Universidad de Valladolid

Escuela de Ingeniería Informática de Valladolid

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención Tecnologías de la Información

AuditSApp: Método para auditar la seguridad de una aplicación móvil

Autora:

Jiménez del Bosque, Teresa

Tutora:

Martínez González, María de las Mercedes

Agradecimientos

En primer lugar, agradecer a mis padres por ser mi apoyo y estar siempre ahí. Siempre habéis sido y seréis mi mayor inspiración. Esto es tan mío como vuestro.

En segundo lugar, agradecer a mi pareja, por estar presente durante todo este tiempo. Gracias por todo el apoyo y los ánimos, tanto en los momentos de felicidad como en los días más duros.

En tercer lugar, dar las gracias a todos los familiares y amigos que han estado durante todo este tiempo, tanto los que ya estaban como las nuevas amistades forjadas a lo largo de la carrera. Sin vosotros no hubiera sido lo mismo.

Por último, agradecer a mi tutora, Mercedes y a Alejandro, por todo el apoyo, los ánimos, correcciones, indicaciones y el conocimiento proporcionado a lo largo de este proyecto.

Resumen

En la actualidad, la privacidad de los datos de los usuarios se ha vuelto un aspecto sumamente relevante debido al aumento del uso de aplicaciones móviles y, al ataque de éstas por parte de terceros para extraer información de los usuarios o de la aplicación. Por este motivo se han redactado diferentes normativas que regulan la protección de los datos y diferentes guías que indican los pasos a llevar a cabo para detectar las vulnerabilidades existentes y, de ese modo, saber actuar en caso de que las exploten. Es por esto por lo que surge este proyecto, cuyo objetivo es detectar las vulnerabilidades presentes en una aplicación móvil. Para ello se realizará una búsqueda de las diferentes normativas y guías actuales relacionadas con el tema, posteriormente se elaborará el diseño de las pruebas que se lanzarán primero en un entorno controlado y después sobre la aplicación real. Una vez ejecutadas las pruebas, el siguiente paso es proporcionar una serie de medidas y recomendaciones para minimizar el impacto de las vulnerabilidades encontradas. Finalmente, se expondrán las conclusiones a las que se ha llegado a lo largo del desarrollo del proyecto.

Abstract

Nowadays, the privacy of user data has become an extremely relevant aspect due to the increase in the use of mobile applications and the attacks on them by third parties to extract information from users or from the application. For this reason, different regulations have been drafted to regulate data protection and different guides that indicate the steps to be taken to detect existing vulnerabilities and, thus, to know how to act in case they are exploited. This is the reason for this project, the aim of which is to detect the vulnerabilities present in a mobile application. To do this, a search of the different regulations and current guides related to the subject will be carried out, after which the design of the tests will be developed, which will be launched first in a controlled environment and then on the real application. Once the tests have been executed, the next step is to provide a series of measures and recommendations to minimise the impact of the vulnerabilities found. Finally, the conclusions reached during the development of the project will be presented.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto

En la actualidad el uso de los dispositivos móviles se ha vuelto algo rutinario y prácticamente indispensable. El uso de los datos que estas aplicaciones manejan es elevado y su tendencia sigue en aumento. Una gran parte de estos datos son aquellos que se denominan como *datos de carácter personal* de los usuarios que en ocasiones se recogen sin que lo usuarios tengan conocimiento, y por ende, lo autoricen. Es por ello por lo que diferentes organizaciones han desarrollado normativas con la regulación del uso de estos datos como puede ser el *Reglamento General del Protección de Datos* (**RGPD**) [1] a nivel europeo, como la *Ley orgánica de Protección de Datos Personales y Garantías de Derechos Digitales* (**LOPDGDD**) [2] a nivel nacional, donde se requiere que la información de los usuarios se proteja para que no se pueda identificar a un individuo por medio de esos datos, a menos que exista un consentimiento expreso para ello por parte de los implicados o haya un interés legítimo.

Aunque hacer que las aplicaciones no muestren de forma directa estos datos de carácter personal es importante para la seguridad de la información, en ocasiones existen una serie de vulnerabilidades dentro de dichas aplicaciones que permitan que terceros puedan acceder a estos datos. Motivo por el que hay que contar con una serie de medidas de seguridad.

El presente proyecto busca encontrar las diferentes vulnerabilidades de seguridad que pueda tener una aplicación móvil que se encuentra accesible para todos los usuarios desde Google Play y, en caso de encontrarlas, proponer unas medidas para mitigarlas o eliminarlas.

1.2. Motivación

Hoy en día la privacidad, y en especial la privacidad de la información, se ha convertido en un aspecto de suma importancia ya que con solo unos pocos datos es posible identificar a una persona. Según las diferentes normativas de seguridad, aquellos datos que permiten identificar a un usuario deben protegerse para que personas no autorizadas o con intereses ilegítimos no puedan acceder a ellos.

El hecho de que un atacante obtenga datos personales de otro usuario puede conllevar, entre otros, suplantaciones de identidad, robo de información, etc; y, con el uso de las aplicaciones móviles estos ataques han aumentado considerablemente. Aunque este incremento sea evidente y esté a la orden del día, aún existen empresas que desarrollan aplicaciones móviles que ignoran este suceso o la propia seguridad básica pensando que el ataque llegará de una forma sofisticada y complicada.

La motivación de este proyecto viene dada por lo ya mencionado anteriormente, con el fin de demostrar de una forma sencilla si una aplicación no se encuentra bien protegida, lo que quiere decir de forma simple si puede obtener información sensible. Por otro lado, se suma el espíritu por aprender cómo se tratan los datos personales en un entorno real y actual en el que la seguridad informática debe considerarse un pilar fundamental.

1.3. Objetivos

El principal objetivo de este proyecto es proponer una metodología para auditar una aplicación móvil sobre la que se llevará a cabo una búsqueda de vulnerabilidades que puedan afectar a la seguridad de dicha aplicación. A continuación se explican los objetivos de forma más detallada:

- Llevar a cabo un estudio de las normativas que rigen las medidas de seguridad que se deben implementar en un aplicación móvil, así como el de las guías de auditoría de seguridad hacia dispositivos móviles.
- Proponer una metodología de trabajo para llevar a cabo una auditoría sobre una aplicación móvil.
- Diseñar y lanzar sobre una aplicación móvil una auditoría de seguridad con el fin de detectar las vulnerabilidades que pueda contener y proponer una salvaguarda para éstas.

1.4. Organización del documento

- **Capítulo 1. Introducción:** Breve descripción del proyecto junto con los aspectos fundamentales y los objetivos de éste.
- **Capítulo 2. Planificación del proyecto:** Incluye todo lo relacionado con la planificación inicial en cuanto a horas para el desarrollo del proyecto y las fechas previstas para la realización de cada una de las tareas del proyecto junto a los riesgos relacionados.
- **Capítulo 3. Estado del arte:** Recopilación de información sobre la actualidad en lo que a normativas de protección de datos y las metodologías para realizar auditorías de seguridad sobre aplicaciones móviles.
- **Capítulo 4. Análisis de la aplicación:** Estudio de la funcionalidad de la aplicación a auditar.
- **Capítulo 5. Análisis de la metodología OWASP:** Exploración de la metodología OWASP, donde se profundiza en sus categorías y se relacionan con los principios de seguridad informática.
- **Capítulo 6. Diseño:** Se muestran los criterios utilizados para la selección de las pruebas a lanzar sobre la aplicación final y el diseño de cada una de las pruebas.

- **Capítulo 7. Lanzamiento de las pruebas sobre InsecureBankv2:** Análisis de la aplicación vulnerable *InsecureBankv2* [3], configuración de ésta en su entorno controlado y lanzamiento de las pruebas para verificar que se encuentran bien diseñadas.
- **Capítulo 8. Lanzamiento de las pruebas sobre AquaCyL:** Configuración del entorno controlado sobre el que se lanzan las pruebas y explicación del proceso de lanzamiento de pruebas sobre la aplicación final, resultados del lanzamiento y recomendaciones en caso de encontrarse fallos de seguridad o vulnerabilidades.
- **Capítulo 9. Conclusiones:** Reflexión de los resultados obtenidos tanto para las pruebas como para el proyecto en general. Se incluye una propuesta de trabajo futuro sobre la que se podría ampliar el alcance del proyecto.
- **Anexo I. Pruebas para realizar una auditoría de seguridad móvil según la metodología OWASP:** Información con todas las pruebas que OWASP propone para realizar una auditoría de seguridad móvil junto a una breve descripción de cada una, si está en uso o no y si es viable para su lanzamiento sobre AquaCyL.

Capítulo 2

Planificación del proyecto

En el presente capítulo se muestra la planificación desarrollada para el proyecto. Para ello se ha realizado un análisis de sus características, se ha seleccionado una metodología a emplear y se ha desarrollado una planificación inicial de la que se han extraído los riesgos que pueden tener lugar durante el desarrollo de éste. Existen múltiples metodologías para el desarrollo de proyectos, como pueden ser el modelo en cascada o el modelo en espiral. En el libro *Software project management.* [4], aparecen dichas metodologías y los diferentes motivos por los que seleccionar una u otra en base a las necesidades de cada proyecto. En los siguientes apartados, se explicará el porqué y la metodología seleccionada para el actual proyecto.

2.1. Características del proyecto

El proyecto que se presenta tiene como objetivos los ya mencionados anteriormente en la sección 1.3. Al tratarse de un proyecto hay que seguir una serie de pasos para que este se complete satisfactoriamente. A continuación se muestra un pequeño análisis de las etapas que conformarán el proyecto:

- **Búsqueda:** En la que se exploran las diferentes normativas existentes relacionadas con el tema, así como las guías disponibles y las vulnerabilidades actuales.
- **Estado del arte:** Donde se profundiza en las normativas y guías de seguridad encontradas que mejor se adaptan a lo requerido para explotar el máximo número de vulnerabilidades.
- **Análisis de la aplicación:** En la que se investiga el tratamiento de los datos y se estudia su funcionamiento.
- **Diseño:** En la que se seleccionarán las pruebas a llevar a cabo de modo que se pueda detectar el mayor número de vulnerabilidades posible. Para ello se seguirá la metodología propuesta por OWASP, más concretamente La guía Mobile Application Securiry Testing Guide (MASTG).
- **Lanzamiento de pruebas:** Donde se evaluarán las pruebas obtenidas en el paso anterior. Primero en un entorno controlado para asegurar que todo se lleve a cabo de la forma esperada, y, más adelante sobre la aplicación seleccionada. Posteriormente se valora la información obtenida de las pruebas y se analizan las vulnerabilidades encontradas. En base a lo anterior se recomiendan diferentes medidas para securizar la aplicación móvil eliminando o mitigando las vulnerabilidades detectadas.

- **Conclusiones:** En la que se estudian los resultados obtenidos una vez realizado el proyecto y se propone trabajo futuro para ampliar su alcance.

2.2. Metodología empleada

Una vez contempladas las características del proyecto, el siguiente paso es seleccionar una metodología para el desarrollo de éste. En este caso se ha optado por un **modelo en espiral** [5], lo que implica que se establezcan una serie de hitos. De este modo cuando se cumpla cada hito, éste se someterá a revisión periódicamente.

Con este método, al tratarse de un bucle en espiral, cada uno de los hitos pueden revisarse en cada iteración que se produzca. Por tanto, es más inmediato reparar en errores y solucionarlos.

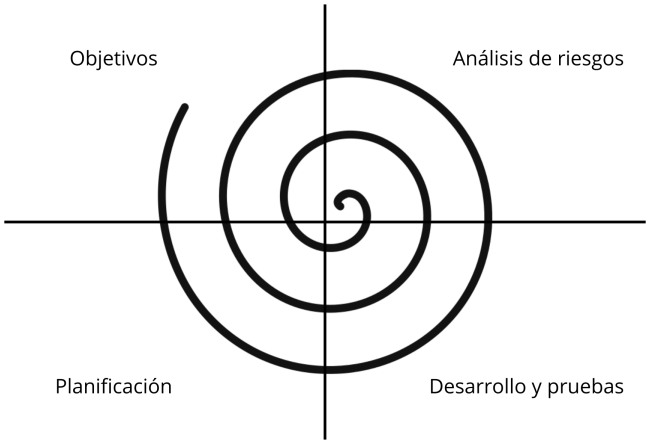

Figura 2.1: Estructura del modelo en espiral.

2.3. Planificación inicial

La planificación del proyecto se ha realizado para que se lleve a cabo entre el **10 de febrero de 2025** y el **30 de mayo de 2025**. Puesto que durante un periodo del cuatrimestre se realizan las prácticas de empresa dispuesta en el plan de estudios, la distribución horaria se ha dividido en dos grupos: el primero entre el 10 de febrero y el 10 de abril, y el siguiente, entre el 11 de abril y el 30 de mayo hasta que se completen las 300 horas en las que se reparte el proyecto, tal y como se muestra a continuación:

Periodo	**Horario**	**Horas**
Del 10/02 al 10/04	L-M-X-V	2
	S-D	3,5
Total		**15 horas/semana**

Tabla 2.1: Horas de trabajo en el primer periodo

Periodo	Horario	Horas
Del 11/04 al 30/05	L-M-X-V	4,5
	S-D	3,5
Total		**25 horas/semana**

Tabla 2.2: Horas de trabajo en el segundo periodo

Para contar con una planificación más detallada de las diferentes tareas, se ha realizado el Diagrama de Gantt [6] que se muestra a continuación:

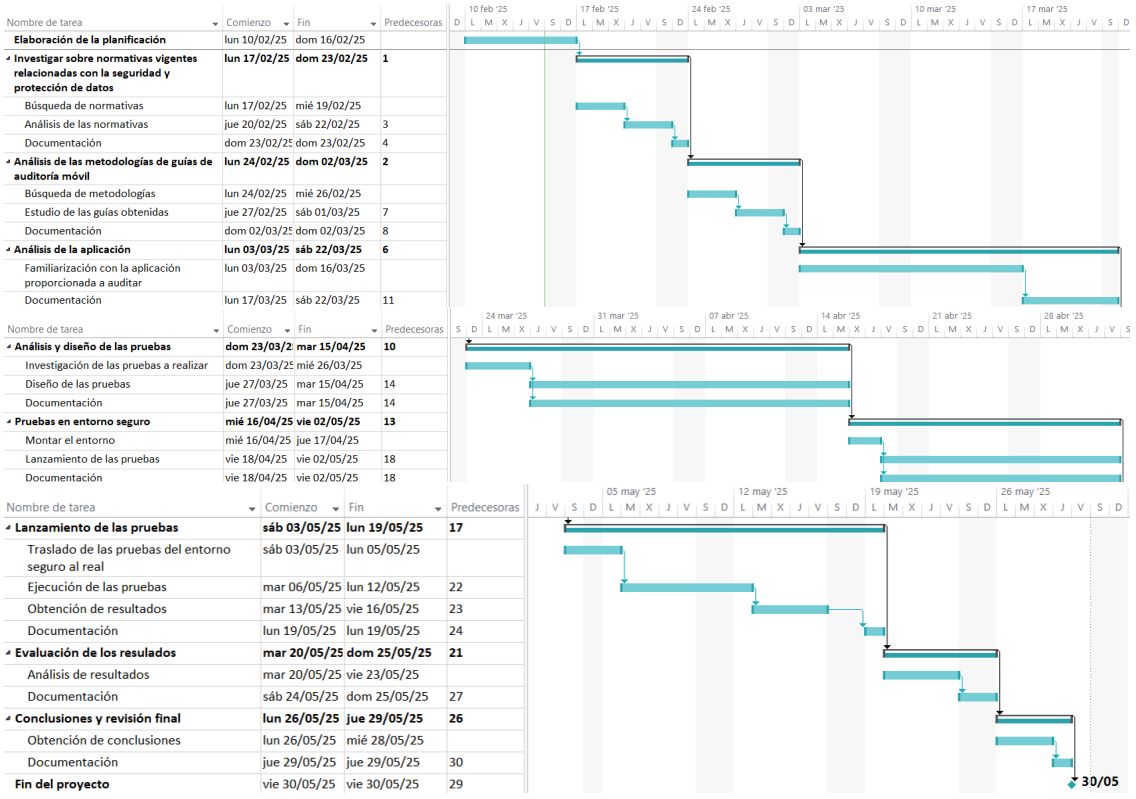

Figura 2.2: Diagrama de Gantt con las actividades del proyecto.

En lo que al anterior diagrama respecta, se puede ver cómo las diferentes tareas se subdividen en diferentes actividades de una determinada duración. Cada una de las tareas cuentan con una actividad de documentación en la que plasmará en el presente documento todos los resultados y conclusiones obtenidos a lo largo del desarrollo del proyecto.

Una vez definidas las actividades y su planificación a lo largo del tiempo, se han obtenido los diferentes hitos a llevar a cabo a lo largo del proyecto:

Hito	Fecha planificada
Planificación realizada	16/02/2025
Adquisición del conocimiento de las normativas	23/02/2025
Descubrimiento de las metodologías necesarias	02/03/2025
Familiarización con la aplicación	22/03/2025
Diseño de pruebas y configuración del entorno seguro	17/04/2025
Detección de vulnerabilidades en la aplicación	19/05/2025
Evaluación de vulnerabilidades obtenidas según criticidad	23/05/2025
Propuesta de medidas de securización	25/05/2025
Finalización del proyecto	30/05/2025

Tabla 2.3: Hitos a llevar a cabo en el transcurso del proyecto

2.4. Riesgos

Una vez descritas las diferentes actividades, hay que realizar un análisis de los riesgos que puedan poner en peligro el éxito del proyecto.

Para realizar dicho análisis se ha utilizado una **matriz de riesgos** de modo que cada uno de los riesgos se ha cuantificado según la métrica **probabilidad x impacto**. Esto quiere decir que dependiendo del valor asignado a la probabilidad de que un riesgo tenga lugar, y del impacto asociado a la ocurrencia del riesgo, se determina la prioridad de dicho riesgo.

		Probabilidad				
		Muy poco frecuente	Poco frecuente	Normal	Frecuente	Muy frecuente
Impacto	Muy bajo	Bajo	Bajo	Bajo	Medio	Medio
	Bajo	Bajo	Bajo	Medio	Medio	Medio
	Medio	Medio	Medio	Medio	Alto	Alto
	Alto	Medio	Medio	Alto	Alto	Muy alto
	Muy alto	Medio	Alto	Alto	Muy alto	Muy alto

Figura 2.3: Matriz de riesgos

Una vez identificado y cuantificado un riesgo, hay qué decidir que medidas se van a llevar a cabo con él. Éstas pueden ser *evitarlo*, *reducirlo*, *aceptarlo* o *transferirlo*.

Con lo anterior se han identificado los siguientes riesgos:

Riesgo	**Probabilidad**	**Impacto**	**Valor**
Mala estimación de los plazos	Normal	Alto	Alto
Retraso en la fecha de finalización	Frecuente	Muy alto	Muy alto
Sucesos extraordinarios que no permitan la realización del proyecto (*p.e.* enfermedad)	Poco frecuente	Alto	Medio
Incumplimiento de los horarios establecidos	Normal	Alto	Alto
No contar con el material necesario, (*p.e.* averías en el equipo personal)	Muy poco frecuente	Alto	Medio
Cambio de objetivos	Poco frecuente	Alto	Medio
No disponer de la aplicación a auditar	Poco frecuente	Muy alto	Alto
Aparición de riesgos no contemplados	Poco frecuente	Muy alto	Medio
Diseño incorrecto de las pruebas	Normal	Muy alto	Alto

Tabla 2.4: Riesgos identificados y su valor cuantificado.

Para cada uno de los riesgos se han propuesto las siguientes medidas para minimizar el impacto en lo que al proyecto respecta:

- **Mala estimación de los plazos:** A lo largo de la realización del proyecto, si se detectase que se ha realizado una estimación incorrecta de los plazos, la solución podría ser hacer más horas de las establecidas. Si la estimación incorrecta se detectase al final de un entregable, la forma de actuar sería en función del tipo de tares. Si se trata de una tarea del camino crítico, ésta se realizaría de forma secuencial con la mayor antelación posible. Si por el contrario no es una tarea del camino crítico, se valoraría desarrollarla en paralelo a la tarea que se estuviera realizando en ese momento.

- **Retraso en la fecha de finalización:** Hablar de la situación con la tutora y barajar la posibilidad de entregarlo en la convocatoria extraordinaria.

- **Sucesos extraordinarios que no permitan la realización del proyecto (p.e. enfermedad):** Evaluar cómo suceden los hechos y aumentar el número de horas posteriormente.

- **Incumplimiento de los horarios establecidos:** Si se detectase que los horarios establecidos no se adaptan a las circunstancias personales, éstos se reajustarían.

- **No contar con el material necesario (p.e. averías en el equipo personal o no disponer del software necesario):** Contar con copias de seguridad en la nube y trabajar desde otro dispositivo. Buscar alternativas de software que permitan el desarrollo del proyecto.

- **Cambio de objetivos:** Si a lo largo del desarrollo del proyecto se produce un cambio de los objetivos, éstos se integrarán de modo que afecten lo menos posible a la finalización del proyecto

- **No disponer de la aplicación a auditar:** Buscar otra aplicación para llevar a cabo la auditoría.

- **Aparición de riesgos no contemplados:** Actuar según la guía de riesgos del capítulo 7 del libro [4].

- **Diseño incorrecto de las pruebas:** Validarlo con la tutora.

2.5. Seguimiento de la planificación.

En el transcurso del desarrollo del presente proyecto, el seguimiento no ha seguido la planificación inicial tal y como se esperaba. Aunque se diseñó teniendo en cuenta los posibles riesgos, siendo uno de ellos el retraso del proyecto, por diversos motivos no se ha llegado a seguir dicha planificación.

A lo largo de los primeros meses, se avanzó de una forma más lenta. No fue hasta finales de abril que se produjo un cambio significativo, donde se realizó una reorganización del trabajo y se comenzó a trabajar de una forma más intensa y de forma regular.

El cambio de dinámica hizo que se recuperase el tiempo que se perdió al inicio del proyecto, de modo que se realizó la selección y diseño de las pruebas.

Una vez realizado lo anterior, se procedió a lanzar las pruebas tanto en una aplicación vulnerable como en AquaCyL, se analizaron los resultados y se mencionaron ciertas recomendaciones de seguridad.

Por último se finalizó el proyecto con las conclusiones, dándolo por acabado.

Gracias a la reorganización que se produjo, el proyecto se ha podido concluir, aunque más tarde de lo estimado, aún en plazo de presentación, realizándose entre el **10 de febrero** y el **1 de julio**.

A continuación se muestra una tabla en la que se explica de una forma más detallada el desarrollo del proyecto en relación a las fechas en las que han tenido lugar realmente:

Tarea	**Fecha de finalización**
Elaboración de la planificación	20/02/2025
Análisis de normativas y metodologías existentes	05/03/2025
Estudio de la aplicación	20/03/2025
Análisis de la metodología OWASP	27/03/2025
Diseño de las pruebas	04/05/2025
Configuración del entorno controlado	12/05/2025
Lanzamiento de las pruebas sobre la aplicación vulnerable	31/05/2025
Lanzamiento de las pruebas sobre la aplicación final	28/06/2025
Conclusiones	01/07/2025

Tabla 2.5: Seguimiento del proyecto.

Capítulo 3

Estado del arte

En el capitulo actual se lleva a cabo un análisis de las condiciones de privacidad y seguridad que las aplicaciones móviles deben cumplir según las normativas de privacidad de datos actuales. También se realiza una breve descripción de las metodologías actuales relacionadas con las auditorías de seguridad para aplicaciones móviles más usadas en la actualidad, y cómo mediante ellas se pueden detectar vulnerabilidades que pueden poner en riesgo aspectos como los datos personales de los usuarios.

3.1. Datos personales

Según el diccionario panhispánico del español jurídico, elaborado por la Real Academia Española (RAE) [7], se define como dato personal todo tipo de información ya sea alfabética, numérica, gráfica, acústica, fotográfica o de cualquier otro tipo que concierna a personas físicas identificadas o identificables [8].

En base a la anterior definición, se consideran datos personales los siguientes:

- Nombre y apellidos.
- DNI, pasaporte y NSS[1].
- Datos bancarios.
- Datos médicos.
- Dirección.
- Datos genéticos o biométricos.
- Certificados electrónicos y firma.
- Datos personales que revelen el origen racial o étnico.
- Opiniones políticas.
- Convenciones religiosas o filosóficas.
- Afiliación sindical

[1] Número Seguridad social.

Algunos de los anteriores datos personales, se consideran como sensibles y por tanto deben tratarse de una forma específica como por ejemplo los datos médicos o aquellos relacionados con la información genética o biométrica, los cuales se pueden usar únicamente con el fin de identificar a un ser humano.

Por otro lado, hay que tener en cuenta que el tratamiento de los datos personales no es igual para un adulto que para un menor de edad [9]. Para poder recabar datos personales sobre estos últimos se tienen que tener en cuenta algunas consideraciones:

- Los datos pueden ser recabados siempre que se sigan las normas establecidas en el RGPD relacionadas con la recolección de datos personales.
- Para recoger datos de menores es necesario que el consentimiento sea expreso y que, si el menor tiene 14 años o menos, ese consentimiento debe ser aportado por sus padres o tutores legales.

3.2. Normativas de protección de datos: RGPD y LOPDGDD.

En la actualidad son dos las principales fuentes reguladoras en lo que a protección de datos se refiere, el *Reglamento General de Protección de Datos* (**RGPD)**, con alcance europeo y la *Ley orgánica de Protección de Datos Personales y Garantías de Derechos Digitales* **LOPDGDD** a nivel nacional, cuyo objetivo es la adaptación de la legislación española a la europea.

Dentro del *RGPD* se definen una serie de principios que los responsables del tratamiento de datos personales deben tener en cuenta:

- **Principio de licitud, transparencia y lealtad**, referido a que los datos deben tratarse siguiendo la ley y de forma transparente para el usuario interesado.
- **Principio de minimización de datos**, aplicar las medidas necesarias para garantizar que el tratamiento de los datos sea únicamente el preciso para el fin especificado.
- **Principio de limitación del plazo de conservación**, determina que la conservación de los datos debe limitarse en el tiempo hasta que se logren los fines por los que se persigue el tratamiento. Tras alcanzar dichas finalidades, los datos deben borrarse, bloquearse o anonimizarse para evitar que se pueda identificar a los interesados.
- **Principio de responsabilidad activa o responsabilidad demostrada**, obliga a los responsables del tratamiento de los datos a mantener dirigencia permanente para proteger y garantizar los derechos y libertades de las personas cuyos datos son tratados por medio de un análisis de los riesgos que el tratamiento supone y representa para dichos derechos y libertades. El responsable debe poder garantizar y estar en condiciones de demostrar que el tratamiento se ajusta dentro de lo que se rige en el *RGPD* y en la *LOPDGDD*.
- **Principio de seguridad**, que obliga a los responsables del tratamiento de los datos el análisis de riesgos necesario orientado a determinar las medidas organizativas y técnicas requeridas para garantizar la integridad, confidencialidad y disponibilidad de los datos que se traten.
- **Principio de exactitud**, impone a los responsables a contar con medidas para que los datos estén actualizados, se eliminen o modifiquen cuando sean inexactos respecto a los fines por los que se tratan.

- **Principio de finalidad**, implica la obligación de que los datos se traten para finalidades determinadas, legítimas y explícitas y prohíbe que los datos recogidos para un fin se utilicen posteriormente de una manera incompatible con estos fines.

Con los anteriores principios se determina como deben de tratarse los datos de los usuarios, si dentro de una aplicación móvil se encontrase que estos principios no se cumplen, se estaría violando el RGPD por lo que es un aspecto a tener en cuenta a la hora de realizar la auditoria móvil.

3.3. Metodologías de pruebas de auditoría de aplicaciones móviles.

Para poder detectar e identificar las diferentes situaciones en las que se ponga en riesgo la privacidad y los datos personales de los usuarios en una aplicación móvil, es necesario lanzar una serie de pruebas sobre ellas. Es por este motivo por el que algunas organizaciones han desarrollado una serie de metodologías que rigen los pasos para lanzar estas pruebas y que a día de hoy se han vuelto estándares. A continuación se hace un estudio de diferentes metodologías relacionadas con las auditorías de seguridad en aplicaciones móviles.

3.3.1. OWASP

El proyecto Abierto de Seguridad de Aplicaciones Web (**OWASP**, *The Open Source Web Application Security Project* es un fundación sin animo de lucro que ofrece metodologías para que el software sea seguro.

Dentro de OWASP, se define **MAS**, el proyecto para la seguridad de las aplicaciones móviles, donde se proporciona un estándar de seguridad para las aplicaciones móviles **OWASP MASVS** y una guía de pruebas **OWASP MASTG** que explica las técnicas y herramientas y pruebas para evaluar la seguridad de las aplicaciones móviles.

OWASP Mobile Top 10.

OWASP recopila las 10 vulnerabilidades más explotadas en *OWASP Mobile Top 10* [10], donde explica cada vulnerabilidad, cómo se produce y cómo evitarla. La última publicación se produjo en 2024, donde, a parte de las nuevas vulnerabilidades, se muestra una comparación entre el Top 10 de la anterior publicación, 2016, y la nueva:

Comparison Between 2016-2024		
OWASP-2016	**OWASP-2024-Release**	**Comparison Between 2016-2024**
M1: Improper Platform Usage	M1: Improper Credential Usage	New
M2: Insecure Data Storage	M2: Inadequate Supply Chain Security	New
M3: Insecure Communication	M3: Insecure Authentication / Authorization	Merged M4&M6 to M3
M4: Insecure Authentication	M4: Insufficient Input/Output Validation	New
M5: Insufficient Cryptography	M5: Insecure Communication	Moved from M3 to M5
M6: Insecure Authorization	M6: Inadequate Privacy Controls	New
M7: Client Code Quality	M7: Insufficient Binary Protections	Merged M8&M9 to M7
M8: Code Tampering	M8: Security Misconfiguration	Rewording [M10]
M9: Reverse Engineering	M9: Insecure Data Storage	Moved from M2 to M9
M10: Extraneous Functionality	M10: Insufficient Cryptography	Moved from M5 to M10

Figura 3.1: Comparación OWASP Top 10 2024-2016.

