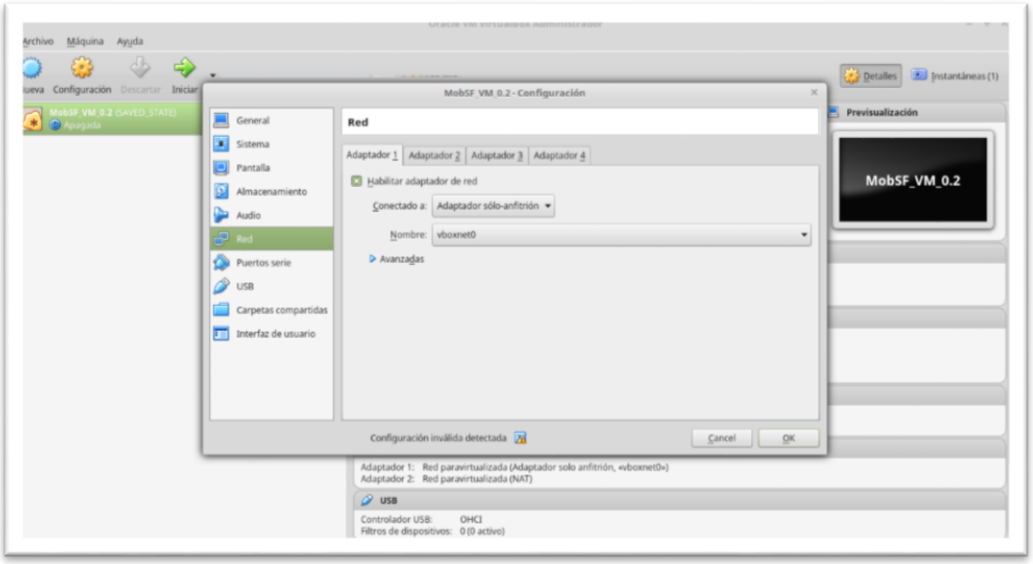


Imagen 52: Configuración Adaptador de red 1

- Adaptador 2: Lo habilitamos y configuramos para NAT:

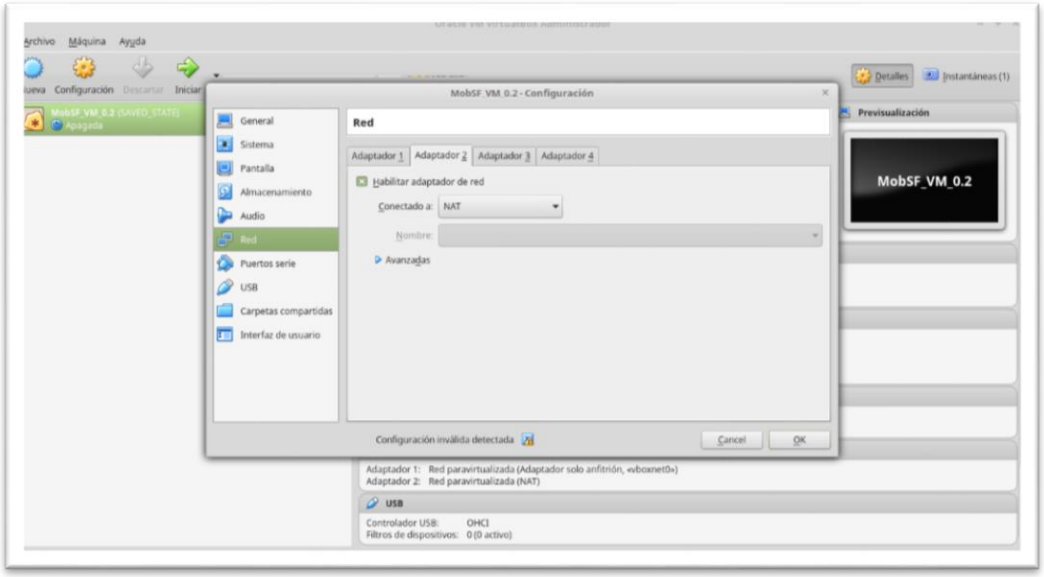


Imagen 53: Configuración de Adaptador de red 2

3. Arrancamos la máquina virtual MobSF. Durante el proceso de arranque podemos ver en pantalla la IP de nuestra máquina virtual, la cual debemos anotar para completar la configuración más adelante.

