Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

Aplicación de la metodología OWASP Mobile a la auditoría de un conjunto de aplicaciones móviles

Autor:
León de la Hera, Gerardo

Tutores:
Martínez González, María de las Mercedes
Aparicio de la Fuente, Amador

Agradecimientos

Me gustaría comenzar agradeciendo a mis padres, que han sido los que me han brindado la oportunidad de poder estudiar lo que yo quería y han hecho un gran esfuerzo para que pudiera estudiar fuera de casa, sin tener que preocuparme de nada más que de lo mío. Además, sus ánimos y exigencias por que diese lo mejor de mí mismo han sido vitales para que haya podido acabar esta carrera. Quiero agradecer también a mi hermana por alegrarme los días siempre que estoy con ella, y a todos mis abuelos por su apoyo incondicional en todo momento.

Agradezco también a todos mis amigos de mi pueblo y hogar, Saldaña, a los de toda la vida y a los que he ido conociendo estos últimos años. Volver todos los fines de semana y pasar el tiempo con ellos ha sido sin duda vital para quitarme el estrés que a veces me producían los estudios y ser más feliz estos 5 años.

Además de ellos, también quiero agradecer a todas las personas que he conocido durante mi estancia en la carrera, por su ayuda, y todos los momentos compartidos durante cada semestre y los periodos de exámenes, tanto los buenos como los malos. Sin ellos, la experiencia no habría sido la misma.

Por último, expreso mi gratitud a cada profesor que me ha impartido clase a lo largo de estos cuatro años, y especialmente a mis tutores de TFG, Mercedes y Amador, por todo el apoyo brindado, la atención, los ánimos y las valiosas correcciones a mi trabajo.

Resumen

Nuestros nombres y apellidos, documentos identificativos, datos médicos o bancarios e incluso opiniones políticas; todos estos son ejemplos de datos que las aplicaciones de nuestros dispositivos móviles utilizan día tras día. La RAE define privacidad como el *ámbito de la vida privada que se tiene derecho a proteger de cualquier intromisión*, y en el contexto digital actual, se ha convertido en un aspecto crucial, debido a que se recaban y procesan enormes cantidades de datos privados de los usuarios. La protección de la privacidad es esencial para salvaguardar los derechos individuales y prevenir el uso indebido de datos sensibles, y, en consecuencia, surgen proyectos como este, en el que se trata de identificar y poner a prueba mediante auditorías móviles las vulnerabilidades que puedan comprometer la protección de los datos.

A lo largo de este informe, se llevará a cabo una auditoría de privacidad sobre diversas aplicaciones móviles utilizando la metodología OWASP Mobile, un marco reconocido internacionalmente para la evaluación de la seguridad de aplicaciones móviles. Esta metodología proporciona un conjunto estructurado de pruebas, de entre las cuales seleccionaré las que considero que se encuentren más destinadas a mitigar las vulnerabilidades que pueden comprometer la privacidad de los usuarios. El informe se dividirá en varias secciones, incluyendo en primer lugar una revisión de la normativa de privacidad vigente y de la metodología OWASP Mobile; la descripción de las pruebas seleccionadas para la auditoría; y, por último, el proceso de ejecución de cada una de estas pruebas con la documentación de sus resultados obtenidos.

Abstract

Our names and surnames, identification documents, medical or banking information and even political opinions; These are all examples of data that the applications on our mobile devices use day after day. The RAE defines privacy as the *area of private life that one has the right to protect from any intrusion*, and in the current digital context, it has become a crucial aspect, given that enormous amounts of users' private data are collected and processed. Privacy protection is essential to safeguard individual rights and prevent the misuse of sensitive data, and because of this, projects like this one arise, in which the aim is to identify and test vulnerabilities that may compromise the data through mobile audits. data protection.

Throughout this report, a privacy audit will be carried out on various mobile applications using the OWASP Mobile methodology, an internationally recognized framework for evaluating the security of mobile applications. This methodology provides a structured set of tests, from which I will select those that I consider to be most intended to mitigate vulnerabilities that may compromise user privacy. The report will be divided into several sections, including first a review of current privacy regulations and the OWASP Mobile methodology; the description of the tests selected for the audit, and, finally, the execution process of each of these tests with the documentation of the results obtained.

ÍNDICE GENERAL

Índice general

ÍNDICE GENERAL

ÍNDICE GENERAL

ÍNDICE GENERAL

Índice de Figuras

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

Índice de Tablas

INDICE DE TABLAS

Capítulo 1

Introducción

1.1. Contexto

En la actualidad, el uso de dispositivos móviles se ha vuelto un factor tan cotidiano como necesario para el día a día de las personas. Según un estudio del *Instituto Nacional de Estadística* (**INE**) [51], el 99,5% de los hogares españoles disponen de al menos un dispositivo móvil para uso diario. Este hecho, cada vez menos sorprendente por otra parte, refleja como nuestra sociedad se encuentra completamente digitalizada y, por ende, como la cantidad de datos que son manejados diariamente por las aplicaciones móviles es cada vez más y más significativa. Muchos de estos datos son de carácter personal de los usuarios y su protección frente a posibles atacantes, cuyo número ha crecido de igual forma que el de los usuarios, se ha vuelto indispensable para asegurar su privacidad.

A razón de todo lo comentado, se explica la existencia de las normativas sobre Protección de Datos, que se estipulan tanto a nivel europeo, con el *Reglamento General de Protección de Datos* (**RGPD**) [1], como a nivel nacional con la *Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de Derechos Digitales* (**LOPDGDD2018**) [2]. Estas normativas imponen a las aplicaciones que se proteja la identidad de los individuos, asegurando que no sea posible identificar personas a partir de datos públicos, excepto en casos donde exista consentimiento explícito o interés legítimo por parte de los interesados.

Adicionalmente, para asegurar el cumplimiento de la legislatura expuesta por cada una de estas normativas, han ido surgiendo con el paso de los años organismos y entidades que proporcionan sus propias directrices sobre el manejo de datos en entornos móviles, como la *Agencia Española de Protección de Datos* (**AEPD**) y el *Consejo Europeo de Protección de Datos* (**CEPD**) [3]. Por otro lado, existen también diversas metodologías que buscan exponer las vulnerabilidades en aplicaciones móviles, como el *Estándar de Seguridad de Aplicaciones Móviles* (**OWASP MASVS** (Mobile Application Security Verification Standard)) [4].

El presente proyecto se centrará en identificar vulnerabilidades en la protección de la privacidad de los usuarios presentes en dos aplicaciones android de acceso público: InsecureBanck y RigthsApp. Para ello se utilizará una auditoría creada siguiendo las recomendaciones de la metodología OWASP MOBILE.

1.2. Motivación

Paralelo al crecimiento del número de usuarios propietarios de dispositivos móviles, y de la cantidad de datos procesada por las aplicaciones de estos dispositivos, también encontramos como el número de ataques y delitos relacionados con violaciones de la privacidad de estos usuarios ha crecido exponencialmente. *Según el Observatorio Español de Delitos Informáticos* (**OEDI**) [5], en el año 2022 se registraron 374.737 delitos informáticos en España, siguiendo con una tendencia al alza de años anteriores. Muchos de estos delitos, como la extorsión o el phishing[1], siguen ocurriendo pese a la existencia de normativas que estipulan que la información personal utilizada en las aplicaciones debe ser resguardada para evitar que sea accesible por personas no autorizadas o posibles atacantes.

Como se ha expuesto en el apartado del contexto, existen multitud de legislaturas, normativas u organismos destinados a garantizar la protección de la privacidad de los usuarios, y pese a ello se siguen registrando multitud de delitos, los cuales buscan, en su gran mayoría, la obtención de datos personales que pongan en peligro la integridad de la privacidad de los individuos. En ese contexto surgen las **auditorías móviles**, que buscan evaluar que las aplicaciones móviles cumplan los requisitos de seguridad establecidos por metodologías como la definida por OWASP.

De todo lo explicado anteriormente surge la motivación general detrás de este proyecto. Desde un primer momento consideré interesante el tema de la privacidad y el manejo de datos personales, cada vez más al alza por el aumento exponencial de los mismos. A su vez llegué a la conclusión de que la forma más práctica de aprender sobre este tema sería realizando una auditoría sobre una aplicación que me diese una visión general de cómo se maneja y protege la privacidad, y de qué vulnerabilidades pueden aparecer. Quería, además, seguir la metodología de auditoría propuesta por OWASP, debido a la gran reputación que sus pruebas tienen en la comunidad de la ciberseguridad. Buscando información sobre las mismas, me di cuenta de que esta metodología se centra en evaluar de forma más general la seguridad del dispositivo y cumplir con los requisitos de seguridad establecidos, sin reunir toda su atención en la búsqueda de debilidades relativas a la privacidad. Ante este hecho, consideré muy interesante crear una auditoría que se centrase en evaluar la protección de los datos personales y sensibles de los usuarios, tomando las pruebas que más se ajustasen a ello de las propuestas por OWASP.

Por último, me decanté por hacer esta auditoría sobre una aplicación móvil y no sobre una aplicación web debido a que, para informarme del todo del proceso de realización de una auditoría en un proyecto de fin de grado como este, revisé el trabajo de fin de grado "*AuditaWebPriv: Auditoría aplicada a la Privacidad en Aplicativos Web*" [52] entregado por **José Francisco Villatoro Meyer** en 2022. Este proyecto me ha servido de gran inspiración y he seguido una estructura muy similar a la de mi compañero para redactar mi informe, pero quise distanciarme del mismo centrándome en realizar la auditoría sobre aplicaciones móviles, sobre las cuales, también sea dicho, nunca había trabajado y consideré un reto adicional para mi trabajo.

[1] Método de fraude en el que se engaña a las personas para obtener información confidencial haciéndose pasar por una entidad confiable.

1.3. Objetivos

El objetivo principal del proyecto es desarrollar un **marco[2] de auditoría** de seguridad móvil Android siguiendo la metodología OWASP MOBILE, que se centre completamente en la búsqueda de vulnerabilidades relacionadas con la privacidad de los datos de los usuarios. A continuación, se especifican los objetivos concretos:

- Crear una **versión de auditoría de seguridad** siguiendo la metodología OWASP para aplicaciones móviles, que se centre completamente en determinar la privacidad de los usuarios.

- Desarrollar una **guía de auditoría** que documente los pasos seguidos durante las pruebas y los resultados obtenidos, para que cualquier persona con un mínimo de conocimientos informáticos pueda replicar si problema.

1.4. Organización de la memoria

- **Capítulo 1. Introducción**: Se presenta una breve visión general del proyecto, destacando sus aspectos esenciales y objetivos fundamentales.

- **Capítulo 2. Planificación del Proyecto**: Comprende la organización inicial del trabajo por horas y días de la semana, así como las fechas estimadas de finalización tanto de tareas individuales como del proyecto en su conjunto. Incluye también un plan detallado de gestión de riesgos.

- **Capítulo 3. Estado del Arte**: Se recopila información sobre las normativas de protección de datos vigentes y las metodologías de pruebas en aplicaciones móviles definidas por OWASP.

- **Capítulo 4. Metodología**: Detalla la forma en la que se ha trabajado en cada una de las pruebas de la auditoría, estableciendo que se sigue el marco de trabajo propuesto por la metodología OWASP MOBILE.

- **Capítulo 5. Diseño**: Este capítulo define las posibles vulnerabilidades relativas a la privacidad que puede tener una app android, y abarca la categorización de pruebas definidas por la metodología MASVS de OWASP, según su impacto en la protección de

[2] Estructura que define el alcance, los objetivos, los criterios y los procedimientos para llevar a cabo una auditoría.

datos personales. Finalmente presenta y describe las pruebas seleccionadas para la explotación de las vulnerabilidades de privacidad.

- **Capítulo 6. Auditoría de las aplicaciones**: Describe el proceso de ejecución de cada una de las pruebas y el analiza los resultados en las dos aplicaciones auditadas.

- **Capítulo 7. Conclusiones**: Se representan a grandes rasgos los resultados obtenidos de la realización del proyecto, y se describe un posible trabajo futuro.

- **Anexo I. RGPD**: Se mencionan algunos artículos relevantes del Reglamento General de Protección de Datos (RGPD) relacionados con la protección de datos personales.

- **Anexo II. Herramientas para las pruebas sobre aplicaciones móviles**: Se lista y describe el funcionamiento de cada una de las aplicaciones utilizadas para la realización de las auditorías.

Capítulo 2

Planificación del proyecto

Cuando se comienza a realizar un proyecto de esta índole, resulta indispensable delimitar una planificación que resulte **coherente** con los objetivos propuestos, el nivel de conocimiento de la persona que lo realizará y el tiempo disponible para su realización.

Sabiendo todo esto, me dispongo en el presente capítulo a realizar la planificación de mi proyecto, definiendo en primer lugar las fases propias del mismo; a continuación, escogiendo una metodología de trabajo y dibujando a partir de ella la distribución de los distintos hitos a lo largo del tiempo disponible para el trabajo; y finalmente concretando los diversos riesgos posibles y las posibles acciones para su mitigación.

2.1. Fases del Proyecto

Como se ha podido ver en el apartado de objetivos, la meta principal de este proyecto es completar un modelo de auditoría que aúne las pruebas de la metodología OWASP que más relación tengan con la protección de los datos personales de los usuarios y aplicarla sobre dos aplicaciones Android. Esto implica que el proyecto cuente con una serie de **fases indispensables** para asegurar la coherencia del trabajo realizado en el mismo:

- **Estudio de Contexto de Protección de Datos**: se deberá realizar un estudio sobre las normativas vigentes de protección de datos, así como de las posibles vulnerabilidades relacionadas. Además, se deberá de tener claro cuáles son los datos privados que comprometen la integridad de los usuarios que se están buscando.

- **Estudio del Contexto de las Auditorías Móviles**: una vez conocido el estado actual de la legislación respectiva a la privacidad, se deberá hacer un estudio sobre la metodología OWASP de la que se hará uso en el proyecto y sobre todo, de las diversas pruebas que propone.

- **Diseño de las Pruebas**: en esta etapa se partirá de la lista de pruebas definida por la metodología OWASP, y, teniendo en cuenta todo lo aprendido respecto a la privacidad y la protección de los datos, se seleccionarán las pruebas que más se adecuen al cumplimiento de los objetivos del proyecto.

- **Puesta a punto y Conocimiento de las Herramientas**: cada una de las pruebas seleccionadas se encuentra debidamente documentada y en ellas se definen las herramientas necesarias para su correcta ejecución. En esta fase se estudiará el funcionamiento de todas las herramientas a utilizar y se configurarán para su correcto funcionamiento.

- **Ejecución de las Pruebas de las Auditorías:** esta es la etapa en la que se llevará a cabo la ejecución de todas las pruebas que se han recogido a lo largo del periodo de diseño. Se seguirá cada paso estipulado y se documentará todas las acciones realizadas y todos los resultados obtenidos.

- **Conclusiones**: en esta última fase se tratará de juntar todos los resultados obtenidos y aprendizajes adquiridos a lo largo del proyecto para obtener un balance del mismo.

Cada una de las fases descritas resultan fundamentales para la obtención de los objetivos de este proyecto. Resulta también tremendamente importante ejecutar cada una de ellas siguiendo el **orden** estipulado, puesto que así se asegura una transición fluida entre etapas en la que se van adquiriendo los conocimientos necesarios para una correcta aplicación de las metodologías estudiadas.

2.2. Metodología de trabajo

El desarrollo de proyectos implica la existencia de metodologías[3] de trabajo que definan la forma en la que se debe abordar la **planificación** y **gestión** de los mismos. Para decidir qué metodología es la más idónea para el trabajo sobre un proyecto, se deben conocer claramente las características propias que este posee, así como las diversas fases que se vislumbran para su ejecución.

Uno de los campos que define de forma más clara varias metodologías de trabajo para la planificación de sus proyectos es el *Desarrollo del Software*. Esto se debe a la necesidad de una **planificación eficiente** que estos proyectos requieren frente a las grandes cargas de trabajo que registran, organizándolas en múltiples hitos[4]. Aunque este proyecto no trabaje en ningún momento con software, considero que existen similitudes respecto a los proyectos de desarrollo del mismo, más específicamente, en la organización en fases o hitos que ambos tienen. Es por ello, que, aprovechando que tengo definidas las fases de este proyecto que pueden funcionar a modo de hitos, se hará uso de una de las metodologías de trabajo software.

Partiendo de la base de que esta es mi primera vez desarrollando un proyecto de este tipo, considero fundamental que la metodología escogida me pueda permitir cierta flexibilidad a la hora de moverme entre los distintos hitos definidos en caso de haber hecho alguna estimación errónea. A su vez, como ya he comentado, es muy importante ir completando las fases marcadas

[3] Enfoque estructurado para planificar, gestionar y ejecutar proyectos.

[4] Puntos clave en un proyecto que marcan la finalización de tareas importantes o fases críticas.

una por una, para poder llevar a cabo el proyecto de la forma más idónea posible. Teniendo en cuenta todo esto y considerando todas las opciones posibles, finalmente se ha decidido hacer uso del **modelo en espiral** [53] como método de trabajo, debido a la flexibilidad que este ofrece en el trabajo entre hitos.

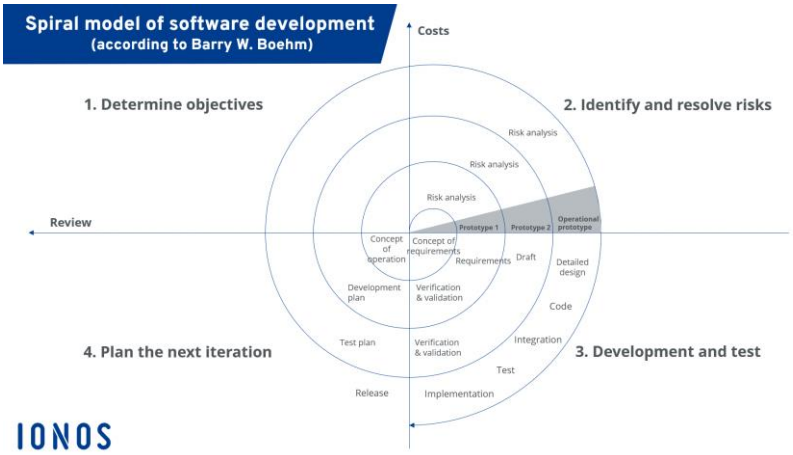


Ilustración 1: Modelo en Espiral

Este enfoque, además, implica una **revisión de riesgos periódica** con cada consecución de un hito, por lo que ayudará a mitigar una incorrecta definición de riesgos, preocupación propia de la inexperiencia en este tipo de proyectos.

2.3. Planificación Inicial

Definida la metodología y los hitos o fases, cada uno de ellos con revisión junto a los tutores, será momento de dibujar la planificación del trabajo realizado. Para ello, se deberá de tener en cuenta el tiempo disponible para su realización. El proyecto se desarrollará en un total de 300 horas, distribuidas desde el 25 de marzo de 2024 hasta el 5 de julio de 2024 (15 semanas). Teniendo en cuenta mi disponibilidad semanal, la distribución inicial de horas resultante es la siguiente:

DIAS	HORARIO	TOTAL DE HORAS
Lunes	9:00-13:00	4
Martes	9:00-13:00	4
Miércoles	9:00-13:00	4
Jueves	16:00-20:00	4
Viernes	9:00-13:00	4
TOTAL		20 horas/semana

Tabla 2.1: Distribución del trabajo del proyecto

Gracias a esta distribución de trabajo se podrán usar los sábados y los domingos de respaldo en caso de que no se hubiese podido trabajar durante alguno de los días de la semana por cualquier motivo.

Siguiendo este esquema, se necesitarían 12 semanas para completar las 300 horas de trabajo estipuladas. Se puede observar, por tanto, como sobrarían 3 semanas de las estipuladas para el desarrollo del proyecto. Estas 3 semanas servirán como colchón en caso de que el proyecto se viera retrasado por la aparición de alguno de los riesgos que más adelante definiremos.

A continuación, se presentará el **Diagrama de Gantt**[5] en el que se manifiesta la planificación del proyecto, mostrando cada uno de los hitos definidos, sus diferentes subtareas, y el tiempo que se ha estimado para completar cada una de ellas.

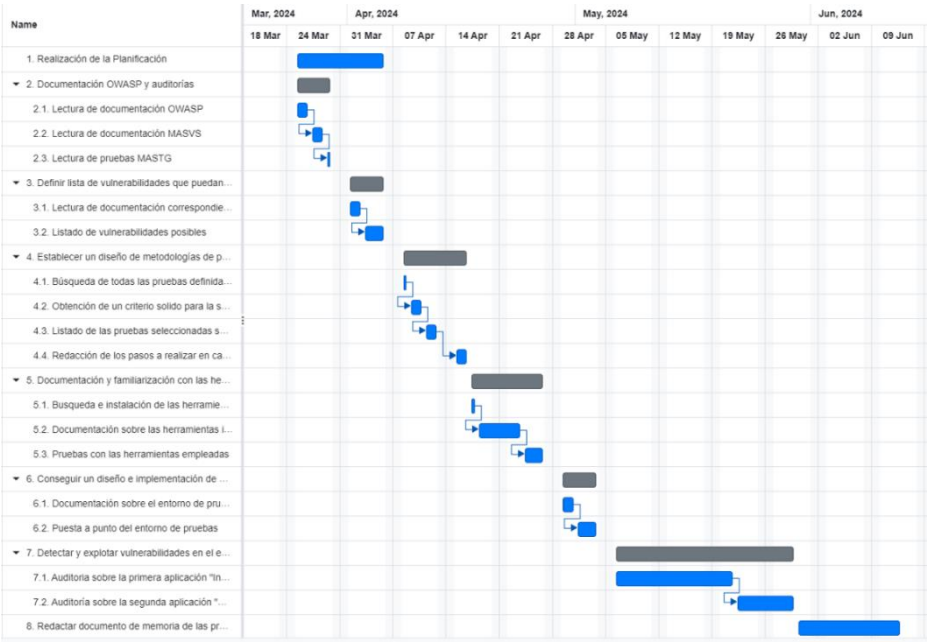


Ilustración 2: Diagrama de Gantt con distribución del trabajo

[5] Herramienta de gestión de proyectos que muestra visualmente el cronograma de actividades o tareas a lo largo de un período de tiempo determinado.

La siguiente tabla muestra de manera más visual cada uno de los **hitos del proyecto**, y además presenta las fechas de finalización de cada uno de ellos, tal y como se representó en el anterior Diagrama de Gantt [6].

HITO	FECHA
Documentación sobre normativas de datos y metodología OWASP para auditorías	29/03/2024
Establecer lista de vulnerabilidades relativas a la privacidad para apps móviles	05/04/2024
Establecer un diseño de metodologías de pruebas de OWASP	16/04/2024
Documentación y familiarización con las herramientas empleadas para la auditoría	26/04/2024
Conseguir un diseño e implementación de un entorno de pruebas controlado	03/05/2024
Detectar y explotar vulnerabilidades en el entorno de pruebas	29/05/2024
Redactar documento de memoria de las pruebas	12/06/2024
Fin del proyecto	05/07/2024

Tabla 2.2: Calendario de hitos del proyecto

La estimación temporal realizada sobre las tareas y subtareas del proyecto, que se muestra en la figura y tabla anteriores, establecerá el ritmo de trabajo que deberá seguirse para la correcta realización de este proyecto. Aun así, y como ya se ha comentado antes, existen entre el último hito y el final del proyecto tres semanas libres, que podrán utilizarse en caso de inconvenientes de cualquier tipo.

2.4. Riesgos

En esta última sección, se tratarán de **definir todos los riesgos** que puedan aparecer durante el desarrollo del proyecto, provocando daños y retrasos en la planificación del mismo. Una correcta **estimación de riesgos** puede ser un factor clave a la hora de que el trabajo realizado en un proyecto sea un éxito o no, por lo tanto, nos encontramos ante una de las secciones más importantes de este apartado de planificación.

No solo valdrá con identificar los posibles riesgos, sino que también será necesario estimar el daño que estos pueden provocar sobre el proyecto en caso de aparecer, y finalmente, proponer un plan de mitigación sobre los mismos.

Para estimar los daños que la aparición de cada riesgo puede provocar, se hará uso de la siguiente **matriz de riesgos**[6], obtenida del libro *Software Project Management 5ª edición* [10]. Esta matiza muestra la peligrosidad de los riesgos que pueden aparecer en un proyecto escalando su probabilidad e impacto en valores del 1 al 4 (bajo, moderado, significativo y alto).

[6] Herramienta para identificar, evaluar y priorizar riesgos en un proyecto según su probabilidad e impacto.

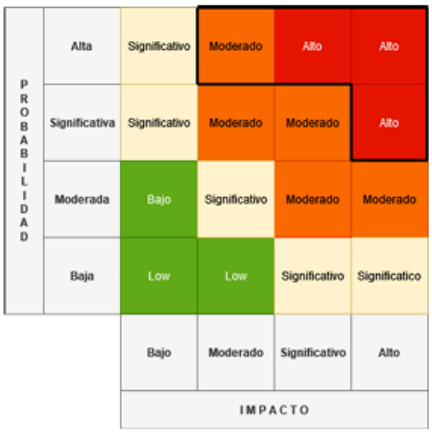


Ilustración 3: Matriz de riesgos

Se le dará mayor prioridad al riesgo cuando más cercano esté de la esquina superior derecha, es decir, cuanto mayor probabilidad tenga de suceder y mayor impacto produzca.

En la siguiente tabla, se presentan los **diez riesgos** que he considerado más probable que aparezcan, y paralelamente, se ha estimado la **probabilidad** de que dicho riesgo ocurra y el **impacto** que provocaría sobre este proyecto en caso de aparecer, obteniendo como resultado el grado de **daño** según la matriz de riesgos anterior.

Número	Riesgo	Probabilidad	Impacto	Daño
1	Estimación de trabajo inferior a la real	4	3	Alto
2	Incumplimiento de horarios de trabajo	2	1	Bajo
3	Daños en el ordenador personal	1	4	Significativo
4	Incumplimiento de plazos	1	4	Significativo
5	Bloqueo al llevar a la práctica los objetivos	2	3	Moderado
6	Modificación de alguno de los objetivos	1	2	Bajo
7	No montar de forma adecuada un entorno de pruebas	1	4	Moderado
8	No detectar todas las vulnerabilidades	3	3	Moderado
9	Fecha de finalización posterior a la prevista	3	4	Alto
10	No estar disponible a lo largo del proyecto	1	4	Significativo

Tabla 2.3: Análisis de los riesgos del proyecto

Con todos los riesgos identificados, tendremos cuatro posibles opciones a modo de actuación: **aceptar** su aparición y proponer medidas en caso de que ocurran, **evitar** que ocurran, **reducir** o **mitigar** su impacto en caso de que ocurran, o **transferirlos**. Como es imposible transferir ningún

riesgo entre hitos debido a que todos ellos forman parte de un mismo **camino crítico**[7], se optará por tratar de evitarlos en primer lugar, y si no es posible, de reducir su probabilidad y minimizar el impacto que puedan acarrear. Como última opción, también se podrá aceptar el riesgo asumiendo un retraso en el proyecto.