En la anterior figura se puede ver cómo entre 2016 y 2024, la mayoría de las vulnerabilidades son nuevas, aunque siguen existiendo algunas que, o se han fusionado, o se han movido en la clasificación.

A continuación, se detalla cada uno de los riesgos clasificados en OWASP Top 10 en el año 2024 y como evitarlos:

M1: Uso inadecuado de credenciales. [11]

Factor	**Descripción**
Agentes de amenaza	Los agentes que explotan credenciales pueden incluir ataques automatizados. Estos ataques pueden explotar credenciales o aprovechar vulnerabilidades.
Vectores de ataque	Los atacantes pueden explotar credenciales codificadas así como el uso indebidos de éstas.
Debilidad de seguridad	Usar credenciales codificadas junto a un manejo inadecuado puede causar vulnerabilidades de seguridad graves.
Impactos técnicos	Que un usuario acceda a información confidencial de una aplicación puede causar filtraciones de datos y actividades fraudulentas.
Mitigación	▪ **Evitar el uso de credenciales codificadas**: Un atacante podría descubrirlas fácilmente y acceder a datos de los usuarios. ▪ **Manejar adecuadamente las credenciales de usuario**: Las credenciales deben almacenarse, transmitirse y autenticarse de forma segura.

Tabla 3.1: Descripción de cada factor para el riesgo *M1: Uso inadecuado de credenciales.*

M2: Seguridad inadecuada de la cadena de suministro. [12]

Factor	**Descripción**
Agentes de amenaza	El atacante puede manipular la funcionalidad de la aplicación por medio de vulnerabilidades dentro de la cadena de suministro de la aplicación. Esto puede permitir a dicho atacante robar datos o incluso tomar el control del dispositivo móvil.
Vectores de ataque	Un atacante puede inyectar código malicioso en la fase de desarrollo de la aplicación y posteriormente comprometer las claves de firma para firmar el código maliciosos como confiable.

Factor	Descripción
Debilidad de seguridad	Esta vulnerabilidad tiene lugar por la falta de prácticas de codificación seguras, revisiones de código y pruebas insuficientes de forma que se incluyen vulnerabilidades en la aplicación sin saberlo cuando con los anteriores mecanismos se podrían evitar.
Impactos técnicos	▪ **Filtración de datos:** El atacante puede robar datos personales como credenciales. Si filtrase esos datos, las personas a las que se las ha robado la información podrían sufrir suplantaciones de identidad. ▪ **Infección de malware:** Si el atacante introduce malware en la aplicación, el dispositivo en el que se instale puede verse gravemente perjudicado como consecuencia de la ejecución del código malicioso sobre el dispositivo. ▪ **Acceso no autorizado:** El atacante puede acceder al servidor de la aplicación y realizar actividades no autorizadas como la denegación del servicio del dispositivo atacado. ▪ **Compromiso del sistema:** Comprometer la aplicación móvil puede suponer la pérdida total de control del sistema lo que puede suponer una pérdida permanente de los datos y un daño a largo plazo sobre la reputación del desarrollador de la aplicación móvil.
Mitigación	▪ Establecer controles de seguridad de forma periódica para buscar vulnerabilidades o fallos de seguridad. ▪ Monitorizar para detectar incidentes de seguridad mediante pruebas de seguridad o escaneos. ▪ Usar únicamente bibliotecas y funciones de terceros confiables y validados para reducir vulnerabilidades. ▪ Implementar prácticas de codificación seguras y pruebas durante todo el ciclo de vida de desarrollo del proyecto. ▪ Asegurarse que los procesos de firma y distribución de la aplicación sean seguros con el fin de evitar que los atacantes puedan firmar la aplicación con código malicioso.

Tabla 3.2: Descripción de cada factor para el riesgo *M2: Seguridad inadecuada de la cadena de suministro.*

M3: Autenticación/autorización insegura. [13]

Factor	Descripción
Agentes de amenaza	Los atacantes explotan este tipo de vulnerabilidades mediante ataques automatizados que usan herramientas existentes o personalizadas.
Vectores de ataque	Cuando el atacante entiende las vulnerabilidades relacionadas con la autenticación y la autorización, puede explotarlas falsificándolas o eludiéndolas o accediendo a la aplicación con las credenciales de un usuario legítimo suplantando su identidad.
Debilidad de seguridad	Para detectar los esquemas de autenticación y autorización deficiente se pueden realizar ataques binarios contra la aplicación y probar a ejecutar funciones privilegiadas que solo deben poder ejecutarlas usuarios con más privilegios.
Impactos técnicos	Dependiendo de la funcionalidad a la que el atacante logre acceder, el impacto puede ser más o menos severo. Las repercusiones se producen cuando no se puede identificar al usuario que está realizando la acción.
Mitigación	▪ **Evitar patrones débiles**: Seguir buenas prácticas cómo evitar almacenar la contraseña dentro de la función *Recordarme* o no permitir que el usuario utilice códigos PIN de 4 o menos dígitos.

Tabla 3.3: Descripción de cada factor para el riesgo *M3: Autenticación/autorización insegura.*.

M4: Validación de entrada/salida insuficiente. [14]

Factor	Descripción
Agentes de amenaza	Las aplicaciones que no validan o desinfectan de forma correcta los datos de entrada o salida pueden correr el riesgo de sufrir ataques como inyecciones.
Vectores de ataque	Este tipo de vulnerabilidades pueden causar accesos no autorizados, ejecución de código malicioso y manipulación de los datos.
Debilidad de seguridad	Si los datos de entrada y salida no se validan o se sanitizan correctamente, pueden darse ataques de manipulación e incluso ataques de inyección como SQL o XSS.
Impactos técnicos	Pueden darse diversos impactos técnicos como la ejecución de código malicioso dentro de la aplicación o permitir que loa atacantes manipulen la entrada o la salida de los datos para conducir al acceso no autorizado y a la extracción de los datos confidenciales.

Factor	Descripción
Mitigación	▪ Validación de entrada. ▪ Sanitización de salida. ▪ Validación específica del contenido. ▪ Pruebas de seguridad periódicas. ▪ Comprobaciones de integridad de los datos. ▪ Prácticas de codificación segura.

Tabla 3.4: Descripción de cada factor para el riesgo *M4: Validación de entrada/salida insuficiente.*

M5: Comunicación insegura. [15]

Factor	Descripción
Agentes de amenaza	Las aplicaciones móviles modernas cambian datos con diferentes servidores remotos. Al trasmitir los datos, si un tercero intercepta el tráfico puede obtenerlos e incluso modificarlos antes de que lleguen al destino.
Vectores de ataque	Las aplicaciones dependen de protocolos criptográficos, pero en ocasiones la implementación puede tener fallos como el uso de protocolos obsoletos o estar configurados de forma incorrectos
Debilidad de seguridad	Aunque las aplicaciones modernas suelen proteger el tráfico de red puede haber inconsistencias en su implementación, lo que puede generar vulnerabilidades con las que se exponen los datos del ususario o los datos de la sesión.
Impactos técnicos	Esta vulnerabilidad puede exponer datos del usuario, lo que puede implicar apropiación de cuentas de usuario, fugas de datos o suplantación de identidad.
Mitigación	▪ Utilizar cifrados sólidos con longitudes de de clave adecuadas. ▪ No enviar datos confidenciales por canales como SMS. ▪ Alertar a los usuarios a través de la interfaz de usuario si la aplicación detecta un certificado no válido.

Tabla 3.5: Descripción de cada factor para el riesgo *M5: Comunicación insegura.*

M6: Controles de privacidad inadecuados. [16]

Factor	Descripción
Agentes de amenaza	Los controles de seguridad se encargan de proteger la información de identificación personal (PII). Los atacantes podrían suplantar la identidad de la víctima o hacerle chantaje con sus datos. En general, la información personal podría filtrarse, manipularse, destruirse o bloquearse.
Vectores de ataque	Obtener información de identificación personal requiere que el atacante primero vulnere la seguridad a otro nivel, por ejemplo espiando la comunicación de red.
Debilidad de seguridad	Los riesgos de violación de la privacidad aumentan por el manejo de forma descuidada por parte de los desarrolladores. La información debe procesarse siempre teniendo en cuenta la posibilidad de que un atacante pueda acceder al almacenamiento.
Impactos técnicos	Si se manipulan los datos del usuario, el sistema puede quedar inutilizable. Por otro lado, si los datos están mal formados, el backend puede verse gravemente afectado si no se limpian o se gestionan las excepciones adecuadas.
Mitigación	La información de identificación personal no necesaria no debe almacenarse ni transmitirse a menos que sea extrictamente necesario.

Tabla 3.6: Descripción de cada factor para el riesgo *M6: Controles de privacidad inadecuados.*

M7: Protección binaria insuficiente. [17]

Factor	Descripción
Agentes de amenaza	El binario de la aplicación puede contener secretos valiosos como secretos criptográficos codificados que los atacantes pueden usar de forma indebida. A parte de recopilar información, también podrían querer acceder gratis a funciones de pago o eludir mecanismos de seguridad.
Vectores de ataque	El binario podría estar sujeto a dos tipos de ataques: ingeniería inversa y manipulación del código.
Debilidad de seguridad	Puesto que todas las aplicaciones tienen ficheros binarios, deben implementar contramedidas para defenderse de los posibles atacantes el tiempo necesario hasta que el atacante se de por vencido. Hay que tener en cuenta que las aplicaciones modificadas posiblemente sean reistribuídas desde las tiendas de aplicaciones por lo que resulta conveniente contar con mecanismos de detección y denuncia dentro de las propias aplicaciones.

Factor	Descripción
Impactos técnicos	Si se filtran secretos como consecuencia de un ataque de ingeniería inversa, estos deben reemplazarse lo más rápido posible en todo el sistema. Las fugas de información del binario puede revelar vulnerabilidades de seguridad en el backend.
Mitigación	▪ **Evitar la ingeniería inversa:** Para evitar este riesgo, el binario de la aplicación debe ser incomprensible. Para ello hay que ofuscar su contenido con algoritmos robustos. ▪ **Romper mecanismos de seguridad:** Deben existir mecanismos de seguridad locales implementados por el backend. Por otro lado, las comprobaciones de integridad ayudan a detectar la manipulación del código.

Tabla 3.7: Descripción de cada factor para el riesgo *M7: Protección binaria insuficiente.*

M8: Configuración incorrecta de seguridad. [18]

Factor	Descripción
Agentes de amenaza	La configuración incorrecta de seguridad en las aplicaciones móviles está referido a la configuración incorrecta de permisos, controles de seguridad o ajustes, lo que puede llevar a accesos no autorizados o vulnerabilidades. Los atacantes pueden explotarlo obteniendo acceso no autorizado a datos de carácter confidencial o realizar acciones maliciosas.
Vectores de ataque	Las configuraciones incorrectas se pueden explotar con controles de acceso inadecuados o mal configurados, almacenamiento de datos sin protección, gestión de sesiones de usuario mal configurada o un cifrado hash débiles o mal implementados, con los que se puede obtener acceso a información confidencial.
Debilidad de seguridad	Este tipo de vulnerabilidad es común en las aplicaciones por factores como falta de tiempo, de conocimiento o errores humanos a lo largo del desarrollo. Se pueden detectar mediante la revisión manual del código.
Impactos técnicos	La configuración incorrecta de la seguridad puede implicar acceso no autorizado a datos confidenciales y secuestro o suplantación de cuentas de usuarios.

Factor	Descripción
Mitigación	▪ Evitar almacenar archivos de aplicaciones con demasiados permisos. ▪ Configurar la red de forma segura, impidiendo tráfico sin formato. ▪ Deshabilitar la depuración del código. ▪ Abstenerse de usar credenciales predeterminadas. ▪ Solicitar solamente los permisos necesarios para el funcionamiento correcto de la aplicación.

Tabla 3.8: Descripción de cada factor para el riesgo *M8: Configuración incorrecta de seguridad.*

M9: Almacenamiento de datos inseguro. [19]

Factor	Descripción
Agentes de amenaza	Almacenar datos de forma insegura puede atraer múltiples agentes de amenazas que quieren explotar las vulnerabilidades y obtener acceso no autorizado a la información confidencial.
Vectores de ataque	El almacenamiento de los datos de forma insegura expone vulnerabilidades a múltiples vectores de ataque que los atacantes pueden explotar. Estos vectores incluyen acceso al sistema no autorizado o la interceptación de transmisiones de datos.
Debilidad de seguridad	La ausencia de protocolos seguros de transmisión de datos causa que los datos sean vulnerables a la interceptación del tráfico entre la aplicación y los servidores, lo que puede llevar a filtraciones de datos.
Impactos técnicos	Almacenar de forma insegura los datos puede causar fallos como problemas de integridad y manipulación de los datos, filtraciones de datos y daños a la reputación y confianza del desarrollador de la aplicación móvil.
Mitigación	▪ Uso de un cifrado robusto. ▪ Transmisión segura de los datos. ▪ Implementar controles de acceso adecuados. ▪ Validar la entrada y sanitizar los datos. ▪ Implementar mecanismos de almacenamiento seguro.

Tabla 3.9: Descripción de cada factor para el riesgo *M9: Almacenamiento de datos inseguro..*

M10: Criptografía insuficiente. [20]

Factor	Descripción
Agentes de amenaza	Para explotar la vulnerabilidad, los atacantes pueden socavar la confidencialidad, autenticidad e integridad de la información sensible.
Vectores de ataque	El vector de ataque para la criptografía insegura en una aplicación consiste en explotar vulnerabilidades en los mecanismos criptográficos usados para proteger la información confidencial.
Debilidad de seguridad	La criptografía insegura puede introducir vulnerabilidades de seguridad que pueden incluir el uso de algoritmos de cifrado débiles o longitudes de clave inadecuadas. Las funciones hash inseguras plantean vulnerabilidades de seguridad graves por los conflictos que estas funciones causan.
Impactos técnicos	Esta vulnerabilidad provoca la recuperación no autorizada de la información confidencial del dispositivo.
Mitigación	▪ Usar algoritmos de cifrado robustos. ▪ Implementar el cifrado de forma correcta. ▪ Asegurarse que la longitud de la clave es suficiente. ▪ Almacenamiento de claves de cifrado seguro. ▪ Usar una capa de transporte segura. ▪ Realizar pruebas de seguridad periódicas.

Tabla 3.10: Descripción de cada factor para el riesgo *M10: Criptografía insuficiente.*.

Estándar de Verificación de Seguridad de Aplicaciones Móviles (MASVS) y Guía de Pruebas de Seguridad para Aplicaciones Móviles (MASTG).

Con el auge de las aplicaciones móviles, la seguridad en este ámbito se ha convertido en algo crítico dentro del desarrollo y la auditoría. Es por ello que OWASP, ha desarrollado diferentes guías específicas para las aplicaciones móviles entre las que se encuentran *Mobile Application Security Verification Standart* (**MASVS**) y *Mobile Application Security Testing Guide* (**MASTG**).

MASVS [21] define qué debe buscarse y verificarse dentro de la aplicación, mientras que **MASTG** [22] muestra como hacerlo. Con ello, ambas guías permiten estructurar las auditorías de seguridad mostrando a quién debe encargarse de la tarea cómo hacerlo y que buscar.

Mobile Application Security Verification Standart (MASVS).

MASVS es un estándar que defina los requisitos mínimos de seguridad que una aplicación móvil debería cumplir en función del nivel de protección deseado. El objetivo principal es ser un punto en común con los desarrolladores y auditores de seguridad para establecer unos estándares comunes sobre qué se considera una aplicación segura.

MASVS se organiza en diferentes niveles:

- **MASVS-L1:** Requisitos de seguridad básicos que todas las aplicaciones, independientemente

de su naturaleza, deberían cumplir.

- **MASVS-L2:** Requisitos de seguridad adicionales para aquellas aplicaciones que manejen datos sensibles o que necesiten una protección reforzada.
- **MASVS-R:** Requisitos relacionadas con la resiliencia de la aplicación frente a ataques de ingeniería inversa o manipulación del entorno de ejecución.

Dentro de cada uno de los anteriores niveles, se encuentran una serie de controles que se clasifican según su área:

- **MASVS-STORAGE:** Almacenamiento seguro de datos confidenciales.
- **MASVS-CRYPTO:** Funcionalidad criptográfica usada para proteger los datos confidenciales.
- **MASVS-AUTH:** Mecanismos de autorización y autenticación usados por la aplicación.
- **MASVS-NETWORK:** Comunicación de red segura entre la aplicación y el exterior.
- **MASVS-PLATFORM:** Interacción segura con la plataforma móvil y otras aplicaciones instaladas.
- **MASVS-CODE:** Mejores prácticas de seguridad para el procesamiento de datos y para mantener la aplicación actualizada.
- **MASVS-RESILIENCE:** Resiliencia a ingeniería inversa e intentos de manipulación.
- **MASVS-PRIVACY:** Controles de privacidad para proteger la privacidad del usuario.

Mobile Application Security Testing Guide (MASTG).

En base a los controles que OWASP *MASVS* define, se basa OWASP *MASTG*, una guía con las diferentes puebas de seguridad para las aplicaciones móviles. Completa a *MASVS* ya que proporciona los procedimientos y herramientas para verificar si una aplicación cumple o no con los requisitos que se establecen en *MASVS*.

Para cada una de las pruebas que propone, tanto para Android como para IOS, el sistema operativo de Apple, la guía se organiza por las categorías que se proponen en *MASVS*.

La guía *MASTG* incluye pruebas de análisis estático y de análisis dinámico para cubrir todos los controles y niveles que OWASP establece. Cada una de las pruebas cuenta con:

- Identificador único.
- Descripción de la prueba.
- Objetivo de la prueba.
- Metodología propuesta.
- Comandos y herramientas sugeridas.

Con ello, tanto el desarrollador como el auditor pueden conocer la forma de evaluación de la aplicación y ser capaces de llevar a cabo sus tareas con el objetivo de garantizar la seguridad en la aplicación siguiendo esta metodología.

3.3.2. NIST SP 800-163

El Instituto Nacional de Estándares y Tecnología de Estados Unidos (**NIST**, *National Institute of Standards and Technology* [23] propone una guía para evaluar la seguridad de las aplicaciones móviles. Dicha guía esta pensada para:

- Evaluar los riesgos de seguridad de las aplicaciones.
- Integrar los procesos de seguridad.
- Estandarizar el proyecto de análisis mediante pruebas tanto de análisis estático como de análisis dinámico.

3.3.3. Categorías de análisis para la realización del test de seguridad.

A la hora de realizar un análisis de seguridad sobre una aplicación móvil, hay que tener en cuenta que existe una clasificación en función al tipo de análisis que se va a realizar. Estos pueden ser:

- **Análisis estático:** Consiste en analizar el código fuente de la aplicación sin ejecutarlo o en reposo. Se busca estudiar la estructura del código y del software disponible con el objetivo de buscar vulnerabilidades, errores de lógica o malas prácticas de programación.
- **Análisis dinámico:** Se ejecuta la aplicación, y con ella corriendo se observa su comportamiento en tiempo real dentro de un entorno controlado, que puede ser tanto un dispositivo real como un emulador.

Capítulo 4

Análisis de la aplicación.

En el presente capítulo se realiza un análisis tanto a nivel técnico como funcionalidad de la aplicación móvil *AquaCyL* [24], desarrollada por Pablo Varela Vázquez para que los usuarios puedan conocer el estado de las diferentes áreas de baño disponibles en Castilla y León.

Se realizará un análisis de la aplicación en base a la información que se ofrece en Google Play teniendo en cuenta permisos, datos que se recogen y como se gestionan los datos que ésta usa. Finalmente se ejecutará la aplicación con el fin de descubrir su finalidad y familiarizarse con ella.

4.1. ¿Qué es AquaCyL?

AquaCyL [24] es una guía con la que se puede obtener información sobre las zonas de baño disponibles en Castilla y León. Con ella se puede consultar el estado de las zonas de baño, el pronóstico del tiempo en dichos puntos, la localidad en la que se encuentra y puntos de interés en la zona. Es posible publicar comentarios y ver la experiencia de otros usuarios.
Para instalarla, hay que hacerlo desde Google Play con un dispositivo Android que cuente con la versión 10 o superior. Hay que tener en cuenta que está clasificada como PEGI3 [25], lo que quiere decir que se considera adecuada para todos los grupos de edad.

Figura 4.1: Logo AquaCyL

4.2. Seguridad de los datos

La seguridad de los datos de una aplicación dentro de Google Play es información que el desarrollador proporciona con el fin de informar a los usuarios cómo recoge, trata y comparte los datos dentro de la aplicación. [26]

Se indica que la aplicación no comparte datos con terceros. Esto significa que la información que recoge del usuario no se comparte con otras organizaciones u empresas.

Datos que se recogen:

Cuando un desarrollador desea subir su aplicación a GooglePlay, uno de los pasos necesarios es elegir dentro de una serie de tipos de datos fijos cuáles son los que la aplicación va a recoger [27]. En la siguiente tabla se muestran los datos que AquaCyL recoge y con qué finalidad lo hace:

DATOS PERSONALES		FINALIDAD			
CLASIFICACIÓN	TIPO DE DATO	ANÁLISIS	FUNCIONALIDAD DE LA APLICACIÓN	PERSONALIZACIÓN	GESTIÓN DE LA CUENTA
Actividad en la aplicación	Interacciones con la aplicación	✓			
Fotos y videos	Fotos		✓		
IDs de dispositivos o de otro tipo	IDs de dispositivo o de otro tipo	✓			
Información y rendimiento de la aplicación	Registros de fallos	✓			
	Diagnósticos	✓			
Información personal	Nombre		✓	✓	✓
	Correo electrónico		✓	✓	✓
	IDs de usuario		✓	✓	✓

Tabla 4.1: Datos recogidos y objetivo.

En la anterior tabla se muestran los diferentes datos que se recogen y el fin por el que lo hacen. Puede ser para diferentes fines entre los que se encuentran los siguientes:

- Analizar aspectos como las interacciones con la aplicación.
- Comprobar la funcionalidad revisando el registro de fallos.
- Personalizar la aplicación mediante el nombre o el correo electrónico.

- Gestionar la cuenta gracias a los identificadores de usuario.

Prácticas de seguridad:

- Transferencia de datos segura.
- Posibilidad de eliminar los datos si se solicita.

4.3. Permisos

Al igual que el resto de aplicaciones, ésta también requiere de permisos que se solicitan al usuario con el fin de poder utilizar ciertos elementos del dispositivo móvil. Con ello, la aplicación puede acceder a los siguientes elementos en función de cada categoría:

- **Fotos/multimedia/archivos**
  - Leer el contenido del almacenamiento USB.
  - Modificar o eliminar contenido del almacenamiento USB.
- **Micrófono**
  - Grabación de sonido.
- **Cámara**
  - Realizar vídeos o fotografías.
- **Almacenamiento**
  - Modificar o eliminar contenido del almacenamiento USB.
  - Leer el contenido del almacenamiento USB.
- **Otro motivo**
  - Acceso completo a red.
  - Ver conexiones de red.
  - Recibir datos de internet.
  - Impedir que el dispositivo se suspenda.
  - Control de la vibración.
  - Lectura de la configuración de los Servicios de Google.
  - Comprobación de licencia de Google Play.

4.4. Familiarización con la aplicación.

Tal y como se verá en el capítulo 8, para lanzar la aplicación se ha utilizado el emulador con Android 16 para ejecutar la aplicación.

Lo primero es abrir el proyecto. Una vez hecho, e iniciado el emulador, hay que ejecutar la APK con la aplicación. Para ello hay que abrir una terminal dentro de Android Studio y ejecutar el siguiente comando:

```
adb install AquaCyL_1.0.0_010000000.apk
```

En este caso *adb* no se encuentra instalado por lo que hay que hacerlo con el comando:

```
sudo apt install adb
```

4.4.1. Aplicación sin iniciar sesión.

Tras instalarlo y volver a ejecutar lo anterior en el emulador ya aparece la aplicación. Una vez se pulsa en el icono de la aplicación, esta se abre. Lo primero que se pregunta es si se desea permitir que la aplicación envíe notificaciones, a lo que se ha optado por no permitirlo.

El siguiente paso es decidir si iniciar sesión o continuar como invitado. Ya que el objetivo de esta sección es familiarizarse con la aplicación se ha continuado como invitado. La siguiente pantalla es el menú de inicio en el que se pueden ver los botones de inicio, ajustes, filtros, ordenación de resultados y perfil.

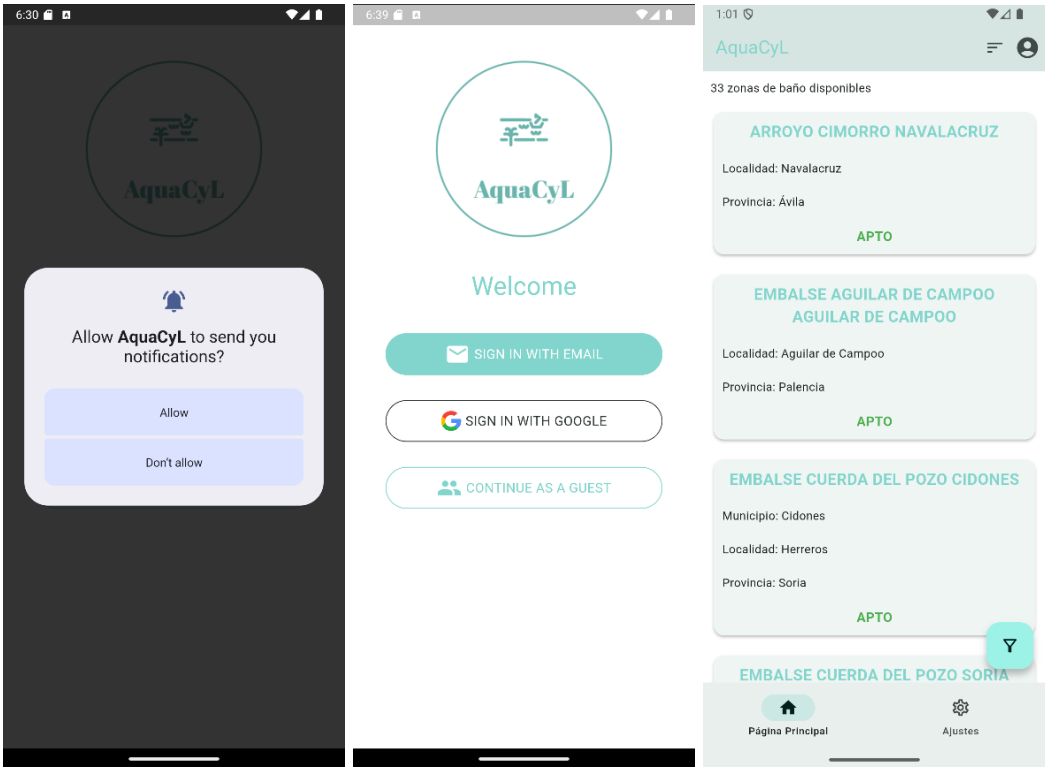

Figura 4.2: Primer arranque de la aplicación con las pantallas de notificaciones, inicio de sesión y home.

En la siguiente figura se puede ver como se ha accedido al menú de ajustes para, a parte de navegar por él, cambiar el idioma de la aplicación al español para un mejor entendimiento y comprensión del entorno. Por otro lado se puede ver como las zonas de baño se pueden ordenar por nombre, municipio, localidad, provincia o estado. Este último no se refiere al país si no a cómo se encuentra el lugar de baño.
También es posible filtrar por provincia, ya que es una aplicación de Castilla y León, solo se pueden filtrar según las nueve provincias pertenecientes a la comunidad, y por estado de la zona de baño.

Estados por los que se puede filtrar:

- Apto.
- No apto.
- Agua sin analizar.
- Fuera de temporada.
- Desconocido.

Provincias por las que las que filtrar:

- Ávila.
- Burgos.
- León.
- Palencia.
- Salamanca.
- Segovia.
- Soria.
- Valladolid.
- Zamora.

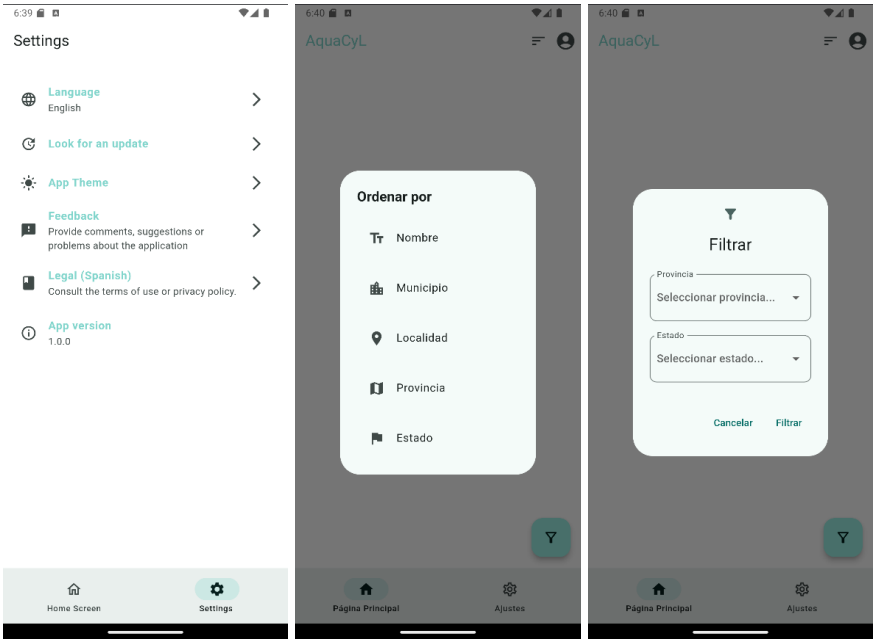

Figura 4.3: Primer arranque de la aplicación con las pantallas de ajustes, ordenar y filtrar.