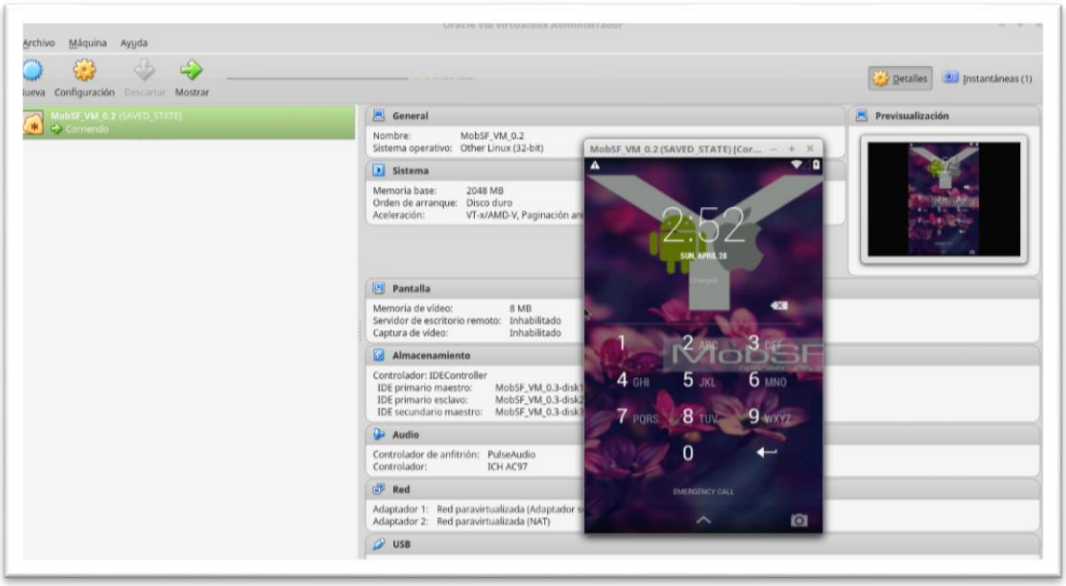


Imagen 54: MoBSF VM arrancado desde Oracle Virtual Box

4. Una vez tenemos nuestra Máquina virtual arrancada, accedemos a la configuración Wi-Fi para configurar los valores adecuados de Host/Proxy IP que se corresponden con la IP de nuestro host, y puerto número 1337.

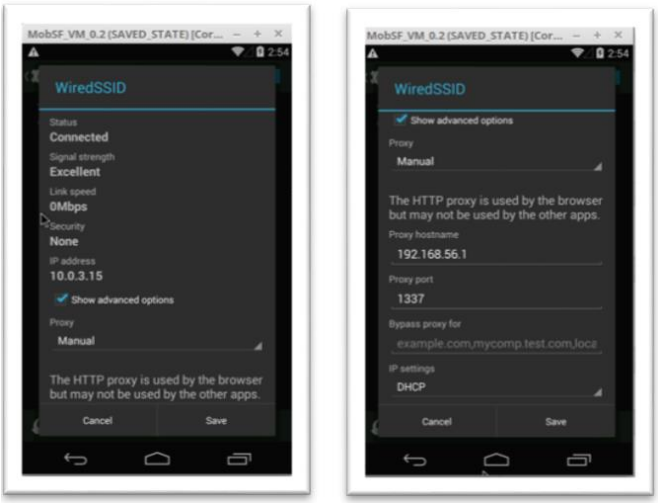


Imagen 55: Configuración Wi-Fi para MobSF VM

5. Editamos el archivo *MobSF_VM_X.X.vbox* para obtener los valores de VM UUID y Snapshot UUID, necesarios también para la configuración posterior.

6. Finalmente, para MobSF instalado localmente para el Análisis Estático, editamos el archivo MobSF/*settings.py* y configuramos los valores apropiados para vincular nuestra máquina virtual a la aplicación, con los valores que hemos ido apuntando durante el proceso de instalación, concretamente:

    - UUID = VM UUID
    - SUUID = Snapshot UUID
    - VM_IP = VM IP
    - PROXY_IP = Host/Proxy IP
    - ANDROID_DYNAMIC_ANALYZER = "MobSF_VM"

Imagen 56: Configuración del archivo settings.py de MobSF

15.2.3 EJECUCIÓN

Si hemos completado los pasos descritos anteriormente con éxito, el análisis dinámico nos funcionará sin problemas. Para ello, una vez ejecutado el análisis estático, desde el menú lateral o bien desde la sección “Options” hacemos click en el botón: *Start Dynamic Analysis*.