En la siguiente tabla se detallan las soluciones propuestas ante los riesgos definidos anteriormente:

Numero	Acción para mitigación de riesgo
1	Consultar a los tutores y avanzar en otras tareas
2	Planificar el proyecto de modo que haya algunas semanas libres
3	Copias de seguridad del trabajo del proyecto y buena documentación de lo realizado hasta el momento
4	Horas extras de trabajo en fines de semana
5	Pedir consejo a los tutores
6	Asegurarse de fijar bien los objetivos al inicio del proyecto y de tener flexibilidad a la hora de retornar a ciertos puntos del proyecto
7	Seguir correctamente la documentación para el montaje correcto del entorno de pruebas
8	Consultar con tutores por las vulnerabilidades encontradas
9	Realizar la entrega y defensa en una fecha posterior a la prevista
10	Emplear más horas de las establecidas cuando se prevean problemas con la disponibilidad

Tabla 2.4: Plan de actuación ante aparición de riesgos previstos

Por último, se muestra una descripción del proceso de actuación que se espera tener en caso de aparición de algunos de los riesgos anteriores.

- Debido a la **poca experiencia** en las acciones a realizar sobre el entorno de pruebas, existe la posibilidad de que el tiempo estimado esté alejado de la realidad. Para evitar esto, se acelerarán otras subtareas del hito y, si es necesario, se emplearán horas extras de trabajo para mantener la fecha de cumplimiento del objetivo.

- **Modificar los objetivos** en las primeras etapas no supondría muchos problemas, pero hacerlo casi al final del proyecto podría implicar repetir todo el desarrollo. Por ello, es crucial definir bien los objetivos desde el principio

- Si el ordenador personal sufre **daños**, afectaría considerablemente al desarrollo del proyecto debido a la falta de una máquina de reserva. Aunque es poco probable, en caso de producirse se mitigará haciendo copias de seguridad en dispositivos externos y en la nube.

[7] Secuencia de tareas que determina la duración mínima de un proyecto.

- Paralelas a la realización del proyecto, existen **otras actividades** universitarias y extra universitarias que el estudiante debe realizar al mismo tiempo, lo que puede provocar que en ocasiones se deban emplear las horas previstas de trabajo en dichas actividades. Se buscará que el desarrollo del proyecto no se vea afectado realizando horas de trabajo fuera del horario establecido para compensar la situación.

- Si se prevé que se ha **de incumplir la fecha** de un hito, ya sea por mala estimación o por carga de trabajo externa, se emplearán horas extras de trabajo para cumplir con el plazo definido, o se reajustará los hitos posteriores tomando días de las semanas "cochón".

- Ante posibles **bloqueos** propios de la inexperiencia a la hora de llevar a cabo proyectos de este tipo, se pedirá inmediatamente ayuda a los tutores.

.

- Si por cualquier **circunstancia externa** el estudiante se ve incapaz de poder avanzar en el proyecto, se hará uso de las semanas "colchón" que se han dejado para el final del proyecto.

Capítulo 3

Estado del Arte

Este capítulo se trata de uno de los más importantes del documento, puesto que se encargará de aportar el **contexto** necesario para que se entienda correctamente qué se está haciendo y qué se presentará en cada una de las pruebas realizadas en las auditorías. Además, se reflejará de forma técnica todo el estudio del contexto de las **normativas de privacidad** y de las **metodologías** relacionadas con ellas, que he tenido que hacer para la correcta realización del proyecto.

Comenzaremos realizando un análisis sobre los requisitos de privacidad que deben cumplir las aplicaciones móviles definidos por las normativas vigentes en nuestro país. Además, se proporcionará información importante respecto al manejo de datos y los diferentes tipos de los mismos. A continuación, se definirá brevemente el concepto de auditoría móvil, para que finalmente, se complete dicho concepto presentando a OWASP y sus diferentes metodologías.

3.1. Normativa de Protección de Datos: RGPD y LOPDPDG2018

Actualmente, la protección de datos está regulada por dos leyes principales: el Reglamento General de Protección de Datos (**RGPD**) a nivel europeo y la Ley Orgánica 3/2018 de Protección de Datos Personales y Garantía de Derechos Digitales (**LOPDPDG2018**) a nivel nacional en España. La LOPDPDG2018 adapta la legislación española a la normativa europea [7].

Analizando el RGPD, este reglamento define qué son los **datos personales**, las circunstancias en las que es legítimo tratarlos y una serie de principios técnicos para su protección. Estos tres elementos son esenciales para este proyecto, cuyo objetivo es identificar y explotar mediante auditorías de seguridad, vulnerabilidades que puedan poner en riesgo el derecho a la protección de datos personales. En particular, se deben establecer:

- **Qué datos de una persona se consideran personales** para poder identificarlos cuando se explota una vulnerabilidad.
- **Cómo deben protegerse los datos personales**, proponiendo medidas de seguridad que prevengan riesgos para la protección de datos sensibles.
- **Qué datos puede manejar una aplicación móvil**, para asegurar el uso adecuado de los datos recogidos y almacenados.

En las siguientes secciones se detallan estos aspectos.

3.1.1. Datos de Carácter Personal

Según el RGPD, los datos personales son aquellos que permiten **identificar** directa o indirectamente a una persona [8]. Ejemplos de datos personales incluyen:

- DNI, NUSS y pasaporte.
- Nombre completo (nombre y apellidos).
- Dirección física, ubicación y dirección IP.
- Número de teléfono y correo electrónico.
- Número de cuenta bancaria y tarjetas de crédito o débito.
- Datos biométricos, como imágenes fotográficas o huellas dactilares.
- Firmas y certificados electrónicos.
- Datos médicos.
- Características físicas, fisiológicas, genéticas, psíquicas, económicas, culturales o sociales.
- Seudónimos.
- Matrícula de coche u otros vehículos.

3.1.2. Principios de Protección de Datos

El RGPD establece una serie de **principios** fundamentales para evitar la exposición de datos sensibles, que se consideran indispensables a la hora de desarrollar una aplicación que vaya a recoger datos personales y que busque protegerlos. Estos principios son:

- **Minimización de datos**: Solo se deben recopilar los datos estrictamente necesarios.

- **Limitación de almacenamiento**: Los datos deben almacenarse únicamente durante el tiempo necesario.

- **Integridad y confidencialidad**: Los datos almacenados deben protegerse adecuadamente para garantizar su seguridad, integridad y confidencialidad mediante técnicas como cifrado y anonimización. Si los datos están cifrados, serían inútiles para un atacante que comprometa la seguridad, evitando sanciones por haber aplicado medidas de protección.

3.1.3. Casos en los que se Pueden Procesar Datos

El RGPD define los casos en los que es legal procesar datos personales [9], permitiendo la recopilación, almacenamiento y procesamiento de datos sólo en situaciones específicas. Si una aplicación móvil utiliza datos fuera de estos casos, estaría violando el derecho a la protección de datos. Por lo tanto, este aspecto también debe considerarse durante la auditoría móvil.

3.1.4. Diferencias entre tipos de datos

En el ámbito de las aplicaciones móviles, es crucial entender las diferencias entre los tipos de datos que pueden manejarse. Estos datos se categorizan principalmente en datos **personales**, datos

confidenciales y datos **sensibles**, cada uno con características y requisitos de protección específicos. A continuación, se explica la diferencia entre estos tipos de datos.

- **Datos Personales vs Datos Confidenciales**

  - Datos Personales

    Los datos personales son aquellos que permiten **identificar** directa o indirectamente a una persona física. Según el RGPD, ejemplos de datos personales incluyen el nombre, la dirección, el número de teléfono, la dirección IP, datos biométricos, y otros mencionados anteriormente. Estos datos se utilizan para identificar o contactar a una persona específica.

  - Datos Confidenciales

    Los datos confidenciales, por otro lado, son aquellos que, independientemente de su capacidad para identificar a una persona, requieren una protección especial debido a su naturaleza sensible o su importancia estratégica para una organización. Incluyen secretos comerciales, propiedad intelectual, estrategias empresariales, acuerdos legales, y cualquier información que, si se divulga sin autorización, podría perjudicar a la organización o individuos asociados. Mientras que todos los datos personales pueden ser considerados confidenciales en contextos específicos, no todos los datos confidenciales son necesariamente datos personales.

- **Datos Personales vs Datos Sensibles Datos Sensible**

  Los datos sensibles son un subconjunto de los datos personales, que requieren un nivel de protección adicional debido a la naturaleza **delicada**, fruto de la información que contienen. Según el RGPD, estos incluyen datos que revelen el origen étnico o racial, opiniones políticas, creencias religiosas o filosóficas, afiliación sindical, datos genéticos, datos biométricos destinados a identificar de manera unívoca a una persona física, datos relativos a la salud, y datos relativos a la vida sexual u orientación sexual de una persona. [10]

  - Diferencias Clave:

    - **Naturaleza y Riesgo**: los datos personales pueden incluir información básica de contacto y de identificación, mientras que los datos sensibles abarcan información que puede llevar a discriminación, perjuicio, o invasión significativa de la privacidad si se divulga.

    - **Nivel de Protección**: los datos sensibles requieren medidas de seguridad más estrictas, tales como niveles más altos de cifrado, acceso limitado estrictamente controlado, y procedimientos adicionales de anonimización para reducir el riesgo de exposición indebida.

    - **Regulación y Cumplimiento**: la legislación, como el RGPD, establece requisitos adicionales para el manejo de datos sensibles, incluyendo la necesidad de obtener el consentimiento explícito de los individuos para el procesamiento de estos datos, y la implementación de evaluaciones de impacto en la privacidad antes de su procesamiento [11].

3.2. Auditorías móviles

Las auditorías móviles son procesos diseñados para **evaluar la seguridad** y funcionalidad de aplicaciones móviles. Estas auditorías son esenciales para **identificar vulnerabilidades** y asegurar que las aplicaciones móviles cumplan con los estándares de seguridad establecidos [12]. En un panorama en el que el uso de dispositivos móviles sigue creciendo de manera exponencial, la seguridad móvil se ha convertido en una prioridad crítica para desarrolladores y empresas por igual.

El objetivo principal de una auditoría móvil es detectar y mitigar riesgos que podrían comprometer la seguridad de los datos y la integridad del sistema. Esto incluye la identificación de vulnerabilidades en el código fuente, la configuración de la aplicación, y las interacciones de la aplicación con otros sistemas. Además, las auditorías móviles examinan la implementación de **medidas de seguridad** como la autenticación, la autorización, la criptografía y la gestión de sesiones.

La motivación detrás de las auditorías móviles proviene de la necesidad de proteger información sensible que las aplicaciones manejan, como datos personales, financieros y corporativos, ayudando a usuarios y organizaciones a adelantarse a estas amenazas, implementando prácticas de seguridad robustas desde las primeras etapas del desarrollo.

Debido a la creciente cantidad de regulaciones estrictas en cuanto a la protección de datos y la privacidad, como *el Reglamento General de Protección de Datos* (**GDPR**) [13] en Europa o la **LOPDGDD2018** en España, la realización de auditorías móviles periódicas asegura que las aplicaciones no solo sean seguras, sino que también cumplan con las normativas vigentes, evitando así posibles sanciones y daños a la reputación.

Para llevar a cabo una auditoría, se hace uso de una combinación de **herramientas automatizadas** y **técnicas manuales**. Las herramientas automatizadas pueden escanear el código y la infraestructura de la aplicación para identificar vulnerabilidades comunes, mientras que los auditores humanos realizan análisis más detallados y pruebas personalizadas que las herramientas automatizadas no pueden abordar. Esta combinación de enfoques asegura una evaluación completa y detallada de la seguridad de la aplicación.

Finalmente, las auditorías no son un proceso único, sino un **esfuerzo continuo**. La seguridad es un objetivo en constante cambio debido a la evolución constante de las amenazas y las tecnologías. Por lo tanto, deberán realizarse regularmente, especialmente cuando se realizan cambios significativos en la aplicación o se lanzan nuevas versiones. De esta manera, las organizaciones pueden mantener una postura de seguridad sólida y adaptarse rápidamente a las nuevas amenazas.

3.3. Metodología OWASP

La Metodología **OWASP** (Open Web Application Security Project) es un marco reconocido internacionalmente para mejorar la seguridad de las aplicaciones web y móviles. OWASP es una organización sin fines de lucro que proporciona herramientas, documentación y estándares de seguridad de aplicaciones abiertas a la comunidad global. Fundada en 2001, y que tiene como objetivo ayudar a las organizaciones a desarrollar, adquirir y mantener aplicaciones que puedan ser confiables. Uno de los aportes más significativos de OWASP es su lista **OWASP Top 10**, que identifica y describe las principales vulnerabilidades de seguridad en aplicaciones web y móviles. Esta lista es actualizada periódicamente para reflejar las amenazas emergentes y proporcionar a los desarrolladores y profesionales de seguridad una guía práctica sobre los riesgos más críticos a mitigar. El OWASP Top 10 es ampliamente utilizado como referencia en la industria y es considerado un estándar de facto para la seguridad de aplicaciones.

Sin embargo, la Metodología OWASP no se limita solo al OWASP Top 10. También incluye proyectos y guías como *el OWASP Application Security Verification Standard* (**ASVS**) [14], que proporciona un marco detallado para verificar la seguridad de las aplicaciones en diferentes **niveles de rigor**. ASVS cubre aspectos como la autenticación, autorización, gestión de sesiones[8], validación de entradas[9] y manejo de errores, entre otros. Es una herramienta esencial para las auditorías de seguridad y la certificación de aplicaciones seguras.

Otro recurso importante en proyectos específicos para la seguridad móvil, es el *OWASP Mobile Security Project*. Este proyecto incluye el *Mobile Security Testing Guide* (**MSTG**) [15] y el *Mobile Application Security Verification Standard* (**MASVS**) [16], que, en sus diferentes versiones, proporcionan directrices y estándares específicos para evaluar y mejorar la seguridad de las aplicaciones móviles. Estos recursos son vitales para las organizaciones que desarrollan aplicaciones móviles y buscan proteger sus datos y usuarios de amenazas específicas del entorno móvil.

La **comunidad** OWASP es una de las mayores fortalezas de la organización, ya que está compuesta por miles de miembros, incluyendo profesionales de seguridad, desarrolladores, investigadores y entusiastas de todo el mundo. Esta comunidad colabora en la creación y actualización de proyectos OWASP, compartiendo conocimientos y mejores prácticas. La participación en eventos y conferencias de OWASP también proporciona oportunidades valiosas para el networking[10] y el aprendizaje continuo.

En resumen, la Metodología OWASP proporciona un **marco sólido** y recursos extensivos para mejorar la seguridad de las aplicaciones. Su enfoque colaborativo y abierto permite a las organizaciones de todos los tamaños acceder a herramientas y conocimientos de seguridad avanzados, facilitando la creación de aplicaciones más seguras y confiables. Adoptar las prácticas y estándares de OWASP es un paso fundamental para cualquier organización comprometida con la seguridad de sus aplicaciones y datos.

[8] Proceso de controlar y mantener información sobre las interacciones de un usuario con un sistema durante una sesión específica.

[9] Proceso de verificar que los datos ingresados en un sistema son correctos, completos y seguros.

[10] Práctica de establecer y mantener relaciones profesionales y personales para intercambiar información y recursos.

3.4. OWASP MOBILE TOP 10

El OWASP Mobile Top 10 es una lista que identifica y describe las **principales vulnerabilidades** de seguridad en aplicaciones móviles [17]. Esta lista proporciona a los desarrolladores y profesionales de seguridad una guía práctica sobre los riesgos más críticos que deben ser mitigados en el desarrollo de aplicaciones móviles. A continuación, se presentan los detalles del OWASP Mobile Top 10 más reciente:

- **M1: Uso Incorrecto de la Plataforma**
  - Descripción:

    Ocurre cuando los desarrolladores no siguen las guías y prácticas recomendadas por la plataforma móvil, lo que puede resultar en vulnerabilidades.
  - Ejemplos de Ataques:
    - Uso incorrecto de permisos
    - Fallos en la implementación de TouchID/FaceID[11]
  - Métodos de Prevención:
    - Adherirse a las guías de desarrollo de la plataforma
    - Realizar revisiones de seguridad del código
- **M2: Almacenamiento Inseguro de Datos**
  - Descripción:

    Las aplicaciones móviles a menudo almacenan datos de forma insegura en el dispositivo, exponiendo información sensible.
  - Ejemplos de Ataques:
    - Extracción de datos de almacenamiento interno
    - Acceso a datos a través de copias de seguridad no cifradas
  - Métodos de Prevención:
    - Cifrar todos los datos sensibles almacenados
    - Utilizar APIs de almacenamiento seguro proporcionadas por la plataforma
- **M3: Comunicación Insegura**
  - Descripción:

    Las vulnerabilidades en la comunicación insegura ocurren cuando las aplicaciones no utilizan protocolos de comunicación seguros, exponiendo datos durante su transmisión.
  - Ejemplos de Ataques:

[11] Tecnologías de autenticación biométrica utilizadas en dispositivos móviles para desbloquear y acceder a funciones mediante la huella dactilar o reconocimiento facial del usuario.

- Ataques Man-in-the-Middle (MitM)[12]
- Interceptación de datos no cifrados

- Métodos de Prevención:

  - Usar TLS para todas las comunicaciones
  - Validar certificados SSL/TLS correctamente

- **M4: Autenticación Insegura**

  - Descripción:

    Las aplicaciones que manejan la autenticación de manera inadecuada permiten a los atacantes comprometer cuentas y datos de usuarios.

  - Ejemplos de Ataques:

    - Credential stuffing[13]
    - Uso de contraseñas débiles o predecibles

  - Métodos de Prevención:

    - Implementar autenticación multifactor (MFA)
    - Enforzar políticas de contraseñas fuertes

- **M5: Criptografía Insegura**

  - Descripción:

    Las vulnerabilidades de criptografía insegura se producen cuando las aplicaciones utilizan algoritmos de cifrado débiles o implementan incorrectamente la criptografía

  - Ejemplos de Ataques:

    - Desencriptación de datos sensibles
    - Uso de algoritmos de cifrado obsoletos

  - Métodos de Prevención:

    - Usar bibliotecas de criptografía aprobadas
    - Seguir las mejores prácticas para la implementación de criptografía

- **M6: Autorización Inadecuada**

  - Descripción:

    Las fallas en la autorización permiten a los atacantes realizar acciones no autorizadas o acceder a datos que no deberían ser accesibles.

  - Ejemplos de Ataques:

    - Escalada de privilegios
    - Acceso a funciones restringidas

[12] Ataques en los que un tercero intercepta y potencialmente modifica la comunicación entre dos partes, sin que ninguna de ellas lo sepa, comprometiendo así la seguridad y la integridad de la información transmitida.
[13] Uso automatizado de credenciales filtradas para intentar acceder a cuentas en diferentes servicios en línea.

- Métodos de Prevención:
  - Implementar controles de acceso basados en roles (RBAC)[14]
  - Verificar la autorización en el servidor

- **M7: Calidad del Código Cliente**
  - Descripción:

    La falta de control sobre la calidad del código puede llevar a vulnerabilidades explotables en el cliente móvil.
  - Ejemplos de Ataques:
    - Inyección de código
    - Ejecución de scripts maliciosos
  - Métodos de Prevención:
    - Revisar y auditar el código fuente regularmente
    - Utilizar herramientas de análisis estático y dinámico
- **M8: Manipulación del Código**
  - Descripción:

    Las aplicaciones pueden ser vulnerables a la manipulación del código si no se implementan protecciones adecuadas contra la ingeniería inversa.
  - Ejemplos de Ataques:
    - Modificación del código de la aplicación
    - Inyección de código malicioso
  - Métodos de Prevención:
    - Usar técnicas de ofuscación del código
    - Implementar verificaciones de integridad del código
- **M9: Funcionalidad Insegura**
  - Descripción:

    Las funcionalidades inseguras pueden introducir vulnerabilidades si no se controlan adecuadamente las funcionalidades expuestas al usuario.
  - Ejemplos de Ataques:
    - Abuso de funciones de depuración
    - Uso de APIs expuestas sin control

[14] Modelo de control de acceso que asigna permisos a usuarios según sus roles dentro de una organización, simplificando la gestión de derechos de acceso.

- Métodos de Prevención:
    - Limitar las funcionalidades expuestas en las versiones de producción
    - Realizar pruebas de seguridad exhaustivas

- **M10: Falta de Procedimientos de Respuesta**
    - Descripción:

      La falta de procedimientos de respuesta adecuados puede llevar a una respuesta lenta o ineficaz ante incidentes de seguridad.
    - Ejemplos de Ataques:
        - Incapacidad para detectar y responder a intrusiones
        - Falta de planes de contingencia
    - Métodos de Prevención:
        - Implementar un plan de respuesta a incidentes
        - Realizar simulacros regulares de respuesta a incidentes

3.5. Estándar MASVS y guía MSTG

El *Mobile Application Security Verification Standard* (**MASVS**) [16] y *la Mobile Security Testing Guide* (**MSTG**) [15] son dos componentes clave del proyecto de seguridad móvil de OWASP. Estos recursos proporcionan **directrices** y **estándares** específicos para evaluar y mejorar la seguridad de las aplicaciones móviles, y son utilizados ampliamente por desarrolladores, auditores de seguridad y organizaciones para asegurar que sus aplicaciones móviles sean seguras y confiables.

El **MASVS** es un estándar que establece un conjunto de requisitos de seguridad que las aplicaciones móviles deben cumplir para ser consideradas seguras. Este estándar cuenta con diferentes versiones aceptadas por toda la comunidad de la ciberseguridad. Actualmente en concreto se están utilizando sus versiones 1.5 y 2. La versión 1.5 aúna diversas pruebas en función de los campos de la aplicación sobre los que busca vulnerabilidades (almacenamiento, criptografía, red…), y dentro de cada campo describe diversas pruebas. Por otro lado, la versión 2 también emplea esa división por campos, pero a su vez divide cada uno de estos campos en varios niveles de verificación, que van desde los requisitos de seguridad básicos hasta los avanzados, y que juntan las pruebas que se definían en la versión anterior. El nivel 1 (**MASVS-L1**) se enfoca en las prácticas de seguridad que todas las aplicaciones móviles deberían implementar, mientras que el nivel 2 (**MASVS-L2**) se destina a aplicaciones que manejan datos altamente sensibles y requieren una mayor seguridad. Adicionalmente, existe el nivel **MASVS-R** para aplicaciones que requieren requisitos específicos de resiliencia, como las que necesitan protegerse contra ataques de ingeniería inversa y manipulación.

Los **requisitos** de MASVS cubren una amplia gama de áreas, incluyendo la arquitectura y diseño de la aplicación, el manejo de datos sensibles, la autenticación y autorización, la gestión de sesiones, la criptografía, la interacción con la plataforma móvil y las comunicaciones de red. Cada requisito está acompañado de una explicación detallada y ejemplos prácticos, lo que facilita su implementación y verificación.

Por otro lado, la Mobile Security Testing Guide (**MSTG**) complementa el MASVS proporcionando una **guía** práctica y detallada para realizar pruebas de seguridad en aplicaciones móviles. Esta guía cubre una variedad de técnicas de prueba, desde la revisión de código fuente y el análisis estático, hasta el análisis dinámico y la evaluación de la interacción de la aplicación con la plataforma móvil. La guía también incluye ejemplos de ataques comunes y cómo prevenirlos, lo que la convierte en un recurso muy valioso para los evaluadores de seguridad.

La MSTG se organiza en diferentes **secciones** que corresponden a las diversas fases del ciclo de vida de la seguridad móvil. Estas incluyen la configuración del entorno de prueba, la recopilación de información, el análisis estático y dinámico, y la evaluación de las interacciones de la aplicación con la plataforma móvil. Cada sección ofrece metodologías específicas, herramientas recomendadas y ejemplos detallados, proporcionando una **hoja de ruta** clara para realizar pruebas de seguridad exhaustivas y efectivas. Una característica a destacar es su enfoque en las pruebas prácticas y la participación activa de la comunidad. La guía se actualiza regularmente para incorporar nuevas técnicas y herramientas.

Capítulo 4

Metodología

Para establecer un marco **claro** y **replicable** en la realización de las pruebas de privacidad de mi auditoría, he adoptado la metodología de trabajo propuesta por OWASP en su estándar para auditorías móviles **MASVS**, más específicamente en su versión v1.5, ya que proporciona una estructura bien definida, paso por paso y ampliamente aceptada en la industria para identificar y mitigar vulnerabilidades en aplicaciones móviles.

Haré uso de la **versión 1.5** del estándar MASVS, debido a que considero que será más fácil seleccionar entre una serie de pruebas separadas en campos, sobre la separación por niveles que se puede ver en la versión v2.

Para la implementación de las pruebas que propone este estándar, se ha procedido a seguir los pasos que MSTG define de manera canónica. Esto implica comenzar con una **fase de planificación** detallada, donde se analizan los objetivos de cada una de las pruebas y se busca alinearlos con los objetivos específicos del proyecto. Esta fase es crucial para garantizar que las pruebas sean efectivas y eficientes, abordando tanto las necesidades del proyecto como los requisitos de seguridad relevantes.

A continuación, se procederá con la fase de **recopilación de información**, donde se identifican y enumeran los posibles vectores de ataque[15] y la información sobre las tecnologías utilizadas, todo ello con el objetivo de orientar adecuadamente las pruebas y de poder familiarizarme correctamente con las herramientas necesarias para la prueba.

Finalmente, se llevan a cabo las **pruebas** reales de búsqueda de vulnerabilidades relativas a la privacidad siguiendo las directrices detalladas por MSTG. El proceso de cada prueba comienza siempre con el **análisis estático** del código fuente o del manifiesto de la aplicación[16], para comprender las bases del funcionamiento de la misma. Generalmente se hará uso de herramientas de inspección de código y se buscarán las clases, funciones o palabras clave que se definen en los estándares de la guía. A continuación, se llevará a cabo el **análisis dinámico** con la aplicación en ejecución. Cada prueba, como ya se ha dicho, sigue al pie de la letra los pasos indicados por MSTG, y se ha documentado meticulosamente, describiendo los pasos seguidos, las herramientas utilizadas y los resultados obtenidos, asegurando así que cualquier persona que siga el mismo enfoque pueda replicarlas con precisión y obtener resultados comparables.