Para cada una de las zonas de baño se puede ver la previsión del tiempo de la semana actual, así como la opción de ver los comentarios que otros usuarios han dejado de ese embalse, información sobre como llegar a esa zona de baño mediante una redirección a Google Maps [28] y los alojamientos disponibles en la zona redirigiendo a EscapadaRural [29], que es un sitio web para ver y reservar alojamientos en zonas rurales:

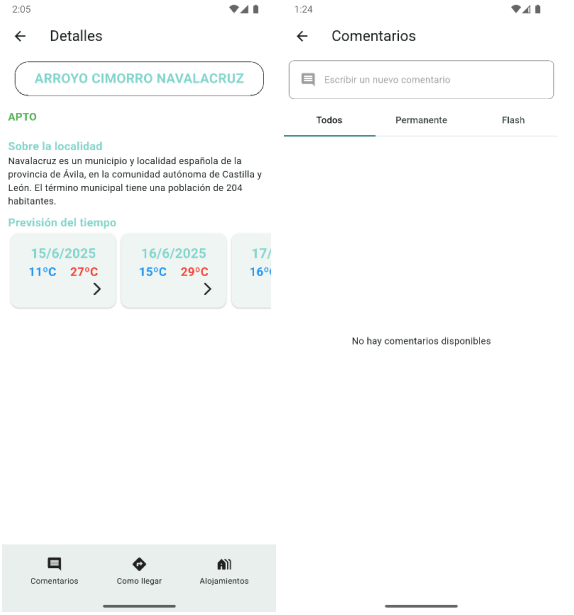

Figura 4.4: Detalles de una de las zonas de baño y vista de los comentarios.

En la anterior figura puede verse que en la pantalla de comentarios, si se desease comentar algo, esto no se podría dar puesto que dicho campo se encuentra deshabilitado. Por otro lado, los comentarios se clasifican en *Permanentes*, lo cuales, tal y como indica la palabra, se encuentran siempre disponibles; o en comentarios *Flash* que se eliminan tras 24 horas.

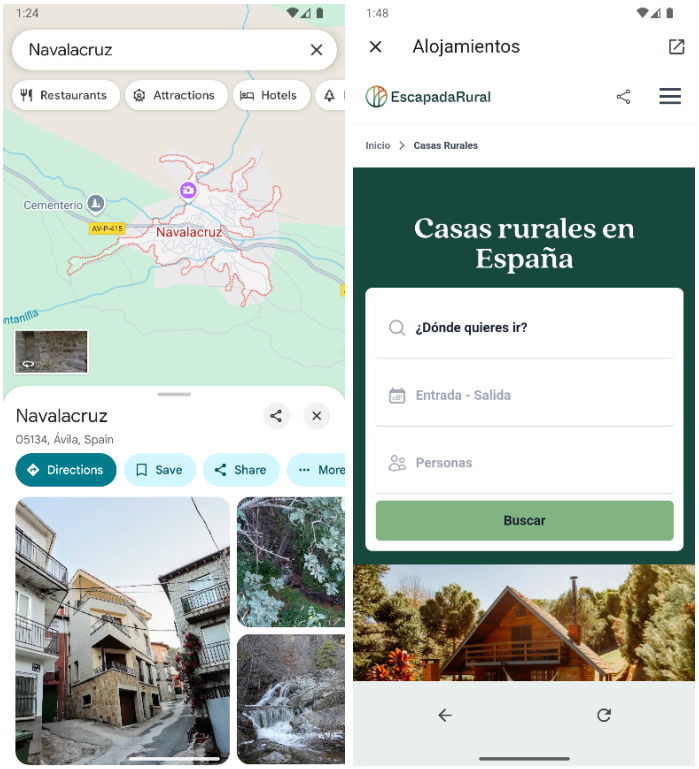

Figura 4.5: Vista de las webs, con la ubicación del arroyo y con el sitio web dónde encontrar alojamiento.

Si se continúa como invitado, no da la opción de tener zonas favoritas, o de hacer comentarios, habilitando únicamente su visualización, pero si se inicia sesión o se crea una cuenta estas funciones se encuentran disponibles.

4.4.2. Aplicación con la sesión iniciada.

Para crear una cuenta, los datos que se solicitan son los siguientes:

- Nombre.
- Correo electrónico.
- Contraseña.
- Repetir contraseña.

Una vez introducidos los datos, es necesario verificar la cuenta recién creada mediante un enlace que se envía al correo indicado.

Tras su validación, ya se puede iniciar sesión con la cuenta recién creada.

Para iniciar sesión se solicitan:

- Correo electrónico.
- Contraseña.

Una vez iniciada la sesión, se puede ver como aparece el botón de favoritos, donde se pueden añadir aquellas zonas de baño de preferencia.

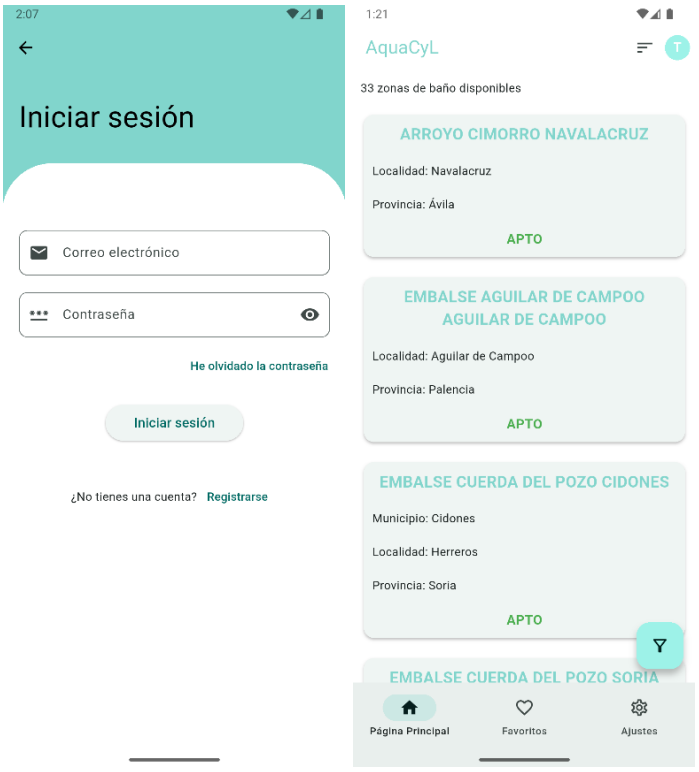

Figura 4.6: Interfaz de inicio de sesión y menú principal con sesión iniciada.

En la siguiente figura se puede observar cómo en la vista de una zona de baño, en la esquina superior derecha, aparece un icono de un corazón, desde donde se puede añadir a favoritos esa zona de baño. Por otro lado, en la sección *Favoritos* se puede ver la lista de todas las zonas añadidas a dicha categoría. Finalmente, si se quiere hacer un comentario, aparece el nombre de usuario y el correo electrónico, junto con la posibilidad de elegir si en el comentario se quiere que se vean las credenciales del usuario que lo publica, el tipo de comentario y una caja de texto donde introducir el comentario deseado.

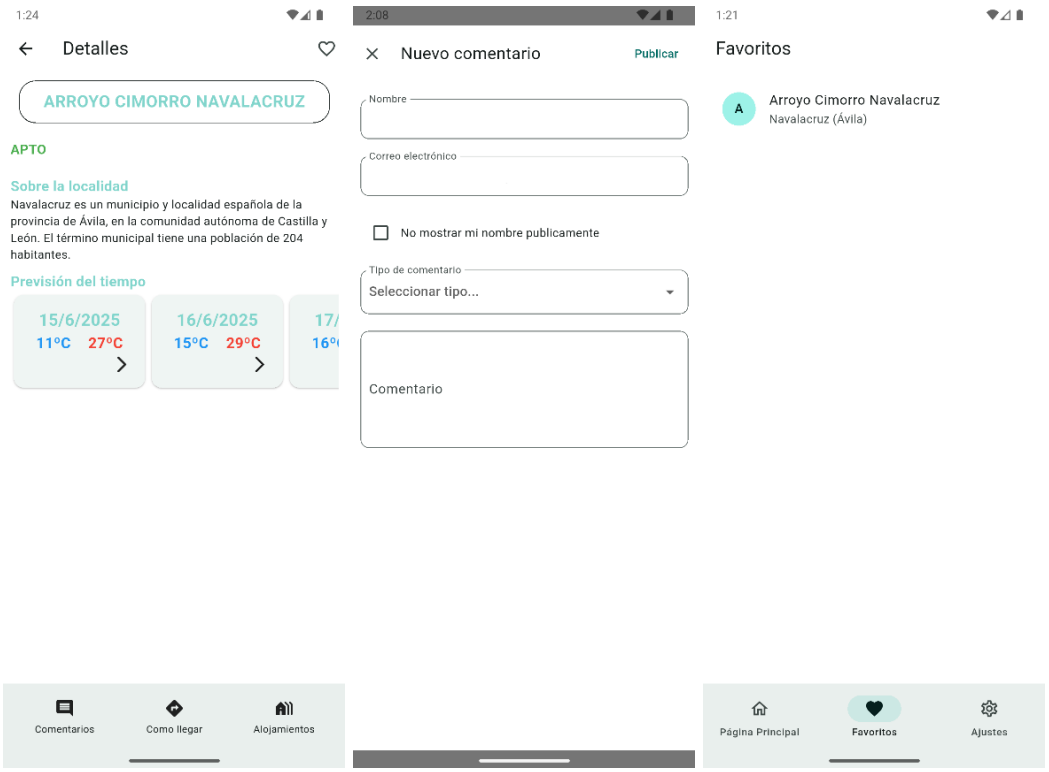

Figura 4.7: Interfaces de una zona de baño, la sección de comentarios y la de favoritos.

En lo que al resto de la funcionalidad respecta, se mantiene igual tanto con la sesión iniciada como entrando como invitado.

Capítulo 5

Análisis de la metodología OWASP.

OWASP (Open Web Application Security Project) [30], tal y como se ha mencionado en capítulos anteriores, es una fundación sin ánimo de lucro que promueve buenas prácticas de seguridad y trabaja para mejorar la seguridad del software. Para las aplicaciones móviles OWASP propone una serie de controles y tests para probar, por medio de auditorías móviles, la seguridad de las aplicaciones. [31]

Según OWASP, una aplicación móvil es vulnerable si se viola alguno de los requisitos de seguridad de los que se proponen en OWASP-MASVS (Mobile Application Security Verification Standard) [21]. Para verificar si dicha violación se produce, se puede comprobar con la evidencia técnica que se puede obtener mediante las pruebas que se proponen en OWASP-MASTG (Mobile Application Security Testing Guide) [22].

Según lo anterior, la vulnerabilidad tiene como fundamento tres grandes pilares:

- **Estándar MASVS:** Que define los controles de seguridad en las diferentes categorías para estructuras las distintas pruebas que aparecen en MASTG. Cada uno de los controles se encuentra estructurado en niveles de seguridad.
- **Verificación con las pruebas MASTG:** Con las pruebas proporcionadas por MASTG, se puede comprobar de forma técnica si se cumplen los controles propuestos por MASVS. Dichas pruebas pueden requerir de:
  - **Análisis estático:** de la APK descompilada o del código fuente.
  - **Análisis dinámico:** sobre la aplicación en ejecución sobre un emulador o un dispositivo real.
- **Impacto sobre la seguridad de la aplicación a auditar:** Si realizando una de las pruebas que OWASP propone, ésta encuentra una vulnerabilidad, se considera que conlleva un riesgo potencial para la seguridad de la aplicación. Es importante que una vez se encuentre dicha vulnerabilidad hay que documentarla y definir el impacto que tiene sobre la aplicación.

En la seguridad informática, se definen cuatro principios fundamentales a modo de base sobre

la que se diseñan, implementan y evalúan los mecanismos de protección de un sistema o una aplicación móvil. [32] Estos principios son de gran utilidad para establecer criterios para identificar vulnerabilidades en una aplicación. Se trata de los siguientes:

- **Autenticidad:** Asegura que la información de la que se cuenta sobre la aplicación es la que realmente es. El software debe ser capaz de verificar que el usuario que firma un mensaje es quien dice ser.
- **Confidencialidad de la información:** O privacidad de la información, se refiere a que la información solo debe ser conocida por aquellos que necesitan conocerla y cuentan con la correspondiente autorización para ello.
- **Disponibilidad de la información:** Referido a que la información debe estar disponible cada vez que las personas autorizadas a acceder a ella y modificarla cuando quieran. Por otro lado debe poder recuperarse en caso de que tenga lugar un incidente de seguridad que implique su corrupción o pérdida.
- **Integridad de la información:** Hace referencia a que la información que se haya almacenado o que haya sido transmitida, no ha sido manipulada por terceros. De este modo se garantiza que la información no va a ser modificada por personas no autorizadas.

Los anteriores principios se pueden relacionar con OWASP y las vulnerabilidades que busca detectar de modo que cada una de las categorías que aparecen en OWASP-MASVS responde a la necesidad de proteger alguno de los principios que se han mencionado anteriormente.

A continuación se muestra como se vincula cada uno de los principios con los controles que MASVS ofrece. De este modo se puede justificar por qué una aplicación se puede considerar vulnerable si incumple alguno de los principios:

Autenticidad:
La autenticidad es un control que OWASP aborda mediante controles que comprueban la identidad legítima de usuarios y la restricción del acceso en función a los permisos con los que se cuente. Se relaciona con la categoría **MASVS-AUTH**, de modo que se considera que existe una vulnerabilidad si se puede acceder a funcionalidades restringidas o acceder a la aplicación sin pasar por los diferentes métodos de autenticación.

Confidencialidad:
En OWASP la confidencialidad se muestra mediante canales orientados a la protección de la información sensible frente a los intentos no autorizados ya sea en reposo o en tránsito. Está fuertemente relacionado con las categorías **MASVS-STORAGE**, **MASVS-NETWORK** y **MASVS-CRYPTO**. Se considera que hay una vulnerabilidad si se encuentran datos de carácter sensible accesibles sin que haya medidas de protección.

Disponibilidad:
La disponibilidad se evalúa de forma indirecta en relación con OWASP, lo que conlleva que se compruebe la robustez de la aplicación frente a fallos, manipulaciones o interrupciones del servicio. OWASP busca que la aplicación siga funcionando a pesar de fallos provocados o no provocados. Se relaciona con **MASVS-RESILIENCE** y **MASVS-PLATFORM**, de forma que se valora si existe una vulnerabilidad si la aplicación deja de prestar servicio o si se muestra inestable en ciertas condiciones.

Integridad:
OWASP refleja la integridad asegurando que ni los datos ni el código fuente de la aplicación pueden ser alterados sin que se detecte. Está relacionado con **MASVS-CODE**, **MASVS-PLATFORM** y **MASVS-RESILIENCE**. Se considera vulnerable si la aplicación se puede modificar sin que

haya mecanismos de detección o que lo impidan.

En la siguiente tabla se muestra de una forma mas visual, la relación entre los principios de la seguridad informática ya mencionados antes con las categorías propuestas por OWASP en su guía OWASP-MASVS:

Principio de seguridad	**Categorías relacionadas**
Autenticidad	▪ MASVS-AUTH
Confidencialidad	▪ MASVS-CRYPTO ▪ MASVS-NETWORK ▪ MASVS-STORAGE
Disponibilidad	▪ MASVS-PLATFORM ▪ MASVS-RESILIENCE
Integridad	▪ MASVS-CODE ▪ MASVS-PLATFORM ▪ MASVS-RESILIENCE

Tabla 5.1: Relación de los principios de la seguridad informática con las categorías que OWASP-MASVS propone.

5.1. Categorías MASVS-OWASP

OWASP propone una agrupación de las pruebas según las categorías de requisitos de seguridad que se muestran en la tabla 5.2. Cada una de ellas incluye una serie de pruebas que evalúan distintos controles:

Categoría	**Nombre**	**Descripción**
Almacenamiento	MASVS-STORAGE	Comprobación del almacenamiento de datos de forma segura en el dispositivo en el que se ejecuta la aplicación.
Autenticación y autorización	MASVS-AUTH	Revisión de las credenciales de acceso, gestión de las sesiones de usuario dentro de la aplicación, así como de los tokens de sesión [33] o valores alfanuméricos asignados por el servidor al cliente cuando se inicia sesión.

Categoría	**Nombre**	**Descripción**
Código	MASVS-CODE	Evaluación del código fuente y su protección, ingeniería inversa, ofuscación y validación de su integridad.
Comunicación de red	MASVS-NETWORK	Verificar que las conexiones sean seguras. Comprobar que se utiliza el protocolo HTTPS y el almacenamiento de una copia del certificado del servidor en el dispositivo local.
Criptografía	MASVS-CRYPTO	Uso adecuado de los diferentes algoritmos criptográficos, así como el almacenamiento y la transmisión segura de los datos con su debida protección previa.
Plataforma	MASVS-PLATFORM	Correcta utilización de las interfaces de programación de aplicaciones o API [34] de la plataforma Android o IOS. En el caso de este proyecto, Android.
Resiliencia	MASVS-RESILIENCE	Resistencia a la manipulación en tiempo de ejecución o dinámica, debugging y análisis de malware.

Tabla 5.2: Categorías de seguridad de OWASP-MASVS.

Capítulo 6

Diseño

En el presente capítulo se recoge el diseño de las pruebas realizadas para evaluar las vulnerabilidades con las que puede contar la aplicación que se va a auditar. Resulta una fase esencial para garantizar que la auditoría se realice según establece la guía OWASP. Para ello se ha tomado como referencia la guía de pruebas OWASP-MASTG, que define los tests que hay que realizar, clasificándolos en las diferentes categorías y controles plasmados en la guía OWASP-MASVS.

En primer lugar, se han detectado una serie de riesgos que podrían afectar a la seguridad de la aplicación en caso de producirse. Seguidamente se ha realizado una selección de pruebas en función a una serie de criterios con los que se justifican las decisiones tomadas según la funcionalidad de la aplicación, la viabilidad técnica y el impacto en la seguridad.

Posteriormente se realiza una planificación para la ejecución de los pasos que se describen a lo largo del presente capítulo.

Finalmente, se diseñan las pruebas seleccionadas, donde se muestra el objetivo de cada una de ellas así como en qué consistirá la prueba en lo que a análisis estático y dinámico se refiere.

6.1. Riesgos detectados en la aplicación

Un vez realizado el análisis de la aplicación a auditar, como se ha visto en el capítulo 4, y antes de comenzar a establecer los criterios de selección de las pruebas previo diseño, hay que identificar los riesgos de seguridad que puede tener la aplicación.

A continuación se muestra una lista de los posibles riesgos de seguridad que pueden darse en la aplicación:

- **Acceso no autorizado:** El atacante accede a partes de la aplicación donde no debería.
- **Suplantación de identidad:** Un tercero accede a la aplicación con las credenciales de otro usuario haciéndose pasar por él.
- **Fuga de datos personales:** Los datos de los usuarios se ven comprometidos ante ataques de terceros.
- **Almacenamiento inseguro:** La información sensible de la aplicación se almacena sin cifrar y, por tanto, de forma no segura.

- **Tráfico de datos no seguro:** Los datos en tránsito se envían sin cifrar por lo que está sometida a ataques en los que la información se ve comprometida.
- **Uso de algoritmos criptográficos obsoletos o inseguros:** La información se almacena con algoritmos criptográficos no seguros u obsoletos.
- **Existencia de datos personales en WebViews [35]:** Los datos personales del usuario se comparten en las webs a las que se acceden desde la aplicación.
- **Manipulación del código:** Los atacantes alteran el código para acceder a datos sensibles.
- **Fugas de información a través de la interfaz:** Se muestran datos sensibles en la interfaz, vía notificaciones, capturas de pantalla, etc.
- **Persistencia de datos tras cerrar la sesión:** Una vez un usuario cierra la sesión, los datos del usuario se mantienen presentes.

Una vez identificados los riesgos en la aplicación, el siguiente paso es definir los criterios de selección de las pruebas que se van a lanzar para comprobar la seguridad.

6.2. Criterios de selección de las pruebas.

A la hora de seleccionar las pruebas que se van a lanzar sobre la aplicación móvil para realizar la auditoría, se han tenido en cuenta una serie de criterios para garantizar una cobertura lo más completa posible.

6.2.1. Criterios de selección según la metodología OWASP.

- **Validez y versión de las pruebas:** Han sido excluidas aquellas pruebas que OWASP marca como obsoletas en su versión oficial en MASTG, así como aquellas pruebas de la versión 2 (v2 beta) con el fin de garantizar fiabilidad y estabilidad de las pruebas.
- **Cobertura de las categorías que establece OWASP-MASVS:** Las pruebas se han elegido de modo que se cubren todas las categorías que se definen en MASVS para contar con una auditoría que dé cobertura en los campos de seguridad que se especifican.
- **Equilibrio de pruebas de análisis estático y dinámico:** Se ha buscado incluir tanto pruebas de análisis estático (revisión de la configuración, recursos y código fuente) así como análisis dinámico (observación del comportamiento de la aplicación en tiempo de ejecución) con el fin de obtener una visión lo más completa posible de cómo de segura es la aplicación y de las vulnerabilidades con las que cuenta.
- **Viabilidad técnica:** Para la selección de las pruebas también se han teniendo en cuenta los recursos técnicos disponibles, es decir, herramientas como Android Studio y su emulador, para poder ejecutar la aplicación, y otras como Frida [36] para poder comprobar la seguridad de la misma.

6.2.2. Criterios de selección según AquaCyL.

De forma adicional a los criterios anteriores, las pruebas se han elegido también en base al conocimiento con el que se cuenta de AquaCyL, obtenido en la fase de familiarización de la aplicación, qué se muestra en el capítulo 4. Con esto, los criterios que se han tenido en cuenta son los siguientes:

- **Presencia de mecanismos de autenticación:** La aplicación permite navegación tanto como invitado como mediante el registro y posterior inicio de sesión a través de correo

electrónico y contraseña. Por ello se han seleccionado pruebas relacionadas con el tratamiento de información personal y su debida protección.

- **Manejo de datos de carácter confidencial y almacenamiento local:** Puesto que se utiliza almacenamiento local, se recopilan identificadores y se permite gestionar el perfil, se han incluido una serie de pruebas que evalúan cómo se procesan y almacenan los datos sensibles y si se utilizan algoritmos criptográficos seguros y actuales.
- **Interacción con Webs:** Dado que la aplicación hace uso de sitios web dentro de sí misma, se ha seleccionado una batería de pruebas orientadas a detectar posibles vulnerabilidades relacionadas con ejecución de código no deseado y persistencia de datos en dichas webs.
- **Resiliencia frente ataques:** Puesto que se busca evaluar qué tan robusta frente a manipulaciones es la aplicación, se han incluido pruebas que evalúen si la aplicación es capaz de detectar si está siendo ejecutada en un emulador.
- **Uso de permisos y comunicación con el Sistema Operativo:** Ya que la aplicación hace uso de diferentes permisos relacionados con almacenamiento, micrófono y cámara, se ha considerado relevante incluir pruebas que evalúan la interacción de la aplicación con los servicios de Android.
- **Uso de la interfaz:** La aplicación gestiona notificaciones y datos que se pueden mostrar por pantalla, por lo tanto se han elegido pruebas para analizar cómo se usa la información en la interfaz y cómo se depura dicha información.

6.3. Tests OWASP para auditoría móvil.

OWASP Mobile Application Security Verification Standard (MASVS) [21] define el modelo de seguridad de una aplicación móvil y lista los requisitos generales de seguridad para esta. Por otro lado OWASP Mobile Application Security Testing Guide (MASTG) [22] es la guía para lanzar las pruebas en base a los requisitos establecidos en OWASP MASVS.

En base a lo anterior, en MASTG se establecen las siguientes categorías, las cuales cuentan con las pruebas específicas:

- **MASVS-AUTH:** Autenticación y Autorización.
- **MASVS-CODE:** Calidad del Código y Mitigación de Exploits.
- **MASVS-CRYPTO:** Criptografía.
- **MASVS-NETWORK:** Comunicación de Red.
- **MASVS-PLATFORM:** Interacción con la plataforma Móvil.
- **MASVS-RESILIENCE:** Antimanipulación y Antireversión.
- **MASVS-STORAGE:** Almacenamiento de Datos y Privacidad.

6.3.1. Controles

Para cada una de las categorías mencionadas anteriormente, OWASP define distintos controles que son medidas técnicas o prácticas que describen cómo se debe proteger la aplicación móvil. En la siguiente tabla se muestran dichos controles junto a una breve descripción en la que se habla de qué consiste cada uno:

Control	**Descripción**
MASVS-AUTH-1	La aplicación usa protocolos de autenticación y autorización seguros siguiendo buenas prácticas.
MASVS-AUTH-2	La aplicación realiza la autenticación local de forma segura según las mejores prácticas de la plataforma.
MASVS-AUTH-3	La aplicación protege las operaciones sensibles mediante autenticación adicional.
MASVS-CODE-1	La aplicación requiere una versión actualizada de la plataforma.
MASVS-CODE-2	La aplicación cuenta con un mecanismo para cumplir las actualizaciones de la aplicación.
MASVS-CODE-3	La aplicación únicamente usa componentes software sin vulnerabilidades conocidas.
MASVS-CODE-4	La aplicación valida y sanitiza todas las entradas no confiables.
MASVS-CRYPTO-1	La aplicación utiliza criptografía sólida y la usa según las mejores prácticas.
MASVS-CRYPTO-2	La aplicación realiza la gestión de claves según las mejores prácticas.
MASVS-NETWORK-1	La aplicación securiza todo el tráfico de red según las mejores prácticas actuales.
MASVS-NETWORK-2	La aplicación realiza la fijación de la identidad para cada uno de los puntos finales remotos bajo el control del desarrollador.
MASVS-PLATFORM-1	La aplicación utiliza comunicación entre procesos de forma segura.
MASVS-PLATFORM-2	La aplicación usa WebViews[1] de forma segura.
MASVS-PLATFORM-3	La aplicación usa la interfaz de usuario de forma segura.
MASVS-PRIVACY-1	La aplicación minimiza el acceso a los datos sensibles y recursos.
MASVS-PRIVACY-2	La aplicación impide la identificación del usuario.
MASVS-PRIVACY-3	La aplicación es transparente sobre la recolección de datos y su uso.
MASVS-PRIVACY-4	La aplicación ofrece al usuario control sobre sus datos.
MASVS-RESILIENCE-1	La aplicación valida la integridad de la plataforma.
MASVS-RESILIENCE-2	La aplicación implementa mecanismos antimanipulación.
MASVS-RESILIENCE-3	La aplicación implementa mecanismos de análisis antiestático.
MASVS-RESILIENCE-4	La aplicación implementa técnicas de análisis antidinámicos.
MASVS-STORAGE-1	La aplicación almacena de forma segura los datos sensibles.
MASVS-STORAGE-2	La aplicación previene la fuga de datos confidenciales.

Tabla 6.1: Controles OWASP para la protección de aplicaciones móviles.

6.3.2. Pruebas

Dentro de cada uno de los controles, se definen una serie de pruebas para comprobar si en una aplicación móvil se cumplen los controles propuestos anteriormente.

En la siguiente tabla se muestran las pruebas seleccionadas para realizar la auditoría de seguridad en la aplicación móvil AquaCyL, una breve descripción de cada una y la categoría de OWASP-MASVS en la que se clasifica:

Categoría	**Prueba**	**Descripción**
MASVS-AUTH	MASTG-TEST-0017	Prueba para confirmar credenciales.
MASVS-CODE	MASTG-TEST-0002	Prueba del almacenamiento local para la validación de los datos de entrada.
MASVS-CODE	MASTG-TEST-0026	Prueba de intenciones implícitas.
MASVS-CODE	MASTG-TEST-0027	Prueba de carga de URL en WebViews.
MASVS-CODE	MASTG-TEST-0036	Prueba de actualización forzada.
MASVS-CODE	MASTG-TEST-0043	Errores de corrupción de memoria.
MASVS-CRYPTO	MASTG-TEST-0014	Prueba de la configuración del algoritmo estándar de criptografía.
MASVS-NETWORK	MASTG-TEST-0023	Prueba del proveedor de seguridad.
MASVS-PLATFORM	MASTG-TEST-0008	Comprobación de la divulgación de datos confidenciales a través de la interfaz de usuario.
MASVS-PLATFORM	MASTG-TEST-0037	Prueba de limpieza de WebViews.
MASVS-RESILIENCE	MASTG-TEST-0040	Prueba de símbolos de debugging.
MASVS-RESILIENCE	MASTG-TEST-0047	Prueba de comprobación de integridad de archivos.
MASVS-RESILIENCE	MASTG-TEST-0049	Prueba de la detección del emulador.
MASVS-STORAGE	MASTG-TEST-0004	Determinar si se comparten datos confidenciales con terceros a través de embebidos.
MASVS-STORAGE	MASTG-TEST-0011	Prueba de memoria de datos confidenciales.

Tabla 6.2: Pruebas OWASP seleccionadas para la auditoría de AquaCyL.

[1]Componente nativo para embeber dentro de las apps contenido web.

En la anterior tabla, se puede ver como se cubren todas las categorías que OWASP presenta. Teniendo en cuenta lo anterior y observando las pruebas seleccionadas, se puede comprobar que se cumplen los criterios de selección que se han mencionado anteriormente y, con ello se puede comenzar a realizar el diseño de ellas.

6.4. Diseño de las pruebas.

Una vez seleccionadas las pruebas a lanzar, el siguiente paso es realizar su diseño. Para ello hay que tener en cuenta los diferentes tipos de pruebas que existen, así como las categorías de análisis que se utilizan. Con lo anterior se puede realizar el diseño de cada una de las pruebas escogidas.

6.4.1. Tipos de pruebas.

A la hora de realizar las diferentes pruebas, ésta se pueden clasificar según el conocimiento que la persona encargada para realizar las pruebas tenga sobre el código fuente de la aplicación [37]:

- **Caja negra:** Se prueba la funcionalidad de la aplicación desde el punto de vista del usuario final, sin conocer el código fuente o la estructura de datos que se analiza.
- **Caja blanca:** Se cuenta con el código fuente de la aplicación, de modo que se puede analizar de forma profunda el código y por ende la estructura a bajo nivel de la aplicación.
- **Caja gris:** Se tiene acceso parcial al código fuente, así como un conocimiento limitado del sistema. Es una mezcla de las pruebas de caja blanca y caja negra aprovechando el conocimiento ambas sin llegar a ser pruebas de ninguno de los dos tipos.