Se nos abrirá una segunda interfaz web que nos permitirá inicializar el ambiente de trabajo en primer lugar, lo cual configurará el proxy, instalará el APK a analizar y lo inicializará. Una vez el ambiente haya sido creado con éxito podremos ver e interactuar con la aplicación de manera normal desde nuestra máquina virtual para comenzar con el proceso de análisis dinámico.

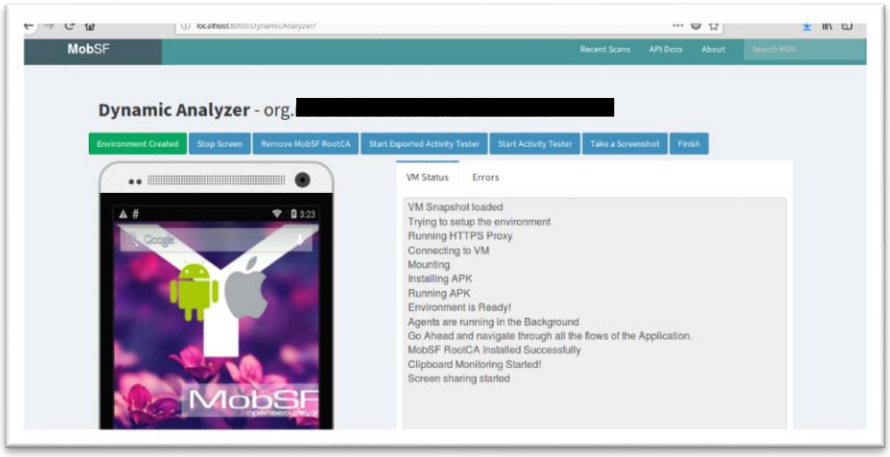


Imagen 57: Entorno de MobSF para análisis dinámico

16. Anexo C. Instalación de Inspeckage

16.1.1 REQUISITOS

Inspeckage (Android Package Inspector) es un marco de análisis para aplicaciones Android que funciona como un módulo de Xposed Installer y que permite ver en tiempo real los eventos que ocurren en el teléfono.

Para su instalación necesitamos disponer del siguiente entorno:

- Dispositivo Android Rooteado (en nuestro caso con Android v.8.1.0 API 27)
- Xposed Installer instalado en el dispositivo (v 3.1.5) con su Framework actualizado y activado (en nuestro caso versión 90-beta3), siguiendo los pasos a continuación:
    1. Descargamos el archivo .apk correspondiente desde su página oficial: *https://repo.xposed.info/module/de.robv.android.xposed.installer* y lo instalamos siguiendo las instrucciones del asistente.
    2. Ejecutamos el programa y procedemos a activar su Framework, encargado de procesar los distintos módulos, desde la opción Framework > Instalar/Actualizar.

16.1.2 INSTALACIÓN Y CONFIGURACIÓN

Podemos descargar Inspeckage directamente desde el repositorio Descargas de *Xposed* y una vez instalado, debemos habilitarlo desde la sección de Módulos, y reiniciar el dispositivo:

Imagen 58: Módulo Inspeckage habilitado en Xposed Installer

16.1.3 EJECUCIÓN

Desde Módulos, abrimos Inspeckage y seleccionamos la aplicación que deseamos analizar desde *choose target*:

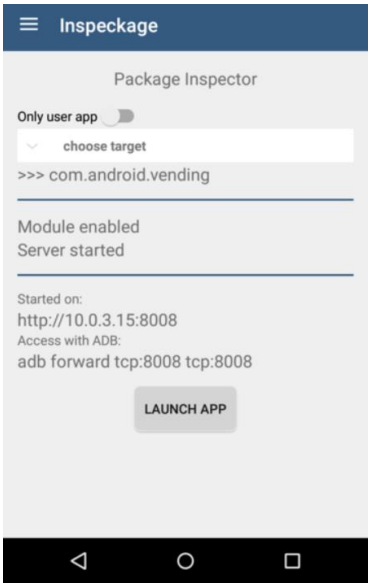


Imagen 59: Ventana principal de configuración de Inspeckage

Tras seleccionar la aplicación e iniciarla con el botón *Launch App*, comenzará el análisis dinámico de su funcionamiento.