[15] Forma específica en que un atacante intenta explotar vulnerabilidades para comprometer la seguridad de la aplicación o el dispositivo.

[16] Archivo que contiene información esencial sobre la aplicación, como su nombre, versión, componentes y permisos necesarios para funcionar correctamente en un dispositivo.

CAPITULO 4. METODOLOGIA

El uso de la metodología OWASP MOBILE se da debido a su enfoque práctico y su amplia **aceptación en la comunidad** de seguridad informática. Esta metodología no solo proporciona una guía clara para realizar pruebas de seguridad efectivas, sino que también facilita la comunicación de los hallazgos de manera estandarizada y comprensible para todos los interesados. Además, al seguir esta metodología, puedo asegurar que mis resultados sean **comparables** y **verificables**, cumpliendo así con los estándares de calidad y rigor necesarios para un TFG en el ámbito de la seguridad informática.

Capítulo 5

Diseño

En el presente capítulo se definirá la **batería de pruebas** a la que se someterán las aplicaciones auditadas, así como los pasos a seguir en cada una de ellas para un correcto análisis de sus vulnerabilidades buscadas. Este proceso de selección es uno de los puntos más importantes del proyecto, puesto que se encuentra directamente relacionado con los objetivos definidos al comienzo del mismo, y su correcta realización medirá el éxito que tengamos a la hora de definir la privacidad de las aplicaciones auditadas.

Este proceso de diseño de auditoría supondrá, por tanto, revisar todas las pruebas de la guía MSTG que propone la metodología **MASVS v1.5** para la realización de auditorías móviles, y seleccionar las que consideremos que nos ayudarán más a buscar vulnerabilidades relacionadas con la privacidad de los usuarios.

Será, por lo tanto, vital establecer cuáles son las vulnerabilidades más importantes que puedan aparecer en una aplicación móvil, para llevar a cabo la selección de las pruebas que más se centren en encontrarlas.

Finalmente, y una vez realizada la selección de las pruebas, se presentarán cada uno de los pasos establecidos por MSTG para la correcta realización de las mismas.

5.1. Vulnerabilidades principales para la auditoría

Como ya se ha establecido a lo largo del presente documento, las auditorías a realizar tendrán como objetivo la búsqueda de las vulnerabilidades que pongan en peligro la privacidad de los usuarios que hacen uso de las apps. El hecho de que una aplicación proteja la privacidad de los usuarios conlleva asegurar que se protege su información personal y sensible, y, además, que se mantienen esos datos de forma segura previniendo su robo. La confianza de los usuarios en una aplicación y en la marca que la respalda se ve significativamente afectada por la percepción de cómo se maneja su privacidad.

Para asegurar una correcta gestión de la privacidad hay dos puntos principales sobre los que los **desarrolladores** de las aplicaciones deben poner especial atención [18]:

- **Transparencia**: las aplicaciones deben informar claramente a los usuarios sobre qué datos recopilan, cómo se utilizan y con quién se comparten.

- **Seguridad de los datos**: implementación de medidas de seguridad robustas para proteger los datos contra accesos no autorizados, tanto a nivel de almacenamiento como de transmisión.

La definición de una **política de privacidad** [19] al comienzo de la actividad de la aplicación se ha convertido en la práctica más común a la hora de informar a los usuarios sobre las razones de la recopilación de algunos de sus datos. Las políticas de privacidad son documentos fundamentales que describen detalladamente cómo una aplicación recopila, utiliza, almacena y comparte la información personal de los usuarios. Según el RGPD, una política de privacidad debe incluir varios elementos esenciales para garantizar la transparencia y la protección de datos:

- **Identidad y datos de contacto del responsable del tratamiento**: debe indicarse quién es el responsable de la recopilación y el tratamiento de los datos personales, proporcionando su identidad y cómo puede ser contactado.
- **Finalidades del tratamiento de los datos**: la política debe explicar claramente las razones por las cuales se recopilan los datos personales y para qué serán utilizados. Esto incluye especificar si los datos serán utilizados para mejorar la experiencia del usuario, para fines de marketing, para análisis de datos, entre otros.
- **Base jurídica del tratamiento**: es necesario indicar la base legal que justifica el tratamiento de los datos personales, ya sea el consentimiento del usuario, la necesidad de ejecutar un contrato, el cumplimiento de una obligación legal, la protección de intereses vitales, el cumplimiento de una misión de interés público o el interés legítimo del responsable del tratamiento.
- **Destinatarios de los datos**: la política debe identificar a terceros que pueden tener acceso a los datos personales, como proveedores de servicios, socios comerciales o autoridades públicas, y en qué circunstancias se compartirán los datos.
- **Transferencias internacionales de datos**: si los datos personales serán transferidos a países fuera del Espacio Económico Europeo[17], la política debe informar sobre las medidas de protección que se implementarán para asegurar un nivel adecuado de protección de los datos.
- **Período de conservación de los datos**: se debe especificar cuánto tiempo se conservarán los datos personales antes de ser eliminados o anonimizados. Este período debe estar justificado según las finalidades del tratamiento.
- **Derechos de los usuarios**: el RGPD otorga a los usuarios varios derechos sobre sus datos personales, incluyendo el derecho a acceder, rectificar, borrar, limitar el tratamiento, oponerse al tratamiento y a la portabilidad de los datos. La política de privacidad debe explicar cómo los usuarios pueden ejercer estos derechos y a quién deben dirigirse para hacerlo.
- **Medidas de seguridad**: es crucial detallar las medidas técnicas y organizativas que se implementan para proteger los datos personales contra accesos no autorizados, pérdida, destrucción o alteración.

Las políticas de privacidad son esenciales no solo para cumplir con las normativas legales como el RGPD, sino también para construir y mantener la confianza de los usuarios. Una política clara y detallada demuestra el compromiso de la aplicación con la **protección de los datos** personales y la transparencia en sus prácticas de tratamiento de datos.

Definidos todos los compromisos que desarrolladores y aplicaciones deben cumplimentar para asegurar el correcto manejo de los datos sensibles de los usuarios, será momento de conocer las principales **vulnerabilidades** que pueden provocar el incumplimiento de los mismos.

Una vulnerabilidad en el contexto de estudio en el que nos encontramos, se tratará de un punto mal configurado de la aplicación en el que los datos personales o sensibles de un usuario puedan

[17] Área que comprende los países de la Unión Europea junto con Islandia, Liechtenstein y Noruega, donde se aplican principios de libre circulación de bienes, servicios, personas y capitales.

verse comprometidos. Las principales vulnerabilidades que pueden afectar a la privacidad de los datos de una aplicación móvil serán las siguientes: [20]

- **Intercepción de datos**: sin las adecuadas medidas de cifrado, los datos transmitidos entre el dispositivo del usuario y los servidores de la app pueden ser interceptados por terceros.
- **Almacenamiento inseguro**: si los datos personales se almacenan sin la debida protección en los dispositivos o en los servidores, pueden ser accesibles para atacantes que busquen explotar esta información.
- **Permisos excesivos**: algunas aplicaciones solicitan permisos que no son necesarios para su funcionamiento, permitiendo el acceso a datos innecesarios para el funcionamiento de las mismas.
- **Fugas de datos**: las aplicaciones pueden filtrar datos de manera inadvertida debido a errores de programación o configuraciones incorrectas.

La búsqueda y explotación de estas vulnerabilidades será el **punto principal** en el que se deberá centrar la auditoría a diseñar en los siguientes apartados. Comprendiendo, además, las consecuencias que su aparición tendría sobre la integridad de los usuarios, se pueden entender de forma más clara la tremenda importancia de las auditorías móviles.

Como se ha visto a la hora de definir las vulnerabilidades anteriores, los ataques que se pueden realizar para comprometer la misma, engloban gran parte de los puntos vulnerables que define OWASP en su TOP 10. Un error en la configuración del almacenamiento de datos, la criptografía, la comunicación entre dispositivos o la autenticación/autorización puede comprometer fatalmente los datos sensibles de los usuarios, por eso es que, al centrar las auditorías de este proyecto en este ámbito, también podremos tener una **imagen general** de cómo está configurada la seguridad de las aplicaciones.

5.2. Presentación de pruebas MSTG

El presente apartado se trata de un punto clave para la consecución de los objetivos del proyecto. En él, procederemos a realizar la **presentación** de todas las pruebas que OWASP define en su estándar para auditar apps móviles MASVS. Cada uno de los puntos que componen este estándar serán pruebas de la guía MSTG en la que se describen los pasos a seguir para tratar de buscar las diferentes vulnerabilidades.

Al tratarse MASVS de una metodología enfocada en asegurar la **seguridad** de las aplicaciones móviles, podremos encontrar una gran multitud de pruebas, de entre las cuales no todas se relacionan directamente con la búsqueda de vulnerabilidades relativas a la privacidad. Es por ello que se deberá seguir un proceso de selección para quedarnos únicamente con las pruebas que más nos puedan acercar a la consecución de los objetivos del proyecto.

El proceso de selección que seguiremos, comenzará con la presentación de las pruebas junto con una breve explicación de cómo pueden afectar o no a la privacidad y que tan relacionadas están con las vulnerabilidades expuestas en el apartado anterior. A continuación, se procederá a calificarlas según un criterio propio que ayudará con la criba de las pruebas totales.

El criterio seleccionado, distingue en primer lugar, entre pruebas de seguridad[18] (**SG**) o pruebas de privacidad[19] (**PR**). Una auditoría móvil busca vulnerabilidades relacionadas con la **seguridad** de una aplicación cuando se centra en identificar y mitigar riesgos que puedan permitir accesos no autorizados, pérdida, alteración o destrucción de información almacenada o transmitida por la aplicación. Por otro lado, se buscarán vulnerabilidades relacionadas con la **privacidad** de los usuarios cuando se enfoca en asegurar que la información personal y sensible de los usuarios se recopile, utilice y comparta de manera transparente y conforme a las normativas de protección de datos, como el RGPD.

Adicionalmente, se añadirá una capa de complejidad más al criterio, filtrando entre el tipo de datos obtenibles en la prueba haciendo la diferenciación entre la capacidad o no de obtener datos personales y sensibles de los usuarios en la misma. Como ya se explicó en el apartado 3.1.4, existen tres tipos de datos presentes en las aplicaciones móviles: datos confidenciales, datos sensibles y datos personales. En las auditorías que se llevarán a cabo, nos centraremos en hacer uso de pruebas que busquen la existencia de los dos últimos tipos de datos. Definiré, pues, el siguiente código identificativo:

- **SG**: la prueba en cuestión está centrada principalmente en buscar vulnerabilidades relacionadas con la **seguridad de la aplicación**.
- **PR-N**: la prueba en cuestión está centrada principalmente en buscar vulnerabilidades relacionadas con la privacidad de la aplicación, pero no explotará filtraciones de datos personales y sensibles.
- **PR-S**: la prueba centrada en la privacidad busca vulnerabilidades con las que se pueden obtener **datos personales y sensibles** de los usuarios.

Prueba	Descripción	Utilidad
MASVS-STORAGE		
[MSTG-STORAGE-1, MSTG-STORAGE-2] Pruebas en almacenamiento local para buscar datos confidenciales [54]	La información sensible relacionada con el usuario debe encontrarse almacenada de forma segura dentro del contenedor de la aplicación[20] o el almacenamiento de credenciales del sistema. Es posible encontrar vulnerabilidades de **almacenamiento inseguro** de datos personales o sensibles.	**PR-S**
[MSTG-STORAGE-3] Registros de logs[21] para buscar datos confidenciales [55]	Los registros (logs) de la aplicación no deben mostrar información sensible de los usuarios. Es posible encontrar vulnerabilidades de **intercepción de datos** personales o sensibles.	**PR-S**
[MSTG-STORAGE-8] Pruebas en copias de seguridad para buscar datos confidenciales [56]	La información sensible no se incluye en copias de seguridad generadas por el sistema operativo. Es posible encontrar vulnerabilidades de **intercepción de datos** personales o sensibles.	**PR-S**

[18] Evaluaciones para detectar y corregir vulnerabilidades en aplicaciones móviles, protegiendo así la información del usuario.

[19] Evaluaciones diseñadas para verificar cómo una aplicación móvil gestiona y protege los datos personales de los usuarios, asegurando el cumplimiento de normativas de privacidad y protección de datos.

[20] Entorno seguro que encapsula una aplicación y sus datos, proporcionando aislamiento del sistema operativo y otras aplicaciones para mejorar la seguridad y la gestión de la aplicación móvil.

[21] Registros automáticos que capturan eventos y actividades relevantes dentro de un sistema informático, útiles para el diagnóstico, la seguridad y el análisis posterior.

[MSTG-STORAGE-10] Pruebas en memoria para buscar datos confidenciales [57]	La aplicación no debe conservar ninguna información sensible en la memoria más allá del tiempo necesario y se debe borrar la memoria después de su uso. Es posible encontrar vulnerabilidades de **almacenamiento inseguro** de datos personales o sensibles.	**PR-S**
[MSTG-STORAGE-11] Pruebas en la política de seguridad de acceso al dispositivo [58]	La aplicación requiere que exista una política de seguridad mínima en el dispositivo, si esta no es lo suficientemente robusta puede provocar el acceso a datos confidenciales.	**SG**
MASVS-CRYPTO		
[MSTG-CRYPTO-1] Prueba de criptografía simétrica [59]	La aplicación no depende únicamente de criptografía simétrica[22] cuyas claves son encontrado directamente en el código fuente, puesto que si fuesen encontradas permitiría el acceso a los datos encriptados que puedan ser sensibles. Es posible encontrar vulnerabilidades de **intercepción de datos** personales o sensibles.	**PR-S**
[MSTG-CRYPTO-2, MSTG-CRYPTO-3, MSTG-CRYPTO-4] Prueba de la configuración de algoritmos estándar criptográficos[23] [60]	La aplicación debe utilizar implementaciones de criptografía probadas, con controles de seguridad que sean apropiadas y que siguen las buenas prácticas de la industria, para así asegurarse de que todos los datos encriptados lo estén de la forma más segura posible. El resultado de ejecutar esta prueba busca descifrar los posibles datos encriptados en el código de la app, por lo que es posible encontrar vulnerabilidades de **intercepción de datos** personales o sensibles.	**PR-S**
[MSTG-CRYPTO-5] Prueba los propósitos de las claves [61]	La aplicación no debe reutilizar la misma clave criptográfica para diversos fines puesto que facilitaría la explotación de los datos sensibles encriptados. Esta prueba comprueba la implementación de una medida de seguridad.	**SG**
[MSTG-CRYPTO-6] Prueba de generación de números aleatorios [62]	Los valores aleatorios se generan usando un generador de números aleatorios suficientemente seguro que asegure que ningún atacante pueda descifrarlos fácilmente. Esta prueba comprueba la implementación de una medida de seguridad.	**SG**

[22] Método de cifrado donde la misma clave se utiliza tanto para cifrar como para descifrar los datos, facilitando una comunicación segura entre dos partes que comparten esta clave secreta.

[23] Conjunto de pasos y reglas matemáticas que define cómo se cifran y descifran los datos para garantizar la seguridad y privacidad en las comunicaciones y almacenamiento de información [21].

MASVS-AUTH		
[MSTG-AUTH-1] Prueba de autenticación biométrica[24] [63]	Si la aplicación provee acceso a un servicio remoto, se implementa un mecanismo de autentificación aceptable que no permita a un tercero acceder en otra cuenta y visualizar información confidencial. Esta es una prueba para medir el acceso a la aplicación, y por lo tanto su seguridad.	**SG**
[MSTG-AUTH-8] Pruebas de confirmación de credenciales [64]	La autenticación biométrica, si la hay, no debe estar asociada a eventos (p. ej. usando una API que simplemente retorna "true" o "false"), sino basada en el desbloqueo del keychain/keystore (almacenamiento seguro). Esta es una prueba para medir el acceso a la aplicación, y por lo tanto su seguridad.	**SG**
MASVS-NETWORK		
[MSTG-NETWORK-1] Prueba del cifrado de datos en la red [65]	La información se debe enviar cifrada utilizando TLS y un canal seguro es usado consistentemente en la aplicación, para así evitar que terceros puedan llevar a cabo un ataque del tipo Man in the Middle. Un tercero podría leer datos personales o sensibles de una comunicación si no está correctamente configurada, por lo que es posible encontrar vulnerabilidades de **intercepción de datos** personales o sensibles.	**PR-S**
[MSTG-NETWORK-2] Prueba de la configuración TLS [66]	Las configuraciones del protocolo TLS [25] deben seguir las buenas prácticas de la industria, o deben hacerlo lo mejor posible en caso de que el sistema operativo del dispositivo no soporte los estándares recomendados, para así asegurar la comunicación segura entre cliente y servidor. Es posible encontrar vulnerabilidades de **intercepción de datos** personales o sensibles.	**PR-S**
[MSTG-NETWORK-3] Verificación de identificación del endpoint de prueba [67]	La aplicación debe verificar el certificado X.509 [26] del sistema remoto al establecer el canal seguro y sólo se deben aceptar certificados firmados por una autoridad certificadora (CA) de confianza, con el propósito de reducir el riesgo de usar certificados falsificados. Esta prueba mide la seguridad de los certificados de red utilizados.	**SG**
[MSTG-NETWORK-4] Prueba de almacenes de certificados	La aplicación debe utilizar su propio almacén de certificados o realiza pinning	**SG**

[24] Verifica la identidad usando características físicas únicas como huellas dactilares o reconocimiento facial.

[25] Protocolo de seguridad que garantiza la privacidad y la integridad de los datos transmitidos en Internet mediante el cifrado de la comunicación entre clientes y servidores [22].

[26] Estándar utilizado para verificar la autenticidad de entidades en línea, como sitios web, asegurando la confianza y la seguridad mediante la firma digital de una autoridad certificadora.

personalizados y pinning de certificados [68]	[27]del certificado o la clave pública del servidor. Bajo ningún concepto establecerá conexiones con servidores que ofrecen otros certificados o claves, incluso si están firmados por una autoridad certificadora (CA) de confianza. De esta forma se previene el desvío de datos a servidores maliciosos y se llevará un control más estricto de los certificados válidos. Esta prueba mide la seguridad de los certificados de red utilizados.	
[MSTG-NETWORK-6] Probar el proveedor de seguridad [69]	La aplicación sólo debe depender de bibliotecas de conectividad y seguridad actualizadas, para prevenir la explotación de vulnerabilidades nuevas que certificados desfasados no contemplen. Esta prueba mide la configuración de seguridad con el proveedor de red.	**SG**
MASVS-PLATFORM		
[MSTG-STORAGE-6] Determinar si los datos almacenados han sido expuestos a través de mecanismos de IPC[28] [70]	La información personal no debe exponerse a través de mecanismos de comunicación entre procesos (IPC). Los distintos proveedores de contenido no deben pasarse entre sí información personal. Es posible encontrar vulnerabilidades de **intercepción o fuga de datos** personales o sensibles.	**PR-S**
[MSTG-STORAGE-7] Comprobación de divulgación de datos confidenciales a través de la interfaz de usuario [71]	La información confidencial, como, no debe exponerse a través de la interfaz que todos los usuarios ven puesto que puede ser captada por terceros. La preocupación en esta prueba es que no se divulgue información confidencial.	**PR-N**
[MSTG-STORAGE-9] Encontrar información confidencial en capturas de pantalla generadas automáticamente [72]	La aplicación debe eliminar toda la información confidencial de la vista cuando pasa a segundo plano. La preocupación en esta prueba es que no se divulgue información confidencial.	**PR-N**
[MSTG-PLATFORM-1] Prueba de permisos de aplicaciones [73]	La aplicación debe requerir la mínima cantidad de permisos necesaria para que así asegurar que no se acceda a partes del dispositivo que no sean necesarias para el funcionamiento de la misma. Los permisos aceptados por el usuario en la política de privacidad aparecen aquí y pueden comprometer sus datos personales y sensibles si no están correctamente configurados. Es posible encontrar vulnerabilidades de **permisos excesivos.**	**PR-S**

[27] Técnica que asegura conexiones al fijar certificados o claves de servidor específicos para evitar ataques de intermediarios.
[28] Mecanismo que facilita la comunicación y el intercambio de datos entre procesos o aplicaciones en un sistema informático. [23]

[MSTG-PLATFORM-3] Prueba de enlaces profundos[29] [74]	Todo dato ingresado por el usuario o cualquier fuente externa debe ser validado y, si es necesario, saneado. Si no se validan y sanean adecuadamente los datos ingresados por el usuario o provenientes de fuentes externas, los atacantes podrían inyectar datos maliciosos a través de enlaces profundos, permitiendo el acceso no autorizado a información privada del usuario, o la ejecución de acciones no deseadas en la aplicación. Esta prueba busca rastrear la implementación de medidas de seguridad contra fuentes externas.	**SG**
[MSTG-PLATFORM-4] Pruebas de implementación vulnerable de PendingIntent[30] y Pruebas de exposición a funcionalidades sensibles mediante IPC [74]	La aplicación no debe exponer ninguna funcionalidad sensible a través de mecanismos IPC salvo que dichos mecanismos estén debidamente protegidos. Si la aplicación expone funcionalidades sensibles a través de IPC, podría permitir a aplicaciones maliciosas o atacantes acceder a información confidencial del usuario. Esta prueba busca medir la seguridad de la implementación de mecanismos IPC.	**SG**
[MSTG-PLATFORM-5] Prueba de ejecución de JavaScript en WebViews [76]	JavaScript debe estar deshabilitado en los WebViews [31]salvo que sea necesario, puesto que puede permitir a sitios web ejecutar scripts potencialmente maliciosos que podrían robar datos confidenciales de la aplicación.	**PR-N**
[MSTG-PLATFORM-6] Prueba de controladores de protocolo WebView [77]	Las WebViews se deben configurar para permitir el mínimo número de protocolos requeridos (idealmente sólo https). Aquellos considerados como peligrosos deben estar deshabilitados (p. ej. file, tel y app-id) y ser un problema para la seguridad de la app. Esta prueba busca medir la seguridad del protocolo de WebViews.	**SG**
[MSTG-PLATFORM-7] Prueba de objetos Java expuestos a través de WebViews [78]	Si los métodos nativos son expuestos en WebViews, debe verificarse que cualquier componente JavaScript se carga exclusivamente desde el contenedor de la aplicación, para que dichos componentes JavaScript maliciosos no accedan a estos métodos, potencialmente accediendo a información confidencial o ejecutando código arbitrario en el contexto de la aplicación. Esta prueba busca medir la seguridad del uso de WebViews.	**SG**

[29] URLs que dirigen a usuarios a contenido específico dentro de una aplicación móvil, mejorando la experiencia de usuario al llevarlos directamente a secciones relevantes sin necesidad de navegación manual.
[30] Permite a una aplicación realizar acciones en nombre del usuario en un momento posterior, incluso cuando la aplicación está inactiva.
[31] Permite mostrar contenido web dentro de una aplicación, facilitando la integración de páginas web y servicios en línea en aplicaciones móviles [24]

[MSTG-PLATFORM-9] Pruebas de ataques de superposición (overlay attacks[32]) [79]	La aplicación debe protegerse contra ataques de tipo screen overlay, en los que el usuario puede introducir sin saberlo credenciales o datos de otro tipo. Esta prueba busca que la app no permita que elementos externos recaben información del usuario, tratando así de mejorar la seguridad de la misma.	**SG**
[MSTG-PLATFORM-10] Prueba de limpieza de WebViews [80]	La caché, el almacenamiento y los recursos cargados (JavaScript, etc.) de las WebViews deben borrarse antes de destruir la WebView. Este caché puede tener información confidencial relativa a la aplicación.	**PR-N**
MASVS-CODE		
[MSTG-CODE-5] Comprobación de debilidades en bibliotecas de terceros[33] [81]	Todos los componentes de terceros deben ser identificados y revisados en busca de vulnerabilidades comunes. Las bibliotecas de terceros pueden introducir vulnerabilidades en la aplicación si no se revisan adecuadamente. Estas vulnerabilidades pueden ser explotadas por atacantes para acceder a datos privados del usuario.	**SG**
[MSTG-CODE-9] Asegúrese de que las funciones de seguridad gratuitas estén activadas [82]	Las funcionalidades y las herramientas de seguridad gratuitas deben estar activadas. Esto incluye simplificación del código, protección de la pila, soporte PIE y conteo automático de referencias, sin embargo, todos estos cuidados no se encuentran muy ligados con protección de la privacidad de los usuarios.	**SG**
MASVS-RESILIENCE		
[MSTG-CODE-1] Asegurarse de que la aplicación esté correctamente firmada [83]	La aplicación debe estar firmada y provista con un certificado válido, cuya clave privada está debidamente protegida, para evitar que los usuarios sean vulnerables a la instalación de versiones no oficiales o modificadas de la aplicación.	**SG**
[MSTG-CODE-2] Probar si la aplicación es depurable [84]	La aplicación debe estar publicada en modo release y con sus configuraciones apropiadas. Publicar la aplicación en modo release con las configuraciones apropiadas evita que los datos confidenciales de la misma sean expuestos a través de herramientas de depuración.	**SG**

[32] Colocan ventanas emergentes maliciosas sobre aplicaciones legítimas para engañar a los usuarios y obtener información sensible.
[33] Conjuntos de código prescrito desarrollados por otros que se utilizan en aplicaciones para agregar funcionalidades específicas sin necesidad de desarrollarlas desde cero.