6.4.2. Categorías de análisis.

Por otro lado, las pruebas pueden ser clasificadas según el estado de la aplicación en el momento de la realización de la prueba. Existen dos tipos de análisis en función de lo anterior [38]:

- **Análisis estático:** Análisis del código fuente sin ejecutarlo o en reposo. Se analiza la estructura del código y del software disponible. Se trata de buscar vulnerabilidades, malas prácticas o errores de lógica.
- **Análisis dinámico:** Se realiza con la aplicación en ejecución. Se busca observar el comportamiento de la aplicación en tiempo real dentro de un entorno controlado, que puede ser un emulador o un dispositivo real.

6.4.3. Diseño de las pruebas seleccionadas.

Dentro de todas las pruebas disponibles, se ha realizado una selección de 15 pruebas cuyos resultados, en caso de explotar vulnerabilidades, podrían dar acceso a datos de carácter personal. El resto de pruebas de las que se ha hablado previamente pueden ser lanzadas con el fin de explotar más vulnerabilidades. No obstante, esto supondría una extensión mayor a lo que abarca el presente proyecto.

A continuación se detalla el diseño de cada una de las pruebas elegidas:

MASTG-TEST-0002	**Prueba del almacenamiento local para la validación de los datos de entrada.**
Objetivos	Constatar que los datos que se almacenan de forma local no se utilicen sin validar antes de volver a usarse.
Pruebas	▪ **Análisis estático:** • Inspección del código fuente para buscar si se utiliza *SharedPreferences* puesto que con ello no es posible verificar si los datos de tipo int, boolean o long se sobreescriben. • En caso de que se use *SharedPreferences* [39], comprobar que se valida su uso.

Tabla 6.3: Diseño de la prueba MASTG-TEST-0002: Prueba del almacenamiento local para la validación de los datos de entrada.

MASTG-TEST-0004	**Determinar si se comparten datos confidenciales con terceros a través de embebidos.**
Objetivos	Averiguar si la aplicación comparte datos sensibles del usuario con terceros sin consentimiento o conocimiento explícito de dicho usuario.
Pruebas	▪ **Análisis estático:** • Revisar el código fuente, los permisos que se solicitan y verificar si existen vulnerabilidades conocidas. • Verificar que los datos que se envían a terceros se anonimizan para evitar que se exponga información de carácter personal que permita a éstos identificar al usuario. Los identificadores que se asignan a la cuenta no deben ser enviados a terceros. ▪ **Análisis dinámico:** • Lanzar un ataque Man-in-the-middle [40] para interceptar y analizar el tráfico entre el cliente y el servidor. Con ello se puede tratar de rastrear el tráfico entre aplicación y servidor de modo que aquellas solicitudes de la aplicación que no se envían de forma directa al servidor deben revisarse para localizar información confidencial. • Revisar las solicitudes a servicios externos a la aplicación para hallar si se comparte información confidencial.

Tabla 6.4: Diseño de la prueba MASTG-TEST-0004: Determinar si se comparten datos confidenciales con terceros a través de embebidos.

MASTG-TEST-0008	**Comprobación de la divulgación de datos confidenciales a través de la interfaz.**
Objetivos	Verificar que no se comparten datos de carácter confidencial por medio de la interfaz.
Pruebas	▪ **Análisis estático:** Buscar en el código fuente la gestión de las notificaciones, en concreto el uso de *NotificationManager* [41] en busca de indicios de que la aplicación envíe datos personales y asegurarse que en los campos *EditText* [42] las contraseñas se enmascaren (se use *android:inputType="textPassword"*). ▪ **Análisis dinámico:** Revisar la aplicación y su funcionalidad buscando formas de activar las notificaciones evaluando si contiene información confidencial.

Tabla 6.5: Diseño de la prueba MASTG-TEST-0008: Comprobación de la divulgación de datos confidenciales a través de la interfaz.

MASTG-TEST-0011	**Prueba de memoria de datos confidenciales.**
Objetivos	Determinar si los datos sensibles permanecen almacenados en memoria en texto plano mientras se usan, o después, o si por el contrario se eliminan de forma adecuada. Es de especial relevancia que se encuentren cifrados para que ningún atacante pueda tener acceso a estos.
Pruebas	▪ **Análisis estático:** Para identificar datos confidenciales expuestos en memoria hay que: • Asegurarse que los datos confidenciales sean manejados por el menor número posible de componentes. • Identificar los componentes de la aplicación y averiguar dónde se utilizan los datos. • Asegurarse de que se solicite la recolección de basura una vez se hayan eliminado las referencias. • Afianzar que las referencias de los objetos se eliminan de forma correcta y segura una vez que el objeto que cuenta con los datos confidenciales ya no sea necesario. • Cerciorarse de que los datos de carácter confidencial se sobrescriben tan pronto como ya no sean necesarios: ◦ Se evitan los tipos de datos no primitivos. ◦ Se preste atención a los componentes de terceros como APIs públicas, frameworks o bibliotecas. ◦ No represente datos confidenciales con datos inmutables. ◦ Sobrescriba las referencias antes de eliminarlas fuera de la cláusula *finalize*[2]. ▪ **Análisis dinámico:** Realización de un volcado y análisis explícito del heap de Java mediante las diferentes herramientas integradas en Android Studio de modo que, durante el análisis se busquen: • Patrones indicativos de datos confidenciales. • Nombres de campo indicativos como puede ser contraseña, pin, secreto, etc. • Secretos conocidos como por ejemplo algún dato personal.

Tabla 6.6: Diseño de la prueba MASTG-TEST-0011: Prueba de memoria de datos confidenciales.

[2]Permite controlar la finalización.

MASTG-TEST-0014	**Prueba de la configuración del algoritmo estándar de criptografía.**
Objetivos	Comprobar que la aplicación: ▪ Utiliza algoritmos criptográficos seguros (evitando aquellos inseguros como SHA-1 o MD5). ▪ Usa bibliotecas en uso y seguras en lugar de implementarlas desde cero.
Pruebas	▪ **Análisis estático:** Identificar las clases, funciones, interfaces o excepciones relacionadas con la criptografía dentro del código fuente de la aplicación. ▪ **Análisis dinámico:** Ejecutar la aplicación con el fin de: • Interceptar llamadas a funciones criptográficas. • Verificar si las contraseñas se almacenan haciendo uso de funciones inseguras. • Inspección en tiempo real de los datos cifrados y de los algoritmos usados.

Tabla 6.7: Diseño de la prueba MASTG-TEST-0014: Prueba de la configuración del algoritmo estándar de criptografía.

MASTG-TEST-0017	**Prueba para confirmar credenciales.**
Objetivos	▪ Verificar que la aplicación solicita al usuario confirmar sus credenciales antes de realizar diferentes operaciones como: • Eliminar la cuenta. • Cambiar información sensible del perfil. • Modificar la contraseña. ▪ Evitar que los usuarios no autorizados realicen acciones críticas con la cuenta de otro usuario.
Pruebas	▪ **Análisis estático:** Confirmar que la clave desbloqueada se use durante el flujo de la aplicación. Si solamente comprueba si el usuario ha desbloqueado la clave, puede ser vulnerable a una omisión de autenticación local. ▪ **Análisis dinámico:** En el caso en el que *setUserAuthenticationRequired*[3] se utiliza, hay que validar el tiempo durante el cual se autoriza el uso de la clave una vez el usuario se ha autenticado correctamente.

Tabla 6.8: Diseño de la prueba MASTG-TEST-0017: Prueba para confirmar credenciales.

[3]Habilita o deshabilita que el usuario tenga que autenticarse en el momento de la conexión.

MASTG-TEST-0023	**Prueba de proveedor de seguridad.**
Objetivos	Asegurar que la aplicación no requiera de implementaciones no seguras de los servicios de Google (*GooglePlayServices*) y del proveedor de seguridad de Android (*Android Security Provider*), encargado de certificados, criptografía, etc; y que las versiones que utiliza son fiables y se encuentran en su última versión.
Pruebas	▪ **Análisis estático:** • Verificar que la aplicación está basada en el SDK de Android y si es el caso que dependa de GooglePlayServices. • Asegurar que la clase *ProviderInstaller*[4] se llame con *installIfNeeded*[5] o *installIfNeededAsync*[6]. • Comprobar que un componente de la aplicación llame a *ProviderInstaller* lo antes posible y que las excepciones generadas por los métodos se detecten y gestionen de forma correcta. • Verificar que la aplicación gestione de forma correcta las excepciones relacionadas con las actualizaciones del proveedor de seguridad y que informe al BackEnd cuando ésta funciona con un proveedor de seguridad sin parchear. ▪ **Análisis dinámico:** Seguir los siguientes pasos: • Ejecutar la aplicación en modo depuración. • Crear un punto de interrupción donde la aplicación se comunica primero con los puntos finales. • Evaluar la expresión resaltada. • Escribir *Security.getProviders()*[7]. • Verificar los proveedores e intentar encontrar *GmsCore_OpenSSL*[8].

Tabla 6.9: Diseño de la prueba MASTG-TEST-0023: Prueba de proveedor de seguridad.

[4]Clase para instalar un proveedor de forma dinámica.
[5]Instala el proveedor de seguridad dinámicamente si no se encuentra instalado.
[6]Instala de forma asíncrona el proveedor de seguridad de forma dinámica si no está instalado.
[7]Devuelve todos los proveedores instalados.
[8]Tipo de proveedor utilizado con *ProviderInstaller*.

MASTG-TEST-0026	**Prueba de intenciones implícita.**
Objetivos	▪ Comprobar si la aplicación es vulnerable a ataques de inyección o a filtraciones de datos de carácter confidencial. ▪ Evaluar si existen intenciones implícitas [43], que son aquellas que no se refieren a un componente específico, si no que declaran una acción genérica que un componente de otra aplicación puede ejecutar. El uso de dichas intenciones pueden producir riesgos de seguridad, como la filtración de datos confidenciales.
Pruebas	▪ **Análisis estático:** Inspeccionar el fichero *AndroidManifest*[9] con el fin de buscar firmas definidas en sus bloques internos, donde se especifica el conjunto de otras aplicaciones con las que la aplicación busca interaccionar y verificar si contiene alguna acción del sistema como pueden ser *android.intent.action.GET_ CONTENT*[10], *android.intent.action.PICK*[11] o *android.media.action.IMAGE_ CAPTURE*[12]. ▪ **Análisis dinámico:** Usar Frida o Frida-trace [44], conectar los métodos *startActivityForResult*[13] y *onActivityResult*[14] e inspeccionar las intenciones proporcionadas y aquellos datos que contienen.

Tabla 6.10: Diseño de la prueba MASTG-TEST-0026: Prueba de intenciones implícita.

[9]Fichero que cuenta con información esencial de la aplicación para las herramientas de compilación de Android, el sistema operativo y Google Play.
[10]Muestra diferentes datos para que el usuario seleccione el que requiera.
[11]Devuelve una lista de recursos que el usuario selecciona para devolverlo al elemento que lo ha llamado.
[12]Permite capturar una foto.
[13]Inicia una actividad de la que se desea obtener un resultado al acabar.
[14]Devuelve un objeto del tipo definido en *ActivityResultContract*.

MASTG-TEST-0027	**Prueba de carga de URL en WebViews.**
Objetivos	Comprobar que la aplicación carga URLs de forma segura, de modo que se impida que se pueda redirigir al usuario a sitios fraudulentos o maliciosos, así como ejecutar ataques de inyección o phishing.
Pruebas	▪ **Análisis estático:** • Comprobar la anulación del manejo de la navegación de la página. Para ello hay que inspeccionar lo que devuelven las siguientes funciones: ◦ *shouldOverrideUrlLoading*: Permite que la web cargue la URL si devuelve *false* o que cancele la carga con contenido sospechoso al devolver *true*. Hay que tener en cuenta las siguientes consideraciones: ◇ No se llama para solicitudes del tipo POST[15]. ◇ No se llama para *iFrames*[16], *XmlHttpRequest*[17] ni atributos *src* incluidos en el HTML o en las etiquetas *script*[18]. De esto debe encargarse *shouldInterceptRequest*. ◦ *shouldInterceptRequest*: Deja que la aplicación retorne los datos de las distintas solicitudes de los recursos. En caso de retornar un valor nulo, la *WebView* seguirá cargando el recurso de forma normal. De lo contrario se utilizan aquellos datos que retorna el método. Consideraciones: ◇ Si la navegación segura se encuentra habilitada, las URLs se someten a las verificaciones de navegación segura aunque el desarrollador puede permitir la URL o ignorar la advertencia mediante el retorno de la llamada *onSafeBrowsingHit*[19]. ◇ No se requiere para *JavaScript* [45] ni *blob*[20] o para recursos accedidos vía *file:///android_asset/*[21] o *file:///android_res/*[22]. En el caso de redirecciones, únicamente se requiere para URL inicial, no para las que se redirige. ◇ El retorno de la función se invoca para varios URL como pueden ser *http(s):*[23], *file:*[24], *data:*[25], etc. • Verificar que *EnableSafeBrowsing*[26] se encuentra habilitado. Hay que revisar el fichero AndroidManifest.xml[27] y ver el estado de dicho componente. ▪ **Análisis dinámico:** Ejecutar Frida o frida-trace [44] para conectar los métodos *shouldOverrideUrlLoading* y *shouldInterceptRequest* y hacer click en los enlaces dentro de la web. A mayores hay que conectar otros métodos como *getHost*[28], *getScheme*[29] o *getPath*[30] para inspeccionar las solicitudes, listas de denegación o patrones conocidos.

Tabla 6.11: Diseño de la prueba MASTG-TEST-0027: Prueba de carga de URL en WebViews.

MASTG-TEST-0036	**Prueba de actualización forzada.**
Objetivos	Verificar si la aplicación: ▪ Tiene soporte para actualizaciones. ▪ Se implementa correctamente, de modo que el usuario no pueda seguir usando la aplicación sin actualizarla primera.
Pruebas	▪ **Análisis estático:** Analizar el código fuente en busca de fragmentos relacionados con el control de versiones, condiciones de funcionamiento en caso de versiones inferiores o si existe una lógica que active algún tipo de mecanismo que impida el funcionamiento de la aplicación y que obligue a la actualización en caso de contar con una versión anterior. ▪ **Análisis dinámico:** • Probar a descargar una versión anterior de la aplicación para comprobar si deja ejecutarla o si por el contrario obliga a actualizar. • Tratar de modificar el número de versión de la aplicación a la vez que se intercepta su tráfico mediante un proxy MIMT [46] para ver como responde el backend.

Tabla 6.12: Diseño de la prueba MASTG-TEST-0036: Prueba de actualización forzada.

[15]Método para enviar datos a un servidor.
[16]Elemento HTML para insertar contenido de otro sitio web dentro de la página web actual.
[17]Objeto que permite a un navegador web realizar peticiones HTTPS y HTTP a un servidor y recibir respuestas sin recargar la página.
[18]Etiqueta para insertar o referenciar código ejecutable dentro de las webs.
[19]Servicio por el que se detecta si un sitio web es o no seguro advirtiendo al usuario si acceder o no.
[20]Tipo de dato usado para almacenar grandes cantidades de datos.
[21]Identificador de Recurso Uniforma referido al directorio de recursos de una aplicación.
[22]Esquema de URL usado para referenciar recursos dentro de una aplicación.
[23]Indica que la dirección se refiere a un recurso en linea.
[24]Indica que la dirección se refiere a un fichero local.
[25]Esquema que permite insertar datos en la URL de forma directa.
[26]Permite comprobar si una WebView es segura.
[27]Fichero que cuenta con información esencial de la aplicación para las herramientas de compilación de Android, el sistema operativo y Google Play.
[28]Método para obtener el host de un sitio web.
[29]Permite recuperar información de la estructura de datos de una web.
[30]Método para obtener la ruta de una web.

MASTG-TEST-0037	**Prueba de limpieza de WebViews.**
Objetivos	Probar que las aplicaciones que hacen uso de WebViews eliminan los datos de esta WebView (datos sensibles, historial, caché, trazas, etc...) tras cerrar la aplicación, la sesión o al desinstalar.
Pruebas	▪ **Análisis estático:** Identificar el uso de las API de WebView que se muestran a continuación: • **Inicialización:** La aplicación podría inicializar la WebView con el objetivo de evitar almacenar cierta información. • **Caché:** La clase WebView[31], ofrece el método *clearCaché* [47] para borrar la caché en todas las WebViews usadas por la aplicación. Cuenta con el parámetro *includeDiskFiles*, si devuelve *true*, borra todos los recursos almacenados incluida la RAM [32], si por el contrario, devuelve *false* solo se borra la caché RAM. • **API de WebStorage[33]:** *WebStorage.deleteAllData* [48] se puede usar para borrar el almacenamiento que usan las API de almacenamiento de JavaScript [45], incluyendo las bases de datos SQL [49] web y las API [34] de almacenamiento web de HTML5 [50]. • **Cookies:** Verificar que las cookies existentes se eliminan. Para ello se puede usar *CookieManager.removeAllCookies*. • **API de archivos:** Comprobar que se eliminan de forma correcta los datos, incluyendo los de ciertos directorios que contienen información del usuario, en los que se hace de forma manual. ▪ **Análisis dinámico:** Abrir una WebView que acceda a datos de carácter confidencial y cerrar la sesión de la aplicación. Acceder al contenedor de almacenamiento de la aplicación asegurándose de eliminar todos los ficheros relacionados con esa WebView. Los siguientes ficheros y directorios suelen estar relacionados con las WebViews: • Cookies[34] • app_webview[35] • blob_storage[36] • pref_store[37] • Session Storage[38] • Web Data[39] • Service Worker[40]

Tabla 6.13: Diseño de la prueba MASTG-TEST-0037: Prueba de limpieza de WebViews.

MASTG-TEST-0040	**Prueba de símbolos de debugging.**
Objetivos	Verificar que los ficheros de la aplicación no cuenten con símbolos de depuración, datos innecesarios que pueden facilitar ingeniería inversa o metadatos.
Pruebas	▪ **Análisis estático:** Puesto que normalmente, los símbolos se eliminan durante el proceso de compilación, se requiere el código en bytes y las bibliotecas compiladas, con el fin de asegurar que se hayan descartado los metadatos innecesarios. • En primer lugar hay que buscar el binario *nm*[41] en el NDK de Android[42] y exportarlo. • Mostrar los símbolos de depuración o verificar los símbolos manualmente mediante un desensamblador. ▪ **Análisis dinámico:** Se debe usar el análisis estático para llevar a cabo esta prueba.

Tabla 6.14: Diseño de la prueba MASTG-TEST-0040: Prueba de símbolos de debugging.

[31]Componente nativo para embeber dentro de las aplicaciones contenido web.
[32]Tipo de memoria que almacena información de forma temporal.
[33]API del almacenamiento de la WebView.
[34]Almacena las cookies de la WebViews.
[35]Guarda la información necesaria para que se pueda mostrar la WebView desde la aplicación.
[36]Almacena objetos inmutables.
[37]Almacena datos relacionados con las preferencias de la WebView.
[38]Guarda información sobre el los datos que se almacenan en una sesión de usuario.
[39]Almacena información sobre la WebView.
[40]Guarda información relacionada con las notificaciones, caché y gestión de recursos de la WebView dentro de la aplicación.
[41]Parte del nombre del binario que hay que buscar para exportarlo y lanzar la prueba.
[42]Conjunto de herramientas que permite a los desarrolladores incorporar código C o C++ en las aplicaciones.

MASTG-TEST-0043	**Errores de corrupción de memoria.**
Objetivos	Comprobar que la aplicación no es vulnerable a los diferentes errores de memoria como pueden ser: ▪ Uso de la memoria tras liberarla. ▪ Desbordamiento de búfer. ▪ Condiciones de carrera. ▪ Escritura fuera de los límites establecidos.
Pruebas	▪ **Análisis estático:** Buscar dentro del código los siguientes elementos: • **Código Java o Kotlin [51]:** Buscar problemas de serialización/deserialización. • **Código nativo:** Si es el caso, revisar problemas de corrupción de memoria. ▪ **Análisis dinámico:** Seguir los siguientes pasos: • Si hay código nativo, usar *Valgrind* [52] o con el fin de analizar el uso de la memoria y las llamadas a memoria que realiza el código. • Si hay código en Java/Kotlin, tratar de volver a compilar la aplicación y usarla con *Squares leak canary* [53]. • Para vulnerabilidades de serialización seguir *Android Java Deserialization Vulnerability Tester* [54]. • Comprobar el generador de perfiles de memoria de AndroidStudio en caso de existencia de fugas.

Tabla 6.15: Diseño de la prueba MASTG-TEST-0043: Errores de corrupción de memoria.

MASTG-TEST-0047	**Prueba de comprobación de integridad de archivos.**
Objetivos	Comprobar que la aplicación implementa comprobaciones de integridad para detectar cambios no autorizados, para evitar la manipulación de la APK, ingeniería inversa, inyección de código, etc...
Pruebas	▪ **Comprobación de integridad de la fuente de la aplicación:** Hay que responder a las siguientes preguntas: • ¿Cómo de difícil resulta la identificación de código anti-depuración mediante análisis estático y dinámico? • ¿Fue necesario escribir código para desactivar las defensas? • ¿Se pueden eludir los mecanismos de forma trivial? ▪ **Comprobación de integridad del almacenamiento:** Al igual que en el anterior caso, hay que responder ciertas preguntas: • ¿Se pueden eludir los mecanismos de forma trivial? • ¿Cómo de fácil es obtener la clave privada asimétrica o la clave HMAC (Hash-based Message Authentication Code o Código de Autenticación de Mensaje basado en Hash) [55]? • ¿Hubo que desarrollar código para desactivar las defensas?

Tabla 6.16: Diseño de la prueba MASTG-TEST-0047: Prueba de comprobación de integridad de archivos.

MASTG-TEST-0049	**Prueba de la detección del emulador**
Objetivos	Verificar si la aplicación cuenta con algún mecanismo para identificar si está siendo ejecutada en algún emulador, en vez de en un dispositivo real. La ejecución en un emulador puede tener como finalidad la automatización de ataques o el análisis dinámico.
Pruebas	▪ **Análisis dinámico:** Ejecutar la aplicación en un emulador. La aplicación debe detectar que está siendo ejecutada en un emulador y, como respuesta finalizar o rechazar la ejecución de la funcionalidad que debe proteger.

Tabla 6.17: Diseño de la prueba MASTG-TEST-0049: Prueba de detección del emulador.

Capítulo 7

Lanzamiento de las pruebas sobre InsecureBankv2.

Antes de realizar la auditoría de seguridad sobre AquaCyL, se ha considerado realizar una primera ejecución de las pruebas ya seleccionadas y diseñadas sobre una aplicación vulnerable conocida. De este modo se puede verificar la viabilidad técnica y si el diseño que se ha realizado es el más adecuado de cara a la aplicación real.

Con el anterior objetivo, se ha seleccionado InsecureBankv2, un aplicación que simula una aplicación bancaria, con sus funcionalidades típicas, desarrollada con vulnerabilidades con la finalidad de enseñar a quien esté interesado a auditar y aprender aspectos de seguridad. Se puede obtener desde su repositorio oficial en GitHub [56]. Es una de las aplicaciones que OWASP propone para explotar vulnerabilidades, siendo la aplicación denominada como *MASTG-APP-0010* [57].

En el presente capítulo se documenta la famiilarización con el entorno de la aplicación, así como el resultado de lanzar las pruebas seleccionadas con el objetivo de comprobar si el diseño realizado permite identificar las vulnerabilidades que puedan existir en la aplicación.

7.1. Preparación del entorno de pruebas.

En el repositorio de GitHub de InsecureBankv2, aparece un manual de configuración de la aplicación. Para ejecutar la aplicación hay que completar dos pasos: configuración de la aplicación dentro del emulador y puesta en marcha del servidor.

7.1.1. Instalación y configuración del emulador.

Para poder ejecutar la aplicación, es necesario crear un dispositivo virtual. Para ello, hay que crear una cuenta en Genymotion [58], el emulador que desde la documentación de InsecureBankv2 propone.

Para crear la cuenta se solicitan los siguientes datos:

- Correo electrónico.

- Contraseña.
- Uso: Seleccionar *Development and testing*[1].
- Tipo de empresa: Seleccionar *Others*[2].
- País.

Una vez creada la cuenta, hay que activarla mediante un enlace que se envía al correo electrónico proporcionado.

Tras activar la cuenta, hay que navegar al menú *Get Licence*[3], donde se elegirá la licencia gratuita de uso personal. Seguidamente, hay que descargar la versión del emulador para el sistema operativo que se desee. En este caso Linux.

El siguiente paso es iniciar Genymotion y seleccionar *Sí* para añadir un nuevo dispositivo virtual. Posteriormente hay que iniciar sesión con la cuenta que se ha creado previamente y, una vez hecho esto hay que configurar un nuevo dispositivo virtual, con los datos:

```
Google Nexus 5X { Android 8.0 { API 26 { 1080x1920
```

Lo siguiente es iniciar el dispositivo virtual y asegurarse de que funciona correctamente y ejecutar el siguiente comando en la máquina local para instalar *adb*:

```
sudo apt install android-tools-adb
```

7.1.2. Puesta en marcha del servidor.

En primer lugar hay que clonar el repositorio a la máquina local con el comando:

```
git clone https://github.com/dineshshetty/Android-InsecureBankv2
```

El siguiente paso es navegar hacia el directorio *AndroLabServer* e instalar las dependencias que el servidor requiere [59]:

```
cd Android-InsecureBankv2/AndroLabServer
pip install -r requirements.txt
```

Una vez hecho lo anterior, hay que ejecutar el servidor en el puerto por defecto, (*8888 HTTP*:

```
python2 app.py
```

[1] Desarrollo y pruebas.
[2] Otros.
[3] Obtener licencia.

La decisión de utilizar Python2 frente a Python3 es puesto que app.py está escrito en sintaxis para Python2 [60]. Esto se puede comprobar al ejecutarlo con Python3, ya que aparece el siguiente error:

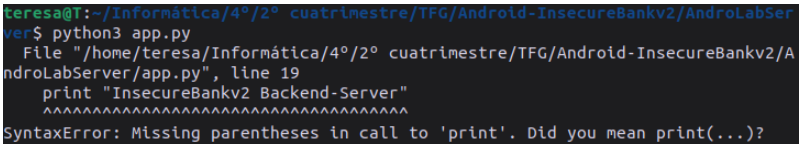

Figura 7.1: Error de ejecución de app.py con Python3.

Al ejecutarlo con Python2, el servidor está corriendo en el puerto 8888.

7.1.3. Configuración de la aplicación en el emulador.

Una vez corriendo el servidor y con el emulador en marcha, el siguiente paso es realizar la configuración para ejecutar la aplicación en el emulador.

Para ello hay que utilizar el comando *adb* desde el directorio para instalar la aplicación en el navegador:

```
adb install InsecureBankv2.apk
```

Con lo anterior, ya se puede acceder a InsecureBankv2 desde el emulador ya configurado.

7.2. Familiarización con la aplicación.

El primer paso nada más abrir la aplicación, es acceder a *Preferences* (preferencias), ubicado en el panel superior izquierdo pulsando sobre los tres puntos. Ahí hay que comprobar que la dirección IP del servidor es la **10.0.2.2**[4] y que el puerto en el que está siendo ejecutado es el **8888**.

Con lo anterior, ya se puede acceder a la aplicación con cualquiera de las credenciales:

Usuario	**Contraseña**
dinesh	Dinesh@123$
jack	Jack@123$

Tabla 7.1: Credenciales disponibles en InsecureBankv2.

En este caso se ha optado por iniciar sesión con el usuario **jack**, aunque la interfaz y el funcionamiento de la aplicación es similar para ambos usuarios.

[4]En ocasiones, la dirección IP que hay que poner es la 10.0.2.3. Para averiguar cual de las dos es, basta con abrir el navegador disponible en el emulador y buscar la IP con el formato *http://IP:8888/*, la IP correcta será aquella que se muestre con un mensaje de *Not Found*, esto es porque aún no hay nada asignado a / en la web pero el servidor es encontrado por el emulador.

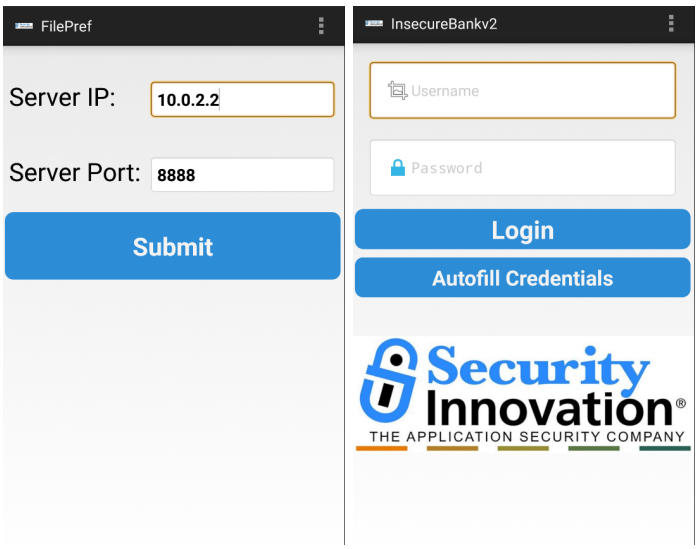

Figura 7.2: Interfaces *Preferences* e *Inicio de sesión* de la aplicación InsecureBankv2.

En la figura anterior se pueden ver las interfaces de inicio y de preferencias. Esta última con la información correspondiente para la correcta conexión con el servidor ya configurado anteriormente.

Para iniciar sesión es únicamente necesario usuario y contraseña. Aunque también se da la opción de autorellenar las credenciales (*Autofill Credentials)*, en el primer arranque aparece un mensaje que indica que aún no se ha iniciado sesión con ningún usuario. No obstante, si se inicia sesión, se cierra la aplicación y se vuelve a abrir. Al intentar hacerlo si se pulsa sobre dicho botón, aparecen las credenciales (usuario y contraseña) del último usuario que se ha iniciado sesión.