Podremos acceder a los resultados generados desde el navegador del dispositivo en la IP y puerto indicados desde Inspeckage:

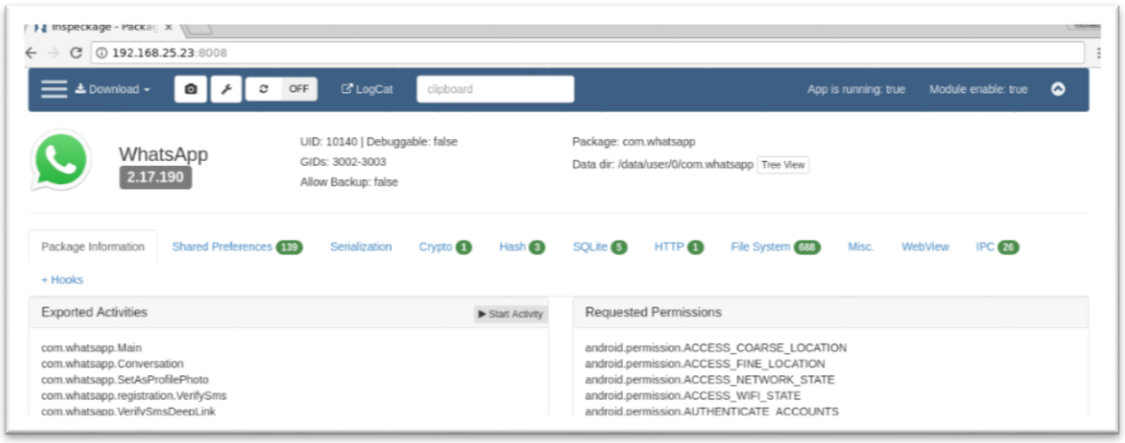


Imagen 60: Resultados del análisis dinámico ejecutado por Inspeckage

17. Anexo D. Instalación de OWASP ZAP

17.1.1 DESCARGA

Descargamos la última versión de OWASP ZAP, en nuestro caso la ZAP 2.7.0 Standard, desde [36].

17.1.2 CONFIGURACIÓN

Instalamos OWASP ZAP descargado anteriormente y lo inicializamos:

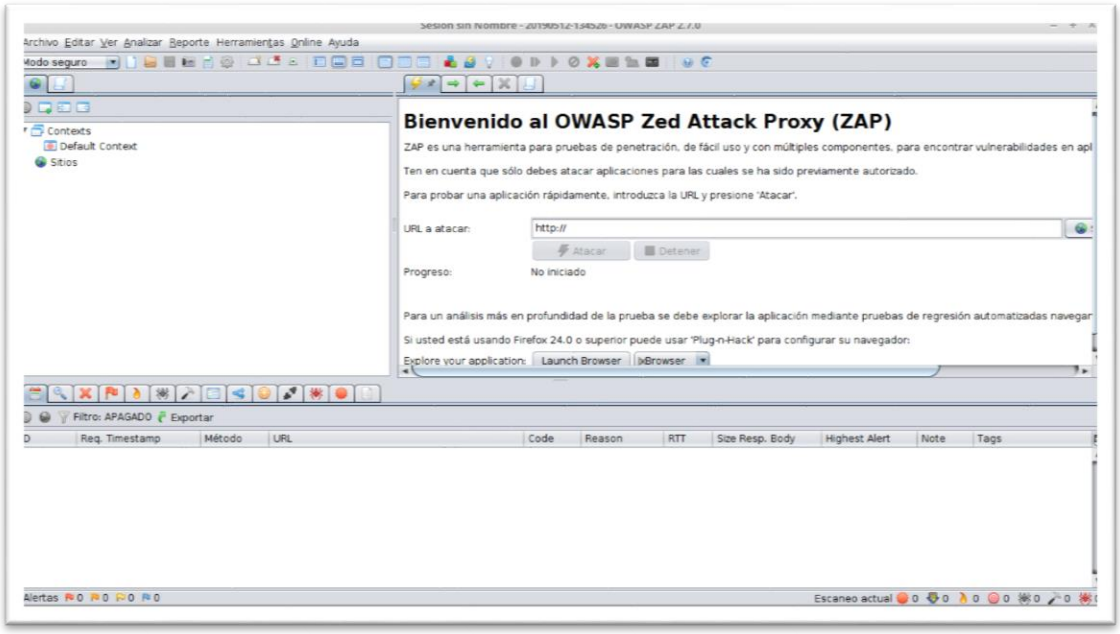


Imagen 61: Ventana de inicio de ZAP

Desde Herramientas > Opciones > Local Proxies, configuramos nuestra IP local y mantenemos el puerto indicado por defecto.