[MSTG-CODE-3] Prueba de símbolos de depuración [85]	Los símbolos de depuración deben estar eliminados de los binarios nativos. Aunque estos pueden dar información adicional sobre la estructura interna de la aplicación, facilitando la ingeniería inversa, no se considera un riesgo que afecte a la privacidad directamente.	**SG**
[MSTG-CODE-4] Pruebas de código de depuración y registro de errores detallado [86]	Cualquier código de depuración y/o de asistencia al desarrollador (p. ej. código de test, backdoors, configuraciones ocultas) debe ser eliminado. La aplicación no debe hacer logs detallados de errores ni de mensajes de depuración, ya que el código de depuración y los registros detallados pueden contener información confidencial.	**SG**
[MSTG-RESILIENCE-1] Prueba de detección de raíces [87]	La aplicación detecta y responde a la presencia de un dispositivo rooteado, ya sea alertando al usuario o finalizando la ejecución de la aplicación, aunque esto no pone en riesgo la privacidad de los usuarios de manera demasiado directa.	**SG**
[MSTG-RESILIENCE-2] Prueba de detección antidepuración [88]	La aplicación impide la depuración o detecta y responde a la misma. Se deben cubrir todos los protocolos de depuración. Impedir la depuración o detectar y responder a ella protege contra la extracción de datos sensibles y la manipulación de la aplicación durante la ejecución	**SG**
[MSTG-RESILIENCE-3] Prueba de comprobaciones de integridad de archivos [89]	La aplicación detecta y responde a cualquier modificación de ejecutables y datos críticos de la propia aplicación, ayudando a asegurar que la aplicación no ha sido alterada para incluir código malicioso.	**SG**
[MSTG-RESILIENCE-4] Prueba de detección de herramientas de ingeniería inversa[34] [90]	La aplicación detecta la presencia de herramientas de ingeniería inversa o frameworks[35] comúnmente utilizados. Responder ante estas herramientas protege contra la extracción y análisis del código de la aplicación, lo cual podría revelar vulnerabilidades explotables.	**SG**
[MSTG-RESILIENCE-5] Prueba de detección del emulador [91]	La aplicación detecta y responde a ser ejecutada en un emulador protegiendo así, contra entornos controlados por atacantes donde se podrían analizar y explotar las debilidades de la aplicación.	**SG**
[MSTG-RESILIENCE-6] Prueba de comprobaciones de integridad en tiempo de ejecución [92]	La aplicación detecta y responde ante modificaciones de código o datos en su propio espacio de memoria, ayudando a prevenir que un atacante manipule la aplicación en tiempo real.	**SG**

[34] Proceso de desmontar y comprender cómo funciona un producto o sistema, a menudo con el objetivo de duplicarlo, mejorar su funcionamiento o detectar posibles vulnerabilidades de seguridad.
[35] Conjuntos de herramientas y funciones predefinidas que facilitan y aceleran el desarrollo de aplicaciones al proporcionar una estructura y funcionalidades comunes.

[MSTG-RESILIENCE-9] Prueba de ofuscación [93]	La ofuscación[36] se aplica a las defensas del programa, lo que a su vez impide la desofuscación mediante análisis dinámico, dificultando a los atacantes el acceso a la lógica interna y a los datos confidenciales.	**SG**

Tabla 2.5: Pruebas MSTG definidas por MASVS

A lo largo de los **7 campos** definidos en el estándar MASVS las pruebas definidas han obtenido los siguientes resultados en la clasificación de utilidad para el estudio de la privacidad de la app que se ha creado:

Tipo	**Cantidad**
SG	25
PR-N	5
PR-S	10
TOTAL	40

Tabla 2.6: Resultados obtenidos de la calificación de pruebas

En total, de las 40 pruebas MSTG que recoge el estándar, se ha obtenido que 25 de ellas están mayoritariamente relacionadas con **vulnerabilidades relativas a la seguridad** de la aplicación. De las 15 pruebas restantes, la explotación de 5 de ellas dará como resultado la exposición de **datos confidenciales** de la aplicación, pero no personales ni sensibles. Por último, hemos obtenido 11 pruebas mediante las cuales se pueden explotar vulnerabilidades que expongan **datos personales o sensibles** de los usuarios.

De estas 10 pruebas, se ha decidido dejar de lado una de ellas debido a que consideramos que las vulnerabilidades que busca pueden verse de forma más completa en otra prueba que sí que realizamos. Estas pruebas son MSTG-CRYPTO-1, la cual busca los puntos con criptografía simétrica del código, y [MSTG-CRYPTO-2, MSTG-CRYPTO-3, MSTG-CRYPTO-4], que analizan la configuración de algoritmos estándar criptográficos. Consideramos que, al llevar a cabo el análisis sobre los algoritmos criptográficos definidos en la segunda prueba, acabaremos rastreando las claves empleadas por los mismos y determinando si la aplicación utiliza criptografía simétrica o no.

Así pues, acabamos con **9 pruebas** que completan nuestra auditoría, y con las que considero que podremos abarcar todas las posibles vulnerabilidades relacionadas con la privacidad que se definieron anteriormente. Estas 9 pruebas son sin duda las que más se centran en la privacidad de los usuarios dentro de las definidas por MASVS, por lo que se espera que su resultado nos dé una muestra fiel de cuál es el grado de privacidad presente en las aplicaciones auditadas.

5.3. Pruebas de la auditoría de privacidad

Este último apartado se centrará en definir los pasos a seguir para la consecución de cada una de las pruebas seleccionadas. Como se ha especificado en el capítulo 4 de la metodología seguida para la ejecución de las pruebas MASVS, antes de comenzar con ejecución de la prueba, es

[36] Proceso de modificar el código fuente de una aplicación para dificultar su comprensión por parte de terceros, con el objetivo de proteger la propiedad intelectual y aumentar la seguridad del software.

importante definir los objetivos que se esperan conseguir con su realización, tratando de que coincidan con los que se establecieron para este proyecto. Adicionalmente se debe entender que es lo que pretende cada paso tomado en la realización de la prueba, así como el funcionamiento de las herramientas cuyo uso pueda ser requerido.

La siguiente tabla se encarga de mostrar todo lo comentado, comenzando con un apartado en el que se muestran los objetivos generales de la prueba y las comprobaciones que se hacen en ella de forma general. Luego se especifican las herramientas que serán empleadas durante el proceso de ejecución de la misma, y finalmente, se muestra una descripción de los pasos a realizar para la finalización de la prueba.

[MSTG-STORAGE-1, MSTG-STORAGE-2] [54]	**Pruebas en almacenamiento local para buscar datos confidenciales**
Objetivos	Identificar datos potencialmente personales **almacenados** por la aplicación y verificar si están almacenados de forma segura. Se deben realizar las siguientes comprobaciones: ▪ Analizar el almacenamiento de datos en el código fuente. ▪ Verificar todos los archivos generados y modificados por la aplicación y asegurar que el método de almacenamiento sea lo suficientemente seguro.
Herramientas	▪ **ADB** para acceder al almacenamiento del dispositivo. ▪ **Sqlite** para acceder al contenido de las bases de datos de la aplicación.
Pruebas	*Análisis Estático* ▪ Identificación del **Tipo de Almacenamiento**: • Determina el tipo de almacenamiento utilizado por la aplicación y verificar si maneja datos sensibles de manera insegura. • Revisar el archivo `AndroidManifest.xml` para permisos de lectura/escritura en almacenamiento externo, por ejemplo, `android.permission.WRITE_EXTERNAL_STORAGE` . ▪ Revisión del **Código Fuente**: • Busca palabras clave y llamadas a APIs que se usan para almacenar datos: - Permisos de archivo como `MODE_WORLD_READABLE` o `MODE_WORLD_WRITABLE`. Se deben evitar estos modos ya que permiten que cualquier aplicación lea o escriba en los archivos. - Uso de clases y funciones como `SharedPreferences`, `FileOutputStream`,

	`getExternal*`, `getWritableDatabase`, `getReadableDatabase`, `getCacheDir` y `getExternalCacheDirs`. *Análisis Dinámico* ▪ Instalar y utilizar la aplicación ejecutando todas sus funciones al menos una vez para generar y rastrear datos. ▪ Revisar el almacenamiento local **interno** y **externo** para archivos creados por la aplicación que contengan datos sensibles. ▪ Verificar si las **bases de datos** SQLite están disponibles y si contienen información sensible. Comprueba si están cifradas y cómo se protege la contraseña de la base de datos. ▪ Revisar las **Preferencias Compartidas** almacenadas como archivos XML para información sensible, y verifica si están encriptadas. ▪ Revisar los **permisos** de los archivos en `/data/data/<nombre-del-paquete>`.
[MSTG-STORAGE-3] [55]	**Registros de logs para buscar datos confidenciales**
Objetivos	Identificar cualquier dato personal de la aplicación dentro de los **registros logs** del sistema y de la aplicación. Se deben realizar las siguientes comprobaciones: ▪ Analizar el código fuente para registrar el código relacionado. ▪ Verificar el directorio de datos de la aplicación para ver los archivos de registro. ▪ Recopilar mensajes y registros del sistema y analizarlos en busca de datos personales y sensibles.
Herramientas	▪ **Logcat** para revisar la salida de los registros del sistema
Pruebas	*Análisis Estático* ▪ Identificar clases y funciones de Log: • Búsqueda en el código fuente del uso de clases y funciones de registro (logging) con las siguientes palabras clave: • `android.util.Log` • `Log.d`, `Log.e`, `Log.i`, `Log.v`, `Log.w`, `Log.wtf` • `Logger` • `System.out.print`, `System.err.print` • `logfile`, `logging`, `logs` *Análisis Dinámico* ▪ Ejecución de la aplicación: • Usar todas las funciones de la aplicación al menos una vez. Identificar el directorio de datos de la aplicación y

	buscar archivos de log en `/data/data/<nombre-del-paquete>`. • Verificar si se han generado datos de log y si la aplicación crea y almacena sus propios logs en el directorio de datos. ▪ Monitoreo de salida del sistema: • Usar "Logcat" [25] para determinar qué datos se imprimen directamente por "System.out.println" o "printStackTrace". ▪ Filtrado de Logcat: • Filtrar la salida de "Logcat" para un paquete específico • Si ya conoces el PID de la aplicación[37], puedes especificarlo directamente usando la opción `--pid`.
[MSTG-STORAGE-8] [56]	**Pruebas en copias de seguridad para buscar datos confidenciales**
Objetivos	Identificar la aparición de datos personales en las **copias de seguridad automáticas** que la app genera. ▪ Comprobar que ningún dato sensible relativo a los usuarios se haya anotado en las copias de seguridad. ▪ Comprobar cómo realiza la app las copias de seguridad y asegurarse de que en el proceso no se anota ningún dato personal.
Herramientas	▪ **ADB** para realizar la copia de seguridad de la aplicación. ▪ **Android Backup Extractor** para extraer el archivo de backup en formato .tar.
Pruebas	*Análisis Estático* ▪ Verificación de la configuración de respaldo en el AndroidManifest.xml: • Revisar el archivo "AndroidManifest.xml" para encontrar la siguiente flag[38]: ```xml android:allowBackup="true"``` • Si el valor de esta flag es `true`, determina si la aplicación guarda algún tipo de dato sensible. ▪ Revisión de backups en la nube: • Independientemente de si se utiliza backups de clave/valor o backups automáticos, determinar lo siguiente: • Archivos enviados a la nube: Identificar qué archivos son enviados a la nube (por ejemplo, `SharedPreferences`). • Contenido de los archivos: Verificar si los archivos contienen información sensible.

[37] Número único asignado por el sistema operativo para identificar y gestionar cada instancia de un programa en ejecución.
[38] Configuración que define comportamientos específicos o características de una aplicación, como la orientación de la pantalla o la compatibilidad con hardware específico.

	• Encriptación de la información sensible: Asegúrese de que la información sensible esté encriptada antes de ser enviada a la nube. ▪ Configuración del Backup Automático: • El backup automático se configura mediante el atributo booleano `android:allowBackup` dentro del archivo de manifiesto de la aplicación. • El atributo `android:fullBackupOnly` puede usarse para activar el backup automático al implementar un agente de respaldo, pero este atributo está disponible solo para versiones de Android 6.0 y superiores. ▪ Verificación del Backup de Clave/Valor[39]: • Para habilitar el backup de clave/valor, se define el agente de backup en el archivo de manifiesto: ```xml android:backupAgent``` • Para implementar el backup de clave/valor, se extiende una de las siguientes clases: ▪ `BackupAgent` ▪ `BackupAgentHelper` *Análisis Dinámico* ▪ Ejecución de la aplicación y generación de Backup: • Ejecutar todas las funciones disponibles de la aplicación. • Intentar realizar un backup a través de adb usando el siguiente comando: adb backup -apk -nosystem <package-name> • ADB debería responder con "Now unlock your device and confirm the backup operation" y el dispositivo Android debería pedir una contraseña opcional para el backup. ▪ Aprobación del backup y conversión del archivo: • Aprobar el backup desde tu dispositivo seleccionando la opción "Back up my data". ▪ Extracción del backup y comprobación de los datos que genera el mismo

[39] Proceso que permite a las aplicaciones Android almacenar y recuperar datos de manera segura y eficiente, facilitando la restauración de información importante en caso de pérdida o cambio de dispositivos.

[MSTG-STORAGE-10] [57]	**Pruebas en memoria para buscar datos confidenciales**
Objetivos	Identificar si dentro de la **memoria interna** del sistema[40] se almacena información sensible, y si es el caso, definir cuál es dicha información sensible que está almacenada en la memoria. ▪ Comprobar que esta información se expone lo más brevemente posible.
Herramientas	
Pruebas	*Análisis Estático* ▪ Identificación y mapeo de componentes de la aplicación: • Identificar los componentes de la aplicación y mapear dónde se utilizan los datos sensibles. • Asegurarse de que los datos sensibles son manejados por la menor cantidad de componentes posible. ▪ Eliminación adecuada de referencias de objetos: • Verificar que las referencias de objetos que contienen datos sensibles se eliminen adecuadamente una vez que el objeto ya no es necesario. • Solicitar la recolección de basura después de eliminar las referencias. ▪ Sobrescritura de datos sensibles: • Asegurarse de sobrescribir los datos sensibles tan pronto como ya no sean necesarios. • No usar estructuras de datos inmutables (como `String` y `BigInteger`) para representar datos sensibles, ya que su nulificación será inefectiva. ▪ Uso de tipos de datos primitivos: • Almacenar la información sensible en tipos de datos primitivos, como `byte[]` y `char[]`. • Evitar almacenar la información en tipos de datos no primitivos mutables.
[MSTG-CRYPTO-2, MSTG-CRYPTO-3, MSTG-CRYPTO-4] [60]	**Prueba de la configuración de algoritmos estándar criptográficos**
Objetivos	Asegurar que los **algoritmos criptográficos** empleados para la encriptación de información de carácter personal o sensible cumplen unos

[40] La memoria interna del sistema en Android es el espacio de almacenamiento utilizado por el sistema operativo para ejecutar aplicaciones y almacenar datos temporales y esenciales para el funcionamiento del dispositivo [26].

	requisitos de seguridad mínimos y siguen las mejores prácticas definidas por la industria. ▪ Identificar todos los puntos de la app en los que se estén empleando algoritmos criptográficos. ▪ Definir el tipo de algoritmo utilizado y su seguridad respecto a los estándares actuales. ▪ Comprobar que los algoritmos criptográficos empleados sean robustos frente a ataques.
Herramientas	▪ **CypherChef** para desencriptar las claves cifradas del código ▪ **Base64Decoder** para decodificar las cadenas de texto codificadas en base 64
Pruebas	*Análisis Estático* ▪ Identificación de Primitivas Criptográficas en el código: • Buscar instancias de las clases `Cipher`, `Mac`, `MessageDigest`, `Signature`. • Buscar interfaces como `Key`, `PrivateKey`, `PublicKey`, `SecretKey`. • Buscar funciones `getInstance`, `generateKey`. ▪ Identificación de Implementaciones Criptográficas personalizadas: • Examinar el código en busca de cualquier implementación criptográfica personalizada. • Verificar que las implementaciones sigan las mejores prácticas y no utilicen algoritmos inseguros o desactualizados. ▪ Revisión de buenas prácticas criptográficas: • Revisar el código para detectar algoritmos inseguros y configuraciones comunes incorrectas.
[MSTG-NETWORK-1] [65]	**Prueba del cifrado de datos en la red**
Objetivos	Comprobar que la información intercambiada entre la app (cliente) y el servidor correspondiente se encuentre **cifrada** correctamente para evitar que terceros puedan interceptar el flujo de comunicación y recoger posibles datos personales intercambiados. ▪ Comprobar que las comunicaciones se realizan de forma cifrada. ▪ Asegurar que todos los intercambios en red se hacen vía protocolo HTTPs y no HTTP [27]. ▪ Identificar todas las solicitudes de red realizadas desde el código y asegurarse de que se realizan de la forma correcta.
Herramientas	▪ **Wireshark** para interceptar el tráfico de red entre la app y el servidor

Pruebas	*Análisis Estático* ▪ Identificación de **solicitudes de red**: • Buscar en el código fuente todas las solicitudes de red para asegurarte de que no se utilizan URLs de HTTP sin cifrar. • Verificar que la información sensible se envía a través de canales seguros utilizando `HttpsURLConnection` o `SSLSocket` (para comunicación a nivel de socket utilizando TLS). ▪ Uso seguro de **API de red**: • Incluso al usar una API de bajo nivel que se supone realiza conexiones seguras (como `SSLSocket`), asegurar que esté implementada de manera segura. • `SSLSocket` no verifica el nombre de host, por lo que debes usar `getDefaultHostnameVerifier` para verificar el nombre de host ▪ Verificación de tráfico en texto claro: • Asegurar que la aplicación no permita el tráfico HTTP en texto claro. ▪ Revisión del Manifest y la Configuración de Seguridad de Red • Revisar el archivo `AndroidManifest.xml` para asegurar que el atributo `android:usesCleartextTraffic` no permita tráfico en texto claro. android:usesCleartextTraffic="false" • Verifica que la configuración de seguridad de red no permita el tráfico en texto claro. <domain-config cleartextTrafficPermitted="false"> *Análisis Dinámico* ▪ Intercepción del **Tráfico de Red**: • Interceptar el tráfico de red entrante y saliente de la aplicación y asegurarse de que esté cifrado.
[MSTG-NETWORK-2] [66]	**Prueba de la configuración TLS**
Objetivos	Comprobar que la **configuración** del protocolo TLS empleado por la app sigue las buenas prácticas de la industria, para así asegurar la comunicación segura entre cliente y servidor.
Herramientas	▪ **MobSF TLS tester** para realizar las pruebas sobre la configuración TLS
Pruebas	*Análisis Dinámico*

	Ejecución del tester TLS de MobSF que lleva a cabo las siguientes pruebas: ▪ Cleartext Traffic Test ▪ TLS Misconfiguration Test ▪ TLS Pinning/Certificate Transparency Bypass Test ▪ TLS Pinning/Certificate Transparency Test
[MSTG-STORAGE-6] [70]	**Determinar si los datos almacenados han sido expuestos a través de mecanismos de IPC**
Objetivos	Comprobar si datos personales han sido mostrados a la hora de hacer uso de mecanismos de IPC. ▪ Identificar el funcionamiento del proveedor de contenido de la app. ▪ Asegurarse de que ningún tipo de información comprometida se comparte a través del proveedor.
Herramientas	▪ **Drozer** para comprobar el contenido de los proveedores de contenido de la aplicación.
Pruebas	*Análisis Estático* ▪ Inspeccionar el archivo AndroidManifest.xml: • Identificar todos los elementos `<provider>` en el archivo AndroidManifest.xml. • Determinar si el valor del atributo `android:exported` es "true". Si se define un `<intent-filter>` para el proveedor, el valor predeterminado será "true". • Verificar si los datos están protegidos por una etiqueta de permiso (`android:permission`). • Comprobar si el atributo `android:protectionLevel` tiene el valor "signature", lo que indica que los datos están destinados a ser accedidos solo por aplicaciones de la misma empresa (firmadas con la misma clave). ▪ Inspeccionar el código fuente para comprender cómo se supone que se debe utilizar el proveedor de contenido[41], buscando las siguientes palabras clave: • `android.content.ContentProvider` • `android.database.Cursor` • `android.database.sqlite` • `.query` • `.update` • `.delete` *Análisis Dinámico* ▪ Pruebas de proveedores de contenido:

[41] Interfaz que permite a otras aplicaciones acceder y modificar datos almacenados privadamente por la aplicación, siguiendo reglas de permisos y seguridad para garantizar la integridad de los datos.

	• Usar la herramienta Drozer [28] para enumerar la superficie de ataque del proveedor de contenido. • Ejecutar `app.provider.info` para obtener información sobre los proveedores de contenido. ▪ Identificar URIs de proveedores de contenido: • Usar el módulo `scanner.provider.finduris` de Drozer para identificar URIs de contenido accesibles. ▪ Extraer datos de proveedores de contenido: • Usar el módulo `app.provider.query` de Drozer para extraer datos. ▪ Automatizar Pruebas de Inyección SQL[42]: • Usa el módulo `scanner.provider.injection` de Drozer para encontrar proveedores de contenido vulnerables.
[MSTG-PLATFORM-1] [73]	**Prueba de permisos de aplicaciones**
Objetivos	Probar los permisos de la aplicación para intentar reducir la cantidad de permisos utilizados por su aplicación al mínimo absoluto. ▪ Intentar evaluar si la aplicación necesita usar los permisos definidos. ▪ Asegurar que la solicitud/respuesta para acceder al permiso se maneje correctamente.
Herramientas	
Pruebas	Análisis Estático ▪ Revisión del AndroidManifest.xml: • Inspecciona las declaraciones de permisos en el archivo `AndroidManifest.xml` para identificar permisos innecesarios. ▪ Confirmar la necesidad de cada permiso: • Trabaja con los desarrolladores para entender el propósito de cada permiso declarado y elimina aquellos que no sean esenciales para el funcionamiento de la aplicación. ▪ Permisos Peligrosos • Revisar la lista de permisos peligrosos [29] que requieren especial atención y asegúrate de que solo se utilicen cuando realmente sean necesarios.

Tabla.2.6: Pruebas de la auditoría de seguridad

[42] Evaluaciones diseñadas para detectar y prevenir vulnerabilidades en aplicaciones web y bases de datos, evitando que los atacantes ejecuten comandos maliciosos a través de entradas de datos no validadas.

CAPITULO 5: DISEÑO

Capítulo 6

Auditoría sobre aplicaciones

En este capítulo llevaremos a cabo la ejecución de la auditoría diseñada durante el apartado anterior de diseño, sobre dos aplicaciones móviles Android: InsecureBank y RightsApp. Para ello, en primer lugar, se explicará cómo funcionan estas aplicaciones, para poder así comprender mejor qué es lo que hacen o qué datos sensibles o personales podemos esperar encontrar en ellas. A continuación, comenzaremos con la realización de cada una de las pruebas de nuestra auditoría, siguiendo siempre los pasos que cada prueba MSTG define para su correcta ejecución. Procederemos a redactar una conclusión producto de los resultados obtenidos en cada prueba.

6.1. Auditoría sobre primera app: InsecureBank

La primera aplicación sobre la que se llevará a cabo la auditoría móvil, se tratará de una aplicación de banco de nombre **InsecureBank** [30]. Esta aplicación cuenta con diversos errores de seguridad colocados por su desarrollador para que sean encontrados llevando a cabo las pruebas definidas por OWASP. Como ya se ha comentado, el foco principal de la auditoría a realizar será comprobar el compromiso de la privacidad de los datos de los usuarios, que además en este caso serán de vital importancia, al tratarse principalmente de datos financieros cuya pérdida puede suponer un gran daño.

Para completar la configuración de esta app, es necesario iniciar un **servidor** que reciba todas las peticiones de la misma. Para ello, simplemente se ejecutará un script que viene dentro de la carpeta de la app (/AndroLabServer/app.py) y comenzará a correr el servicio en el puerto 8888 de la máquina anfitriona. Únicamente habrá que completar la dirección de esta máquina en el apartado de preferencias cuando se inicie la app.

El funcionamiento de esta app es muy sencillo: comienza mostrando una pantalla de login en la que el usuario debe introducir sus datos para entrar. Sabemos gracias a la documentación de la aplicación que existen 2 usuarios previamente creados con las siguientes acreditaciones: dinesh/Dinesh@123$ || jack/Jack@123$

Una vez introducida esta información, la aplicación nos brinda un menú con 3 opciones a elegir:

- Hacer una transferencia
- Ver las operaciones realizadas
- Cambiar la contraseña

Ilustración 4: Pantalla principal de la app InsecureBank

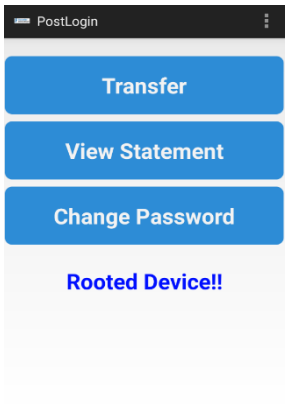


Ilustración 1: Pantalla del menú de la app InsecureBank

La pantalla de operaciones realizadas simplemente muestra una lista con las operaciones que ha realizado el usuario, mientras que la pantalla de cambio de contraseña simplemente permite introducir una contraseña nueva y cambiar la anterior. La pantalla más interesante es sin duda la de las transferencias.

En esta pantalla podremos realizar una transferencia del dinero que queramos desde la cuenta que digamos hacia otra cuenta. Además, se podrá anotar un número de teléfono en la operación.

Por último, la app permite en cada momento su reinicio desde el menú de arriba a la derecha, así como acceder a una pantalla de preferencias en la que se puede especificar la dirección y puerto en los que se encuentra el servidor.

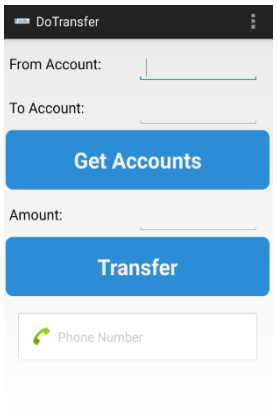


Ilustración 4: Pantalla para la realización de transferencias

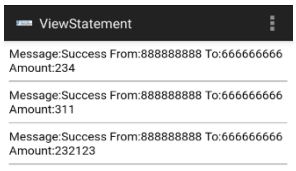


Ilustración 3: Pantalla para la visualización de operaciones realizadas

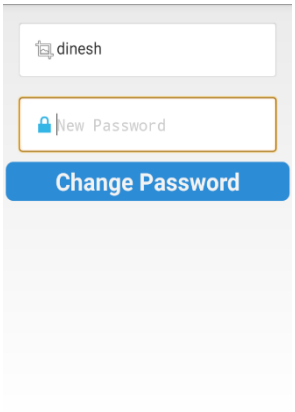


Ilustración 2: Pantalla para el cambio de contraseña

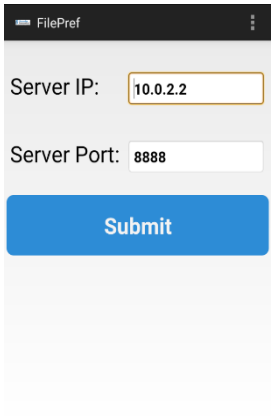


Ilustración 5: Pantalla para la inserción de datos del servidor

6.1.1. [MSTG-STORAGE-1, MSTG-STORAGE-2] Pruebas en almacenamiento local para buscar datos confidenciales

El objetivo de esta prueba radica en identificar datos potencialmente sensibles almacenados por la aplicación y verificar si están **almacenados correctamente** y de forma segura. Para ello, se analizará el código fuente de la aplicación, los datos generados por la aplicación al ejecutar sus funcionalidades y se chequearán todos los archivos generados y modificados.