Una vez las credenciales son las correctas, se redirige a la pantalla de inicio o *PostLogin*, como lo denomina la aplicación. Ahí se muestra el mensaje *Rooted Device!!*, lo que significa que se tiene acceso como root o superusuario en el dispositivo. Esto tiene sentido ya que se trata de un dispositivo con la aplicación con fines de aprendizaje. Por otro lado se muestran tres botones:

- **Transfer:** Redirige a otra pantalla donde se da la opción de transferir. Es necesario la cuenta de origen y la de destino, pero si se pulsa en el botón **Ger Accounts** se muestran unas cuentas por defecto. También se solicita la cantidad a tranferir y un número de teléfono.

- **View Statement:** En el primer arranque de la aplicación, al pulsar sobre el botón aparece un mensaje que dice que no existe. No obstante, si se realiza una transferencia aparece reflejada.

- **Change Password:** Permite cambiar la contraseña de un usuario insertando nombre de usuario y nueva contraseña.

En la siguiente figura se muestran las interfaces de inicio y de transferencia, que cuentan con los elementos previamente explicados:

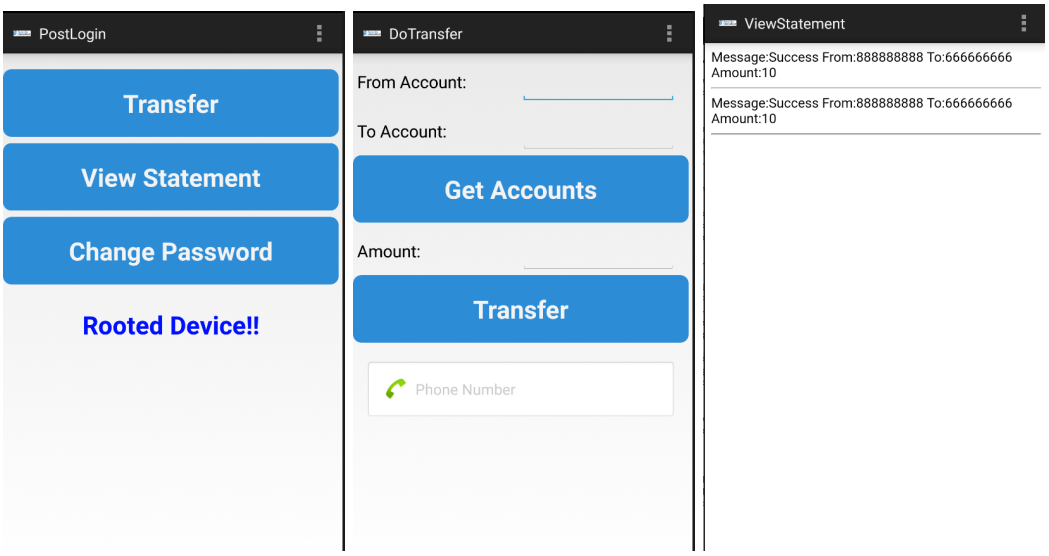

Figura 7.3: Interfaces *PostLogin*, *DoTransfer* y ViewStatement de la aplicación InsecureBankv2.

7.3. Ejecución de la selección de pruebas.

En la presente sección se describe, de forma breve el comportamiento de la aplicación ante cada prueba y se analiza si, en base a los resultados obtenidos, la prueba está correctamente diseñada y por ende, se puede lanzar sobre AquaCyL.

7.3.1. MASTG-TEST-0002 - Prueba del almacenamiento local para la validación de los datos de entrada.

Objetivo: Constatar que los datos que se almacenan de forma local no se utilicen sin validar antes de volver a usarse [61].

Herramientas utilizadas: Editor de texto, comandos cat [62] y grep [63].

Análisis estático: Se ha realizado un análisis directamente sobre el código fuente del proyecto, de modo que abriendo los ficheros Java disponibles con un editor de texto, se ha buscado el uso de la clase **SharedPreferences**, donde se identifica que se usa en múltiples ocasiones con el fin de almacenar información relacionada con el usuario (Por ejemplo en el fichero DoLogin.java). Se puede ver como esta información se recupera posteriormente sin hacer ningún tipo de validación.

Por otro lado, se ha probado desde terminal con los comandos cat y grep, para una búsqueda más rápida. Con el siguiente comando se muestran todas las entradas que contienen **SharedPreferences** dentro de un fichero Java[5]:

```
cat fichero.java | grep SharedPreferences
```

[5]Si se desease ver todas las apariciones de la clase buscando en todos los ficheros del directorio en el que se encuentra, bastaría por sustituir el nombre del fichero por un *.

Resultado en InsecureBankv2: Prueba válida. La aplicación reutiliza datos de carácter sensible sin validar.

7.3.2. MASTG-TEST-0004 - Determinar si se comparten datos confidenciales con terceros a través de datos embebidos.

Objetivo: Averiguar si la aplicación comparte datos sensibles del usuario con terceros sin consentimiento o conocimiento explícito de dicho usuario [64].

Herramientas utilizadas: Editor de texto, Burp Suite [65], Genymotion [58].

Análisis estático: Se ha inspeccionado el código fuente con el fin de localizar los permisos que se utilizan y si se llama a bibliotecas o funciones de terceros. Se ha encontrado que se usa la biblioteca externa *Toasteroid*, que sirve para mostrar las notificaciones de tipo *toast* [66][6]. Dicha biblioteca no realiza conexiones a terceros. Por otro lado, no se ha encontrado evidencia de que se envíen datos a terceros

Análisis dinámico: Se ha lanzado un ataque Man-in-the-middle mediante Burp suite [65] con el fin de interceptar y analizar el tráfico de la aplicación mientras se usa. No se ha detectado comunicación con terceros.

Resultado en InsecureBankv2: InsecureBankv2 no comparte datos confidenciales con terceros.

7.3.3. MASTG-TEST-0008 - Comprobación de la divulgación de datos confidenciales a través de la interfaz.

Objetivo: Verificar que no se comparten datos de carácter confidencial por medio de la interfaz [67].

Herramientas utilizadas: Editor de texto, Genymotion [58].

Análisis estático: Se ha inspeccionado dentro del directorio *res/layout*, cada fichero con el fin de encontrar el uso de *android:inputType="textPassword"*. Se busca únicamente en dicho directorio ya que es en el que se define la interfaz gráfica de usuario. En ninguno de los ficheros, cuando aparece *EditText* se usa *android:inputType="textPassword"*.

Por otro lado, se ha buscado el uso de *NotificationManager*[7] para evaluar la gestión de las notificaciones. En este caso, no se utiliza dicha clase por lo que se puede concluir que no se usan notificaciones.

Análisis dinámico: Se ha ejecutado la aplicación en el emulador iniciando sesión y cambiando la contraseña. Puesto que ambos campos aparecen enmascarados (con puntos), no hay fugas por la interfaz.

[6] Notificaciones de alerta que muestran retroalimentación al usuario.
[7] Notifica al usuario que un evento está teniendo lugar.

En lo que a las notificaciones se refiere, durante el manejo de la aplicación no se ha encontrado evidencia de que aparezcan notificaciones.

Resultado en InsecureBankv2: Aunque no se utilice la función correspondiente para la gestión de contraseñas, en la aplicación en ejecución si que se muestran enmascaradas. InsecureBankv2 no lanza notificaciones.

7.3.4. MASTG-TEST-0011 - Prueba de memoria de datos confidenciales.

Objetivo: Determinar si los datos sensibles permanecen almacenados en memoria en texto plano mientras se usan, o después, o si por el contrario se eliminan de forma adecuada [68].

Herramientas utilizadas: Editor de texto, Android Studio, comando strings [69], Genymotion.

Análisis estático: Se ha analizado el código fuente correspondiente a acciones relacionadas con el inicio de sesión, el resultado obtenido es que se usa el tipo de dato primitivo *String* para almacenar datos sensibles como puede ser la contraseña del usuario, no se eliminan las referencias y no hay evidencia de llamadas a funciones de recolección de basura.

Análisis dinámico: Mediante Android Studio, y con la aplicación ejecutándose desde Genymotion con la sesión iniciada se ha realizado un volcado de memoria, con éste se ha usado el comando strings para comprobar si hay datos en texto plano en memoria. Se ha probado a buscar las credenciales del usario y se han encontrado en texto plano.

Resultado en InsecureBankv2: La aplicación resulta vulnerable tanto desde el punto de vista del análisis estático como del dinámico ya que se puede ver como se almacena la información en texto plano y no se gestiona de forma segura en el código fuente.

7.3.5. MASTG-TEST-0014 - Prueba de la configuración del algoritmo estándar de criptografía.

Objetivo: Comprobar que la aplicación [70]:

- Utiliza algoritmos criptográficos seguros (evitando aquellos inseguros como SHA-1 o MD5 [71]).
- Usa bibliotecas en uso y seguras en lugar de implementarlas desde cero.

Herramientas utilizadas: Comando grep, Frida, Genymotion

Análisis estático: Analizando el código fuente con la ayuda del comando *grep* se ha encontrado el uso de la clase *Cipher*, no obstante aunque aparecen funciones como *getInstance*, la lógica de cifrado se realiza mediante la clase *CryptoClass* lo que indica que se implementa la función criptográfica en vez de usar bibliotecas ya existentes.

Análisis dinámico: Se ha utilizado Frida [36] para interceptar en tiempo real las llamadas criptográficas que realiza la aplicación en ejecución, en este caso al iniciar sesión. El resultado es que la clave que descifra se ha creado con AES [72] y devuelve la clave en hexadecimal lo que lo vuelve sumamente vulnerable por la facilidad de obtención del texto en plano.

Resultado en InsecureBankv2: La implementación criptográfica es incorrecta tal y como se ha comprobado con ambos análisis. A nivel de código se usan funciones propias en vez de bibliotecas conocidas, y en el análisis dinámico se ha visto como la clave criptográfica que puede interceptar en texto claro gracias a Frida.

7.3.6. MASTG-TEST-0017 - Prueba para confirmar credenciales.

Objetivo:

- Verificar que la aplicación solicita al usuario confirmar sus credenciales antes de realizar diferentes operaciones como: [73]
  - Eliminar la cuenta.
  - Cambiar información sensible del perfil.
  - Modificar la contraseña.
- Evitar que los usuarios no autorizados realicen acciones críticas con la cuenta de otro usuario.

Herramientas utilizadas: Editor de texto, Genymotion.

Análisis estático: Se ha analizado el código fuente en busca de evidencias en las que se muestre que se puedan hacer tareas como puede ser cambiar la contraseña sin verificar credenciales. Tampoco hay un tiempo límite en el que la clave del usuario sea válida antes de expirar y requerir nueva autenticación.

Análisis dinámico: Se ha comprobado que no se utiliza *setUserAuthenticationRequired* [74] en ninguna de las funciones por lo que no hay tiempo límite por el que se autoriza el uso de la clave al usuario. Por otro lado, para modificar la contraseña, en ningún momento se solicita al usuario una confirmación de sus credenciales.

Resultado en InsecureBankv2: La aplicación no pasa la prueba ya que no cumple con los requisitos establecidos ni para análisis estático ni dinámica al no solicitar confirmación de las credenciales ni existir un tiempo límite de uso.

7.3.7. MASTG-TEST-0023 - Prueba de proveedor de seguridad.

Objetivo: Asegurar que la aplicación no requiera de implementaciones no seguras de los servicios de Google (*GooglePlayServices*) y del proveedor de seguridad de Android (Android Security Provider), encargado de certificados, criptografía, etc. Y que las versiones que utiliza son fiables y se encuentran en su última versión [75].

Herramientas utilizadas: Editor de texto, comando grep,

Análisis estático: Se ha analizado el código fuente en búsqueda del uso de la clase *ProviderInstaller* [76] o que la aplicación esté basado en el SDK de Android. En este caso no se ha encontrado evidencia de uso para ninguna de los dos elementos, ni elementos relacionados con un proveedor de seguridad.

Análisis dinámico: Puesto que en análisis estático no hay evidencia de uso de proveedor de seguridad, no hay punto de interrupción en el que interceptar su ejecución.

Resultado en InsecureBankv2: No existe referencia al uso de un proveedor de seguridad y por ende el uso de las funciones y elementos de seguridad que éstos proponen.

7.3.8. MASTG-TEST-0026 - Prueba de intenciones implícita.

Objetivo: [77]

- Comprobar si la aplicación es vulnerable a ataques de inyección o a filtraciones de datos de carácter confidencial.
- Evaluar si existen intenciones implícitas, que son aquellas que no se refieren a un componente específico, si no que declaran una acción genérica que un componente de otra aplicación puede ejecutar. El uso de dichas intenciones pueden producir riesgos de seguridad, como la filtración de datos confidenciales.

Herramientas utilizadas: Comandos grep y cat, Genymotion, Frida.

Análisis estático: Se ha analizado el fichero *AndroidManifest.xml* en busca de firmas definidas dentro del bloque *intent*, no obstante no se ha encontrado nada.

Análisis dinámico: Se ha ejecutado Frida con la aplicación en ejecución sobre el emulador conectando los métodos *startActivityForResult*[8] y *onActivityResult*[9] para inspeccionar las intenciones implícitas devueltas. Se ha encontrado que utiliza *startActivityForResult*, no obstante la intención devuelta es *null* por lo que se concluye que no hay intenciones implícitas.

Resultado en InsecureBankv2: Se ha observado como no existen intenciones implícitas ni una declaración de ellas en el código fuente.

7.3.9. MASTG-TEST-0027 - Prueba de carga de URL en WebViews.

Objetivo: Comprobar que la aplicación carga URLs de forma segura, de modo que se impida que se pueda redirigir al usuario a sitios fraudulentos o maliciosos, así como ejecutar ataques de inyección o phishing [78].

[8] Inicia una actividad y recibe un resultado de ella.
[9] Recibe datos de una actividad creada con *startActivityForResult*.

Herramientas utilizadas: Comandos cat y grep, Frida y Genymotion.

Análisis estático: Se ha revisado el código fuente en busca del uso de la clase *WebViewClient* así como el uso de la función *shouldOverrideUrlLoading*[10], las cuales se han encontrado en el fichero *MyWebViewClient.java* donde se sobrescribe la función nativa por otra que admite todas las URL sin comprobar origen. En *AndroidManifest.xml* se ha buscado que *EnableSafeBrowsing*[11] se encuentra habilitado, no obstante este componente no existe en dicho fichero.

Análisis dinámico: Se ha ejecutado Frida con la aplicación en ejecución sobre el emulador para inspeccionar las conexiones a las webs. La salida obtenida indica que no se restringe el origen del contenido. Tal y como también se ha demostrado en el análisis estático.

Resultado en InsecureBankv2: Se ha confirmado que la aplicación no carga de forma segura las URL puesto que no verifica su origen.

7.3.10. MASTG-TEST-0036 - Prueba de actualización forzada.

Objetivo: [79]

- Tiene soporte para actualizaciones.
- Se implementa correctamente, de modo que el usuario no pueda seguir usando la aplicación sin actualizarla primera.

Herramientas utilizadas: Comando grep, vim [80].

Análisis estático: Se ha inspeccionado el código fuente en busca de algún elemento relacionado con el control de la versión y de actualizaciones. No obstante no se ha encontrado ninguna referencia en el código que controle esto.

Análisis dinámico: Se ha probado a cambiar el número de versión de la aplicación y lanzarla, no obstante al tratar de generar la nueva APK da error ya que hay paquetes que no se encuentran disponibles para su descarga por lo que no se ha realizado el análisis dinámico.

Resultado en InsecureBankv2: Aunque no se haya podido realizar el análisis dinámico, se ha visto cómo en el código fuente no se gestiona el control de versiones por lo que es muy posible que dinámicamente la aplicación funcione en versiones anteriores sin impedir la ejecución.

7.3.11. MASTG-TEST-0037 - Prueba de limpieza de WebViews.

Objetivo: Probar que las aplicaciones que hacen uso de WebViews eliminan los datos de esta WebView (datos sensibles, historial, caché, trazas, etc...) tras cerrar la aplicación, la sesión o al desinstalar [81].

[10] Permite que la aplicación tome el control sobre la carga de una URL dentro de una WebView.
[11] Permite comprobar si una WebView es segura.

Herramientas utilizadas: Comandos grep, adb, cd, ls y cat, Genymotion.

Análisis estático: Se ha buscado en el código fuente el uso de APIs relacionadas con WebViews para la inicialización de las vistas, caché, almacenamiento y cookies. No se ha encontrado nada relacionado con esto por lo que se puede decir que la aplicación no elimina la información de la sesión una vez se cierra.

Análisis dinámico: Se ha iniciado sesión en la aplicación y realizado una transferencia. Posteriormente se ha cerrado sesión y se ha accedido al almacenamiento de la aplicación, donde se ha encontrado la existencia de diferentes ficheros y directorios como *Web Data* o *blob_storage*, los cuales al cerrar la sesión en la aplcación deberían haber sido eliminados.

Resultado en InsecureBankv2: La aplicación no elimina los datos de las WebViews una vez se cierra la sesión.

7.3.12. MASTG-TEST-0040 - Prueba de símbolos de debugging.

Objetivo: Verificar que los ficheros de la aplicación no cuenten con símbolos de depuración, datos innecesarios que pueden facilitar ingeniería inversa o metadatos [82].

Herramientas utilizadas: Comandos cat y grep.

Análisis estático: Se ha inspeccionado el código fuente en busca del binario *nm*. No se ha encontrado dicho binario por lo que no usa código nativo por lo que la prueba no aplica a esta aplicación.

Resultado en InsecureBankv2: La aplicación no usa código nativo, por lo que la prueba no aplica.

7.3.13. MASTG-TEST-0043 - Errores de corrupción de memoria.

Objetivo: Comprobar que la aplicación no es vulnerable a los diferentes errores de memoria como pueden ser [83]:

- Uso de la memoria tras liberarla.
- Desbordamiento de búfer.
- Condiciones de carrera.
- Escritura fuera de los límites establecidos.

Herramientas utilizadas: Comando grep.

Análisis estático: Inspeccionando el código fuente no se ha encontrado ningún uso de código nativo, no obstante si que se han encontrado funciones relacionadas con la serialización/deserialización como puede ser *BroadcastReceiber* [84], sin embargo no es causante de fugas puesto que en *AndroidManifest.xml* está declarado de forma estática. Aún así es peligroso ya que sigue habiendo

riesgo de corrupción de memoria por el uso de elementos como *MODE_WORLD_READABLE* [85], elemento inseguro y obsoleto de Java.

Análisis dinámico: Al intentar probar a compilar de nuevo la aplicación y usar LeakCanary [53], aparecen errores ya que las versiones de algunas herramientas como *Gradle* [86], usada en la aplicación es muy antigua y entra en conflicto. Actualizarla no es una opción ya que el código cuenta con funciones obsoletas e incompatibles en versiones más actuales.

Resultado en InsecureBankv2: A pesar de no haber sido posible el análisis dinámico, por medio del estático se ha encontrado que se utilizan funciones que suponen un riesgo de seguridad para la aplicación relacionado con la corrupción de memoria.

7.3.14. MASTG-TEST-0047 - Prueba de comprobación de integridad de archivos.

Objetivo: Comprobar que la aplicación implementa comprobaciones de integridad para detectar cambios no autorizados, para evitar la manipulación de la APK, ingeniería inversa, inyección de código, etc... [87]

Herramientas utilizadas: Comandos adb y grep, vim, frida, Genymotion.

Comprobación de integridad de la fuente de la aplicación: Debido a conflictos con las versiones de la aplicación y firma de la APK para probarla modificada en el emulador, no ha sido posible probar la integridad del código fuente.

Comprobación de integridad del almacenamiento: Se ha modificado el fichero .html correspondiente al usuario *dinesh* y se ha vuelto a subir a la apk. Al ejecutar la aplicación aparece el fichero modificado sin mostrar ningún tipo de error. Por lo que las defensas son fácilmente eludibles sin necesidad de desarrollar código. La clave HMAC es fácilmente accesible si se realiza el inicio de sesión y se intercepta el tráfico con Frida.

Resultado en InsecureBankv2: La aplicación no cuenta con comprobaciones de integridad de almacenamiento.

7.3.15. MASTG-TEST-0049 - Prueba de la detección del emulador.

Objetivo: Verificar si la aplicación cuenta con algún mecanismo para identificar si está siendo ejecutada en algún emulador, en vez de en un dispositivo real. La ejecución en un emulador puede tener como finalidad la automatización de ataques o el análisis dinámico [88].

Herramientas utilizadas: Genymotion.

Análisis dinámico: Se ha ejecutado la aplicación desde un emulador y se ha interaccionado con ella. La aplicación no presenta ningún tipo de rechazo, bloqueo o comportamiento extraño por ser

ejecutada en un emulador.

Resultado en InsecureBankv2: La aplicación no altera su comportamiento al ser ejecutada en un emulador.

Una vez lanzadas todas las pruebas seleccionadas sobre la aplicación vulnerable dentro de un entorno seguro, se puede concluir que las pruebas en su totalidad se encuentran diseñadas correctamente ya que cumplen con su objetivo principal.

Capítulo 8

Lanzamiento de las pruebas sobre AquaCyL.

Una vez se ha corroborado que el diseño de las pruebas seleccionadas es el correcto, el siguiente paso es lanzarlas sobre la aplicación real: **AquaCyL**. Esto tiene como objetivo comprobar si la aplicación cumple con los criterios que OWASP establece. Es por ello por lo que se ejecutarán las pruebas una a una, a nivel estático y dinámico según lo establecido en el diseño.

Con los resultados que se obtengan de la ejecución de cada una de las pruebas, se podrá identificar si la aplicación tiene vulnerabilidades, su criticidad y el riesgo que suponen. A partir de ello, se realizará una serie de recomendaciones con el fin de ayudar a corregir o mitigar las vulnerabilidades encontradas para reforzar AquaCyL y proteger los datos.

El primer paso a llevar a cabo antes del lanzamiento de las pruebas es obtener el código fuente de la aplicación. Para el análisis estático es necesario descompilar la APK. Para ello basta con instalar y ejecutar *apktool* [89] con los comandos que se muestran a continuación:

```
sudo apt install apktook
apktool d AquaCyL_1.0.0_010000000.apk -o AquaCyL_decoded
```

Tras la ejecución de los comandos anteriores, en el directorio *AquaCyL_decoded* se cuenta con el código en formato *smali* [90], que es una representación del código a bajo nivel.

En lo que a nivel dinámico respecta, es necesario contar con un entorno controlado sobre el que ejecutar la aplicación. Para ello se ha seleccionado Android Studio [91], cuya configuración se muestra a continuación:

8.1. Configuración del entorno de pruebas controlado.

En la presente sección se muestran los pasos llevados a cabo para la configuración del entorno de pruebas controlado. Esto consiste en la instalación de Android Studio, entorno sobre el que se

ejecutará la AquaCyL para lanzar las pruebas.

En primer lugar hay que verificar que la máquina sobre la que se va a instaurar el programa cuenta con los requisitos necesarios para su instalación. La máquina en cuestión es una Ubuntu 22.04 por lo que los requisitos mínimos de instalación son los siguientes:

- Cualquier distribución Linux 64 bits compatible con Gnome, KDE o Unity DE.
- Arquitectura CPU x86_64.
- Al menos 8 GB RAM.
- Al menos 8 GB espacio de disco disponible.
- Resolución mínima de pantalla de 1280 x 800.

Para verificar los requisitos en la máquina que se va a utilizar hay que ejecutar los siguientes comandos en una terminal:

```
teresa@T:~$ lscpu
Arquitectura:                         x86_64
  modo(s) de operación de las CPUs:   32-bit, 64-bit
  Address sizes:                      39 bits physical, 48 bits virtual
  Orden de los bytes:                 Little Endian
CPU(s):                               12
  Lista de la(s) CPU(s) en línea:     0-11
ID de fabricante:                     GenuineIntel
  Nombre del modelo:                  Intel(R) Core(TM) i7-10750H CPU @ 2.60G
                                      Hz
    Familia de CPU:                   6
    Modelo:                           165
    Hilo(s) de procesamiento por núcleo: 2
    Núcleo(s) por «socket»:           6
    «Socket(s)»                       1
    Revisión:                         2
    CPU MHz máx.:                     5000,0000
    CPU MHz mín.:                     800,0000
    BogoMIPS:                         5199.98
```

Figura 8.1: Salida de lscpu para ver las características del procesador.

Tal y como puede verse en la anterior figura, al ejecutar el comando *lscpu* [92] se muestran las características del procesador. En este caso, en la linea *Arquitectura* se puede ver que se trata de una **x86_64** tal y como se requiere, y en la linea *Nombre del modelo* se puede ver que éste es un **Intel Core i7**.

```
teresa@T:~$ free -m
               total        usado       libre  compartido   búf/caché  disponible
Mem:           15762         3671         6828          754         5262       11002
Inter:          2047            0         2047
```

Figura 8.2: Salida del comando free para ver la memoria RAM disponible.

Mediante el comando *free -m* [93] se puede ver como hay casi **16 GB** de memoria RAM, más del doble del mínimo por lo que también se cumple con dicho requisito.

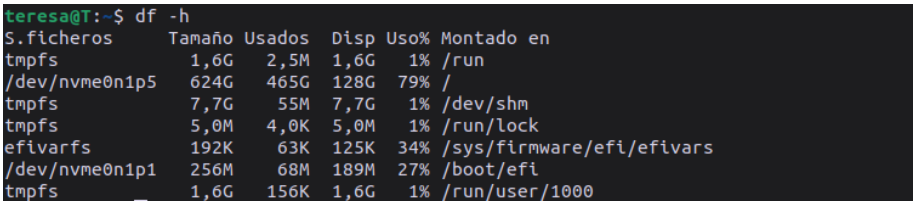

Figura 8.3: Salida de df con el que se ve el espacio de memoria en disco.

En la imagen anterior se muestra la salida del comando *df -h* [94], lo que devuelve cuanto espacio disponible hay en disco. De las múltiples salidas que hay, la que es de interés es la del directorio raíz o root (/). Se puede ver cómo hay disponibles **128 GB**, lo cual supera con creces el mínimo requerido de 8 GB.

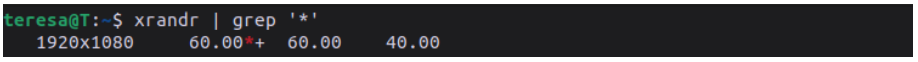

Figura 8.4: Salida del conjunto de comandos xrandr y grep para ver la resolución de pantalla.

Por último, para ver la resolución de la pantalla, basta con ejecutar los comandos *xrandr* [95], encargado de gestionar la configuración de las pantallas junto con *grep* [63] para filtrar por cadenas de texto. Con ello, la salida muestra qué resolución tiene la pantalla, en este caso **1920 x 1080**, lo cual también está por encima del requisito establecido.

Una vez comprobado que se cumplen los requisitos necesarios para la instalación, lo siguiente es descargar el programa desde la página oficial[1]. Desde allí hay que aceptar los términos y condiciones para poder proceder con la descarga. Una vez hecha, hay que extraer el fichero con el siguiente comando:

```
tar -xzvf android-studio-2024.3.2.15-linux.tar.gz [96]
```

Lo siguiente es dirigirse al directorio *android-studio/bin* con el comando *cd* [97] y ejecutar studio.sh de la siguiente forma:

```
./studio.sh
```

Con ello se abre la aplicación. Lo primero es elegir si se desea que se envíen estadísticas a Google o no, en este caso se ha optado por no hacerlo.

[1] https://developer.android.com/studio/?gclid=Cj0KCQiAjJOQBhCkARIsAEKMtO3zEhdK4_I0CEZic3UH4dl-9gVXuHFR9dCl3TOHKjmv3xWLU3UxfhYaApfAEALw_wcB&gclsrc=aw.ds&hl=es-419

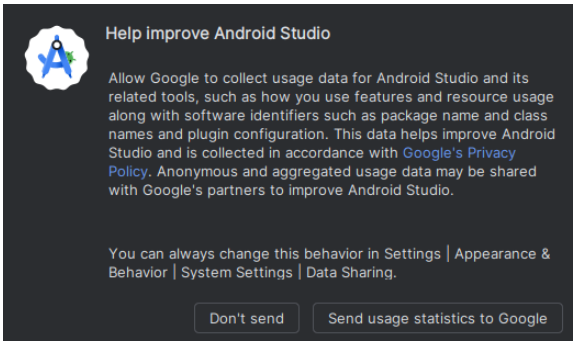

Figura 8.5: Enviar o no estadísticas a Google

Una vez aceptado o rechazado lo anterior, se muestra un asistente de configuración.
En primer lugar, hay que elegir entre el tipo de setup que se desea para la aplicación. Se puede optar entre la versión estándar, con los ajustes predeterminados, o por la versión personalizada, en la que se puede elegir la configuración. Para este caso, ya que la página de instalación de la aplicación lo recomienda [98], se ha optado por elegir la versión estándar tal y como se muestra en la siguiente figura.

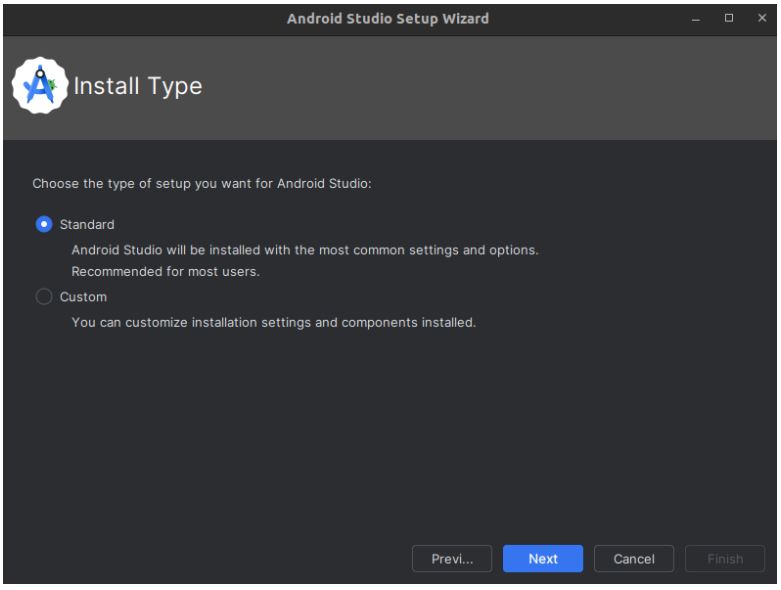

Figura 8.6: Tipo de instalación

Los siguientes pasos consisten en confirmar los ajustes, en los que se muestran los diferentes paquetes y aceptar la licencia de acuerdo. Una vez se completen estos pasos, hay que finalizar el proceso con lo que se instalarán los paquetes necesarios.