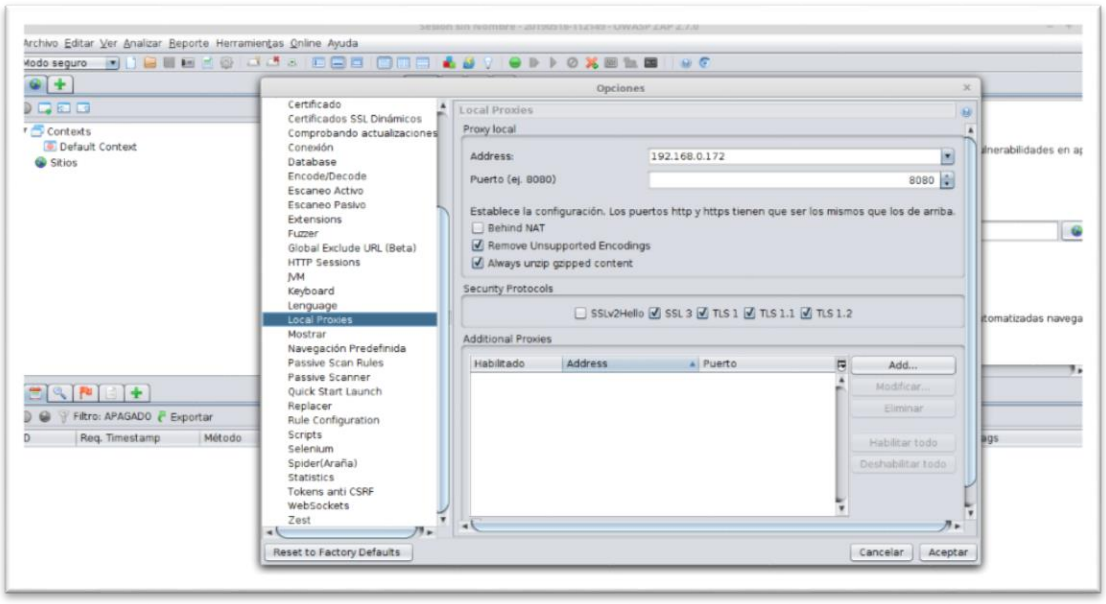


Imagen 62: Configuración de Proxy Local en ZAP

Análogamente en nuestro dispositivo, configuramos la conexión de red desde Ajustes > Wi-Fi, manteniendo seleccionada la red a la cual estamos conectados para modificar sus propiedades. Desde opciones avanzadas, configuramos un proxy manual con la IP y puerto configurado para ZAP:

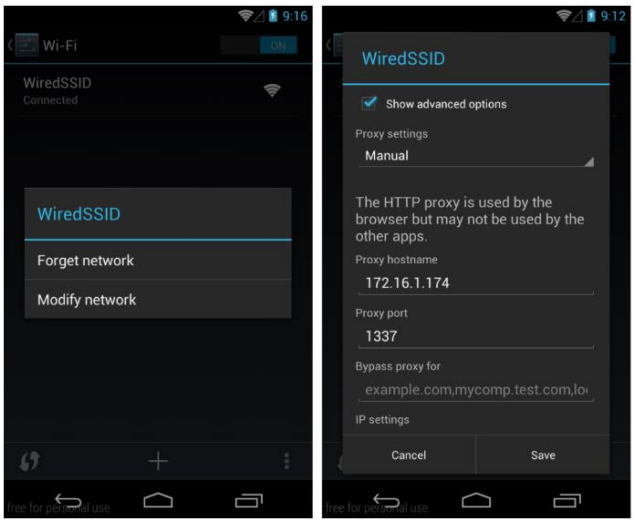


Imagen 63: Configuración de proxy manual en el dispositivo Android.

La configuración hasta este punto nos permite visualizar todo el tráfico HTTP, sin embargo, no podemos visualizar las conexiones protegidas. Para ello necesitamos exportar el certificado OWASP Raíz desde: Herramientas > Opciones > Certificados SSL Dinámicos:

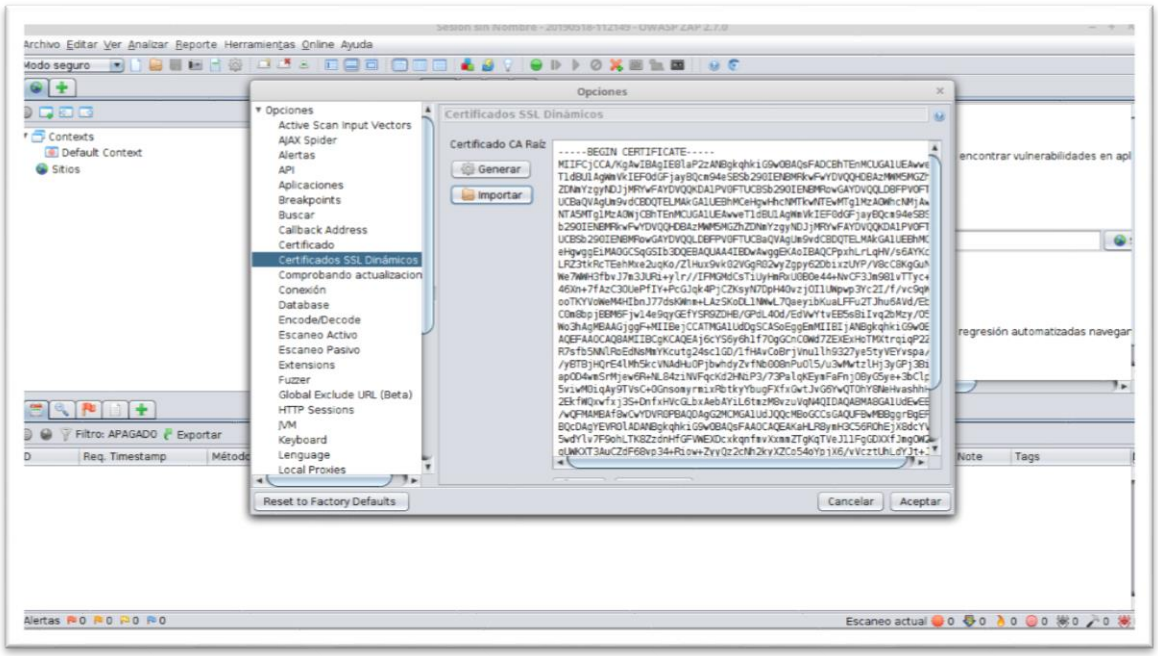


Imagen 64: Generación del Certificado CA Raíz.

Podemos guardar el certificado generado por OWASP que nos aparece en dicha ventana (válido durante un año desde la primera vez que se arrancó la herramienta) o generar uno nuevo. Guardamos el certificado y lo transferimos a nuestro dispositivo Android.

Desde el dispositivo, podemos instalar el certificado desde Ajustes >WiFi >Avanzado >Instalar Certificados y administrarlo desde Ajustes >Seguridad >Certificados de confianza y seleccionando la solapa Usuario:

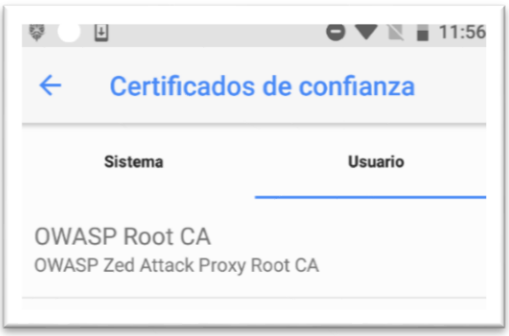


Imagen 65: Certificados de confianza instalados en el dispositivo

Comprobamos si al abrir la aplicación desde el dispositivo, ZAP comienza a monitorizar correctamente todo el tráfico HTTPS ya que, de lo contrario, es posible

que la aplicación cuente con validación de certificados o *SSL Pinning* y en ese caso, deberemos continuar con los pasos siguientes.

El *SSL* o *Certificate Pinning*, es una funcionalidad implementada en algunas aplicaciones móviles para proporcionar una capa extra de seguridad y que consiste en verificar que el servidor al que se está conectando la aplicación es el esperado.

Para poder capturar el tráfico HTTPS de la aplicación en este caso debemos evadir el SSL Pinning instalando el módulo *JustTrustMe* de *Xposed Installer* (ver apartado anterior para su instalación).

Podemos descargarlo desde el repositorio Descargas de *Xposed* y una vez instalado, debemos habilitarlo desde la sección de Módulos, y reiniciar el dispositivo:

Imagen 66: Módulo JustTrustMe habilitado en Xposed Installer

Una vez activado dicho módulo podemos comprobar que finalmente ZAP es capaz de monitorizar correctamente todo el tráfico HTTPS de la aplicación de estudio.

18. Anexo E. Informes

18.1.1 Resultados de análisis estático automatizado con MobSF

18.1.2 Reporte de análisis dinámico con Inspeckage

18.1.3 Reporte y sesiones de análisis con ZAP