Análisis Estático

En primer lugar, será necesario determinar el tipo de almacenamiento empleado por la app Android y cómo procesa la información sensible.

Permisos de Almacenamiento

Comenzaremos comprobando los permisos de lectura/escritura de almacenamiento externo definidos en el AndroidManifest.xml:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<android:uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
android:maxSdkVersion="18" />
```

El manifiesto de la app permite la lectura y escritura del almacenamiento local por lo que este será un punto a tener en cuenta en el análisis de los datos presentes en la app.

Uso de permisos de Ficheros

Para continuar con el análisis del código fuente de la aplicación, buscaremos en el mismo palabras claves y llamadas que se emplean para almacenar información:

- Empleo de permisos de ficheros:
  - MODE_WORLD_READABLE: este permiso da a las apps permiso para leer de ficheros incluso cuando estos se encuentran en el directorio privado de datos de la aplicación.
  - MODE_WORLD_WRITEABLE: este permiso da a las apps permiso para escribir en ficheros incluso cuando estos se encuentran en el directorio privado de datos de la aplicación.

  El código fuente no registra ningún uso de estos dos permisos.

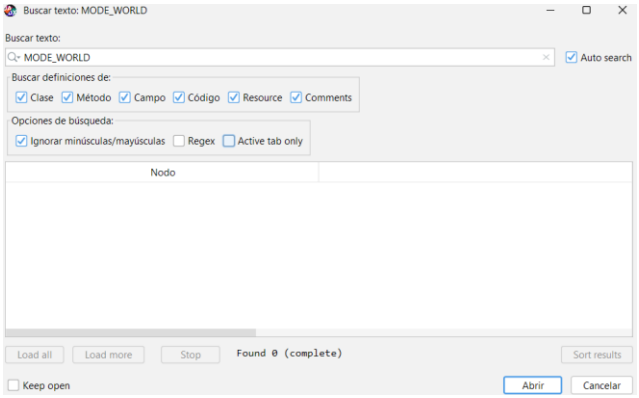


Ilustración 6: Comprobación de existencia de clase MODE_WORLD en el código

Clases y Funciones de Almacenamiento

Se revisará el uso de las siguientes clases y funciones en el código mediante la búsqueda de sus palabras clave en el mismo:

- Empleo de las siguientes clases y funciones:
  - Clase **SharedPreferences**: almacena pares clave-valor. Si se almacenan datos sensibles como tokens de autenticación, credenciales, o información personal sin cifrado adecuado, esto puede representar un riesgo de privacidad en caso de acceso no autorizado. En nuestra app esta clase es referenciada 75 veces, sin embargo, ninguno de los datos asignados a un valor es de carácter personal ni sensible.

  - Clase **FileOutPutStream**: usa almacenamiento externo o interno creando una secuencia de salida de archivo para escribir en el archivo con el nombre especificado. Esta clase se encuentra presente 80 veces a lo largo del código. En todos los usos de la clase, el proceso que se sigue es tomar un valor y guardarlo en el almacenamiento interno de la app. En el caso de que los valores tomados fuesen de carácter personal el problema radicaría en que ninguno de estos datos se encuentran encriptados, sin embargo, para que estos fuesen interceptados sería necesario tener acceso pleno a la app puesto que siempre se almacenan en el almacenamiento interno de la misma. Entre todos sus usos destaca el siguiente, en el que el valor recogido es denominado “clientID”, por lo que presumiblemente almacena el identificador del cliente.

```
zza("Storing clientId", str);
fileOutputStream = context.openFileOutput("gaClientId", 0);
fileOutputStream.write(str.getBytes());
z = true;
fileOutputStream = fileOutputStream;
if (fileOutputStream != null) {
    try {
        fileOutputStream.close();
        fileOutputStream = fileOutputStream;
    } catch (IOException e) {
        zze("Failed to close clientId writing stream", e);
        fileOutputStream = "Failed to close clientId writing stream";
    }
```

Ilustración 7: Uso de la clase FileOutputStream en el código

- Funciones **getExternal***: usa almacenamiento externo. Los datos almacenados en el almacenamiento externo pueden ser accesibles por otras aplicaciones, lo que aumenta el riesgo de filtración de información sensible. Es crucial evitar almacenar datos sensibles en almacenamiento externo [31] o, en su defecto, cifrar adecuadamente la información. En el caso de nuestra app todos los usos de esta función son del tipo getExternalFilesDirs(), por lo que no es preocupante que se almacenen datos sensibles.

- Función **getWritableDatabase**: devuelve una SQLiteDatabase[43] para escritura. Al permitir operaciones de escritura en la base de datos, existe el riesgo de exposición de datos sensibles si no se implementan controles adecuados de acceso y cifrado de la base de datos. También existe el riesgo de modificación malintencionada de datos. El uso de esta función a lo largo del código de la app registra puntos en los que se permite la introducción de datos para su almacenamiento y puntos en los que se leen datos previamente almacenados. Estas consultas de lectura y escritura pueden suponer un problema dependiendo de la naturaleza de los datos que manejan.

- Función **getReadableDatabase**: devuelve una SQLiteDatabase para lectura. Si no se implementan medidas de control de acceso, cualquier entidad que obtenga esta instancia podría leer datos sensibles. Esta función no se emplea ninguna vez a lo largo del código.

En general, podemos observar cómo se emplean diversas clases a lo largo del código que potencialmente podrían almacenar información personal o sensible de los usuarios dentro del almacenamiento interno o externo, Esto unido a que se permite la lectura y escritura en el almacenamiento externo podría suponer algún problema de privacidad.

Análisis Dinámico

Una vez comprendida la configuración del almacenamiento interno de la app y las vulnerabilidades presentes en el mismo, será necesario comprobar cómo se comporta dicho almacenamiento cuando ejecutamos la aplicación y ejecutamos en ella todas las funciones posibles. Con la ejecución de la app se buscará la generación de los datos cuando son introducidos por el usuario, así como el envío de los mismos al endpoint de la app[44].

Acceso al almacenamiento Interno

Para comenzar con las comprobaciones del análisis dinámico se llevará a cabo un chequeo del almacenamiento interno de la app en busca de ficheros creados que puedan contener información sensible. Para ello se hará uso de la herramienta **adb**, la cual tiene registrada dentro de sus

[43] Clase en Android que proporciona una interfaz para gestionar bases de datos SQLite, permitiendo realizar operaciones como crear, leer, actualizar y eliminar datos de manera eficiente dentro de una aplicación.
[44] URL que define un punto de acceso específico donde una aplicación puede enviar o recibir datos, facilitando la comunicación con servicios externos o APIs.

dispositivos remotos reconocidos nuestro simulador Android. Haciendo uso del comando "adb shell", podremos acceder al contenido de nuestro dispositivo en ejecución para analizar todos los elementos presentes en su sistema.

El almacenamiento interno de la app se almacena en el directorio "/data/data/com.android.insecurebank2" y en él se pueden encontrar los siguientes elementos tras la ejecución de la app:

```
genymotion:/data/data/com.android.insecurebankv2 # ls
app_textures app_webview cache code_cache databases shared_prefs
```

Ilustración 8: Contenido del directorio de la memoria interna de la app

Los directorios encontrados son los siguientes:

- **app_textures**: almacenamiento de recursos gráficos.
- **app_webview**: almacenamiento de datos relacionados con componentes WebView.
- **cache**: almacenamiento de ficheros temporales.
- **code_cache**: almacenamiento de código compilado.
- **Databases**: almacenamiento de bases de datos SQLite.
- **shared_prefs**: almacenamiento de ficheros de preferencias compartidas.

El directorio "/**app_textures**", almacena texturas y recursos gráficos de la app, por lo que no contendrá información sensible en su interior.

Por otro lado, el directorio "/**app_webview**" almacena información relacionada con los componentes WebView y puede contener contenido cacheado de la web, cookies[45] y otros datos que pueden ser sensibles. Dentro de este directorio se encuentran varios ficheros y directorios relacionados con el uso de las vistas WebView, sin embargo, entre todos ellos el más susceptible de contener información personal o sensible de los usuarios será el nombrado como "*Web Data*". Este archivo de base de datos SQLite puede almacenar datos del usuario, cookies, credenciales almacenadas… Para inspeccionar su contenido será necesario obtener el directorio "*app_webview*" en nuestra máquina local y acceder a la información de este archivo con la herramienta sqlite3 [32].

```
PS C:\tfg\platform-tools> .\adb.exe pull /data/data/com.android.insecurebankv2/app_webview/
/data/data/com.android.insecurebankv2/app_webview/: 9 files pulled, 0 skipped. 1.0 MB/s (57685 bytes in 0.054s)
```

Ilustración 9: Comando adb con el que se obtiene el directorio especificado en nuestra máquina

Inspeccionando el contenido del archivo se encuentran las siguientes tablas para almacenar la información relativa a las WebViews:

```
sqlite> .tables
autofill                        credit_cards
autofill_model_type_state       masked_credit_cards
autofill_profile_emails         meta
autofill_profile_names          payments_customer_data
autofill_profile_phones         server_address_metadata
autofill_profiles               server_addresses
autofill_profiles_trash         server_card_metadata
autofill_sync_metadata          unmasked_credit_cards
```

Ilustración 10: Tablas almacenadas en la base de datos "Web Data"

[45] Pequeños archivos de datos que los sitios web almacenan en el navegador del usuario para recordar información sobre la visita, como preferencias, sesiones de inicio y actividad en el sitio.

Pese a la cantidad de tablas presentes en esta base de datos `Web Data` del directorio "app_webview", esta se encuentra completamente vacía. Esto puede deberse a que los datos se almacenan en otros lugares, se borran automáticamente, o se sincronizan con un servidor remoto. Aun así, existen en ella tablas potencialmente sensibles como las de la clase "autofill", que almacenan datos de autocompletado; o como las relacionadas con los datos de las tarjetas de crédito y los pagos de los clientes, como "credit_cards", "masked_credit_cards", "unmasked_credit_cards" y "payments_customer_data", que podrían contener datos personales y financieros, implicando riesgos de fuga de información y vulnerabilidades a ataques.

El siguiente directorio a analizar será el "**/cache**", que recoge archivos temporales que pueden contener información sensible. Tras navegar a través de sus múltiples directorios no se ha encontrado ninguna información de valor, por lo que concluiremos que la app no almacena ningún dato personal en su memoria caché. Lo mismo va para el directorio "**/code_cache**", que está vacío.

El directorio "**/databases**", resultará el más crítico a la hora de buscar posibles vulnerabilidades, puesto que contiene bases de datos SQLite que pueden almacenar las credenciales de los usuarios su información financiera. Al acceder al directorio encontramos dos bases de datos, "*mydb*" y "*mydb_journal*". Solo la primera de ellas contiene tablas, de las cuales solo una de ellas contiene información que puede considerarse sensible, los nombres de los usuarios de la aplicación:

```
sqlite> select * from names;
1|dinesh
2|jack
3|jack
4|jack
```

Ilustración 11: Contenido de la tabla "names" dentro de la base de datos "mydb"

Esta base de datos, al encontrarse en el almacenamiento local de la app no será de fácil acceso, y además no recoge información demasiado crítica para la privacidad de los usuarios. Aun así, merece la pena destacar sus vulnerabilidades, como la falta de encriptación de los datos o la nula autentificación que requiere para acceder.

Por último, tenemos el directorio "/**shared_prefs**", el cual contiene los archivos en los que se definen las SharedPreferences [46]de la app, las cuales pueden contener pares clave valor con ajustes de usuarios, tokens y en ocasiones información sensible. Este directorio contiene tres archivos .xml, de los cuales dos contienen información que puede comprometer la privacidad de los usuarios.

```
genymotion:/data/data/com.android.insecurebankv2/shared_prefs # cat com.android.insecurebankv2_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="serverport">8888</string>
    <string name="serverip">172.19.192.1</string>
</map>
```

Ilustración 12: Contenido del archivo "com.android.insecurebankv2_preferences.xml"

[46] Clase en Android que permite almacenar datos clave-valor de manera persistente, facilitando el almacenamiento de configuraciones y preferencias de usuario.

```
genymotion:/data/data/com.android.insecurebankv2/shared_prefs # cat mySharedPreferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="superSecurePassword">v/sJpihDCo2ckDmLW5Uwiw==&#10;    </string>
    <string name="EncryptedUsername">amFjaw==&#13;&#10;    </string>
</map>
```

Ilustración 13: Contenido del segundo archivo del directorio de SharedPreferences

El primer archivo, "com.android.insecurebankv2_preferences.xml", expone la configuración del servidor, en concreto, la información sobre la **dirección IP** y el **puerto** del servidor al que se conecta la aplicación. Si un atacante obtuviese acceso a este archivo, podría conocer detalles de la infraestructura del servidor, facilitando ataques dirigidos como escaneo de puertos, intentos de acceso no autorizado y potenciales ataques de denegación de servicio (DoS)[47].

Por otro lado, el segundo archivo realiza un almacenamiento inseguro de credenciales, ya que contiene una **contraseña** y un **nombre de usuario** aparentemente codificados en base64 [48]. La codificación en base64 no es un método de cifrado, simplemente convierte los datos a una representación diferente, lo que significa que la información sigue siendo fácilmente accesible y no está adecuadamente protegida. Si un atacante obtiene acceso a este archivo, puede decodificar fácilmente el nombre de usuario y la contraseña. Aquí un ejemplo rápido de cómo decodificar el primero de estos datos ("EncryptedUsername") haciendo uso de una herramienta web de **decodificación en base64** [33].

- Nombre de usuario: amFjaw || Decodificado: jack

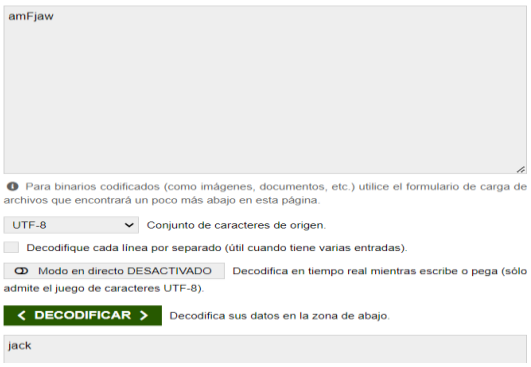


Ilustración 14: Decodificación de username en base64 mediante herramienta web "base64decode"

Para la decodificación de la contraseña (v/sJpihDCo2ckDmLW5Uwiw==) será necesaria alguna acción más, puesto que parece ser una cadena de texto en base64, pero con un cifrado adicional para el cual necesitaremos conocer el método de cifrado y la clave. Aun así, el hecho de obtener el texto cifrado de una contraseña perteneciente a uno de los usuarios ya supone un gran problema de seguridad para la app.

[47] Ataques que buscan sobrecargar un sistema o red, haciendo que los servicios legítimos sean inaccesibles para los usuarios.
[48] Método de codificación que convierte datos binarios en una cadena de texto, permitiendo su transferencia y almacenamiento en sistemas que manejan texto de forma segura.

Revisión de Permisos del Directorio

Como última acción para revisar la configuración de seguridad del directorio del almacenamiento interno de la app, quedaría revisar los permisos configurados para cada uno de los directorios presentes.

```
genymotion:/data/data/com.android.insecurebankv2 # ls -l
total 48
drwxrwx--x 2 u0_a102 u0_a102       4096 2024-06-04 14:22 app_textures
drwx------ 4 u0_a102 u0_a102       4096 2024-06-04 15:59 app_webview
drwxrws--x 4 u0_a102 u0_a102_cache 4096 2024-06-04 14:22 cache
drwxrws--x 2 u0_a102 u0_a102_cache 4096 2024-06-04 13:53 code_cache
drwxrwx--x 2 u0_a102 u0_a102       4096 2024-06-04 13:53 databases
drwxrwx--x 2 u0_a102 u0_a102       4096 2024-06-04 15:59 shared_prefs
```

Ilustración 15: Muestra de permisos del directorio de la memoria interna haciendo uso del comando "ls -l"

La configuración actual de permisos en los directorios del almacenamiento local de la aplicación presenta algunos riesgos de seguridad, especialmente en los directorios “databases” y “shared_prefs”, que permiten el acceso de ejecución a otros usuarios. Sería recomendable restringir los permisos de estos directorios a `drwx------` para limitar el acceso solo al usuario propietario.

Nuestro dispositivo remoto no tiene ningún almacenamiento externo, por lo que no será necesario investigar vulnerabilidades en el mismo.

Conclusiones

Como conclusión a este primer análisis, hemos podido comprobar cómo se realizan algunas malas prácticas a la hora de llevar a cabo el almacenamiento de datos en la memoria interna del dispositivo y se conceden permisos de lectura y escritura que pueden resultar peligrosos. Sin embargo, al comprobar el contenido del almacenamiento interno en el análisis dinámico, no hemos encontrado **ningún dato de carácter personal** ni sensible relativo a los usuarios. El único elemento peligroso es la existencia del username y password (encriptados) dentro de uno de los ficheros de las SharedPreferences, cuya pérdida podría, potencialmente, derivar en la obtención de datos personales y sensibles de los usuarios.

6.1.2. [MSTG-STORAGE-3] Registros de logs para buscar datos confidenciales

Este test se centra en identificar cualquier dato sensible presente tanto en **los logs del sistema** como en los de la aplicación. Denominamos logs al registro de la actividad llevada a cabo en un sistema determinado. En este caso, como tenemos una aplicación corriendo en un sistema móvil remoto, deberemos tomar nota del registro de las acciones llevadas a cabo en la aplicación y en el dispositivo móvil. Para ello, analizaremos el código fuente para buscar el código que genera los registros logs, y recogeremos los mensajes y logs del sistema para analizar cualquier posible información sensible.

Análisis Estático

Uso de Clases de Registro

Las aplicaciones usarán generalmente las clases "**Log**" y "**Logger**" para la creación de registros. Para descubrir esto, será necesario auditar el código fuente de la aplicación para encontrar algunas de estas clases. Estas clases suelen encontrarse buscando las siguientes palabras clave:

- Funciones y clases como:
    - android.util.Log : la clase es importada una gran cantidad de veces en el código fuente, pero no es utilizada en ninguna ocasión.
    - Log.d | Log.e | Log.i | Log.v | Log.w | Log.wtf: la mayoría de usos de estas funciones están relacionados con el registro y notificación de errores durante la ejecución de la app. El problema en estos casos, es, por lo tanto, que en alguno de los mensajes que se registran se imprima algún dato de carácter personal. Revisando todos los usos que tienen estas funciones a lo largo del código fuente, se ha encontrado que solamente la función **Log.d** es susceptible de imprimir información mínimamente comprometedora, puesto que esta función se emplea para **registrar la conexión de la app con el servicio** y en ocasiones puede llegar a imprimir información relativa a los puntos de dicha conexión. Comprobamos pues todos los usos de esta función y encontramos lo siguiente:

```
if (DoLogin.this.result != null) {
    if (DoLogin.this.result.indexOf("Correct Credentials") != -1) {
        Log.d("Successful Login:", ", account=" + DoLogin.this.username + ":" + DoLogin.this.password);
        saveCreds(DoLogin.this.username, DoLogin.this.password);
```

Ilustración 16: Parte del código en el que se imprime el username y password del usuario con la clase "Log.d"

Como puede apreciarse, cuando el usuario inicia sesión correctamente en la aplicación la actividad correspondiente de esta acción, "DoLogin", imprime en el registro log el nombre y contraseña del usuario logueado. Esta es una práctica muy peligrosa desde el punto de vista de la privacidad, y aún más si en el análisis

dinámico comprobamos que efectivamente, los datos imprimidos no están cifrados.

- Logger: esta clase se utiliza para registrar mensajes que pueden ayudar en la depuración y monitoreo del comportamiento de la aplicación. Estos mensajes pueden contener información sobre el estado de la aplicación, errores, advertencias, y otros eventos importantes. En nuestra app su uso se limita a facilitar el registro de mensajes en la aplicación, por lo que será crucial revisar que la información que se está registrando no comprometa la privacidad de los usuarios.

- Palabras clave y salida del sistema:
  - System.out.print | System.err.print: el primero de estos comandos de salida es utilizado numerosas veces en el código fuente de la app, y se ha podido encontrar como en varias ocasiones su uso muestra información personal de los usuarios, más concretamente su número de teléfono y contraseña.

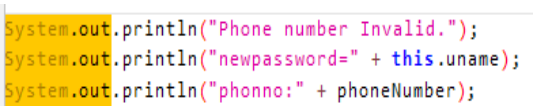


Ilustración 17: Parte del código en la que se imprime información personal del usuario con la clase "System.out.println"

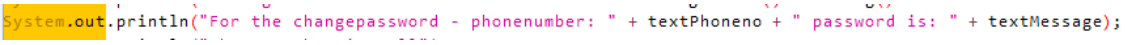


Ilustración 18: Parte del código en la que se imprime información personal del usuario con la clase "System.out.println"

Los resultados obtenidos deberán ser contrastados cuando se lleve a cabo el análisis dinámico de la prueba, pero sobre el papel ninguno de los datos personales que hemos encontrado deberían ser imprimidos en registros de este tipo.

Análisis Dinámico

Cuando se ha ejecutado la aplicación al menos una vez el registro log de la aplicación recogerá todas y cada una de las **acciones realizadas** en la misma durante su ejecución. Este registro log es almacenado por la aplicación en alguno de los directorios del sistema, sin embargo, para visualizar su contenido utilizaremos el siguiente comando:

adb logcat | FINDSTR "$(adb shell ps | FINDSTR com.android.insecurebankv2 | awk '{print $2}')"

Este comando **captura los logs** del sistema Android en tiempo real y filtra únicamente aquellos relacionados con el proceso de la aplicación “com.android.insecurebankv2”. Primero, obtiene la lista de procesos en ejecución en el dispositivo Android y filtra el proceso correspondiente a la aplicación mencionada. Luego, extrae el ID del proceso (PID) de esta aplicación y utiliza este PID para buscar y mostrar solo los logs específicos de ese proceso.

La siguiente es una muestra del resultado obtenido.

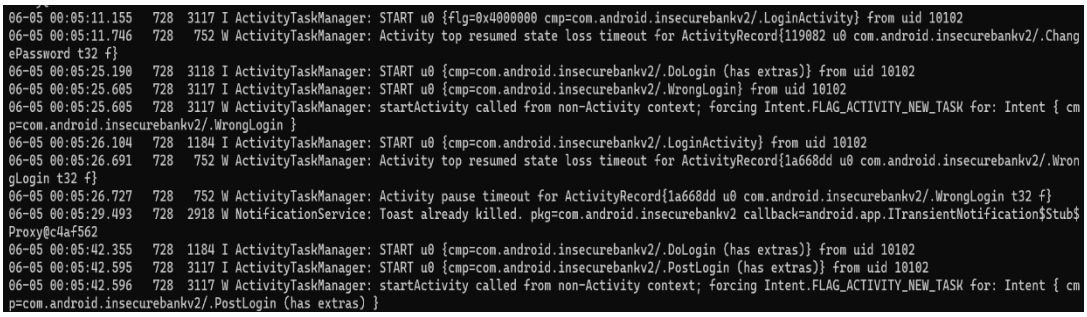


Ilustración 19: Muestra del contenido del Log del sistema cuando se inicia la app

El log recopilado revela **información sobre la aplicación**, incluyendo el inicio y la parada de actividades, errores de permisos y mensajes del sistema. El registro no contiene datos personales directos como nombres o números de tarjetas de crédito, aunque la exposición del flujo de actividades (como inicios de sesión, transferencias y cambios de contraseña) y los intentos de la aplicación de acceder a permisos restringidos pueden ser puntos a tener en cuenta a la hora de definir alguna posible vulnerabilidad.

A continuación, procedo a llevar a cabo todas las funcionalidades importantes de la aplicación. En primer lugar, ante el **inicio de sesión** con una cuenta de usuario se registra el siguiente mensaje:

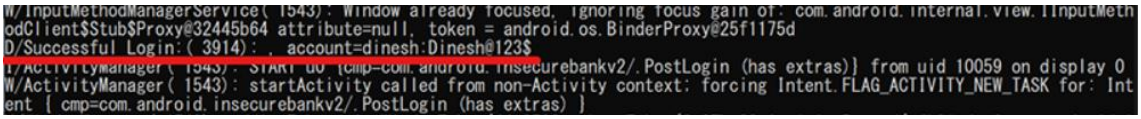


Ilustración 20: Muestra del contenido del Log del sistema cuando un usuario se registra

Tal y como habíamos visto en el análisis estático, la aplicación registra el **nombre** y **contraseña** del usuario que acaba de iniciar sesión exitosamente en su log, ambos además sin ningún tipo de cifrado. La obtención de este registro por parte de algún atacante supondría por lo tanto el filtrado de esta información tan sumamente importante.

Para finalizar con el estudio de la ejecución de los logs, ejecutó el resto de funcionalidades de la app. No obtengo ningún registro al llevar a cabo una transferencia, pero al **cambiar la contraseña** del usuario en el que me encuentro logueado, la aplicación registra lo siguiente:

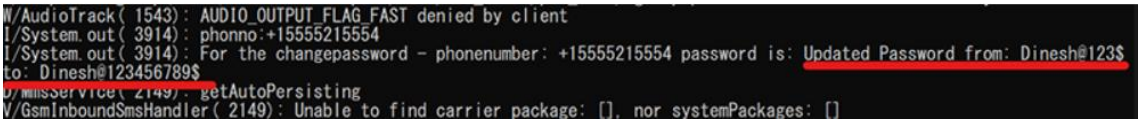


Ilustración 21: Muestra del contenido del Log del sistema cuando un usuario cambia su contraseña

Puede apreciarse como mediante el uso de la salida del sistema la aplicación imprime datos personales como el **número de teléfono** y las **contraseñas** del usuario, la anterior y la nueva tras el cambio. Toda esta información sensible se muestra sin ningún tipo de cifrado, vulnerando así por completo la privacidad de los usuarios en el caso de que estos registros fuesen filtrados.

Conclusiones

Tras lo visto, podemos concluir, por lo tanto, que el uso de los registros logs por parte de la aplicación supone un **riesgo para la privacidad de los usuarios**, dado que se muestran de forma manifiesta datos sensibles de los usuarios.

6.1.3. [MSTG-STORAGE-8] Pruebas en copias de seguridad para buscar datos confidenciales

A lo largo de esta prueba se tratará de comprobar que el contenido de las **copias de seguridad** realizadas por la aplicación no contenga ningún dato sensible que pueda poner en peligro la privacidad de los usuarios. Par ello se deberá buscar en primer lugar como genera la app estas copias de seguridad y donde las guarda.

Análisis Estático

Permisos y Configuración de Backups en AndroidManifest

El primer paso a llevar a cabo en el análisis del código fuente de la aplicación será comprobar si en la configuración del AndroidManifest de la app se permite la realización de **backups.**

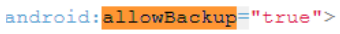


Ilustración 22: Definición del permiso para backups en el manifiesto

En efecto, el manifiesto define que la aplicación permitirá el almacenamiento de información en forma de backup. Además, al estar definido de esta manera en el AndoridManifest, la realización de la copia de seguridad será llevada a cabo de forma automática.

Buscamos ahora, otra vez dentro del manifiesto, si se han activado el resto de opciones posibles, como activación del backup automático implementando un agente de respaldo con "*android:fullBackupOnly*", o la habilitación **del backup de clave/valor**, definiendo el agente de backup en el archivo de manifiesto "*xml android:backupAgent*". Haciendo la misma búsqueda que la realizada anteriormente, no se encuentra ninguna de las definiciones comentadas en el archivo "AndroiManifest.xml".

La aplicación no presenta ningún tipo de **almacenamiento en la nube** por lo que no será necesario ningún análisis en ese sentido.

Análisis Dinámico

Para comprobar cómo realiza la aplicación la generación de backups y qué información almacena en ellos, se deberá ejecutar la aplicación con todas sus funcionalidades al menos una vez, y a

continuación, llevar a cabo los pasos pertinentes para obtener la correspondiente copia de seguridad.

El proceso de realización del backup se realizará ejecutando el comando "adb backup -apk -nosystem <package-name>"

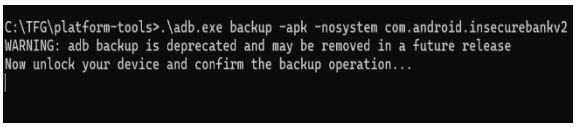


Ilustración 23: Comando adb ejecutado para obtener el backup de la app

A continuación, el sistema donde la aplicación está corriendo solicita introducir una contraseña para poder llevar a cabo el backup. Tras introducirla y seleccionar la opción "BACK UP MY DATA", se genera un archivo **backup.ab** en el directorio desde el que se ejecutó el comando.

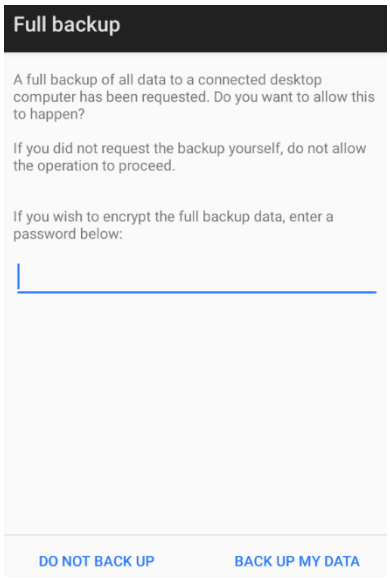


Ilustración 24: Pantalla de aceptación para la realización del backup

Para poder visualizar el contenido de este backup será necesario el uso de la herramienta "Android Backup Extractor" [34], la cual nos ayudará a desempacar el archivo .ab [49]y convertirlo en un .tar[50]. Ejecutamos el siguiente comando:

[49] Se utiliza para archivos de respaldo de datos de aplicaciones en Android, creados mediante la herramienta adb para la copia de seguridad y restauración de datos.
[50] Se utiliza para archivos de archivo TAR, que agrupan múltiples archivos en uno solo para su almacenamiento o distribución sin compresión.