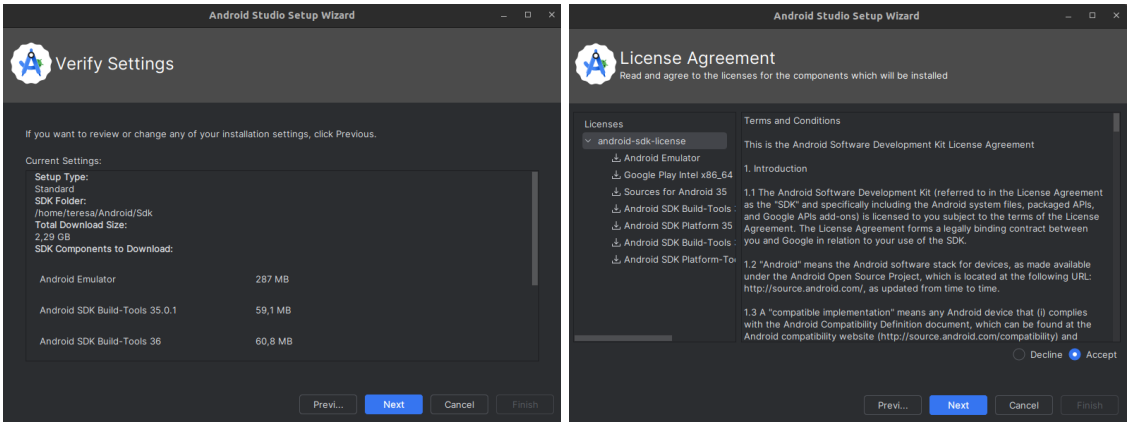

Figura 8.7: Ajustes finales y licencia

Tras finalizar la instalación, se mostrará la interfaz final sobre la que se va a trabajar:

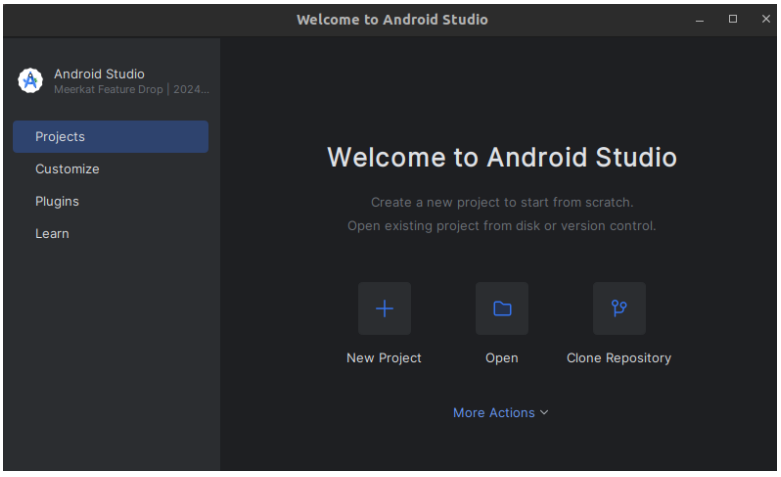

Figura 8.8: Interfaz Android Studio

8.1.1. Funcionamiento del emulador.

Para poder ejecutar una aplicación desde un ordenador portátil es necesario un emulador [99]. En este caso Android Studio cuenta con uno. Para acceder a él hay que navegar por el menú:

```
Tools >Device manager
```

Lo anterior despliega un panel lateral que muestra lo siguiente:

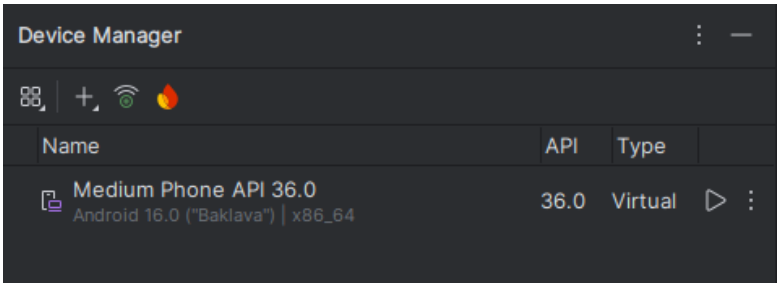

Figura 8.9: Dispositivos disponibles para ejecutar en el emulador.

Tal y como se puede ver en la figura anterior, aparece creado de forma predeterminada un teléfono con sistema operativo Android 16. En caso de querer, se puede crear un dispositivo nuevo desde el símbolo +. En este caso como existe uno ya creado, se va a omitir este último paso y se va a ejecutar el emulador con el dispositivo con Android 16 pulsando sobre el botón *play*[2] que aparece al lado derecho de cada una de las opciones.

8.2. Lanzamiento de las pruebas.

Una vez configurado el entorno controlado, obtenido el código fuente de la aplicación y esta última en ejecución sobre dicho entorno se pueden lanzar las pruebas sobre AquaCyL. En las siguientes secciones se muestran los pasos llevados a cabo para lanzar cada una de las pruebas, los resultados obtenidos y en caso de haber encontrado alguna vulnerabilidad, una serie de recomendaciones para evitarlas o mitigarlas.

8.2.1. MASTG-TEST-0002 - Prueba del almacenamiento local para la validación de los datos de entrada.

Para la ejecución de la prueba, se ha buscado dentro del código fuente el uso de *SharedPreferences*, para comprobar que no se utilicen sin ser previamente validados [61].

Con el comando *grep*, se ha buscado el uso de *SharedPreferences* en el código ya extraído de la APK con el siguiente comando:

```
grep -r SharedPreferences *
```

La salida del comando muestra que se hace uso en múltiples ocasione de *SharedPreferences*. Esto puede verse reflejado en diferentes ficheros, siendo uno de ellos *k.smali*, donde se puede ver que se obtiene la instancia a *SharedPreferences*, se asigna el campo de la clase y se leen los datos persistentes:

```
invoke-virtual p1, p2, v0, Landroid/content/Context;->
```

[2]Botón que permite ejecutar el emulador.

```
getSharedPreferences(Ljava /lang/String;I)Landroid/content/SharedPreferences;


iput-object p1, p0, La6/k;->a:Landroid/content/SharedPreferences;


invoke-interface v0, v1, v2, v3, Landroid/content/SharedPreferences;->getLong(Ljava
/lang/String;J)
```

En la última linea de código se puede ver como se recuperan valores de tipo String directamente desde el almacenamiento.

Por otro lado, se ha comprobado si al usar *SharedPreferences.Editor* para leer o almacenar claves. En las siguientes lineas se puede ver como se escriben elementos en *SharedPreference* sin aparente un control previo:

```
invoke-interface v2, v3, v4, Landroid/content/SharedPreferences$Editor;->
putStringSet(Ljava/lang/String;Ljava/util/Set;)Landroid/content/
SharedPreferences$Editor;


invoke-interface v0, Landroid/content/SharedPreferences$Editor;->commit()Z
```

No se ha encontrado evidencia del uso de operaciones de cifrado para los datos que se manejan con *SharedPreferences*.

Conclusión de la prueba.

En base al análisis estático sobre el código de la aplicación, se puede concluir que se utiliza *SharedPreferences* en múltiples ocasiones como método de persistencia de los datos, lo que implica que se guarda información en el almacenamiento interno de la aplicación. Por otro lado, no se ha encontrado evidencia del uso de funciones de cifrado para proteger los datos que se almacenan con *SharedPreferences*.

Por tanto, se puede concluir que la aplicación no supera la prueba puesto que no garantiza que los datos persistentes de validen antes de usarse, lo que puede implicar que se use información manipulada o no válida.

Recomendaciones.

- Evitar el uso de los valores locales de forma directa sin antes verificar el origen del que provienen o su integridad.
- Considerar aplicar mecanismos de protección mediante funciones criptográficas con el fin de evitar almacenar la información vía *SharedPreferences* en texto plano.

8.2.2. MASTG-TEST-0004 - Determinar si se comparten datos confidenciales con terceros a través de datos embebidos.

Con el fin de descubrir si la aplicación comparte datos de carácter sensible al usuario con terceros, hay que realizar tanto el análisis estático como el dinámico sobre la aplicación [64]:

Análisis estático.

En primer lugar se han buscado los permisos que se solicitan. Para ello hay que buscar en *AndroidManifest.xml*, en el cual se muestra que los permisos que se utilizan son los siguientes [100]:

- **android.permission.INTERNET:** Para realizar operaciones en internet, como puede ser conectarse [101].
- **android.permission.ACCESS_NETWORK_STATE:** Para realizar operaciones en internet, como puede ser conectarse.
- **android.permission.READ_EXTERNAL_STORAGE:** Permite a la aplicación leer desde el almacenamiento externo.
- **android.permission.READ_MEDIA_IMAGES:** Permite que la aplicación lea imágenes desde el almacenamiento externo.
- **android.permission.READ_MEDIA_VIDEO:** Permite que la aplicación lea videos desde el almacenamiento externo.
- **android.permission.READ_MEDIA_VISUAL_USER_SELECTED:** Permite que la aplicación lea imágenes o videos desde un almacenamiento externo que el usuario haya seleccionado.
- **android.permission.CAMERA:** Requerido para acceder a la cámara del dispositivo.
- **android.permission.ACCESS_MEDIA_LOCATION:** Permite que la aplicación acceda a cualquier ubicación geográfica en la colección compartida del usuario.
- **android.permission.RECORD_AUDIO:** Permite grabar video.
- **android.permission.POST_NOTIFICATIONS:** Para publicar notificaciones.
- **com.google.android.gms.permission.AD_ID:** Para monetizar aplicaciones y publicidad [102].

Cada uno de los permisos que se utilizan están justificados por la funcionalidad que presenta la aplicación, o por decisiones del desarrollador para obtener beneficios como puede ser el último permiso que se menciona en la lista.

Por otro lado, se ha inspeccionado el código fuente en busca de bibliotecas de terceros que puedan presentar algún tipo de riesgo de seguridad. Para ello se ha buscado mediante el comando *grep*, la presencia de elementos como *token, password, auth, session, email* o *clave*. Con ello se ha encontrado que se utilizan bibliotecas como *com.google.android.gms.internal.auth*, o *com.google.firebase.auth* para gestionar la autenticación del usuario. Puesto que se trata de bibliotecas de terceros, aunque estén gestionadas por Google, manejan elementos sensibles como puede ser el email del usuario, será en el análisis dinámico donde se comprobará si se utilizan correctamente y de forma segura.

Análisis dinámico.

Con el fin de interceptar el tráfico que sale o entra de la aplicación, se ha utilizado *Burp* [65] para llevar a cabo dicha tarea. En primer lugar se ha encendido el emulador y se ha ejecutado el siguiente comando para activar el proxy [103] para que Burp pueda capturar el tráfico:

```
adb shell settings put global http_proxy 10.0.2.2:8080
```

Posteriormente, dentro de Burp, se ha navegado hacia *Proxy - HTTP history* y se ha comenzado a interaccionar con el emulador [104]. En primer lugar, para comprobar que el tráfico se captura correctamente se ha lanzado una búsqueda sobre el navegador. Posteriormente se ha navegado por la aplicación. No obstante, al comprobar el tráfico capturado se ha descubierto que no es posible capturar el tráfico. Tras investigar los motivos que podrían causar el problema, se ha descubierto que, en versiones actuales de Android, a partir de la 11 no se pueden instalar el certificado que *Burp* requiere ya que el sistema operativo lo impide.

Se ha probado a configurar todo nuevamente desde un Android inferior, primero con la versión 7, pero la aplicación no es compatible con un sistema operativo tan antiguo, y posteriormente sobre Android 11 y nuevamente, no se puede instalar el certificado. En Android 10, el certificado puede instalarse pero la aplicación da error cada vez que se quiere abrir y se cierra inesperadamente.

Es por lo anterior por lo que no se ha podido realizar un análisis dinámico satisfactorio con el que concluir si se envían datos personales en claro y sin la autorización de un tercero.

Conclusión de la prueba.

Aunque solo se tenga en cuenta el análisis estático, ya que con el dinámico no ha sido posible obtener nada, se puede concluir que no hay evidencia de que la aplicación comparta datos confidenciales del usuario con terceros.

8.2.3. MASTG-TEST-0008 - Comprobación de la divulgación de datos confidenciales a través de la interfaz.

Para verificar que no se comparten datos de carácter personal mediante la interfaz se ha realizado el análisis estático sobre el código fuente y el dinámico sobre la aplicación en ejecución [67]:

Análisis estático.

Se ha buscado en el código fuente la gestión de las notificaciones mediante *NotificationManager* con el objetivo de buscar si la aplicación envía algún tipo de dato personal a través de éstas. Para ello se ha utilizado el comando *grep* tal y como se muestra a continuación:

```
grep -r NotificationManager
```

Con ello se ha descubierto que *NotificationManager* se utiliza en múltiples ocasiones, no obstante, no se ha encontrado evidencia de que la aplicación envíe datos de carácter personal a través de las notificaciones.

Por otro lado, se ha buscado que las contraseñas de los campos *EditText* se utilice *android: inputType="textPassword"*. Para ello se han ejecutado los dos siguientes comandos:

```
grep -r EditText
grep -r android:inputType=‘‘textPassword’’
```

Con ello se ha encontrado que *EditText* se usa varias veces, pero sin usarse para contraseñas. Hay que destacar que el segundo comando no devuelve nada. Esto sería un problema en el caso en el que se usara *EditText* para gestionar contraseñas, pero al no ser el caso no supone ningún problema.

Análisis dinámico.

Se ha ejecutado la aplicación en el emulador con el fin de descubrir si por medio de alguna notificación se envían datos personales del usuario. Tras realizar distintas transacciones, como cambiar la contraseña o interaccionar con las diferentes vistas de la aplicación. Las contraseñas se encuentran enmascaradas (se muestran con puntos en vez de en plano).

Resultado de la prueba.

La aplicación pasa la prueba puesto que ni en análisis estático ni en análisis dinámico se han encontrado evidencias de la divulgación de datos personales a través de la interfaz.

8.2.4. MASTG-TEST-0011 - Prueba de memoria de datos confidenciales.

Con el objetivo de determinar si los datos sensibles permanecen en texto plano almacenados en memoria o mientras se usan o si se eliminan correctamente tras usarse se han realizado análisis estático y dinámico sobre la aplicación [68]:

Análisis estático.

Analizando el código fuente, se ha encontrado que el inicio de sesión, que es el principal uso de datos confidenciales, se realiza mediante *Firebase* [105], componente de Google para gestionar la autenticación de los usuarios. Se ha buscado la gestión de usuario, contraseña y correo electrónico mediante el comando *grep*:

```
grep -r email * >email.txt
grep -r password * >paswword.txt
grep -r user * >user.txt
```

Tras inspeccionar la salida del anterior comando en los ficheros anteriores, se ha descubierto que tanto email como contraseña se utilizan en múltiples ficheros. Por otro lado, se ha concluido que se tratan de datos tipo String tal y como se puede ver en lineas como pueden ser:

```
const-string v1, ‘‘email’’
```

```
 invoke-virtual v0, v1, v2, Lorg/json/JSONObject;->optString(Ljava/lang/
String;Ljava/lang/String;)Ljava/lang/String;
```

Esto presenta un problema de seguridad ya que se trata de datos inmutables.

Por otro lado, no se han encontrado indicios del uso de sistemas de eliminación de datos que no se utilizan o recolectores de basura como *System.gb()* [106].

Análisis dinámico.

Para la realización del análisis dinámico hay que realizar un volcado de memoria para comprobar si las credenciales del usuario con la sesión iniciada se pueden obtener del volcado, es decir, si continúan en memoria una vez se ha iniciado sesión.

En primer lugar hay que abrir Android Studio y con la aplicación abierta y con la sesión iniciada navegar a *Profiler* para realizar el *Heap Dump*. No obstante, al intentar hacerlo aparece un error en el que se indica que no se puede realizar el volcado.

Se ha optado por probar otra forma de realizar el volcado con los siguientes comandos:

```
adb shell pm list packages | grep aquacyl
adb shell pidof es.pablo.aquacyl.aqua_cyl
adb shell am dumpheap 4117 /sdcard/heap-aquacyl.hp
```

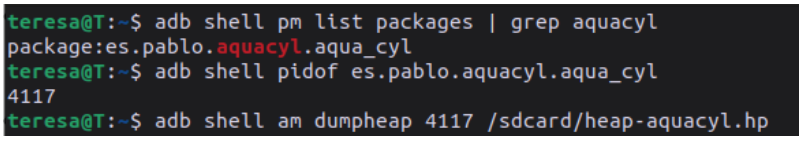

Figura 8.10: Prueba de volcado de memoria.

El último comando devuelve error, lo que significa que la aplicación no es debuggueable por lo que el volcado de memoria no puede realizarse.

Conclusiones de la prueba.

Aunque no ha sido posible realizar el volcado de memoria ya que la aplicación no es debugueable, en el análisis estático se han encontrado algunos fallos de seguridad que provocan que la aplicación no pasa la prueba ya que se utilizan tipos de datos String, y no se realiza ningún tipo de solicitud para la recolección de basura una vez se hayan terminado de utilizar los datos. Esto implica una vulnerabilidad ya que se podría capturar información mientras se usan o tras usarlos.

Recomendaciones.

Utilizar otro tipo de datos diferente a *String* para almacenar tipos de datos sensibles como las direcciones de correo electrónico o las contraseñas de los usuarios. También se recomienda realizar la limpieza de los datos cuando no se utilicen con mecanismos de recolección de basura como puede ser *System.gb()*.

8.2.5. MASTG-TEST-0014 - Prueba de la configuración del algoritmo estándar de criptografía.

Con el objetivo de comprobar si la aplicación utiliza bibliotecas seguras antes que implementar nuevas, y verificar el uso de algoritmos criptográficos seguros se ha realizado el análisis estático y dinámico de la aplicación [70].

Análisis estático.

Se ha analizado el código fuente en busca de diferentes funciones, clases, interfaces o excepciones que indiquen el uso de criptografía para proteger datos sensibles:

```
grep -rE 'Cipher|Mac|MessageDigest|Signature' * >clases.txt
grep -rE 'Key|PrivateKey|PublicKey|SecretKey' * >interfaces.txt
grep -rE 'getInstance|generateKey' * >funciones.txt
grep -rE 'KeyStoreException|CertificateException|NoSuchAlgorithmException' * >
excepciones.txt
grep -rE 'SHA' * >sha.txt
```

Se ha redirigido la salida del comando a diferentes ficheros con el fin de facilitar la comprensión del código ya que la salida de algunos es considerablemente grande y desde un fichero se puede buscar con más facilidad.

Dentro de los diferentes ficheros se ha descubierto que se utilizan bibliotecas como *javax.crypto* [107], biblioteca oficial. Por otro lado, no se han encontrado funciones propias e inseguras criptográficas.

Dentro de los ficheros analizados, se ha encontrado que existen referencias a clases como:

- **Cipher:** Proporciona funcionalidad de un cifrado criptográfico para cifrado y descifrado [108].
- **Mac:** Proporciona la funcionalidad de MAC o *Código de Autenticación de Mensajes* [109].
- **Signature:** Proporciona a la aplicación la funcionalidad de un algortimo de firma digital [110].
- **KeyGenerator:** Facilita la funcionalidad de un generador de clave simétrica [111].
- **KeyStore:** Permite almacenar claves criptográficas y certificados [112].

Por otro lado, en lo que a algoritmos criptográficos se refiere, se ha encontrado el uso de funciones seguras como puede ser *SHA-256* o *SHA-512* [71], que son funciones seguras. No obstante, también se menciona el uso de *SHA-1*, conocida por sus colisiones e inseguridad en ficheros como *m.smali* o *zzach.smali*.

Análisis dinámico.

Con el fin de interceptar llamadas criptográficas, comprobar si las contraseñas de los usuarios se almacenan en texto plano e inspeccionar en tiempo real los datos confidenciales, se ha optado por ejecutar Frida [36], una aplicación para monitorizar la información en tránsito con la aplicación.

Para ejecutar Frida sobre la aplicación hay que tener la aplicación en ejecución sobre el emulador y conectar Frida al proceso de la aplicación:

```
frida -U -n AquaCyL
```

El proceso anterior indica que el servidor no se encuentra en ejecución, por lo que hay que ejecutar:

```
adb push frida-server-16.1.2-android-x86_64 /data/local/tmp/frida-server
adb shell ‘‘chmod 755 /data/local/tmp/frida-server’’
adb shell
su
/data/local/tmp/fridaa-server
```

Los pasos anteriores dan error. Esto es puesto que para que Frida funcione, es necesario que se cumplan o bien ambas de las siguientes condiciones o al menos una:

- El emulador sea rooteable, es decir, tenga permisos de root.
- La aplicación sea debuggueable.

Con el fin de poder lanzar las pruebas se ha probado a realizar los cambios necesarios para que se cumplan las condiciones:

1. Decompilar la aplicación
2. Modificación de *AndroidManifest.xml* añadiendo *android:debuggable="true"*.
3. Compilar la aplicación.
4. Firmar la aplicación.
5. Instalar la aplicación en el emulador

Los pasos anteriores resultan en error por conflicto de firmas. Es por ello por lo que se repitieron los pasos pero con *apk-mitm* [113], ya que de este modo se modifica *AndroidManifest.xml* únicamente sin modificar firmas. No obstante, sigue dando error.

Finalmente, se ha probado a realizar los cambios sobre *AndroidManiefst.xml* pero compilando la aplicación con las bibliotecas de la APK original y no con las modificadas. Tras realizar los pasos, la aplicación se firma correctamente pero sigue sin ser debuggueabke.

Puesto que no ha sido posible hacer que la aplicación sea debuggueable, se ha probado a instalar la aplicación en un emulador rooteado en genymotion. Al intentar ejecutar la aplicación esta se cierra repentinamente.

Con los anteriores resultados se puede concluir que no puede realizarse el análisis dinámico de esta prueba sobre AquaCyL ya que la aplicación no es debuggueable y no se puede transformar, y la aplicación no funciona en entornos con permisos de root.

En términos de seguridad, el hecho de que la aplicación rechace estos mecanismos implica que se encuentra protegida ante vulnerabilidades que impliquen que la aplicación pueda ser debuggueada o su ejecución en entornos root, lo cual puede estar relacionado con ataques malintencionados.

Conclusión de la prueba.

Aunque el análisis dinámico no haya podido realizarse, se ha encontrado que al ser una aplicación no debuggueable y que no se puede ejecutar sobre emuladores con permisos de root, lo cual demuestra que se trata de una aplicación segura en estos casos.

Por otro lado, en el análisis estático se ha encontrado que se usan bibliotecas y clases predefinidas seguras. No obstante se ha encontrado el uso de algoritmos de criptografía inseguros como *SHA-1*, lo cual hace que la aplicación no pase la prueba.

Recomendaciones.

Cambiar el uso de los algoritmos criptográficos inseguros como *SHA-1* por otros maś robustos y seguros.

8.2.6. MASTG-TEST-0017 - Prueba para confirmar credenciales.

Con el objetivo de verificar si existe un tiempo límite establecido con el que la sesión permanezca activa o si para realizar algún cambio se solicitan las credenciales se ha realizado el análisis estático y el análisis dinámico sobre la aplicación [73].

Análisis estático.

Se ha inspeccionado el código fuente con el comando *grep* con el objetivo de buscar si existe alguna restricción en el uso de las claves del usuario, sobre todo a la hora de realizar acciones críticas como la eliminación de la cuenta o el cambio de la contraseña.

Para realizar la búsqueda se han ejecutado los siguientes comandos con el objetivo de encontrar si se utilizan dichas funciones o relacionadas:

```
grep -r setUserAuthenticationRequired[3] *
grep -r UserAuthentication[4] *
```

En ambos casos la salida ha sido vacía, lo que implica que no existen medidas con las que se requiera de autenticación o que exista un tiempo máximo durante la que dicha autenticación es válida para realizar acciones importantes sobre la cuenta.

Análisis dinámico.

Dado que, tal y como se ha comentado en el análisis estático, no se ha encontrado el uso de *setUserAuthenticationRequired* [74], no es necesario medir el tiempo durante el que las credenciales del usuario están disponibles antes de que caduquen ya que este tiempo no existe.

[3] Habilita o deshabilita que el usuario tenga que autenticarse en el momento de la conexión.
[4] Gestiona la autenticación del usuario dentro de la aplicación.

Por otro lado, se han realizado diferentes tareas sobre la aplicación como cambiar la contraseña, cambiar el nombre de usuario, cambiar la foto de perfil o eliminar la cuenta. Únicamente para cambiar la foto de perfil se lleva a la pantalla de inicio de sesión, para el resto con tener la aplicación con la sesión iniciada es suficiente. No obstante, después de iniciar sesión para cambiar la foto de perfil no aparece ninguna página en la que hacerlo, y si se vuelve a navegar hacia esa zona, se vuelven a repetir los pasos anteriormente mencionados:

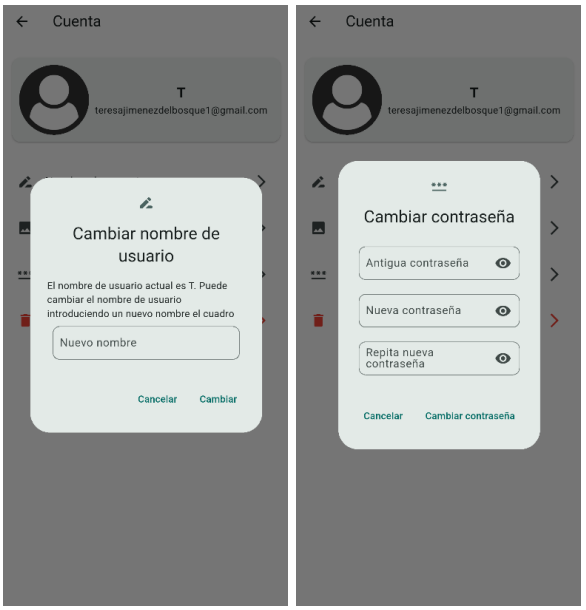

Figura 8.11: Acciones para cambiar el nombre de usuario y la contraseña.

Figura 8.12: Acciones para cambiar la foto de perfil y eliminar cuenta.

Conclusión de la prueba.

No se han encontrado mecanismos que gestione la solicitud de credenciales para eventos críticos como cambiar la contraseña o eliminar la cuenta, ni tampoco la existencia de un tiempo límite en el que las credenciales sean válidas para realizar las diferentes acciones críticas ya mencionadas anteriormente.

Recomendaciones.

- Requerir al usuario sus credenciales cuando se quiera hacer algún cambio en su cuenta como puede ser cambiar la contraseña o eliminar la cuenta.
- Establecer un tiempo máximo en la que se puedan realizar acciones en la cuenta con la sesión iniciada. Esto puede realizarse con *setUserAuthenticationRequired*.

8.2.7. MASTG-TEST-0023 - Prueba de proveedor de seguridad.

Con el fin de asegurar que la aplicación no utilice implementaciones no seguras de los servicios de Google y del proveedor de seguridad se van a lanzar el análisis estático y dinámico sobre AquaCyL [75].

Análisis estático.

Se ha analizado el código fuente para comprobar que la aplicación está basada en el SDK de Android. Para ello se ha ejecutado el siguiente comando desde el directorio en el que se encuentra la APK previamente descompilada:

```
grep -r Android *
grep -r android *
```

El uso de múltiples elementos relacionados con Android confirman que se utiliza el SDK de Android.

Por otro lado se ha buscado el uso de *ProviderInstaller* [76] para verificar su correcto funcionamiento. Se ha ejecutado el siguiente comando:

```
grep -r ProviderInstaller *
```

Tras ejecutar el comando anterior no se ha encontrado que se utilice *ProviderInstaller* en el código. No obstante, se ha descubierto que para aplicaciones que utilizan los Servicios de Google en su versión 5 o superior no requieren de dicho proveedor ya que el servicio se da de forma automática al usar esa versión [114].

Análisis dinámico.

Tal y como se ha mencionado en pruebas anteriores, la aplicación no es debuggueable ni puede ser ejecutada en dispositivos rooteados. Por ende, no puede ser ejecutada en modo depuración ni utilizar Frida para monitorizar el tráfico de datos existente.

Conclusiones de la prueba.

El hecho de que no se haya podido realizar el análisis dinámico sobre la aplicación implica que ésta no es debuggeable ni se puede ejecutar en entornos con permisos de root. Esto aporta seguridad en la aplicación, por lo tanto, pasa la prueba.

Por otro lado, en el análisis estático se ha descubierto que se utiliza el SDK de Android y que, aunque no se utilice la clase ProviderInstaller, al requerir la aplicación de Android 10 o posterior para funcionar, necesita los Servicios de Google en una versión mayor que 5, lo que implica que ya tiene cobertura de proveedor y no es necesario su uso.

8.2.8. MASTG-TEST-0026 - Prueba de intenciones implícita.

Con el fin de comprobar si la aplicación es vulnerable a ataques de inyección o filtración de datos personales o si existen intenciones implícitas, se ha realizado el análisis estático y dinámico sobre la aplicación [77].

Análisis estático.

Se ha buscado en el código fuente, inspeccionando el fichero *AndroidManifest.xml* para buscar firmas definidas, información que se puede buscar dentro de *intent* y se puede ver con el siguiente comando:

```
grep -r intent AndroidManifest.xml
```

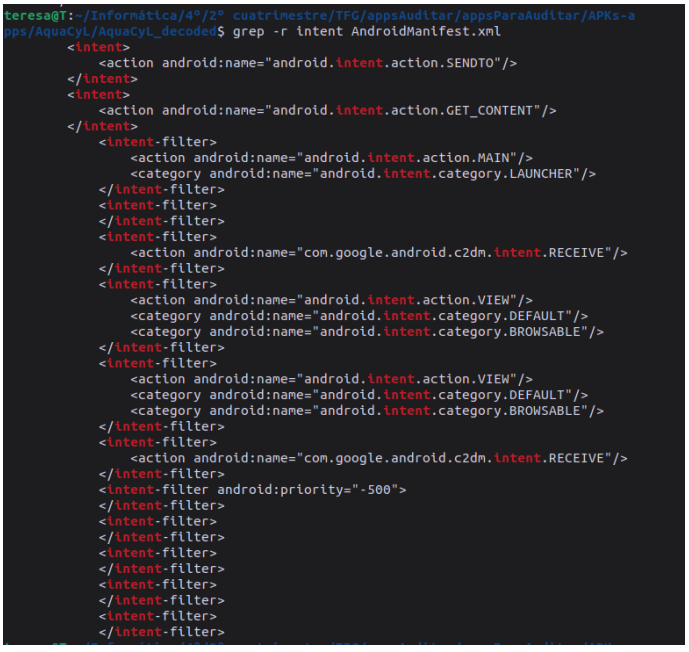

Figura 8.13: Salida de *grep* sobre *AndroidManifest.xml*.

En la figura anterior se pueden ver las diferentes intenciones declaradas para la aplicación. Llama

la atención el uso de *android.intent.action.GET_CONTENT* [115], la cual puede ser especialmente peligrosa si transmite información de caracter sensible.

Por otro lado, tal y como se sugiere para realizar la prueba correctamente, se han buscado las intenciones *android.intent.action.GET_CONTENT*, *android.intent.action.PICK* y *android.media.action.IMAGE_CAPTURES* [116]. Para realizar dicha búsqueda se han lanzado los siguientes comandos:

```
grep -r android.intent.action.GET_CONTENT
grep -r android.intent.action.PICK
grep -r android.media.action.IMAGE_CAPTURES
```

Figura 8.14: Salida de grep para las intenciones que OWASP propone.

En la anterior figura se puede ver cómo la aplicación utiliza las tres intenciones que se buscan. La primera sirve para seleccionar un recurso del dispositivo como un documento o una imagen. La segunda devuelve una clase una vez elegida una actividad según una intención lo que quiere decir que selecciona un elemento de una fuente específica como puede ser una imagen. La tercera permite ejecutar la cámara para hacer una foto. La diferencia entre la primera intención y la segunda es que la primera puede no tener un destino específico y en la segunda éste se encuentra definido.

Es importante descubrir con qué fin se usa *android.intent.action.GET_CONTENT* ya que puede ser usada con fines legítimos, como seleccionar una imagen desde el almacenamiento de dispositivo, o con fines ilegítimos si no se emplean restricciones o validaciones. Como se trata de una intención implícita se puede interceptar por aplicaciones de terceros y manipular la información. Por este motivo, se ha buscado para qué se utiliza dicha intención implícita. Para ello se ha utilizado el comando grep:

```
grep -r GET_CONTENT *
```

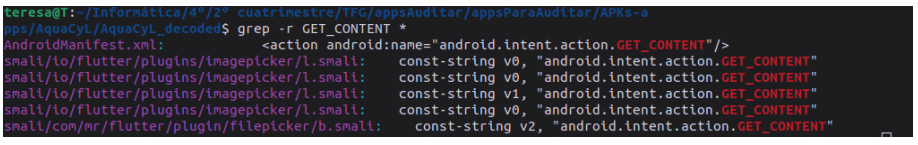

Figura 8.15: Salida de grep para buscar el uso de la intención implícita *android.intent.action.GET_CONTENT*.

En la anterior figura se puede ver como *android.intent.action.GET_CONTENT* se utiliza de forma legítima por Flutter [117], usada para crear la interfaz de usuario, para subir archivos a la aplicación como imágenes.

Análisis dinámico.

Para realizar en análisis dinámico, es necesario lanzar Frida para interceptar las intenciones implícitas. No obstante, tal y como se ha comentado en pruebas anteriores, no se puede lanzar ya que la aplicación no es debuggueable ni se puede ejecutar sobre máquinas con permisos de root.

Conclusiones de la prueba.

Aunque no se ha podido realizar el análisis dinámico ya que la aplicación está protegida a la modificación para transformarla en debuggueable y no permite la ejecución en sistemas con permisos de root, en el análisis estático se ha comprobado que las intenciones implícitas se usan de forma legítima para subir imágenes a la aplicación por medio de la interfaz gestionada por Flutter, por lo que pasa la prueba.

8.2.9. MASTG-TEST-0027 - Prueba de carga de URL en WebViews.

El objetivo de la prueba es comprobar que la aplicación carga las direcciones de las páginas web de forma segura, impidiendo que se redirija al usuario a sitios inseguros, fraudulentos o maliciosos. Para ello se ha realizado el análisis estático y dinámico. Finalmente, se concluirá si la aplicación pasa o no la prueba y por ende es segura para los casos en cuestión que se describen [78].

Análisis estático.

Por un lado, se ha inspeccionado *AndroidManifest.xml* en busca de *EnableSafeBrowsing*, ya que, aunque aparece activado por defecto, en ocasiones el desarrollador puede desactivarlo. Para realizar la búsqueda se ha usado el comando *grep* tal y como se muestra a continuación:

```
grep -r EnableSafeBrowsing AndroidManifest.xml
```

El comando no devuelve nada, lo que significa que *EnableSafeBrosing* se encuentra activo.

Por otro lado, se han inspeccionado el resto del código del que se dispone para analizar lo que devuelven las funciones *shouldOverrideUrlLoading* y *shouldInterceptRequest*.

shouldOverrideUrlLoading:

Se ha ejecutado *grep* para encontrar los ficheros en los que se utiliza dicha función:

```
grep -r shouldOverrideUrlLoading *
```

Lo anterior revela que se utiliza en varios ficheros *smali* como pueden ser *g4aa.smali*, *o5$a.smali* o *o5$c.smali*, en los que se puede ver cómo se hace uso de la función *shouldOverrideUrlLoading*.

En los tres ficheros se define el método de dos formas diferentes. En la primera se llama a un *WebViewClient* personalizado, y en el segundo si la URL que se pasa es de tipo String:

```
.method public shouldOverrideUrlLoading(Landroid/webkit/WebView;Landroid/webkit
/WebResourceRequest;)Z
.method public shouldOverrideUrlLoading(Landroid/webkit/WebView;Ljava
/lang/String;)Z
```

Dentro de cada método se obtiene la URL, se llama a *WebViewClient* y si este cliente devuelve true significa que ya se ha gestionado o bloqueado la dirección. Si por el contrario se devuelve false el código carga la dirección de forma manual. Finalmente siempre se devuelve true al final para bloquear el resto de las direcciones.

shouldInterceptRequest:

Con el fin de buscar si se usa la función *shoudlInterceptRequest*, se ha ejecutado *grep* sobre el directorio en el que se encuentra la aplicación decompilada:

```
grep -r shouldInterceptRequest
```

En la siguiente figura se puede ver la salida resultante de la ejecución del comando *grep*, en ella se observa cómo se usa únicamente en componentes relacionados con reCaptcha [118] o interfaces abstractas para la compatibilidad con ServiceWorkers [119].

```
teresa@T:~/Informática/4º/2º cuatrimestre/TFG/appsAuditar/appsParaAuditar/APKs-apps/AquaCyL/AquaCyL_decod
ed$ grep -r shouldInterceptRequest *
smali/com/google/android/recaptcha/internal/zzeu.smali:.method public final shouldInterceptRequest(Landro
id/webkit/WebView;Ljava/lang/String;)Landroid/webkit/WebResourceResponse;
smali/com/google/android/recaptcha/internal/zzeu.smali:    invoke-super {p0, p1, p2}, Landroid/webkit/Web
ViewClient;->shouldInterceptRequest(Landroid/webkit/WebView;Ljava/lang/String;)Landroid/webkit/WebResourc
eResponse;
smali/org/chromium/support_lib_boundary/ServiceWorkerClientBoundaryInterface.smali:.method public abstrac
t shouldInterceptRequest(Landroid/webkit/WebResourceRequest;)Landroid/webkit/WebResourceResponse;
```

Figura 8.16: Salida de grep para buscar el uso de la función *shouldInterceptRequest*.

Tras analizar el código fuente de las funciones que grep indica que se utiliza dicha función, únicamente se ha encontrado que se delega directamente al comportamiento por defecto de la función sin validar la URL ni filtrando recursos:

```
invoke-super {p0, p1, p2}, Landroid/webkit/WebViewClient;-
>shouldInterceptRequest(Landroid/webkit/WebView;Ljava/lang/String;)
Landroid/webkit/WebResourceResponse;
```

Análisis dinámico.

Para realizar el análisis dinámico de la aplicación hay que utilizar Frida para monitorizarla. Tal y como se ha explicado en anteriores pruebas, la aplicación no permite la modificación de su código para volverla debuggueable ni ser ejecutada en entornos con permisos de root. Por lo que no puede lanzarse la prueba ya que ésta cuenta con mecanismos de seguridad para estos casos.

Conclusión de la prueba.

Por un lado, en el análisis estático se ha concluido que la navegación segura se encuentra habilitada. En cuanto a la función *shouldOverrideUrlLoading*, ésta delega el filtro de URLs al cliente. Por último, en lo que respecta a *shouldInterceptRequest*, únicamente se utiliza para componentes relacionados con Google o compatibilidad. En ambos casos solamente se invoca al comportamiento por defecto sin añadir validaciones personalizadas.

El análisis dinámico no ha podido ser realizado por medidas de seguridad con las que cuenta la aplicación.

Se concluye que la aplicación utiliza una navegación segura basada en un *WebViewClient* que permite configurar que las solicitudes de navegación sean gestionadas por la propia aplicación, y que por tanto pasa la prueba.

8.2.10. MASTG-TEST-0036 - Prueba de actualización forzada.

Con el objetivo de comprobar si se puede seguir utilizando la aplicación en caso de que exista una actualización no instalada en el dispositivo se ha realizado el análisis estático y dinámico sobre la aplicación [79].

Análisis estático.

Se ha inspeccionado el código fuente con el objetivo de buscar fragmentos relacionados con el control de versiones y lógica para controlar la aplicación en caso de contar con una versión anterior. Pare ello se ha usado el comando *grep* desde el directorio en el que se encuentra la APK descompilada:

```
grep -r update * >update.txt
```

La salida se ha redirigido a un fichero ya que salían múltiples coincidencias y de esta forma resultaban mas fáciles de manejar.

Posteriormente se ha inspeccionado el fichero de salida en el que se puede ver que a mayoría de los mensajes corresponden a los diccionarios de idiomas de la aplicación, es decir, lo que se usan en función del idioma que se tenga seleccionado para informar de la actualización de la contraseña, nombre de usuario o correo electrónico.

Llama la atención algunos mensajes que mencionan que a menos que se actualicen los Servicios de Google la aplicación no funciona, pero esto se refiere a servicios externos, no relacionados con la aplicación en sí. En lo que a esto último se refiere no se ha encontrado evidencia de control de actualización.

Análisis dinámico.

Para realizar el análisis dinámico se ha encontrado que en el momento de realizar la prueba, la aplicación descargada desde Google Play se actualizó, de modo que en el emulador se contó con la aplicación sin actualizar en ese momento.

Al abrirla sin la última versión se muestra una pantalla de Google Play en la que se indica que hay una actualización disponible y que para usar la aplicación hay que descargar la última versión. No obstante si se pulsa en la cruz de la esquina superior derecha la aplicación carga y funciona con normalidad sin ninguna restricción.

Posteriormente se ha cerrado la aplicación y se ha vuelto a abrir y la pantalla de actualización no ha vuelto ha mostrarse y para actualizar la aplicación ha sido necesario actualizarla manualmente desde Google Play.

Figura 8.17: Pantalla de actualización en AquaCyL sin actualizar.

Conclusión de la prueba.

La aplicación no cuenta con mecanismos para forzar la actualización de la aplicación en caso de existir una nueva versión y tampoco impide su utilización con normalidad. Esto implica que no pase la prueba.

Recomendaciones.

Para evitar que el usuario utilice una versión no actualizada de la aplicación se recomienda implementar las siguientes medidas:

- Integrar mecanismos de actualización forzada para forzar al usuario a actualizar la aplicación en caso de que haya una nueva versión.
- Mostrar avisos que no se puedan descartar si se detecta que la versión de la aplicación no es la más actual.
- Evitar que la aplicación se ejecute con normalidad si no se cuenta con la versión más actual.

8.2.11. MASTG-TEST-0037 - Prueba de limpieza de WebViews.

Para probar si la aplicación elimina los datos de las WebViews que se utilizan una vez se cierra la aplicación o la sesión, se ha realizado el análisis estático y dinámico sobre ella, finalmente se exponen las conclusiones obtenidas tras realizar la prueba [81].

Análisis estático.

Se ha inspeccionado el código fuente en busca de APIs de WebView relacionadas con caché, inicialización, almacenamiento, cookies y archivos. Para ello se ha buscado con el comando *grep* algunas de las funciones relacionadas con cada una de las APIs. Esto se ha hecho con las funciones que OWASP indica:

- **setDomStorageEnabled:** Inicializa la vista web para evitar que almacenen cierta información.
- **setAppCacheEnabled:** Habilita o deshabilita el almacenamiento caché.
- **setDatabaseEnabled:** Habilita o deshabilita el almacenamiento de la base de datos.
- **android.webkit.WebSettings:** Administra el estado de la configuración de una WebView [120].
- **clearCache:** Elimina la caché.
- **onRenderProcessUnresponsive:** Recibe las llamadas de los eventos de la WebView [121].
- **WebStorage.deleteAllData:** Elimina todos los datos del almacenamiento de la vista web.
- **CookieManager.removeAllCookies:** Elimina todas las cookies almacenadas.
- **java.io.File.deleteRecursively:** Para eliminar manualmente ciertos directorios que contienen información del usuario [122].

Para buscar las funciones anteriores en el código fuente se ha ejecutado *grep* tal y como se muestra a continuación:

```
grep -r setDomStorageEnabled
grep -r setAppCacheEnabled
grep -r setDatabaseEnabled
grep -r android.webkit.WebSettings
grep -r clearCache
grep -r onRenderProcessUnresponsive
grep -r WebStorage.deleteAllData
grep -r CookieManager.removeAllCookies
grep -r java.io.File.deleteRecursively
```

De todas las búsquedas solo han devuelto resultados *clearCache* y *setDomStorageEnabled*. Con ello se puede concluir que las cachés de almacenamiento de la base de datos se encuentran deshabilitadas. Que la caché de las WebViews no se elimina, las cookies, el almacenamiento y los archivos tampoco se eliminan.

De las funciones que se utilizan en la aplicación, se ha descubierto que la ésta inicializa la vista web para evitar que se almacene cierta información y que la caché de las WebViews se elimina, lo cual ambas son buenas prácticas de seguridad.

Análisis dinámico.

Para la realización del análisis dinámico hay que ejecutar la aplicación dentro del emulador y abrir un terminal para ejecutar los siguientes comandos:

```
adb shell
run-as es.pablo.aquacyl.aqua_cyl
```

La salida de los comandos anteriores es la siguiente:

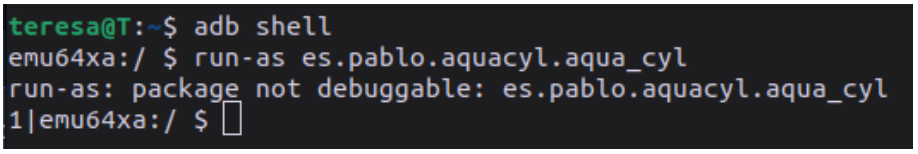

Figura 8.18: Salida de los comandos *adb shell* y *run-as es.pablo.aquacyl.aqua_cyl*.

Tal y como se puede ver en la figura anterior, el análisis dinámico no puede realizarse puesto que la aplicación no es debuggueable, y las medidas de seguridad propias de la aplicación no permite convertirla; el resultado de la prueba es el obtenido por el análisis estático.

Conclusiones de la prueba.

En el análisis estático se ha encontrado que aunque se implementen algunas medidas para la eliminación de información que pueden almacenar las WebViews como puede ser la caché, se ha encontrado que no hay evidencia de la eliminación de otros elementos como cookies que pueden almacenar información relacionada con la aplicación incluso una vez ésta ha sido cerrada o la sesión de usuario haya finalizado. Un tercero podría utilizar dichos datos para fines ilegítimos por lo que no pasa la prueba.

Recomendaciones.

Se recomienda implementar las funciones mencionadas en el análisis estático para que las webs que se utilizan desde la aplicación no almacenen información y se borre la caché, cookies, almacenamiento y archivos, puesto que las WebViews pueden verse comprometidas y terceros con intenciones ilegítimas podrían obtener información de la aplicación o del usuario.

8.2.12. MASTG-TEST-0040 - Prueba de símbolos de debugging.

Para comprobar que los ficheros de la aplicación no cuenten con símbolos de depuración con datos que un atacante pueda usar para realizar ingeniería inversa, se ha realizado el análisis estático sobre el código fuente de la aplicación [82].

Análisis estático

En primer lugar, hay que averiguar si la aplicación utiliza código nativo. Para ello hay que averiguar si una vez descompilada la aplicación existen ficheros con la extensión *.so* [123]:

```
find . -name "*.so"
```

```
pps/AquaCyL/AquaCyL_decoded$ find . -name "*.so"
./lib/armeabi-v7a/libimage_processing_util_jni.so
./lib/armeabi-v7a/libapp.so
./lib/armeabi-v7a/libflutter.so
./lib/x86_64/libimage_processing_util_jni.so
./lib/x86_64/libapp.so
./lib/x86_64/libflutter.so
./lib/x86/libimage_processing_util_jni.so
./lib/arm64-v8a/libimage_processing_util_jni.so
./lib/arm64-v8a/libapp.so
./lib/arm64-v8a/libflutter.so
```

Figura 8.19: Salida comando *find* para buscar si existe código nativo.

En la anterior figura se puede ver la salida del comando anterior, en la que se muestra cómo sí que existen ficheros con código nativo. El siguiente paso es iniciar Android Studio y comprobar que en *SDK Manager - SDK Tools*, *NDK side by side* se encuentra seleccionado:

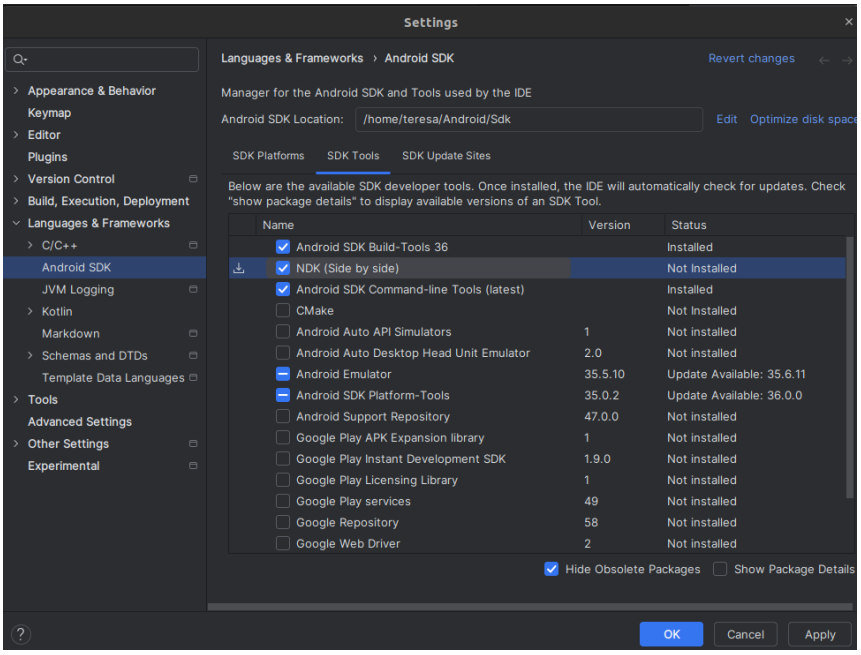

Figura 8.20: *NDK side by side* seleccionado en Android Studio.

El siguiente paso es buscar cuál es la ruta en la que se encuentra *llvm-nm*, necesaria para exportarlo:

```
find  /Android/Sdk/ndk/ -name "llvm-nm"
```

Sabiendo cuál es la ruta se puede exportar el binario *nm* en el NDK de Android:

```
export NM=/home/teresa/Android/Sdk/ndk/29.0.13599879/toolchains/llvm/prebuilt
/linux-x86_64/bin/llvm-nm
```

Para cada uno de los ficheros *.so* encontrados previamente, se han buscado tanto los símbolos de depuración como los símbolos dinámicos. Para ello se han ejecutado los siguientes comandos tanto para símbolos de depuración como para símbolos dinámicos:

```
$NM -a nombreFichero.so
$NM -D nombreFichero.so
```

Para los símbolos de depuración, con cada una de los ficheros con los que se contaba, el comando ha devuelto *"no symbols"*, lo que significa que éstos no están expuestos y no se facilita la ingeniería inversa.

Por otro lado, con los símbolos dinámicos, para cada uno de los ficheros se han devuelto símbolos dinámicos relacionados con funciones como puede ser *Java_androidx_camera_core_ ImageProcessingUtil_nativeShiftPixel@@VERS_1.0*. Esto no implica ningún fallo de seguridad ya que se muestran símbolos para la ejecución de tareas que deben ser visibles.

Conclusiones de la prueba.

Tanto para la búsqueda de símbolos de depuración como para los dinámicos, no se ha encontrado ninguna evidencia de que éstos se encuentren expuestos ni que impliquen fallos de seguridad. Con lo anterior se puede concluir que la aplicación pasa la prueba.

8.2.13. MASTG-TEST-0043 - Errores de corrupción de memoria.

Para comprobar si la aplicación es o no vulnerable a los diferentes tipos de errores de memoria, se ha realizado un análisis estático sobre el código descompilado de la APK de la que se dispone y un análisis dinámico sobre la aplicación en ejecución [83]:

Análisis estático.

Se ha inspeccionado el código fuente en busca de fragmentos relacionados con la serialización. Para ello se ha ejecutado el comando *grep* dentro del directorio en el que se encuentra la APK decompilada tal y como se muestra a continuación:

```
grep -r Serializable *
```

El comando devuelve que la serialización se usa en múltiples ocasiones pero relacionadas con elementos externos como puede ser ReCaptcha, Flutter o Firebase. Por lo tanto, al ser gestionadas y tener como origen recursos seguros, no supone un peligro de seguridad.

En lo que al código nativo se refiere, se han buscado la presencia de métodos relacionados con

la gestión de la memoria como pueden ser *strcpy*[5], *sprintf*[6], *malloc*[7], *free*[8], *memcpy*[9], *new*[10] o *delete*[11]. Para ello se ha ejecutado el comando *strings* junto a *grep* sobre cada uno de los ficheros con extensión *.so* encontrados en la anterior prueba. La estructura del comando es la siguiente:

```
strings ruta/*.so | grep -Ei ‘‘malloc|free|memcpy’’
```

En los ficheros se ha encontrado el uso de los mencionados elementos, no obstante cuando se realizan acciones sobre la memoria se tratan correctamente. Por ejemplo, si se reserva memoria con *malloc*, ésta se libera cuando se termina de usar con *free*.

Análisis dinámico.

Para realizar el análisis dinámico hay que volver a compilar la aplicación realizando un cambio sobre ella y lanzarla con *leakcanary* [53]. No obstante, al igual que ha sucedido en pruebas anteriores, la aplicación no funciona de forma correcta o bien no deja volver a compilarla o firmarla tras realizar modificaciones por la seguridad con la que cuenta por lo que esta parte del análisis no se puede realizar.

Para inspeccionar el código nativo de forma dinámica hay que usar Valgrind [52] para analizar el uso y las llamadas a memoria desde la aplicación. No obstante, para utilizar Valgrind para Android es necesario que el dispositivo esté rooteado por lo que tampoco se puede probar los erroes de memoria de esta forma [124].

Conclusiones de la prueba.

Aunque por restricciones de seguridad de la aplicación el análisis dinámico no se ha podido realizar, con el análisis estático no se ha encontrado evidencia de que existan errores de corrupción de la memoria.

8.2.14. MASTG-TEST-0047 - Prueba de comprobación de integridad de archivos.

Con el fin de verificar si la aplicación implementa algún tipo de comprobación de integridad para detectar cambios no autorizados tanto en el código fuente como en los ficheros relacionados con el de la aplicación, la prueba se divide en la comprobación de integridad de la fuente de ésta y la comprobación de integridad del almacenamiento [87].

Comprobación de integridad de la fuente de la aplicación.

Tal y como se ha comentado en pruebas anteriores, al tratar de modificar partes del código fuente de la aplicación, al volver a firmarla, o bien no permite completar la firma por conflicto con las bibliotecas, o bien si se firma, al ejecutarla se cierra y no permite su funcionamiento. Por lo

[5]Copia una cadena de caracteres en otra.
[6]Formatea texto y lo escribe en una cadena.
[7]Reserva memoria de forma dinámica.
[8]Libera la memoria previamente reservada con *malloc* tras su uso.
[9]Copia un bloque de memoria de un lugar a otro.
[10]Reserva memoria dinámicamente y crea variables en tiempo de ejecución.
[11]Libera la memoria previamente reservada con *new*.

tanto, se puede confirmar que cuenta con mecanismos de seguridad para impedir que se realicen modificaciones en su código fuente.

Comprobación de integridad del almacenamiento.

Al igual que para la anterior comprobación, la aplicación no admite modificaciones de elementos relacionados con el almacenamiento como pueden ser imágenes almacenadas. Se ha intentado lanzar la aplicación para buscar evidencias de defensas eludibles pero no se ha encontrado nada, solo un mensaje en el que se indicaba que la aplicación no es debuggueable, al probar a ejecutar la aplicación con *adb*. La aplicación tiene mecanismos que impiden que se comprometa la integridad del almacenamiento.

Conclusiones de la prueba.

La aplicación cuenta con mecanismos de seguridad tanto para evitar que se comprometa la integridad del código fuente de la aplicación como la integridad de los ficheros que almacena. Se puede concluir que la aplicación pasa la prueba.

8.2.15. MASTG-TEST-0049 - Prueba de la detección del emulador.

Para comprobar si la aplicación es capaz de detectar si está siendo ejecutada en un emulador, y por tanto impedir ciertas acciones o incluso bloquear su uso, se ha realizado el análisis dinámico sobre esta.

Análisis dinámico.

Tal y como se ha visto a lo largo del documento, la aplicación se ha lanzado sobre un emulador sin ningún impedimento. Para realizar la prueba se ha vuelto a navegar por la aplicación nuevamente realizando tareas como ver zonas de baño, filtrar por estado de la zona o por provincia, añadir a favoritos una zona de baño o ver los comentarios de la zona. En ningún momento se bloquea o se impide su ejecución [88].

Conclusiones de la prueba.

Puesto que la aplicación permite su uso en emuladores, la aplicación no pasa la prueba. Un atacante podría aprovechar los emuladores para lanzar ataques contra ésta.

Recomendaciones

Aunque la aplicación no realiza acciones críticas sobre datos del usuario, si que utiliza datos personales como el correo electrónico y la contraseña de la cuenta, sí que podría resultar conveniente que la aplicación impida realizar ciertas acciones en caso de ser ejecutada en un emulador como por ejemplo solo poder iniciar la sesión como invitado.

Para detectar el emulador se puede tratar de obtener la información del modelo, ya que suele contar con información como *emulator* en el nombre. Para comprobarlo desde un dispositivo Linux, por ejemplo, basta con ejecutar el siguiente comando:

```
adb devices
```

En la siguiente figura se puede ver cómo, al tener el emulador en marcha, con *adb* se detecta que hay un emulador llamado *emulator-5554*. Con esta información se puede limitar el uso de la aplicación si se encuentra un patrón así en la información del dispositivo:

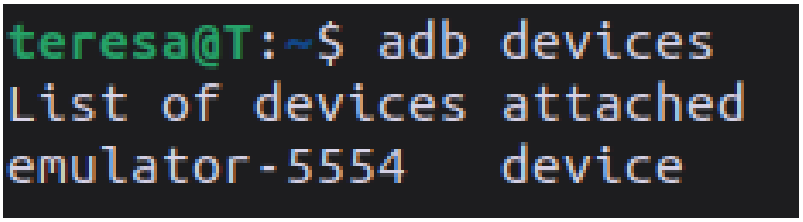

Figura 8.21: Salida de *adb devices* para averiguar el nombre del emulador en uso.

8.3. Resultados.

Una vez lanzadas las pruebas sobre la aplicación y comprobado si esta las pasa, o si por el contrario cuenta con alguna vulnerabilidad de seguridad, se han realizado unas tablas para poder ver de una forma más clara los resultados obtenidos.

En la siguiente tabla se muestran las pruebas que se han lanzado sobre AquaCyL, desglosando su resultado en análisis estático, análisis dinámico y el resultado global. Puesto que en algunas pruebas, debido a las medidas de seguridad de la aplicación o que no procede para dicha prueba, no se ha podido lanzar; estos resultados se muestran en la tabla pero no se tienen en cuenta para el resultado total de la prueba.

El criterio que determina el resultado final de la prueba sobre la aplicación es el siguiente:

- Pasa análisis estático (✓) y pasa análisis dinámico (✓): **Pasa la prueba** (✓).
- Pasa análisis estático (✓) y no pasa análisis dinámico (✗): **No pasa la prueba** (✗).
- No pasa análisis estático (✗) y no pasa análisis dinámico (✗): **No pasa la prueba** (✗).
- Pasa análisis estático (✓) y análisis dinámico no realizado (-): **Pasa la prueba** (✓).
- Análisis estático no realizado (-) y pasa análisis dinámico (✓): **Pasa la prueba** (✓).
- No pasa análisis estático (✗) y análisis dinámico no realizado (-): **No pasa la prueba** (✗).
- Análisis estático no realizado (-) y no pasa análisis dinámico (✗): **No pasa la prueba** (✗).