```
C:\TFG\platform-tools\android-backup-extractor-master>java -jar build/libs/abe-all.jar unpack ..\backup.ab ..\backup.tar
This backup is encrypted, please provide the password
Password:

Calculated MK checksum (use UTF-8: true): 03F43424A0A0F70F570107B2850B073C2143849E0208775FD1FDD962D6C1BB2C
0% 1% 2% 3% 4% 5% 6% 7% 8% 9% 10% 11% 12% 13% 14% 15% 16% 17% 18% 19% 20% 21% 22% 23% 24% 25% 26% 27% 28% 29% 30% 31% 32% 33% 34% 35% 36% 37% 38% 39% 40% 41
% 42% 43% 44% 45% 46% 47% 48% 49% 50% 51% 52% 53% 54% 55% 56% 57% 58% 59% 60% 61% 62% 63% 64% 65% 66% 67% 68% 69% 70% 71% 72% 73% 74% 75% 76% 77% 78% 79% 80
% 81% 82% 83% 84% 85% 86% 87% 88% 89% 90% 91% 92% 93% 94% 95% 96% 97% 98% 99% 100%
3559424 bytes written to ..\backup.tar.
```

Ilustración 25: Comando utilizado para la transformación de .ab a .tar

El backup generado contiene el conjunto de archivos relativos a la aplicación completa, incluyendo todos los elementos de su **almacenamiento interno** los cuales ya analizamos en la primera prueba, su **AndroidManifest** y la propia **apk** para la descarga de la aplicación.

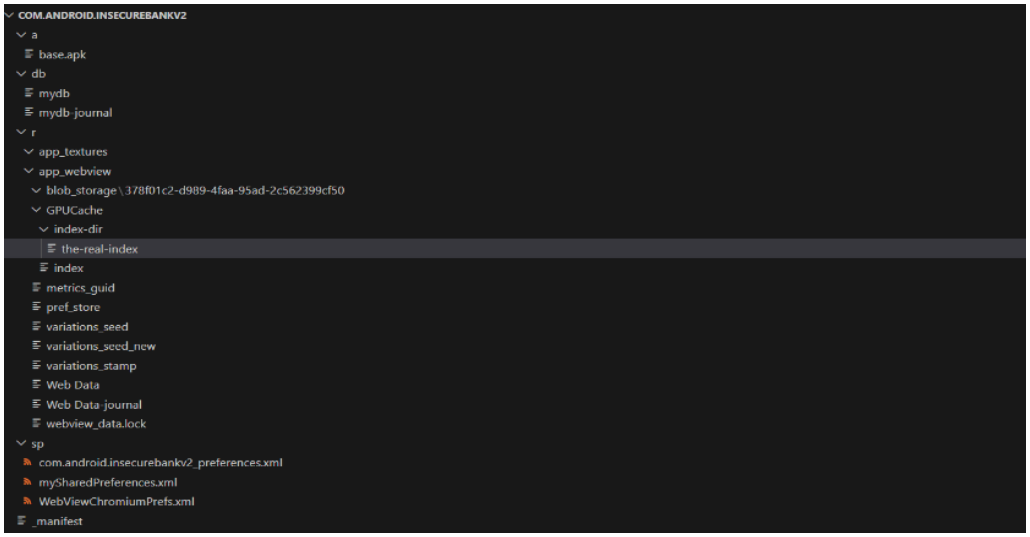


Ilustración 26: Contenido del backup obtenido

Todos los elementos encontrados en la copia del almacenamiento interno son completamente similares a los analizados anteriormente, por lo que, como ya determinamos, su exposición puede suponer un peligro no demasiado grave en clave a la privacidad de los datos de los usuarios. Por la parte del manifiesto su contenido está codificado en binario, por lo que para leerlo sería necesario un análisis forense del que se podría obtener los metadatos o información sobre la aplicación y sus componentes, como permisos o configuraciones que suelen encontrarse en el AndroidManifest de la app. Por último, el archivo base.apk se trata, presumiblemente de la aplicación completa. Para comprobar esta sospecha, se realizó un análisis en **MobSF** como el realizado sobre la apk que recibimos originalmente de la aplicación, y se obtuvo el mismo resultado que en esta. La apk de la copia de seguridad es por lo tanto la aplicación completa

Conclusiones

De este análisis sacamos, por lo tanto, que la app permite la **realización de backups** de su contenido y que dicho contenido se trata del almacenamiento interno que ya analizamos con anterioridad unido a la propia apk. Este hecho no expone de forma manifiesta información sensible o personal, pero podría suponer una puerta para la obtención de este tipo de datos.

6.1.4. [MSTG-STORAGE-10] Pruebas en memoria para buscar datos confidenciales

El propósito de esta prueba será comprobar la existencia de **información sensible dentro de la memoria interna** del sistema en el que la aplicación auditada está siendo ejecutada. La memoria interna de un sistema Android se refiere al almacenamiento no volátil utilizado por el dispositivo para guardar datos permanentes, como aplicaciones, archivos de sistema y datos de usuario. A diferencia de la memoria RAM, de carácter volátil y que se utiliza para ejecutar procesos y aplicaciones en tiempo real, la memoria interna conserva la información incluso cuando el dispositivo está apagado. Esta memoria se divide típicamente en varias particiones[51], como el sistema (donde se encuentran los archivos del sistema operativo), datos (donde se almacenan las aplicaciones instaladas y sus datos), y caché.

La memoria interna está protegida por varias **capas de seguridad**, incluyendo permisos de acceso y encriptación, para prevenir accesos no autorizados y proteger la información personal o sensible. Sin embargo, es posible encontrar vulnerabilidades que permiten el acceso a datos sensibles almacenados.

Análisis Estático

El análisis estático realizado sobre el código de la aplicación buscará principalmente encontrar donde y como se almacenan los datos que puedan considerarse sensibles a lo largo del **ciclo de ejecución** normal de la app. Para ello se deberá llevar una traza de las diferentes actividades que son ejecutadas para llevar a cabo cada una de las funcionalidades que la aplicación ofrece. Tendremos pues que comenzar desde la **actividad de arranque** definida en el manifiesto, "LoginActivity".

*Actividad **LoginActivity**:*

La actividad "LoginActivity" maneja las credenciales del usuario, almacenándolas en "SharedPreferences" y recuperándolas para iniciar sesión. Emplea campos "EditText" para la entrada del usuario y realiza el descifrado de contraseñas almacenadas utilizando "CryptoClass". El manejo de los datos personales presentes es realizado mediante objetos "String" inmutables, que pueden permanecer en la memoria más tiempo del necesario. Además, las referencias a datos personales **no se anulan ni se sobrescriben** sistemáticamente, lo que aumenta el riesgo de retención de datos en memoria no deseada.

```
SharedPreferences settings = getSharedPreferences("mySharedPreferences", 0);
String username = settings.getString("EncryptedUsername", null);
String password = settings.getString("superSecurePassword", null);
```

Ilustración 27: Parte del código en la que se obtienen username y password del usuario desde las SharedPreferences y se almacenan en un String

[51] Divisiones del almacenamiento interno de un dispositivo, cada una dedicada a diferentes funciones como el sistema operativo, datos del usuario y recuperación, facilitando la gestión y protección de la información.

Para mitigar estos riesgos, la aplicación debería almacenar estos datos personales en matrices de bytes, (byte []) en lugar de cadenas, asegurando que estas matrices se sobrescriban con datos no confidenciales antes de ser anuladas. Los métodos "fillData" (encargado de recoger la información de Login del usuario de memoria y autocompletarla) y "performlogin" (encargado de ejecutar el Login del usuario), también deberían solicitar la recolección de basura después de borrar la información confidencial.

*Actividad **DoLogin**:*

Ante un Login exitoso, el flujo de ejecución realizará una llamada a la actividad "DoLogin", que maneja el proceso de **inicio de sesión** y las acciones posteriores. Esta actividad involucra múltiples áreas donde la administración de la memoria y el manejo seguro de datos personales son críticos, como, por ejemplo, la forma en la que se procesan, almacenan y transmiten el **nombre de usuario** y la **contraseña**.

El funcionamiento de esta actividad es el siguiente: primero recupera los datos de nombre de usuario y contraseña de la actividad anterior y los procesa a través de la tarea asincrónica "RequestTask" para comunicarse con el servidor. Estas credenciales se almacenan en variables de instancia y posteriormente se utilizan para solicitudes HTTP POST[52], lo cual supone un riesgo ya que estos detalles confidenciales pueden **permanecer en la memoria** durante períodos prolongados. Durante la ejecución del método "postData", las credenciales se incluyen en una lista "nameValuePairs" para su transmisión.

Después de recibir la respuesta del servidor, esta se almacena en una variable de instancia y se analiza para determinar el resultado del inicio de sesión. Si el inicio de sesión se realiza correctamente, las credenciales se cifran y se guardan en SharedPreferences. Este proceso deja múltiples puntos donde los datos personales pueden **persistir en la memoria**.

```
if (this.serverip != null && this.serverport != null) {
    Intent data = getIntent();
    this.username = data.getStringExtra("passed_username");
    this.password = data.getStringExtra("passed_password");
    new RequestTask().execute("username");
    return;
}
```

Ilustración 29: Almacenamiento de datos personales en variables globales

```
List<NameValuePair> nameValuePairs = new ArrayList<>(2);
nameValuePairs.add(new BasicNameValuePair("username", DoLogin.this.username));
nameValuePairs.add(new BasicNameValuePair("password", DoLogin.this.password));
```

Ilustración 28: Almacenamiento de datos personales en variable de Lista

Los principales puntos peligrosos en la gestión de memoria incluyen los datos personales persistentes, en los que el nombre de usuario y la contraseña se almacenan durante todo el ciclo de vida de la actividad, y potencialmente permanecen en la memoria incluso después de que ya no sean necesarios. Durante el proceso de solicitud HTTP, el nombre de usuario y la contraseña se almacenan temporalmente en **texto sin formato** dentro de la lista "nameValuePairs", lo que aumenta el riesgo de exposición si la memoria se ve comprometida. Además, aunque las contraseñas se cifran antes de almacenarlas en SharedPreferences, el proceso de cifrado en sí mantiene temporalmente los datos no cifrados en la memoria.

Finalmente, la actividad carece de **medidas de limpieza** como el anulado de variables o sobreescritura de datos, lo que puede dejar accesible información confidencial y aumentar el riesgo de ataques basados en la memoria.

52

*Actividad **PostLogin**:*

Desde la actividad “DoLogin”, y una vez se ha comprobado que las credenciales introducidas por el usuario son las correctas, se llama a la actividad “PostLogin”, responsable de presentar al usuario **opciones e información** posteriores al inicio de sesión.

La actividad recibe el nombre de usuario y lo almacena en la variable de instancia "uname". Este nombre de usuario se utiliza luego para personalizar acciones posteriores, como transferir dinero, ver extractos o cambiar la contraseña. Todas estas acciones son accesibles mediante los distintos botones presentes en la interfaz de la actividad, y los cuales desencadenan las actividades restantes. Además de todo esto, la actividad verifica y muestra el estado de root del dispositivo, utilizando métodos que pueden requerir operaciones sensibles de memoria. Este último punto, y la utilización de la variable “uname” suponen el mayor peligro en cuanto al manejo de la memoria del sistema.

La variable “uname”, por su parte, **mantiene** el nombre de usuario en memoria durante todo el ciclo de vida de la actividad y aumenta el riesgo de que quede expuesto si la memoria se ve comprometida. Por otro lado, el proceso de detección de raíz implica ejecutar comandos del sistema y leer sus resultados en búferes, manteniendo temporalmente estos datos en la memoria. Específicamente, el método “doesSUexist” ejecuta un comando del sistema y lee su respuesta, que, aunque no es directamente sensible, implica **operaciones que podrían exponer vulnerabilidades** si no son gestionadas de forma segura.

```
this.uname = intent.getStringExtra("uname");
```

Ilustración 30: Almacenamiento del nombre de usuario en variable "uname"

A partir de la actividad “PostLogin”, como ya se ha dicho, se podrá acceder a las diferentes actividades responsables de las diferentes funcionalidades que la aplicación ofrece y cuyo cometido ya se explicó cuando presentamos la misma.

*Actividad **DoTransfer**:*

La primera de estas actividades, “DoTransfer”, es la encargada gestionar el **proceso de transferencia** de fondos entre las cuentas de los usuarios. La función principal de esta actividad es iniciar y manejar transferencias de fondos mediante tareas asincrónicas. Obtiene las entradas del usuario, que deberá introducir los detalles de las cuentas que intercambiarán fondos y la cantidad de esos fondos; a continuación, se comunicará con el backend para procesar la transferencia.

Los datos personales (nombres de usuario y contraseña), se recuperan de las preferencias compartidas, se decodifican desde Base64 y luego se utilizan en las solicitudes HTTP POST correspondientes con el servidor. Para finalizar, escribirá el estado de la transacción en un archivo almacenado en el almacenamiento del dispositivo, lo que presenta importantes problemas de seguridad y gestión de la memoria.

```
SharedPreferences settings = DoTransfer.this.getSharedPreferences("mySharedPreferences", 0);
String username = settings.getString("EncryptedUsername", null);
byte[] usernameBase64Byte = Base64.decode(username, 0);
try {
    DoTransfer.this.usernameBase64ByteString = new String(usernameBase64Byte, "UTF-8");
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
String password = settings.getString("superSecurePassword", null);
try {
    DoTransfer.this.passNormalized = DoTransfer.this.getNormalizedPassword(password);
} catch (UnsupportedEncodingException | InvalidAlgorithmParameterException | InvalidKeyException
    e1.printStackTrace();
}
```

Ilustración 31: Almacenamiento de username y password del usuario en variables String

```
List<NameValuePair> nameValuePairs = new ArrayList<>(2);
nameValuePairs.add(new BasicNameValuePair("username", DoTransfer.this.usernameBase64ByteString));
nameValuePairs.add(new BasicNameValuePair("password", DoTransfer.this.passNormalized));
```

Ilustración 32: Almacenamiento de username y password del usuario en variable Lista

Durante el proceso descrito, surgen una variedad de problemas relativos al manejo de información personal y el uso de tareas asincrónicas. La actividad **decodifica y almacena** el nombre de usuario y la contraseña en la memoria, donde permanecen **accesibles** durante todo el proceso de transferencia. Además, la actividad lee y escribe datos de flujos de entrada y buffers, como en el método "*convertStreamToString*" y la escritura de registros de transacciones, reteniendo datos temporalmente en la memoria.

```
public String convertStreamToString(InputStream in) throws IOException {
    try {
        this.reader = new BufferedReader(new InputStreamReader(in, "UTF-8"));
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    StringBuilder sb = new StringBuilder();
    while (true) {
        String line = this.reader.readLine();
        if (line != null) {
            sb.append(line + "\n");
        } else {
            in.close();
            return sb.toString();
        }
    }
}
```

Ilustración 33: Clase "convertStreamToStrng" que convierte en String datos de un flujo de entrada

*Actividad **ChangePassword**:*

La siguiente actividad accesible desde la interfaz principal de la aplicación se trata de la actividad "ChangePassword", cuyo cometido es **facilitar los cambios de contraseña** a los usuarios. Cuando un usuario introduce la nueva contraseña que quiere para su cuenta y la envía al servidor, la aplicación validará la nueva contraseña según los criterios de complejidad predefinidos y si la contraseña se considera segura, se enviará la solicitud de cambio al servidor. Una vez recibida la respuesta del servidor, esta será procesada por la aplicación y se proporcionarán los comentarios pertinentes al usuario. Este proceso implica varias interacciones con preferencias compartidas, elementos de la interfaz de usuario y operaciones de red.

Por lo tanto, existen ciertos problemas en la administración de la memoria a lo largo de esta implementación. La **información personal**, como el nombre de usuario y la nueva contraseña, se

almacena en la memoria sin cifrar, lo cual representa un riesgo si la memoria se ve comprometida. El método "*broadcastChangepasswordSMS*", que transmite la nueva contraseña a través de un intento de SMS, es particularmente problemático ya que los intentos pueden ser interceptados por otras aplicaciones en el dispositivo, lo que lleva a una posible exposición de información personal del usuario.

```
List<NameValuePair> nameValuePairs = new ArrayList<>(2);
nameValuePairs.add(new BasicNameValuePair("username", ChangePassword.this.uname));
nameValuePairs.add(new BasicNameValuePair("newpassword", ChangePassword.this.changePassword_text.getText().toString()));
```

Ilustración 34: Almacenamiento de username y password del usuario en variable Lista

```
/* JADX INFO: Access modifiers changed from: private */
public void broadcastChangepasswordSMS(String phoneNumber, String pass) {
    if (TextUtils.isEmpty(phoneNumber.toString().trim())) {
        System.out.println("Phone number Invalid.");
        return;
    }
    Intent smsIntent = new Intent();
    smsIntent.setAction("theBroadcast");
    smsIntent.putExtra("phonenumber", phoneNumber);
    smsIntent.putExtra("newpass", pass);
    sendBroadcast(smsIntent);
}
```

Ilustración 35: Clase para la transmisión de la contraseña del usuario vía SMS

*Actividad **ViewStatement**:*

La última actividad accesible desde este "menú" que resulta ser la interfaz principal de la app, se trata de la actividad "ViewStatement", que permite a los usuarios **ver sus extractos bancarios** cargando un archivo HTML desde el almacenamiento externo. Cuando la actividad se crea, esta recupera el nombre de usuario, construye el nombre de archivo para la declaración HTML y verifica su existencia en el almacenamiento. Si el archivo existe, configura un "WebView" para mostrar el "statement"[53] con los extractos. Si el archivo no existe, la actividad redirige al usuario a la actividad "PostLogin".

```
if (fileToCheck.exists()) {
    WebView mWebView = (WebView) findViewById(2131558530);
    mWebView.loadUrl("file://" + Environment.getExternalStorageDirectory() + "/Statements_" + this.uname + ".html");
    mWebView.getSettings().setJavaScriptEnabled(true);
    mWebView.getSettings().setSaveFormData(true);
    mWebView.getSettings().setBuiltInZoomControls(true);
    mWebView.setWebViewClient(new MyWebViewClient());
    WebChromeClient cClient = new WebChromeClient();
    mWebView.setWebChromeClient(cClient);
```

Ilustración 36: Sección del código para mostrar los extractos bancarios en la interfaz

Esta actividad presenta ciertos fallos relativos a la gestión de la memoria de la aplicación. En primer lugar, **almacenar archivos sensibles**, como extractos bancarios, en un almacenamiento externo resulta bastante inseguro, ya que cualquier aplicación con permisos de almacenamiento podría acceder a ellos. Además, existe la posibilidad de que se guarde información personal o

[53] Documento que detalla las transacciones financieras de una cuenta bancaria durante un período específico, proporcionando un registro completo de ingresos, gastos y saldos

sensible en los datos del formulario, a los que pueden acceder aplicaciones o usuarios maliciosos si el dispositivo se ve comprometido.

Conclusiones

Encontramos a lo largo de este análisis una gran multitud de errores relativos a la configuración de las operaciones con memoria por parte de la aplicación. Diversos datos personales, como el username y la passwprd del usuario se almacenan en variables sin ningún tipo de medidas contra la persistencia en memoria.

Para la mitigación de todos estos problemas, la aplicación debería implementar un almacenamiento seguro para datos personales utilizando el sistema "Keystore" de Android [35] y garantizar que todas las comunicaciones del servidor estén cifradas mediante HTTPS. El manejo de la memoria debería mejorarse borrando los datos personales de la memoria tan pronto como ya no sean necesarios [36]. Además, se debería evitar la práctica de transmitir información personal de los usuarios entre actividades a través de intents para evitar la interceptación.

6.1.5. [MSTG-CRYPTO-2, MSTG-CRYPTO-3, MSTG-CRYPTO-4] Prueba de la configuración de algoritmos estándar criptográficos

A lo largo de este apartado nos centraremos en evaluar cómo se implementan y gestionan los **algoritmos criptográficos** en la aplicación, asegurando que las prácticas empleadas cumplan con los estándares actuales de la industria y las mejores prácticas recomendadas. En el contexto de la auditoría de una aplicación móvil, el análisis del uso de algoritmos criptográficos constituye una pieza fundamental para garantizar la **seguridad y privacidad** de los datos manejados, puesto que esta resulta esencial para proteger la información sensible presente dentro de la misma.

Análisis Estático

El análisis que se llevará a cabo, buscará asegurar que los algoritmos criptográficos empleados para la encriptación de información de carácter personal cumplen unos **requisitos de seguridad** mínimos, identificando todos los puntos de la app en los que se estén empleando, definiendo el tipo de algoritmo utilizado y su seguridad respecto a los estándares actuales; y comprobando que los algoritmos criptográficos empleados sean robustos frente a ataques.

Mediante la exploración del código fuente se tratará de identificar todas las **instancias de implementaciones criptográficas**, con el objetivo de comprobar su funcionamiento dentro de la aplicación, y el proceso de encriptación[54] que se sigue, tratando así de encontrar los posibles errores que pudiesen resultar en la exhibición de los datos personales que se pretendían esconder mediante la encriptación.

Para llevar a cabo estos objetivos se buscarán los siguientes elementos a lo largo del código:

- Clases “Cipher”, “Mac”, “MessageDigest” y “Signature”:

    - La clase “**Cipher**” es empleada para realizar operaciones de cifrado y descifrado de datos. A lo largo del código fuente esta clase se emplea en funciones que requieren que un texto codificado en “AES” [55] sea encriptado o en su defecto desencriptado. Estas funciones, de nombre “aes256decrypt” y “aes256encrypt”, recogen una matriz de bytes para almacenar las especificaciones de los parámetros de encriptación, “ivBytes”, una clave secreta, “keyBytes” y el texto a cifrar o descifrar, “textBytes”.

[54] Proceso de seguridad que transforma datos en formato legible en una forma ilegible (cifrado) utilizando algoritmos matemáticos y claves, para proteger la confidencialidad y seguridad de la información.
[55] Algoritmo de encriptación simétrica ampliamente adoptado para proteger datos mediante operaciones matemáticas avanzadas.

```
public static byte[] aes256decrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes)
    AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
    SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(2, newKey, ivSpec);
    return cipher.doFinal(textBytes);
}
```

Ilustración 37: Clase encargada de llevar a cabo la encriptación en AES256

```
public static byte[] aes256encrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes)
    AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
    SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(1, newKey, ivSpec);
    return cipher.doFinal(textBytes);
}
```

Ilustración 38: Clase encargada de llevar a cabo la desencriptación den AES256

La lectura más interesante que nos brinda el estudio del uso de esta clase en el código es la existencia de una variable de **clave secreta**. Rastreando dentro de los códigos en los que hemos encontrado la clase "Cipher", se encuentra fácilmente la presencia de una variable "keyBytes" o "Key" que guarda el siguiente valor: "This is the super secret key 123". Conocer esta información será de mucha utilidad más adelante cuando probemos a **desencriptar** el mensaje que encontramos en la primera prueba de esta auditoría y que no fuimos capaces de revelar en la misma.

```
byte[] keyBytes = "This is the super secret key 123".getBytes("UTF-8");

String key = "This is the super secret key 123";
```

Ilustración 39: Muestra de claves en texto claro encontradas en el código

Otro descubrimiento que resulta tremendamente importante es el hecho de conocer el algoritmo de encriptación que la app puede estar empleando a la hora de cifrar diversos datos confidenciales.

- La clase "**MessageDigest**" se utiliza para generar **resúmenes** de mensaje o hashes[56], que son representaciones compactas y únicas de datos. A lo largo del código fuente, "MessageDigest" se emplea para generar hashes MD5, con propósitos como la creación de un hash para un arreglo de bytes que posteriormente se combina con datos adicionales o la creación de un hash MD5 de una cadena de texto. El uso de MessageDigest con el algoritmo MD5 presenta **riesgos** significativos para la privacidad y seguridad, puesto que MD5 es un algoritmo hash criptográficamente roto y vulnerable a **colisiones** (diferentes entradas pueden producir el mismo hash). Esto hace de MD5 un algoritmo que no resulta adecuado para propósitos de seguridad como la verificación de integridad de datos sensibles.

[56] Función matemática que convierte datos de cualquier longitud en una cadena de longitud fija, utilizada principalmente para verificar la integridad de datos y proteger la información mediante la creación de un identificador único y difícil de revertir.

```
public String zzax(String str) {
    for (int i = 0; i < 2; i++) {
        try {
            MessageDigest messageDigest = MessageDigest.getInstance("MD5");
            messageDigest.update(str.getBytes());
            return String.format(Locale.US, "%032X", new BigInteger(1, messageDigest.digest()));
        } catch (NoSuchAlgorithmException e) {
        }
    }
    return null;
}
```

Ilustración 40: Muestra de código en el que se usa la clase MessageDigest

- La clase "**Signature**" se utiliza para verificar y crear **firmas digitales**[57], con el objetivo de asegurar la autenticidad e integridad de los datos firmados. En los diversos usos de esta clase a lo largo del código fuente, se emplea para verificar la autenticidad de datos firmados, tanto con una clave pública como con una privada. Cuando se emplea la **clave pública** como medio de firma, se hace uso del algoritmo **SHA-1**[58], el cual presenta ciertas preocupaciones debido a algunas vulnerabilidades.

```
public static boolean zza(PublicKey publicKey, String str, String str2) {
    try {
        Signature signature = Signature.getInstance("SHA1withRSA");
        signature.initVerify(publicKey);
        signature.update(str.getBytes());
        if (signature.verify(Base64.decode(str2, 0))) {
            return true;
        }
        com.google.android.gms.ads.internal.util.client.zzb.zzaz("Signature verification failed.");
        return false;
    } catch (InvalidKeyException e) {
        com.google.android.gms.ads.internal.util.client.zzb.zzaz("Invalid key specification.");
        return false;
    } catch (NoSuchAlgorithmException e2) {
        com.google.android.gms.ads.internal.util.client.zzb.zzaz("NoSuchAlgorithmException.");
        return false;
    } catch (SignatureException e3) {
        com.google.android.gms.ads.internal.util.client.zzb.zzaz("Signature exception.");
        return false;
    }
}
```

Ilustración 41: Uso de la clase "Signature" para realizar firmas con clave pública

[57] Componente criptográfico que garantiza la autenticidad, integridad y no repudio de un documento digital, proporcionando una forma segura de verificar la identidad del remitente y la integridad del contenido.
[58] Algoritmo de hash criptográfico que convierte datos de cualquier longitud en una cadena de 160 bits, utilizado para verificar la integridad y autenticidad de la información.

Por otro lado, para las firmas de datos con una **clave privada**, el algoritmo empleado es el **SHA-256**, lo que, unido al uso de las claves privadas, garantiza la autenticidad y la integridad de la información. [37]

```
static String zza(KeyPair keyPair, String... strArr) {
    try {
        byte[] bytes = TextUtils.join("\n", strArr).getBytes("UTF-8");
        try {
            PrivateKey privateKey = keyPair.getPrivate();
            Signature signature = Signature.getInstance(privateKey instanceof RSAPrivateKey ? "SHA256withRSA" : "SHA256withECDSA");
            signature.initSign(privateKey);
            signature.update(bytes);
            return InstanceID.zzm(signature.sign());
        } catch (GeneralSecurityException e) {
            Log.e("InstanceID/Rpc", "Unable to sign registration request", e);
            return null;
        }
    } catch (UnsupportedEncodingException e2) {
        Log.e("InstanceID/Rpc", "Unable to encode string", e2);
        return null;
    }
}
```

Ilustración 42: Uso de la clase "Signature" para realizar firmas con clave privada

- Instancias “PrivateKey”, “PublicKey” y “SecretKey”:

  - Las instancias “**PrivateKey**” y “**PublicKey**”, como ya se ha mostrado anteriormente, se emplean para realizar la firma digital sobre ciertos archivos cuyo valor se quiere asegurar. El proceso de las firmas digitales utiliza un par de claves, pública y privada, con el fin de asegurar la autenticidad e integridad de un mensaje o documento. Para ello el encargado de crear la firma digital cifra el hash del documento con su clave privada, garantizando así que solo él la haya podido haber creado. Al recibir el documento, el proceso, utilizando la clave pública enviada por el remitente, descifra la firma digital para obtener el hash original y lo compara con un hash generado a partir del documento recibido. Si ambos hashes coinciden, se confirma que el mensaje no ha sido alterado y que proviene auténticamente del remitente.
  - La instancia “**SecretKey**” aparece en el código en forma de “SecretKeySpec”. La funcionalidad y objetivo de la misma es la creación de una clave secreta a partir de una matriz de bytes y la especificación de qué algoritmo de encriptación emplear. En nuestra aplicación esta instancia se emplea en las funciones de **encriptación** y **desencriptación** “aes256” que se analizaron cuando veíamos la clase “Cipher”, donde se pasaba el valor de la clave secreta “This is the super secret key 123”, y el algoritmo AES como método de encriptación.

```
SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
```

Ilustración 43: Uso de la instancia "SecretKey" en el código

- Funciones “getInstance” y “generateKey”:
  - La función “**getInstance**()” es empleada en el marco de la criptografía para obtener instancias de algoritmos de cifrado, **generación de claves** y **hashing**. Esta función proporciona una forma de acceso a los diferentes algoritmos criptográficos disponibles, permitiendo a los desarrolladores trabajar con ellos de manera consistente. A lo largo del código esta función se emplea varias veces en combinación de otras que ya hemos visto anteriormente, haciendo uso de la instancia de algoritmo criptográfico que se especifica.

```
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
MessageDigest messageDigest = MessageDigest.getInstance("MD5");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
zzrK = MessageDigest.getInstance("MD5");
```

Ilustración 44: Uso de la clase "getInstance" en el código

En estos ejemplos podemos observar cómo gracias a la función "getInstance" clases como "MessageDigest", "Cipher" o "KeyFactory" pueden hace uso de algoritmos criptográficos como "RSA", "MD5" o "AES".

- La función "**generateKey**" no se emplea a lo largo del código.

Gracias a el análisis realizado sobre las clases, funciones e instancias anteriores, comprendemos de qué forma está **gestionado** el uso de la criptografía a lo largo del código fuente de la aplicación. También hemos encontrado la presencia de una **clave de encriptación** escrita de forma clara en código, por lo que puede ser interesante intentar buscar donde se usa esta clave y tratar de descifrar lo que pretende encriptar.

Esta clave, es empleada a la hora de encriptar o desencriptar con el algoritmo AES una instancia desconocida. Este proceso se realizaba mediante las funciones "aes256encrypt" y "aes256decrypt" definidas en la clase "CryptoClass", las cuales recibían dos cadenas de bytes, una con la cadena a cifrar o descifrar y otra con la clave de cifrado, y devolvían una cadena con el resultado.

Rastreando dónde son empleadas estas funciones, se han encontrado otras dos funciones de la clase "CryptoClass", "**aesDecryptedString**", la cual **descifra** un texto cifrado (en formato Base64) usando AES con una clave específica y devuelve el texto plano; y "**aesEncryptedString**", que **cifra** un texto plano utilizando AES con una clave específica y devuelve el texto cifrado en formato Base64.

```
public String aesDeccryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyException
    byte[] keyBytes = this.key.getBytes("UTF-8");
    this.cipherData = aes256decrypt(this.ivBytes, keyBytes, Base64.decode(theString.getBytes("UTF-8"), 0));
    this.plainText = new String(this.cipherData, "UTF-8");
    return this.plainText;
}

public String aesEncryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyException,
    byte[] keyBytes = this.key.getBytes("UTF-8");
    this.plainText = theString;
    this.cipherData = aes256encrypt(this.ivBytes, keyBytes, this.plainText.getBytes("UTF-8"));
    this.cipherText = Base64.encodeToString(this.cipherData, 0);
    return this.cipherText;
}
```

Ilustración 45: Clases empleadas por la app para la encriptación y desencriptación de Strings

Puede observarse en el código como a las funciones "aes256" recibe tres variables siendo la segunda, "keyBytes", la que debe contener la **clave secreta de encriptación**. Investigamos pues cual es valor de la variable "this.key" y efectivamente es nuestra clave.

```
String key = "This is the super secret key 123";
byte[] ivBytes = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

Ilustración 46: Variables "key" y "ivBytes" empleadas en la encriptación y desencriptación

El siguiente paso de esta búsqueda, es comprobar dónde se están utilizando estas dos nuevas funciones en el código. La función "aesDecryptedString" se utiliza tres veces en tres actividades distintas: "DoTransfer", "LoginActivity" y "MyBroadCastReceiver". Aunque su uso final difiera, la función que tiene en estas actividades es descifrar una cadena de texto de nombre "password". Esta variable se obtiene al inicio del código de cada una de estas clases y se trata de la instancia "superSecurePassword" que tratamos de revelar con anterioridad sin éxito.

```
String password = settings.getString("superSecurePassword", null);
CryptoClass crypt = new CryptoClass();
String decryptedPassword = crypt.aesDeccryptedString(password);
```

Ilustración 47: Parte del código en la que se usa la clase "aesDecryptedSring" para desencriptar una contraseña

Por parte de la función "aesEncryptesString", es utilizada solamente una vez en la actividad "DoLogin", cuando los usuarios intentan acceder a la aplicación y sus credenciales son almacenadas. Una vez más, se emplea como clave de cifrado una variable de nombre "password", la cual esta vez se corresponde con la contraseña introducida por los usuarios.

```
DoLogin.this.rememberme_username = username;
DoLogin.this.rememberme_password = password;
String base64Username = new String(Base64.encodeToString(DoLogin.this.rememberme_username.getBytes(), 4));
CryptoClass crypt = new CryptoClass();
DoLogin.this.superSecurePassword = crypt.aesEncryptedString(DoLogin.this.rememberme_password);
editor.putString("EncryptedUsername", base64Username);
editor.putString("superSecurePassword", DoLogin.this.superSecurePassword);
```

Ilustración 48: Parte del código en la que se usa la clase "aesEncryptedSring" para encriptar una contraseña

Obtención de datos encriptados:

Puesto que conocemos el **algoritmo de cifrado** empleado, la **clave de cifrado** que se está utilizando y el **valor encriptado de la contraseña** (v/sJpihDCo2ckDmLW5Uwiw==) podemos proceder a descifrar que es lo que esconde dicha contraseña. Para ello, en primer lugar, debemos obtener el valor decodificado de base64 a hexadecimal para poder introducirlo en la herramienta de desencriptación **CyberChef**. Empleo la herramienta online base64.guru [38] y obtengo:

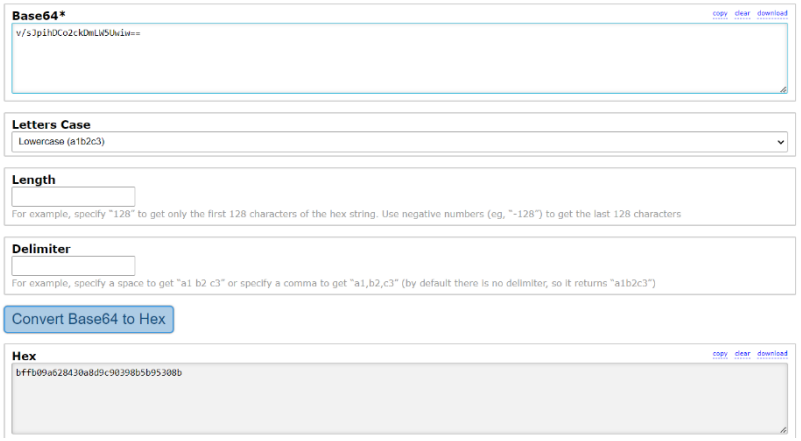


Ilustración 49: Interfaz de la app web "base64" con la que desciframos el texto en base 64

Este valor hexadecimal deberá introducirse a modo de input por bytes de la siguiente forma: “bf fb 09 a6 28 43 0a 8d 9c 90 39 8b 5b 95 30 8b”. Con esto ya solo faltaría introducir el resto de parámetros y definir correctamente los formatos a emplear.

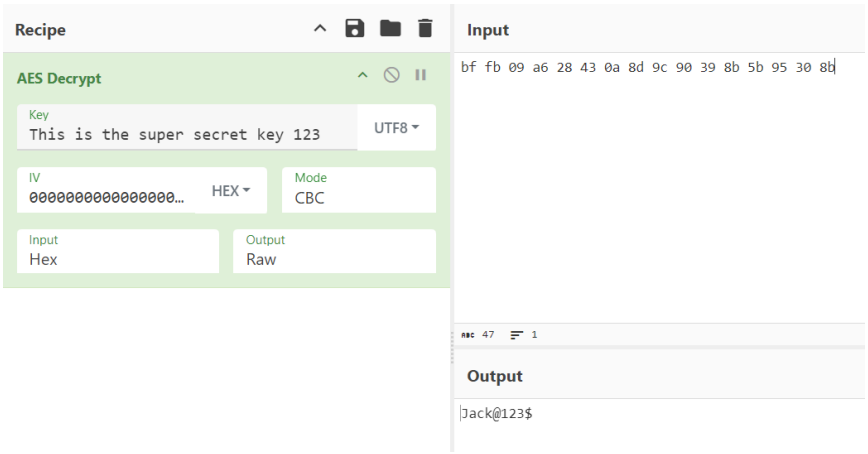


Ilustración 50: Interfaz de la herramienta CypherChef con todos los parámetros y resultado

Como puede apreciarse, hemos obtenido que la clave que se escondía era “**Jack@123$**”, que coincide con la contraseña predefinida por la aplicación para el nombre de usuario “**jack**” que ya se había encontrado previamente. Con esto queda claro que el método de cifrado empleado por esta aplicación es vulnerable y poco fiable, sobre todo a la hora de almacenar la clave secreta.

Conclusiones

En el análisis del cumplimiento de los estándares de seguridad actuales por parte de los algoritmos de cifrados empleados y si son recomendados por la industria, la implementación del algoritmo AES con una clave de 256 bits, como se observa en las funciones "aes256decrypt" y "aes256encrypt", es adecuada y reconocida con las mejores prácticas criptográficas modernas. AES-256 es reconocido por su robustez y resistencia frente a ataques, siempre y cuando las claves

se manejen de forma segura, lo cual no se cumple en este código, puesto que existe la presencia de una clave de encriptación "This is the super secret key 123" escrita de forma clara en el mismo. Esto supone un **punto crítico** y como ya se ha visto supone el descifrado de la contraseña de uno de los usuarios.

El uso del algoritmo MD5 para generar hashes mediante la clase “MessageDigest” también supone un punto de debilidad significativo, ya que MD5 es un algoritmo vulnerable a ataques de colisión y no debe usarse para propósitos de seguridad. En cuanto a la firma digital, aunque el uso del algoritmo SHA-256 en combinación con claves privadas es adecuado, el uso de SHA-1 con claves públicas no es recomendable, debido a las vulnerabilidades conocidas de SHA-1.

6.1.6. [MSTG-NETWORK-1] Prueba del cifrado de datos en la red

La **comunicación** entre cliente y servidor se trata de una de las partes más fundamentales para el correcto funcionamiento de una app. En diversas ocasiones, la información intercambiada entre estas dos partes contiene datos de carácter confidencial para la aplicación o de carácter personal o sensible para los usuarios. Es por ello que resulta indispensable analizar correctamente este **flujo de datos** intercambiados para poder así asegurar que ningún tercero pueda interceptarlo y obtener para sí mismo toda esta información intercambiada. El método empleado para evitar que esto ocurra es principalmente el cifrado de todos los datos salientes y generalmente esto es realizado mediante el uso del protocolo HTTPS.

Análisis Estático

Aunque para la realización de esta prueba lo que podría parecer más natural sería realizar simplemente un análisis dinámico del tráfico intercambiado entre cliente y servidor [39], comprobando la información que se mueve entre extremos, la búsqueda de errores en el código a la hora de llevar a cabo las solicitudes de red resulta también tremendamente importante puesto que nos permitirá tener una visión crítica de cómo se están realizando todas las solicitudes al servidor desde la aplicación.

Análisis de Solicitudes de Red:

Comenzaremos pues identificando todas las **solicitudes de red** realizadas desde el código mediante URLS. Para ello se realizará la búsqueda textual de las solicitudes “http” y “https” presentes a lo largo del código.

```
com.android.insecurebankv2.ChangePassword        String protocol = "http://";
com.android.insecurebankv2.DoLogin               String protocol = "http://";
com.android.insecurebankv2.DoTransfer            String protocol = "http://";
```

Ilustración 51: Empleo de protocolo "http" en actividades de la app

```
com.google.android.gms.appindexing.Action        public static final String TYPE_SEARCH = "http://schema.org/SearchAction";
com.google.android.gms.appindexing.Action        public static final String TYPE_VIEW = "http://schema.org/ViewAction";
com.google.android.gms.appindexing.Action        public static final String TYPE_WANT = "http://schema.org/WantAction";
com.google.android.gms.appindexing.Action        public static final String TYPE_WATCH = "http://schema.org/WatchAction";
com.google.android.gms.common.internal.zzm       private static final Uri zzaaV = Uri.parse("http://plus.google.com/");
```

Ilustración 53: Empleo del protocolo "http" en actividades google

```
com.google.android.gms.common.Scopes             public static final String PLUS_LOGIN = "https://www.googleapis.com/auth/plus.login";
com.google.android.gms.common.Scopes             public static final String PLUS_ME = "https://www.googleapis.com/auth/plus.me";
com.google.android.gms.common.Scopes             public static final String PLUS_MOMENTS = "https://www.googleapis.com/auth/plus.moments.write";
com.google.android.gms.drive.Drive               public static final Scope zzacY = new Scope("https://www.googleapis.com/auth/drive");
com.google.android.gms.drive.Drive               public static final Scope zzacZ = new Scope("https://www.googleapis.com/auth/drive.apps");
com.google.android.gms.games.Games               public static final Scope zzanx = new Scope("https://www.googleapis.com/auth/games.firstparty");
com.google.android.gms.games.internal.GamesClientImpl.zza(Set<Scope>) Set<Sc  Scope scope2 = new Scope("https://www.googleapis.com/auth/games.firstparty");
```

Ilustración 52: Empleo del protocolo "https" en el código fuente

El resultado de la búsqueda textual de los protocolos nos muestra cómo para las peticiones realizadas desde las actividades responsables de la funcionalidad de la aplicación, la misma está empleando el protocolo "**http**", más específicamente, en las actividades "DoLogin", "ChangePassword" y "DoTransfer". Este protocolo, como se muestra en la imagen anterior se emplea en más ocasiones a lo largo del código, pero ninguna de ellas supone una comunicación directa con el servidor. Por su parte, el protocolo "https" también se emplea en diversas ocasiones, pero tampoco en ninguna de ellas la comunicación es con el servidor.

Estudio de Comunicación con el Servidor:

Rastreando las llamadas "**http**" realizadas en las actividades anteriormente señaladas, se encuentra como el string "protocol" al que se le asigna el valor de "http://" se utiliza más adelante en la función "**httpPost**"[59] junto con la dirección ip del servidor, por lo que se puede concluir que todas las comunicaciones realizadas con el servidor hacen uso de este protocolo inseguro.

```
HttpPost httppost = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/login");
HttpPost httppost2 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/devlogin");
```

Ilustración 54: Uso de la función httPost en el código

Continuando con el análisis del método de comunicación con el servidor, revisaremos si al menos esta se realiza a través de canales seguros como los obtenidos al hacer uso de "**HttpsURLConnection**" o "**SSLSocket**". Sorprendentemente, ninguna de estas dos clases es utilizada para llevar a cabo peticiones en la aplicación. Alternativamente a estos dos métodos la aplicación hace uso de la clase "**HttpURLConnection**"[60], la cual crea un canal de transmisión para el protocolo "http".

```
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setConnectTimeout(60000);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setInstanceFollowRedirects(z);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setReadTimeout(60000);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setRequestProperty("User-Agent", str2);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setUseCaches(false);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   public void zza(Context context, String str, boolean z, HttpURLConnection httpURLConnection, boolean z2) {
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setConnectTimeout(60000);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setInstanceFollowRedirects(z);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setReadTimeout(60000);
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setRequestProperty("User-Agent", zzf(context, str));
com.google.android.gms.internal.zzhl.zza(Context, String, boolean, HttpURLCo   httpURLConnection.setUseCaches(z2);
com.google.android.gms.internal.zzhp                                           import java.net.HttpURLConnection;
com.google.android.gms.internal.zzhp.zzdP() void                               HttpURLConnection httpURLConnection = (HttpURLConnection) new URL(this.zzF).openConnection();
com.google.android.gms.internal.zzhp.zzdP() void                               com.google.android.gms.ads.internal.zzo.zzbv().zza(this.mContext, this.zzqr, true, httpURLConnection);
com.google.android.gms.internal.zzhp.zzdP() void                               com.google.android.gms.ads.internal.zzo.zzbv().zza(this.mContext, this.zzqr, true, httpURLConnection, this.zzFP);
com.google.android.gms.internal.zzhp.zzdP() void                               int responseCode = httpURLConnection.getResponseCode();
com.google.android.gms.internal.zzhp.zzdP() void                               httpURLConnection.disconnect();
com.google.android.gms.analytics.internal.zzah.zzc(URL) HttpURLConnection       httpURLConnection.setRequestProperty("User-Agent", this.zzFP);
com.google.android.gms.analytics.internal.zzah.zzc(URL) HttpURLConnection       httpURLConnection.setDoInput(true);
com.google.android.gms.analytics.internal.zzah.zzc(URL) HttpURLConnection       return httpURLConnection;
```

Ilustración 55: Uso de la clase "HttpURLConnection" a lo largo del código fuente

Aunque ciertamente el canal de transmisión empleado mayoritariamente por la aplicación es el definido por "HttpURLConnection", su uso se ve relacionado totalmente a las funcionalidades y servicios del paquete "com.google.android", el cual sirve para ofrecer a las aplicaciones Android acceso a todos los servicios que ofrece Google. Si queremos fijarnos en los canales de transmisión empleados por la infraestructura de la app encargada de llevar a cabo sus funcionalidades, será necesario analizar cómo realizan las actividades de la misma todas sus peticiones al servidor.

[59] Clase en Android que representa una solicitud HTTP del tipo POST, utilizada para enviar datos desde una aplicación a un servidor web, permitiendo la transferencia segura y la actualización de recursos en línea.
[60] Clase en Android que facilita la creación y gestión de conexiones HTTP para enviar y recibir datos entre una aplicación y un servidor web, proporcionando métodos para configurar y realizar solicitudes HTTP de manera eficiente y segura.

Como se ha podido apreciar antes, la forma de enviar peticiones al servidor desde la aplicación es el uso del método "**httpPost**", la cual es una clase de la biblioteca Apache HttpClient que se utiliza para realizar solicitudes de envió de datos al servidor. La comunicación se realiza mediante el protocolo HTTP, que no cifra los datos en tránsito, lo que implica que la información sensible pueda ser interceptada fácilmente por atacantes en la red.

```
public void postData(String valueIWantToSend) throws ClientProtocolException, IOException, JSONException, InvalidKeyException, NoSuchAlgorithmException, NoSuch
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost(ChangePassword.this.protocol + ChangePassword.this.serverip + ":" + ChangePassword.this.serverport + "/changepassword");
    List<NameValuePair> nameValuePairs = new ArrayList<>(2);
    nameValuePairs.add(new BasicNameValuePair("username", ChangePassword.this.uname));
    nameValuePairs.add(new BasicNameValuePair("newpassword", ChangePassword.this.changePassword_text.getText().toString()));
    httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    ChangePassword.this.pattern = Pattern.compile(ChangePassword.PASSWORD_PATTERN);
    ChangePassword.this.matcher = ChangePassword.this.pattern.matcher(ChangePassword.this.changePassword_text.getText().toString());
    boolean isStrong = ChangePassword.this.matcher.matches();
    if (isStrong) {
        HttpResponse responseBody = httpclient.execute(httppost);
```

Ilustración 56: Muestra de comunicación con el servidor para llevar a cabo un cambio de contraseña

```
public void postData(String valueIWantToSend) throws ClientProtocolException, IOException, JSONException, InvalidKeyException, NoSu
    HttpResponse responseBody;
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/login");
    HttpPost httppost2 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/devlogin");
    List<NameValuePair> nameValuePairs = new ArrayList<>(2);
    nameValuePairs.add(new BasicNameValuePair("username", DoLogin.this.username));
    nameValuePairs.add(new BasicNameValuePair("password", DoLogin.this.password));
    if (DoLogin.this.username.equals("devadmin")) {
        httppost2.setEntity(new UrlEncodedFormEntity(nameValuePairs));
        responseBody = httpclient.execute(httppost2);
    } else {
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
        responseBody = httpclient.execute(httppost);
    }
```

Ilustración 57: Muestra de comunicación con el servidor para llevar a cabo un Login

Los dos códigos anteriores se corresponden con las clases "postData" de las actividades "ChangePassword" y "DoLogin" respectivamente, en las cuales se lleva a cabo el envío de **credenciales de usuarios** al servidor. La arquitectura de red empleada en la actividad "DoTransfer" para la realización de transferencias entre usuarios es similar a la mostrada, solo que enviando los datos referentes a la transferencia (cuentas y número de teléfono).

Como puede apreciarse, el proceso para la realización de la solicitud empleada es realmente simple: primero se define una instancia **HttpClient** con la que se ejecutará la petición. A continuación, se crea una variable de tipo **HttpPost** en la que se introducen todos los datos relativos a la comunicación que está a punto de suceder, como su protocolo, la dirección y puerto del servidor, y la vista del servidor encargada de atender la petición ("/login", "/devlogin", "/changepassword"…).