Prueba	**Análisis estático**	**Análisis dinámico**	**Resultado**
MASTG-TEST-0002: Prueba del almacenamiento local para la validación de los datos de entrada	✗	-	✗
MASTG-TEST-0004: Determinar si se comparten datos confidenciales con terceros a través de datos embebidos	✓	-	✓
MASTG-TEST-0008: Comprobación de la divulgación de datos confidenciales a través de la interfaz	✓	✓	✓
MASTG-TEST-0011: Prueba de memoria de datos confidenciales	✗	-	✗
MASTG-TEST-0014: Prueba de la configuración del algoritmo estándar de criptografía	✗	-	✗
MASTG-TEST-0017: Prueba para confirmar credenciales	✗	✗	✗
MASTG-TEST-0023: Prueba de proveedor de seguridad	✓	-	✓
MASTG-TEST-0026: Prueba de intenciones implícitas	✓	-	✓
MASTG-TEST-0027: Prueba de carga de URL en WebViews	✓	-	✓
MASTG-TEST-0036: Prueba de actualización forzada	✗	✗	✗
MASTG-TEST-0037: Prueba de limpieza de WebViews	✗	✗	✗
MASTG-TEST-0040: Prueba de símbolos de debugging	✓	-	✓
MASTG-TEST-0043: Errores de corrupción de memoria	✓	-	✓
MASTG-TEST-0047: Prueba de comprobación de integridad de archivos	✓	✓	✓
MASTG-TEST-0049: Prueba de la detección del emulador	-	✗	✗

Tabla 8.1: Resultados del lanzamiento de las pruebas sobre AquaCyL.

Observando los resultados de las pruebas en la tabla anterior, se puede ver como de las 15 de las lanzadas, sin tener en cuenta aquellas que no se han podido ya sea porque no aplican para la prueba o por tener restricciones de seguridad que implican su lanzamiento, se puede ver como 8 pasan el análisis estático, 2 el dinámico, y 8 resultan positivas, es decir, que pasan la prueba. Esto supone un 53,3 % del total, lo que implica que parte de la aplicación cuenta con los mecanismos de seguridad básicos para protegerse de parte de las vulnerabilidades, no obstante para aquellas para las que no se ha previsto protegerse, la aplicación es totalmente vulnerables.

Capítulo 9

Conclusiones.

El proyecto se ha realizado casi en su totalidad cumpliendo con el objetivo principal. Aunque no se han diseñado y lanzado todas las pruebas viables a probar sobre la aplicación, hacerlo habría extendido con creces el tiempo de trabajo estimado para un Trabajo de Fin de Grado.

En primer lugar, se ha llevado a cabo un estudio de la normativa actual vigente en lo relacionado a la protección de los datos de carácter personal, teniendo como principales el **RGPD** a nivel europeo y la **LOPDGDD** a nivel nacional. En la primera se marcan una serie de principios que se tienen que cumplir para que la gestión de los datos personales sea segura. Por otro lado, se han estudiado diferentes metodologías para llevar a cabo una auditoría móvil con el fin de ampliar el conocimiento sobre como probar la seguridad de una aplicación móvil. Finalmente, se ha realizado un análisis sobre las vulnerabilidades más comunes en la actualidad según refleja *OWASP Mobile Top 10*, indicando en qué consiste cada una, cómo los atacantes la explotan y cómo mitigarla, con el fin de que la aplicación sea más segura.

En segundo lugar, se ha realizado un análisis de la aplicación que se ha auditado, de modo que se ha investigado su funcionamiento y el tratamiento de datos que hace la aplicación, así como los permisos de los que requiere para funcionar. En paralelo a esto, se ha realizado un análisis de la metodología OWASP, con el fin de comprender cómo funciona, sobre todo la orientada a aplicaciones móviles; con ello se han relacionado los diferentes principios de la seguridad informática con las diferentes categorías que OWASP propone para verificar la seguridad en aplicaciones.

Una vez realizado lo anterior, se ha procedido al diseño de las pruebas a lanzar. Para ello se han establecido unos criterios de selección basados en la metodología OWASP y en la aplicación móvil. Una vez seleccionadas, se ha comenzado con su diseño siguiendo OWASP-MASTG. Con el fin de verificar que éste se ha realizado correctamente, se han lanzado las diferentes pruebas sobre una aplicación vulnerable, propuesta por OWASP. Posteriormente se han lanzado sobre AquaCyL, la aplicación elegida para ser auditada.

Tras lanzar las pruebas sobre AquaCyL, se han analizado los resultados una a una, proponiendo ciertas medidas para evitar o mitigar las vulnerabilidades encontradas. Por último se han estudiado los resultados en conjunto con el fin de contar con una visión más general de la seguridad de la

aplicación auditada.

Cabe destacar que el proyecto estaba previsto para desarrollarse completamente entre el 10 de febrero y el 30 de mayo, siendo un total de 16 semanas. No obstante, en realidad ha requerido de casi 5 semanas más de lo establecido, lo que supone aproximadamente un 30 % más de lo estimado en la planificación inicial. Con esto, el principal riesgo ha sido la mala estimación del tiempo, contrarrestándose entregándolo en convocatoria extraordinaria y realizando más horas de trabajo.

9.1. Trabajo futuro.

Como trabajo futuro se plantea completar el trabajo que en sí supondría una auditoría completa de seguridad. Esto es, diseñar cada una de las pruebas que resultan viables para ser lanzadas sobre AquaCyL y lanzarlas, para así contar con una visión completa de la seguridad de la aplicación y conocer todas las vulnerabilidades con las que puede contar según establece la metodología OWASP.

Por otro lado, resultaría una gran mejora automatizar todo el proceso del lanzamiento de las pruebas por medio de scripts para que, únicamente haya que ejecutar el programa que contenga todas las pruebas para obtener el resultado de la auditoría. Esto permitiría obtener conclusiones y fallos de seguridad en una menor cantidad de tiempo, lo que haría que se solucionasen antes, implicando que los atacantes reales cuenten con menos tiempo para explotar dichas vulnerabilidades.

Por último, sería interesante instalar la aplicación sobre un sistema operativo como GrapheneOS [125] y probar a auditar la aplicación desde allí, ya que, aunque la aplicación sea la misma, al tratarse de un sistema operativo que busca ofrecer privacidad al usuario, cabe la posibilidad que la aplicación reaccione de forma diferente en ese tipo de entornos.

Bibliografía

[1] U. Europea. (2025) Reglamento general de protección de datos (rgpd) — eur-lex. Unión Europea. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://eur-lex.europa.eu/ES/legal-content/summary/general-data-protection-regulation-gdpr.html.

[2] B. O. del Estado. (2025) Boe-a-2018-16673 ley orgánica 3/2018, de 5 de diciembre, de protección de datos personales y garantía de los derechos digitales. Boletín Oficial del Estado. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673.

[3] Dineshhetty. (2025) Github - dineshshetty/android-insecurebankv2: Vulnerable android application for developers and security enthusiasts to learn about android insecurities. Dineshhetty. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://github.com/dineshshetty/Android-InsecureBankv2.

[4] B. Hughes and M. Cotterell, *Software project management*, 5th ed. McGraw-Hill, 2009.

[5] Ryte. (2022) Modelo en espiral: todo lo que necesitas saber - ryte wiki. Ryte. Acceso: 15 de febrero de 2025. [En linea]. Disponible en: https://es.ryte.com/wiki/Modelo_en_Espiral.

[6] Atlassian. (2025) ¿qué es un diagrama de gantt? la hoja de ruta para el éxito del proyect. Atlassian. Acceso: 15 de febrero de 2025. [En linea]. Disponible en: https://www.atlassian.com/es/agile/project-management/gantt-chart.

[7] R. A. Española. (2022) Inicio — real academia española. RAE. Acceso: 12 de mayo de 2025. [En linea]. Disponible en: https://www.rae.es/.

[8] ——. (2022) Definición de dato de carácter personal - diccionario panhispánico del español jurídico - rae. RAE. Acceso: 12 de mayo de 2025. [En linea]. Disponible en: https://dpej.rae.es/lema/dato-de-car%C3%A1cter-personal.

[9] A. E. de Protección de Datos. (2025) ¿se pueden recabar y tratar datos personales de menores? — aepd. Agencia Española de Protección de Datos. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://www.aepd.es/preguntas-frecuentes/10-menores-y-educacion/FAQ-1002-se-puede-recabar-y-tratar-datos-personales-de-menores.

[10] OWASP. (2024) Owasp mobile top 10 — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/.

[11] ——. (2024) M1: Improper credential usage — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m1-improper-credential-usage.html.

[12] ——. (2024) M2: Inadequate supply chain security — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m2-inadequate-supply-chain-security.html.

[13] ——. (2024) M3: Insecure authentication/authorization — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m3-insecure-authentication-authorization.html.

[14] ——. (2024) M4: Insufficient input/output validation — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m4-insufficient-input-output-validation.html.

[15] ——. (2024) M5: Insecure communication — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m5-insecure-communication.html.

[16] ——. (2024) M6: Inadequate privacy controls — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m6-inadequate-privacy-controls.html.

[17] ——. (2024) M7: Insufficient binary protection — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m7-insufficient-binary-protection.html.

[18] ——. (2024) M8: Security misconfiguration — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m8-security-misconfiguration.html.

[19] ——. (2024) M9: Insecure data storage — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage.html.

[20] ——. (2024) M10: Insufficient cryptography — owasp foundation. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://owasp.org/www-project-mobile-top-10/2023-risks/m10-insufficient-cryptography.html.

[21] ——. (2025) Owasp masvs - owasp mobile application security. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASVS/.

[22] ——. (2025) Owasp mastg - owasp mobile application security. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/.

[23] C. S. R. Center. (2025) Sp 800-163 rev. 1, vetting the security of mobile applications — csrc. Computer Security Resource Center. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://csrc.nist.gov/pubs/sp/800/163/r1/final.

[24] (2025) Aquacyl - aplicaciones en google play. Google Play. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://play.google.com/store/apps/details?id=es.pablo.aquacyl.aqua_cyl&hl=es&pli=1.

[25] G. Play. (2025) Clasificación del contenido de aplicaciones y juegos en google play - ayuda de google play. Google Play. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://support.google.com/googleplay/answer/6209544?visit_id=638844717946658080-987820150&p=appgame_ratings&rd=1#zippy=%2Ceuropa-y-oriente-medio.

[26] ——. (2025) Entender las prácticas de privacidad y seguridad de las aplicaciones con la sección seguridad de los datos de google play - ordenador - ayuda de google play. Google Play. Acceso:

1 de julio de 2025. [En linea]. Disponible en: https://support.google.com/googleplay/answer/11416267?hl=es&visit_id=638869822440286322-1486938361&p=data-safety&rd=1.

[27] Google. (2025) Proporcionar información sobre la sección seguridad de los datos de google play - ayuda de play console. Google. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://support.google.com/googleplay/android-developer/answer/10787469?hl=en#zippy=%2Cpurposes%2Cdata-types.

[28] G. Maps. (2025) Google maps. Google Maps. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www.google.com/maps.

[29] E. rural. (2025) Escapadarural — reserva la casa rural para tu escapada. Escapada rural. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www.escapadarural.com/.

[30] OWASP. (2025) Owasp foundation, the open source foundation for application security — owasp foundation. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://owasp.org/.

[31] ——. (2025) Owasp mobile application security. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://mas.owasp.org/.

[32] UNIR. (2025) Principios de la seguridad informática: lo que debes conocer. UNIR. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://www.unir.net/revista/ingenieria/principios-seguridad-informatica/.

[33] K. coding. (2025) ¿qué es un token de sesión? [2025] — keepcoding bootcamps. Keep coding. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://keepcoding.io/blog/que-es-un-token-de-sesion/.

[34] M. goodwin. (2025) ¿qué es una api (interfaz de programación de aplicaciones)? — ibm. Michael Goodwin. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www.ibm.com/es-es/think/topics/api.

[35] P. digital. (2025) Webviews: puente nativo a operativas web - paradigma. Paradigma digital. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www.paradigmadigital.com/dev/webviews-puente-nativo-operativas-web/.

[36] Frida. (2025) Gadget — frida • a world-class dynamic instrumentation toolkit. Frida. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://frida.re/docs/gadget/.

[37] ClickUp. (2025) Comprender las pruebas de caja negra, caja blanca y caja gris. ClickUp. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://clickup.com/es-ES/blog/220921/caja-negra-caja-blanca-caja-gris-pruebas.

[38] R. Camacho. (2025) Análisis estático y análisis dinámico. Ricardo Camacho. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://es.parasoft.com/blog/static-analysis-and-dynamic-analysis/.

[39] Android. (2024) Cómo guardar datos simples con sharedpreferences app data and files android developers. Android. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://developer.android.com/training/data-storage/shared-preferences?hl=es-419.

[40] INCIBE. (2025) El ataque del “man in the middle” en la empresa, riesgos y formas de evitarlo — empresas — incibe. INCIBE. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www.incibe.es/empresas/blog/el-ataque-del-man-middle-empresa-riesgos-y-formas-evitarlo.

[41] Android. (2025) Notificationmanager api reference android developers. Android. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/app/NotificationManager.

[42] ——. (2025) Edittext api reference android developers. Android. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/widget/EditText.

[43] A. Developers. (2023) Intents pendientes - security - android developers. Android Developers. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/privacy-and-security/risks/pending-intent?hl=es-419.

[44] Frida. (2025) frida-trace — frida a world-class dynamic instrumentation toolkit. Frida. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://frida.re/docs/frida-trace/.

[45] A. W. Services. (2025) ¿qué es javascript? - explicación de javascript (js) - aws. Amazon Web Services. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://aws.amazon.com/es/what-is/javascript/.

[46] J. Carrión. (2025) Man in the middle: cómo generar tu propio proxy – visión de funnel. Jorge Carrión. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://visiondefunnel.wordpress.com/2023/03/16/man-in-the-middle-como-generar-tu-propio-proxy/.

[47] Android. (2025) Webview api reference android developers. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/webkit/WebView#clearCache(boolean).

[48] ——. (2025) Webstorage api reference android developers. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/webkit/WebStorage#deleteAllData.

[49] A. W. Services. (2025) ¿qué es sql? - explicación de lenguaje de consulta estructurado (sql) - aws. Amazon Web Services. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://aws.amazon.com/es/what-is/sql/.

[50] E. University. Html5: qué es, características y cómo funciona — esic.

[51] Kotlin. (2025) Kotlin programming language. Kotlin. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://kotlinlang.org/.

[52] Valgrind. (2025) Valgrind home. Valgrind. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://valgrind.org/.

[53] L. Canary. (2025) Github - square/leakcanary: A memory leak detection library for android. Leak Canary. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://github.com/square/leakcanary.

[54] modzero. (2025) Github - modzero/modjoda: Java object deserialization on android. modzero. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://github.com/modzero/modjoda.

[55] Keepcoding. (2025) ¿qué son los algoritmos hmac? — keepcoding bootcamps. Keepcoding. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://keepcoding.io/blog/que-son-los-algoritmos-hmac/.

[56] GitHub. (2025) Github · build and ship software on a single, collaborative platform · github. GitHub. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://github.com/.

[57] OWASP. (2025) Mastg-app-0010: Insecurebankv2 - owasp mobile application security. OWASP. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/apps/android/MASTG-APP-0010/.

[58] Genymotion. (2025) Account creation – genymotion android emulator. Genymotion. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www-v1.genymotion.com/account/create/.

[59] L. C. Library. (2025) pip man — linux command library. Linux Command Library. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://linuxcommandlibrary.com/man/pip.

[60] Python. (2025) Python release python 2.7.2 — python.org. Pyhton. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://www.python.org/downloads/release/python-272/.

[61] OWASP. (2025) Mastg-test-0002: Testing local storage for input validation - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0002/.

[62] L. Die. (2025) cat(1): concatenate files/print on stdout - linux man page. Linux Die. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/cat.

[63] ——. (2025) grep(1): print lines matching pattern - linux man page. Linux Die. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/grep.

[64] OWASP. (2025) Mastg-test-0004: Determining whether sensitive data is shared with third parties via embedded services - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0004/.

[65] ——. (2025) Mastg-tool-0077: Burp suite - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tools/network/MASTG-TOOL-0077/.

[66] Xammy. (2025) Toast en .net maui – askxammy. Xammy. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://es.askxammy.com/toast-en-net-maui/.

[67] OWASP. (2025) Mastg-test-0008: Checking for sensitive data disclosure through the user interface - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0008/.

[68] ——. (2025) Mastg-test-0011: Testing memory for sensitive data - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0011/.

[69] L. Die. (2025) strings(1) - linux man page. Linux Die. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/strings.

[70] OWASP. (2025) Mastg-test-0014: Testing the configuration of cryptographic standard algorithms - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-CRYPTO/MASTG-TEST-0014/.

[71] S. Dragon. (2025) Sha1 vs sha2 vs sha256 vs sha512 - ssl dragon. SSL Dragon. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www.ssldragon.com/es/blog/sha1-sha2-sha256-sha-512/.

[72] P. Security. (2025) ¿qué es el cifrado aes? - panda security. Panda Security. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://www.pandasecurity.com/es/mediacenter/cifrado-aes-guia/.

[73] OWASP. (2025) Mastg-test-0017: Testing confirm credentials - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-AUTH/MASTG-TEST-0017/.

[74] Microsoft. (2024) Método setuserauthenticationrequired de la clase win32_tsgeneralsetting - win32 apps microsoft learn. Microsoft. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://learn.microsoft.com/es-es/windows/win32/termserv/setuserauthenticationrequired-win32-tsgeneralsetting.

[75] OWASP. (2025) Mastg-test-0023: Testing the security provider - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-NETWORK/MASTG-TEST-0023/.

[76] Google. (2024) Providerinstaller google play services google for developers. Google. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developers.google.com/android/reference/com/google/android/gms/security/ProviderInstaller.

[77] OWASP. (2025) Mastg-test-0026: Testing implicit intents - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0026/.

[78] ——. (2025) Mastg-test-0027: Testing for url loading in webviews - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0027/.

[79] ——. (2025) Mastg-test-0036: Testing enforced updating - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0036/.

[80] L. Die. (2025) vi(1): Vi improved, programmers text editor - linux man page. Linux Die. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/vi.

[81] OWASP. (2025) Mastg-test-0037: Testing webviews cleanup - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0037/.

[82] ——. (2025) Mastg-test-0040: Testing for debugging symbols - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0040/.

[83] ——. (2025) Mastg-test-0043: Memory corruption bugs - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0043/.

[84] A. Developer. (2025) Broadcastreceiver api reference android developers. Android Developer. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/content/BroadcastReceiver.

[85] S. Overflow. (2025) android - java.lang.securityexception: Mode_world_readable no longer supported - stack overflow. Stack Overflow. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://stackoverflow.com/questions/39121052/java-lang-securityexception-mode-world-readable-no-longer-supported.

[86] Gradle. (2025) Gradle build tool. Gradle. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://gradle.org/.

[87] OWASP. (2025) Mastg-test-0047: Testing file integrity checks - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0047/.

[88] ——. (2025) Mastg-test-0049: Testing emulator detection - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0049/.

[89] Apktool. (2025) Apktool. Apktool. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://apktool.org/.

[90] J. M. Arenas. (2019) Que es smali y como parchear una aplicación android - hackpuntes. José María Arenas. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://hackpuntes.com/que-es-smali-y-como-parchear-una-aplicacion-android/.

[91] Android. (2025) Descarga e instala android studio. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/codelabs/basic-android-kotlin-compose-install-android-studio?hl=es-419#5.

[92] L. die. (2025) lscpu(1): Cpu architecture - linux man page. Linux die. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/lscpu.

[93] ——. (2025) free(3): allocate/free dynamic memory - linux man page. Linux die. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://linux.die.net/man/3/free.

[94] ——. (2025) df(1): report file system disk space usage - linux man page. Linux die. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/df.

[95] ——. (2025) xrandr(1): primitive cli to randr extension - linux man page. Linux die. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/xrandr.

[96] ——. (2025) tar(1): manual page for tar 1.23 - linux man page. Linux die. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/tar.

[97] L. M. Page. (2025) cd(1p) - linux manual page. Linux Manual Page. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://man7.org/linux/man-pages/man1/cd.1p.html.

[98] Android. (2025) Cómo instalar android studio android developers. Android. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://developer.android.com/studio/install?hl=es-419.

[99] IBM. (2025) Emuladores - documentación de ibm. IBM. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://www.ibm.com/docs/es/aix/7.2.0?topic=concepts-emulators.

[100] A. Developers. (2025) Manifest.permission api reference android developers. Android Developers. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/Manifest.permission.

[101] ——. (2025) Cómo conectarse a una red connectivity android developers. Android Developers. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/develop/connectivity/network-ops/connecting?hl=es-419.

[102] Google. (2025) Id de publicidad - ayuda de play console. Google. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://support.google.com/googleplay/android-developer/answer/6048248?hl=es.

[103] OWASP. (2025) Mastg-tech-0011: Setting up an interception proxy - owasp mobile application security. OWASP. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://mas.owasp.org/MASTG/tools/network/MASTG-TOOL-0077/.

[104] PortSwigger. (2025) Configuring an android device to work with burp suite - portswigger. PortSwigger. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://portswigger.net/burp/documentation/desktop/mobile/config-android-device.

[105] Firebase. (2025) Firebase authentication. Firebase. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://firebase.google.com/docs/auth?hl=es-419.

[106] IBM. (2025) ¿qué es garbage collection de java? — ibm. IBM. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://www.ibm.com/mx-es/topics/garbage-collection-java.

[107] Oracle. (2025) javax.crypto (java platform se 8). Oracle. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://docs.oracle.com/javase/8/docs/api/javax/crypto/package-summary.html.

[108] ——. (2025) Cipher (java platform se 8). Oracle. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://docs.oracle.com/javase/8/docs/api/javax/crypto/Cipher.html.

[109] ——. (2025) Mac (java platform se 8). Oracle. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://docs.oracle.com/javase/8/docs/api/javax/crypto/Mac.html.

[110] Microsoft. (2025) Signature clase (java.security) microsoft learn. Microsoft. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://learn.microsoft.com/es-es/dotnet/api/java.security.signature?view=net-android-34.0.

[111] Oracle. (2025) Keygenerator (java platform se 8). Oracle. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://docs.oracle.com/javase/8/docs/api/javax/crypto/KeyGenerator.html.

[112] ——. (2025) Keystore (java platform se 7). Oracle. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html.

[113] niklashigi. (2025) Github - niklashigi/apk-mitm: A cli application that automatically prepares android apk files for https inspection. niklashigi. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://github.com/niklashigi/apk-mitm.

[114] Android. (2025) Cómo actualizar tu proveedor de seguridad para protegerte contra exploits de ssl security android developers. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/privacy-and-security/security-gms-provider?hl=es-419.

[115] ——. (2025) Intent api reference android developers. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/content/Intent.

[116] ——. (2025) Intents de cámara android media android developers. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/media/camera/camera-intents?hl=es-419.

[117] Flutter. (2025) Flutter - build apps for any screen. Flutter. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://flutter.dev/.

[118] Google. (2024) recaptcha google for developers. Google. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developers.google.com/recaptcha?hl=es-419.

[119] ——. (2021) Descripción general del service worker workbox chrome for developers. Google. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.chrome.com/docs/workbox/service-worker-overview?hl=es-419.

[120] Android. (2025) Websettings api reference android developers. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/webkit/WebSettings.

[121] ——. (2025) Webviewrenderprocessclient api reference android developers. Android. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://developer.android.com/reference/android/webkit/WebViewRenderProcessClient.

[122] Kotlin. (2025) deleterecursively. Kotlin. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://kotlinlang.org/api/core/kotlin-stdlib/kotlin.io/delete-recursively.html#.

[123] L. Die. (2025) find(1) - linux man page. Linux Die. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://linux.die.net/man/1/find.

[124] S. Overflow. (2025) permissions - valgrind cannot execute memcheck tool on android os? - stack overflow. Stack Overflow. Acceso: 26 de junio de 2025. [En linea]. Disponible en: https://stackoverflow.com/questions/19641111/valgrind-cannot-execute-memcheck-tool-on-android-os.

[125] GrapheneOS. (2025) Grapheneos: the private and secure mobile os. GrapheneOS. Acceso: 1 de julio de 2025. [En linea]. Disponible en: https://grapheneos.org/.

ANEXO I: Pruebas para realizar una auditoría de seguridad móvil según la metodología OWASP.

En el presente anexo se muestran las pruebas que OWASP ofrece para auditar una aplicación móvil. Por otro lado se indica si la prueba está o no en uso, es decir si se considera o no como obsoleta y si para la aplicación auditada en el presente proyecto, AquaCyL, es viable de realizar o no:

Categoría	**Prueba**	**Descripción**	**En uso**	**Viable**
MASVS-AUTH	MASTG-TEST-0017	Prueba para confirmar credenciales.	Sí	✓
MASVS-AUTH	MASTG-TEST-0018	Prueba de autenticación biométrica.	Sí	✗
MASVS-CODE	MASTG-TEST-0002	Prueba del almacenamiento local para la validación de los datos de entrada.	Sí	✓
MASVS-CODE	MASTG-TEST-0025	Prueba de defectos de inyección.	Sí	✓
MASVS-CODE	MASTG-TEST-0026	Prueba de intenciones implícitas.	Sí	✓
MASVS-CODE	MASTG-TEST-0027	Prueba de carga de URL en WebViews.	Sí	✓
MASVS-CODE	MASTG-TEST-0034	Prueba de persistencia de objetos.	Sí	✓
MASVS-CODE	MASTG-TEST-0036	Prueba de actualización forzada.	Sí	✓
MASVS-CODE	MASTG-TEST-0042	Comprobación de debilidades en bibliotecas de terceros.	No	✗
MASVS-CODE	MASTG-TEST-0043	Errores de corrupción de memoria.	Sí	✓
MASVS-CODE	MASTG-TEST-0044	Prueba de que las funciones de seguridad gratuitas están activadas.	No	✗

Categoría	**Prueba**	**Descripción**	**En uso**	**Viable**
MASVS-CRYPTO	MASTG-TEST-0013	Prueba de criptografía simétrica.	No	✗
MASVS-CRYPTO	MASTG-TEST-0014	Prueba de la configuración del algoritmo estándar de criptografía.	Sí	✓
MASVS-CRYPTO	MASTG-TEST-0015	Prueba del propósito de las claves.	Sí	✓
MASVS-CRYPTO	MASTG-TEST-0016	Prueba del generador aleatorio de números.	No	✗
MASVS-NETWORK	MASTG-TEST-0019	Prueba de cifrado de datos en la red.	No	✗
MASVS-NETWORK	MASTG-TEST-0020	Probando la configuración de TLS[1].	No	✗
MASVS-NETWORK	MASTG-TEST-0021	Prueba de verificación de identidad del endpoint.	Sí	✓
MASVS-NETWORK	MASTG-TEST-0022	Prueba de almacenes de certificados y fijación de certificados.	No	✗
MASVS-NETWORK	MASTG-TEST-0023	Prueba del proveedor de seguridad.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0007	Determinar si se han expuesto datos sensibles almacenados a través de mecanismos de IPC.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0008	Comprobación de la divulgación de datos confidenciales a través de la interfaz de usuario.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0010	Búsqueda de información confidencial en capturas de pantalla generadas automáticamente.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0024	Prueba de permisos de aplicaciones.	No	✗
MASVS-PLATFORM	MASTG-TEST-0028	Prueba de enlaces profundos.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0029	Prueba de exposición de funcionalidad sensible a través de IPC.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0030	Prueba de implementación vulnerable de PendingIntent[2].	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0031	Prueba de ejecución de JavaScript en WebViews.	Sí	✓

Categoría	**Prueba**	**Descripción**	**En uso**	**Viable**
MASVS-PLATFORM	MASTG-TEST-0032	Prueba de controladores del protocolo WebView.	No	✗
MASVS-PLATFORM	MASTG-TEST-0033	Prueba de objetos Java expuestos a través de WebViews.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0035	Prueba de ataques de superposición.	Sí	✓
MASVS-PLATFORM	MASTG-TEST-0037	Prueba de limpieza de WebViews.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0038	Asegurarse de que la aplicación esté correctamente firmada.	No	✗
MASVS-RESILIENCE	MASTG-TEST-0039	Probar si la aplicación se puede depurar.	No	✗
MASVS-RESILIENCE	MASTG-TEST-0040	Prueba de símbolos de debugging.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0041	Prueba de código de depuración y registro de errores detallado.	No	✗
MASVS-RESILIENCE	MASTG-TEST-0045	Prueba de detección de root.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0046	Prueba de detección anti-depuración.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0047	Prueba de comprobación de integridad de archivos.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0048	Prueba de detección de herramientas de ingeniería inversa.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0049	Prueba de la detección del emulador.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0050	Comprobaciones de integridad en tiempo de ejecución.	Sí	✓
MASVS-RESILIENCE	MASTG-TEST-0051	Prueba de ofuscación.	Sí	✓
MASVS-STORAGE	MASTG-TEST-0001	Prueba de almacenamiento local para datos confidenciales.	No	✗
MASVS-STORAGE	MASTG-TEST-0003	Prueba de registros en busca de datos confidenciales.	No	✗
MASVS-STORAGE	MASTG-TEST-0004	Determinar si se comparten datos confidenciales con terceros a través de embebidos.	Sí	✓

Categoría	**Prueba**	**Descripción**	**En uso**	**Viable**
MASVS-STORAGE	MASTG-TEST-0005	Determinar si se comparten datos sensibles a través de las notificaciones.	Sí	✓
MASVS-STORAGE	MASTG-TEST-0006	Determinar si la caché del teclado está deshabilitado para los campos de entrada de texto.	No	✗
MASVS-STORAGE	MASTG-TEST-0009	Prueba de copias de seguridad de datos confidenciales.	No	✗
MASVS-STORAGE	MASTG-TEST-0011	Prueba de memoria de datos confidenciales.	Sí	✓
MASVS-STORAGE	MASTG-TEST-0012	Prueba de la política de seguridad de acceso al dispositivo.	No	✗

Tabla 9.1: Pruebas OWASP para la auditoría de aplicaciones móviles.

En la anterior tabla, se puede ver como algunas de las pruebas que se muestran se encuentran en estado obsoleto. Esto puede deberse a diferentes cambios en el sistema operativo o a cambios en lo que a las buenas prácticas sobre seguridad se refiere.

Por otra parte, a parte de las anteriores pruebas, OWASP cuenta con una serie de test que se denominan *Tests (v2 Beta)*. Para el presente proyecto, se ha optado por no utilizar dichas pruebas ya que las versiones beta aún se encuentran en un estado preliminar y sujeto a cambios. Con ello se busca mantener una solidez de la metodología así como asegurar unos resultados válidos al contar con una metodología revisada y validada.