Para finalizar la configuración del **POST**, solamente faltaría introducir la información que se desea transmitir, la cual se almacena en una lista de nombre "nameValuePairs" y que se añadirá a la variable "httpPost" mediante la declaración:
"httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs))".

Por último, para el envío de la solicitud al servidor se emplea la función "httpclient.execute(httppost)".

Siguiendo el proceso de envío de información al servidor podemos concluir que esta se realiza mediante un **protocolo inseguro** y mediante un canal de transmisión en el que no se cuida de ninguna forma la sensibilidad y confidencialidad de los datos, que viajarán al servidor sin ningún tipo de cifrado, permitiendo que cualquier atacante "**man in the middle**" pueda interceptarlos fácilmente.

Gracias a haber rastreado cómo se realiza el envío de información, se ha podido encontrar también como la aplicación recibe las **respuestas** del servidor. Este proceso se realiza mediante la entidad "**HttpResponse**", que se define antes de llevar a cabo la petición al servidor y cuyo valor se recibe al ejecutar la misma ("responseBody = httpclient.execute(httppost)"). El contenido de la respuesta se obtiene mediante la declaración: "responseBody.getEntity().getContent()",

que devuelve un flujo de entrada (InputStream) representando la respuesta del servidor.

```
HttpResponse responseBody = httpclient.execute(httppost);
InputStream in = responseBody.getEntity().getContent();
ChangePassword.this.result = convertStreamToString(in);
ChangePassword.this.result = ChangePassword.this.result.replace("\n", "");
ChangePassword.this.runOnUiThread(new Runnable() { // from class: com.android.insecurebankv2.ChangePassword.RequestChangePasswordTask.1
    @Override // java.lang.Runnable
    public void run() {
        if (ChangePassword.this.result != null && ChangePassword.this.result.indexOf("Change Password Successful") != -1) {
            try {
                JSONObject jsonObject = new JSONObject(ChangePassword.this.result);
                String login_response_message = jsonObject.getString("message");
                Toast.makeText(ChangePassword.this.getApplicationContext(), login_response_message + ". Restart application to Continue.", 1).show();
                TelephonyManager phoneManager = (TelephonyManager) ChangePassword.this.getApplicationContext().getSystemService("phone");
                String phoneNumber = phoneManager.getLine1Number();
                System.out.println("phonno:" + phoneNumber);
                ChangePassword.this.broadcastChangepasswordSMS(phoneNumber, ChangePassword.this.changePassword_text.getText().toString());
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
```

Ilustración 58: Muestra de la obtención de la respuesta del servidor

Análisis de Permisos:

Para acabar con el análisis estático del cifrado de datos en red, quedaría revisar el AndroidManifest para encontrar como está definido en el mismo la **configuración de seguridad** de red. Buscamos, más específicamente, que el atributo "android:usesCleartextTraffic" no permita **tráfico en texto claro**, y que la configuración de seguridad de red, "domain-config cleartextTrafficPermitted" no permita el tráfico en texto claro.

Como cabía esperar después del análisis del código fuente anterior, ninguna de estos atributos está definido como falso, es más, ni siquiera se encuentran definidos en el manifiesto.

Análisis Dinámico

Conocidas todas las prácticas que la aplicación realiza en su código fuente para llevar a cabo la comunicación con el servidor, será mucho más sencillo comprender la razón de los resultados que se obtendrán en este análisis dinámico.

El procedimiento de esta prueba consiste simplemente en hacer uso de una herramienta de **intercepción de datos de red** y para así comprobar qué datos se están transfiriendo entre cliente y servidor y, sobre todo, si estos datos se encuentran correctamente cifrados o no. Para ejecutar esta prueba se hará uso de WhireShark [40], la cual nos permitirá visualizar claramente el tráfico que circula a través de la red, pudiendo definir el tipo de paquetes que más nos interesan.

Configuramos pues el filtro del tipo de paquetes como “http” y comenzamos a interceptar el tráfico que pasa por la dirección “127.0.0.1” de nuestra máquina de pruebas. Al llevar a cabo el **login** con las credenciales válidas, recogemos un mensaje POST del cliente hacia el servidor con el siguiente contenido:

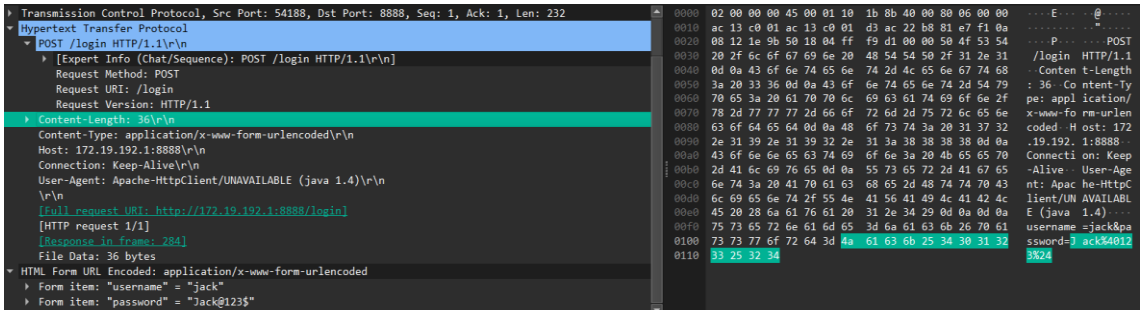


Ilustración 59: Petición del cliente al servidor para Login

El resultado obtenido es el esperado, sin embargo, no deja de sorprender que la aplicación comparta tan claramente tanto el nombre de usuario como la contraseña que se le pasan al servidor para llevar a cabo la autentificación. El mensaje interferido muestra claramente cómo el usuario “jack” con contraseña “Jack@123$”, está intentando acceder a la aplicación cuyo servidor se encuentra en la dirección “172.19.192.1”, y el cual tiene una vista “/login” mediante la cual gestiona el acceso de los usuarios.

A continuación, veremos cuál es la lógica de transmisión cuando un usuario desea **realizar una transferencia**.

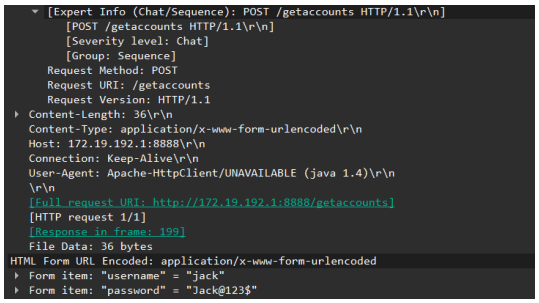


Ilustración 61: Petición del cliente al servidor para obtención de cuentas del usuario

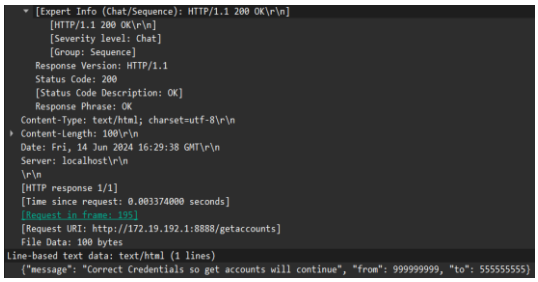


Ilustración 60: Respuesta del servidor al cliente ante la petición de cuentas del usuario

En primer lugar, al pulsar el botón “Get Accounts”, la aplicación envía un mensaje a la vista “/getaccounts” del servidor, compartiendo una vez más, las credenciales del usuario sin cifrar. La respuesta del servidor es un mensaje con las cuentas que el usuario tiene asociadas para las transferencias, sin ningún tipo de cifrado.

Al **ejecutar la transferencia**, la aplicación envía una petición http al servidor con la información relativa a la misma: username y contraseña del usuario que desea hacerla, cuenta de salida, cuenta de entrada y cantidad de dinero a transferir. La vista “/dotransfer” del servidor, responde entonces, después de comprobar que los datos son correctos, con la información de las dos cuentas y la cantidad de la transferencia. Toda esta información se encuentra sin ningún tipo de cifrado.

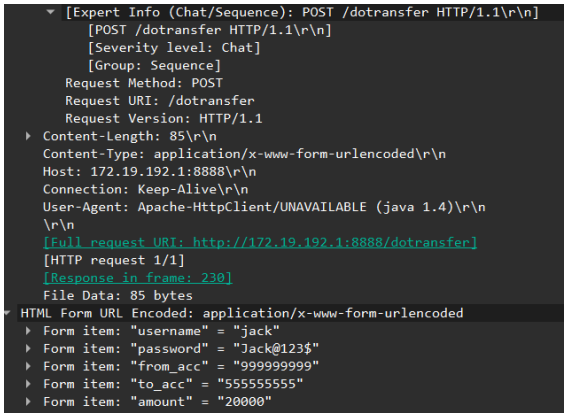


Ilustración 63: Petición del cliente al servidor para realización de transferencia

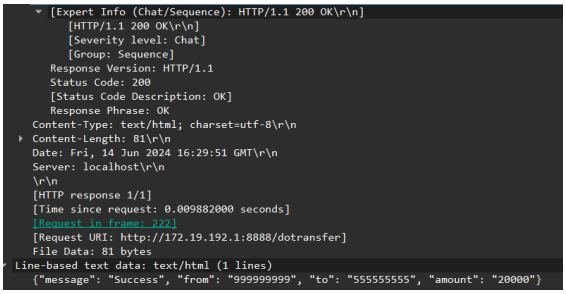


Ilustración 62: Respuesta del servidor a cliente después de transferencia

Por último, cuando un usuario desea **cambiar su contraseña**, la petición enviada al servidor contendrá el nombre del usuario y la nueva contraseña que desea utilizar, una vez más en texto totalmente claro.

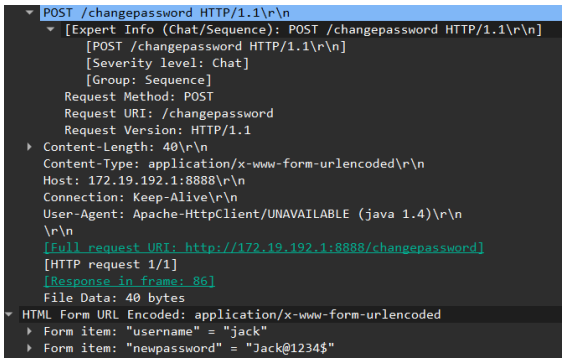


Ilustración 64: Petición de cliente a servidor para realizar cambio de contraseña

Conclusiones

Podemos concluir, por lo tanto, que tal y como podíamos prever todo el flujo de información relativo a la comunicación entre cliente y servidor muestra información personal de los usuarios, dejando así su privacidad totalmente expuesta si se produce un ataque de tipo “Man in the Middle”.

6.1.7. [MSTG-NETWORK-2] Prueba de la configuración TLS

El *Transport Layer Security* (**TLS**) es un protocolo de seguridad diseñado para asegurar la comunicación a través de redes, el cual proporciona de **confidencialidad** e **integridad** a los datos transmitidos entre las aplicaciones y sus usuarios [22]. En el contexto de aplicaciones móviles, el uso correcto de TLS es crucial para proteger la información sensible de los usuarios contra ataques malintencionados.

Para la realización de esta prueba se hará uso de la herramienta de análisis dinámico de MobSF, la cual cuenta con un **tester TLS/SSL** que analiza todo el tráfico presente en la ejecución de la aplicación y ejecuta una serie de pruebas para comprobar la configuración de este protocolo en la aplicación.

Análisis Dinámico

Durante el proceso de ejecución del test TLS, lo único que debemos hacer como usuarios es ejecutar todas las funcionalidades de la aplicación que requieran de comunicación directa con el servidor. Mientras tanto, el tester de MobSF se encarga de llevar a cabo las pruebas pertinentes para obtener el correspondiente informe final. Las pruebas llevadas a cabo son las siguientes.

- **Cleartext Traffic Test**:
    - Descripción: esta prueba verifica si la aplicación transmite datos en **texto claro** (sin cifrar). En caso de no encontrar tráfico en texto claro, la configuración será correcta, ya que se asegurará que toda la información transmitida está cifrada, protegiendo así los datos de los usuarios contra interceptaciones no autorizadas.
    - Resultado: ❌ - El tráfico en texto claro será susceptible a interceptaciones y ataques de intermediarios.
- **TLS Misconfiguration Test**:
    - Descripción: esta prueba evalúa la **configuración de TLS** en el servidor y cliente para detectar configuraciones inseguras. Un resultado negativo aquí indica la presencia de configuraciones incorrectas, como el uso de versiones obsoletas de TLS o cifrados débiles, las cuales pueden comprometer la privacidad de los usuarios, exponiendo sus datos a posibles ataques de intermediario (MITM) o descifrado.
    - Resultado: ☑ - Protege los datos en tránsito, asegurando que solo el cliente y el servidor previstos puedan acceder a la información.

- **TLS Pinning/Certificate Transparency Bypass Test**:

    - Descripción: esta prueba verifica si la aplicación es vulnerable a la omisión del **pinning**[61] **de certificados**[62] o de la **transparencia de certificados**, técnicas esenciales para garantizar que la aplicación solo se comunique con servidores legítimos. El TLS pinning es una técnica que vincula la aplicación a un certificado específico, previniendo que acepte certificados no autorizados, incluso si están firmados por una autoridad de certificación confiable. Un resultado negativo indica que la aplicación puede ser engañada para aceptar certificados falsificados, poniendo en riesgo la integridad y confidencialidad de la comunicación.
    - Resultado: ☑ - Se añade una capa adicional de seguridad, garantizando que los certificados utilizados sean legítimos y no hayan sido comprometidos.

- **TLS Pinning/Certificate Transparency Test**:

    - Descripción: similar a la prueba anterior, esta verifica si el pinning de certificados y la transparencia de certificados están implementados correctamente. Un resultado negativo revela fallos en la implementación, permitiendo potencialmente a un atacante interceptar y modificar la comunicación.
    - Resultado: ☑ - Asegura que solo servidores legítimos puedan establecer comunicaciones con la aplicación, protegiendo así los datos del usuario contra interceptaciones.

Conclusiones

Los resultados de las pruebas realizadas revelan que, aunque la aplicación falla en la configuración del tráfico en texto claro, tiene configuraciones adecuadas en términos de TLS general y de prácticas avanzadas como TLS Pinning y transparencia de certificados.

[61] Técnica de seguridad que protege las conexiones web al verificar y fijar específicamente los certificados o claves públicas de un servidor, asegurando la autenticidad y evitando ataques de intermediarios maliciosos.
[62] Archivos que contienen información de identidad digital y clave pública de un servidor web, utilizados para establecer conexiones seguras mediante protocolos como HTTPS, asegurando la autenticidad y privacidad de la comunicación en línea.

6.1.8. [MSTG-STORAGE-6] Determinar si los datos almacenados han sido expuestos a través de mecanismos de IPC

A lo largo de esta prueba se realizarán las acciones necesarias para determinar si los datos personales están siendo mostrados mientras se hace uso de los mecanismos IPC. Los mecanismos **IPC** (*Inter-Process Communication*) son métodos que permiten a los procesos en un sistema operativo **comunicarse** y **coordinar** sus acciones, para poder llevar a cabo así la interacción entre diferentes componentes y servicios del sistema. El uso inadecuado de estos mecanismos puede tener graves consecuencias para la privacidad de los datos de los usuarios, pudiendo exponer datos sensibles, permitiendo la interceptación de comunicación y facilitando la fuga de datos. Para determinar que el uso de estos mecanismos es el correcto, identificaremos el funcionamiento del proveedor de contenido de la app y nos aseguraremos de que ningún tipo de información se comparta a través del mismo.

Análisis Estático

Análisis de permisos:

El primer paso para llevar a cabo el análisis estático de la aplicación será revisar el AndrodManifest en busca de todos los elementos "<**provider**>", utilizados para gestionar el acceso a conjuntos estructurados de datos, y esenciales para compartir datos entre diferentes aplicaciones de manera controlada y segura.

```
<provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="com.android.insecurebankv2.TrackUserContentProvider" />
```

Encontramos definido solamente un proveedor de contenido de nombre "**com.android.insecurebankv2.TrackUserContentProvider**", el cual contiene el atributo "android:exported" como true, haciendo que cualquier aplicación instalada en el dispositivo puede acceder a los datos gestionados por este proveedor de contenido sin restricciones.

Los datos del proveedor no se encuentran protegidos con la etiqueta "android:permission", significando que no hay controles específicos sobre quién puede acceder a los datos, y permitiendo así, que cualquier aplicación, pueda potencialmente leer y manipular los datos personales. En su lugar se define el atributo "android:authorities", que especifica los nombres de los proveedores de contenido autorizados para acceder a los datos gestionados por este ContentProvider. Por último, tampoco se define ningún atributo "android:protectionLevel", por lo que los datos no estarán destinados a ser accedidos solo por aplicaciones de la misma empresa (firmadas con la misma clave).

Análisis del código fuente:

Pasaremos ahora a realizar una inspección en el código fuente para comprender el funcionamiento del proveedor de contenido. Para ello, se hará una búsqueda de las siguientes palabras clave:

- "android.content.ContentProvider"
- "android.database.Cursor"
- "android.database.sqlite"
- ".query`""
- ".update"
- ".delete"

Al poner especial enfoque en el uso de la clase "**ContentProvider**", puesto que se trata de la base que todas las aplicaciones extienden para hace uso del proveedor de contenido, se ha encontrado archivo relativo al proveedor que encontramos definido en el AndroidManifest, "TrackUserContentProvider". Este ContentProvider es utilizado para gestionar una base de datos SQLite que almacena nombres de usuarios. En él se definen varias de las operaciones anteriormente comentadas (query, insert, delete, y update), que permiten realizar consultas, inserciones, eliminaciones y actualizaciones de registros en una tabla "**names**". La tabla es creada en la base de datos mydb con una columna id como clave primaria autoincremental y una columna "name" que almacena el nombre del usuario.

En cuanto al funcionamiento de este proveedor, el método query construye una consulta a la base de datos y devuelve un **Cursor** con los resultados. El método **insert** añade nuevos registros a la base de datos y devuelve la URI del nuevo registro insertado, mientras que **delete** y **update** realizan operaciones de eliminación y actualización en los registros.

Este proveedor presenta varias **vulnerabilidades** que pueden comprometer la privacidad de los usuarios y la seguridad de la aplicación. En primer lugar, la tabla "names" almacena los datos en texto plano sin ningún mecanismo de cifrado, lo que significa que, si se obtiene acceso a la base de datos, los datos pueden ser leídos directamente sin ningún tipo de protección. Segundo, el método "query" no valida ni filtra adecuadamente las entradas del usuario antes de ejecutar la consulta SQL, lo que puede dejar la aplicación vulnerable a ataques de inyección SQL. Tercero, los métodos "insert", "update" y "delete" tampoco carecen de validaciones de entrada robustas, permitiendo la inserción de datos maliciosos o la manipulación indebida de registros existentes.

```
@Override // android.content.ContentProvider
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(TABLE_NAME);
    switch (uriMatcher.match(uri)) {
        case 1:
            qb.setProjectionMap(values);
            if (sortOrder == null || sortOrder == "") {
                sortOrder = name;
            }
            Cursor c = qb.query(this.db, projection, selection, selectionArgs, null, null, sortOrder);
            c.setNotificationUri(getContext().getContentResolver(), uri);
            return c;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
}
```

Ilustración 65: Inserción de valores en la tabla "names" del proveedor de contenido

```
@Override // android.content.ContentProvider
public int update(Uri uri, ContentValues values2, String selection, String[] selectionArgs) {
    switch (uriMatcher.match(uri)) {
        case 1:
            int count = this.db.update(TABLE_NAME, values2, selection, selectionArgs);
            getContext().getContentResolver().notifyChange(uri, null);
            return count;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
}
```

Ilustración 66: Clase empleada para la actualización del contenido de la tabla del proveedor de contenido

Todo lo descrito, unido a que como ya se comentó anteriormente, la configuración del atributo "android:exported" como true permite que cualquier aplicación en el dispositivo acceda al proveedor sin restricciones, y a que la ausencia del atributo "android:permission" hace que no se hayan definido controles específicos sobre quién puede acceder a los datos, nos deja una muestra lo bastantemente significativa como para afirmar que el proveedor de contenido usado por la aplicación puede vulnerar la privacidad de los usuarios durante la ejecución de sus funcionalidades.

Análisis Dinámico

Para la realización del análisis dinámico del uso del ContentProvider de la aplicación, haremos uso de la herramienta **Drozer**. Su instalación requerirá la ejecución de la apk "drozer-agent" desde el sistema móvil auditado, y una vez que se encuentre en funcionamiento nos deberemos conectar a su consola mediante el comando "drozer console connect".

Ilustración 67: Interfaz de la aplicación Drozer

Ilustración 68: Vista del agente Drozer ejecutado en nuestra máquina

Una vez dentro de la consola del agente, procederemos a enumerar la **superficie de ataque** del proveedor de contenido, ejecutando el comando "app.provider.info" para obtener información sobre los proveedores de contenido presentes en la aplicación.

```
dz> run app.provider.info -a com.android.insecurebankv2
Attempting to run shell module
Package: com.android.insecurebankv2
  Authority: com.android.insecurebankv2.TrackUserContentProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.android.insecurebankv2.TrackUserContentProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False
```

Ilustración 69: Comando y resultados de la enumeración de la superficie de ataque del proveedor de contenido

Obtenemos la información relativa al único proveedor de contenido presente en la aplicación, "**TrackUserContentProvider**", en especial, de los permisos que tiene definidos dicho proveedor.

A continuación, procederemos a identificar todas las **URIs**[63] del proveedor encontrado. Para ello se hará uso del módulo "scanner.provider.finduris" el cual nos devolverá las URIs del proveedor que pueden ser accesibles.

```
dz> run scanner.provider.finduris -a com.android.insecurebankv2
Attempting to run shell module
Scanning com.android.insecurebankv2...
No respone from content URI:      content://com.google.android.gms.games/
Got a response from content Uri:  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers/
No respone from content URI:      content://com.google.android.gms.games
No respone from content URI:      content://com.android.insecurebankv2.TrackUserContentProvider
Got a response from content Uri:  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
No respone from content URI:      content://com.android.insecurebankv2.TrackUserContentProvider/

For sure accessible content URIs:
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers/
```

Ilustración 70: Identificación de todas las URIs del proveedor de contenido

Obtenemos dos URIs accesibles:
"//com.android.insecurenackv2.TrackUserContentProvider/trackusers" y
"//com.android.insecurenackv2.TrackUserContentProvider/trackusers/".

Como ambas resultan similares, continuaremos el estudio con la última de ellas. El siguiente paso se trata de la **extracción de datos** de datos del ContentProvider a través de la URI encontrada anteriormente. Se usará para ello el comando "app.provider.query", el cual nos devuelve la tabla de la base de datos del proveedor en la cual almacena el nombre de los usuarios junto con su id, tal y como habíamos descifrado que lo hacía en el análisis estático.

[63] Uniform Resource Identifiers, son identificadores únicos que permiten acceder y manipular datos almacenados privadamente por una aplicación, garantizando el acceso controlado y seguro a la información.

```
dz> run app.provider.query content://com.android.insecurebankv2.TrackUserCon
tentProvider/trackerusers/
Attempting to run shell module
| id | name     |
| 11 | devadmin |
| 1  | dinesh   |
| 5  | dinesh   |
| 6  | dinesh   |
| 7  | dinesh   |
| 8  | dinesh   |
| 9  | dinesh   |
| 2  | jack     |
| 3  | jack     |
| 4  | jack     |
| 10 | jack     |
| 12 | jack     |
| 13 | jack     |
| 14 | jack     |
```

Ilustración 71: Contenido de la URI del proveedor de contenido

Todos los **nombres de los usuarios** almacenados se encuentran codificados en texto claro, por lo que un ataque contra este proveedor podría resultar crítico contra la privacidad de los usuarios.

Por último, llevaremos a cabo una automatización de **inyección sql** para comprobar si las URIs presentes en el proveedor son vulnerables o no. Haremos uso del comando "scanner.provider.injection"

```
dz> run scanner.provider.injection -a com.android.insecurebankv2
Attempting to run shell module
Scanning com.android.insecurebankv2...
Not Vulnerable:
  content://com.android.insecurebankv2.TrackUserContentProvider/
  content://com.google.android.gms.games/
  content://com.google.android.gms.games
  content://com.android.insecurebankv2.TrackUserContentProvider

Injection in Projection:
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
/

Injection in Selection:
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
/
```

Ilustración 72: Ejecución de prueba de inyección sql automatizada sobre las URIs

El resultado de esta inyección automatizada, nos muestra como las dos URIs accesibles previamente mostradas son vulnerables tanto en inyecciones de **proyección de datos**, lo cual implica la manipulación de las columnas que se devuelven en una consulta SQL; como en inyecciones de **selección de datos**, que permiten a un atacante modificar los criterios de selección de una consulta SQL.

Las vulnerabilidades de inyección de proyección y selección de datos representan una seria amenaza para la privacidad de los usuarios, ya que permiten a los atacantes manipular las consultas SQL enviadas al ContentProvider, accediendo y extrayendo datos sensibles almacenados en la base de datos de la aplicación [41].

6.1.9. [MSTG-PLATFORM-1] Prueba de permisos de aplicaciones

El objetivo principal de esta prueba es evaluar los **permisos** que la aplicación tiene definidos y determinar si cada uno de ellos es esencial para su correcto funcionamiento. Para llevar a cabo esta tarea de manera efectiva, es crucial comprender en detalle todas las acciones que la aplicación realiza, para poder así identificar con claridad qué permisos son absolutamente **necesarios** y cuáles pueden considerarse **prescindibles**. Esta tarea no solo garantizará el óptimo funcionamiento de la aplicación, sino que también contribuirá a mejorar su seguridad y eficiencia al minimizar los permisos innecesarios.

Análisis Estático

El proceso a seguir para llevar a cabo el análisis estático de los permisos definidos por la aplicación consistirá en comprobar cada uno de los permisos definidos en el AndroidManifest y definir si es esencial para el funcionamiento de la aplicación o no. Esta es la lista de permisos definidos:

- android.permission.INTERNET
    - Descripción: Permite que la aplicación acceda a Internet.
    - Necesidad: **SI**. La aplicación necesitará conectarse a Internet para poder llevar a cabo las peticiones necesarias al servidor.
    - Nivel de protección: Normal
- android.permission.WRITE_EXTERNAL_STORAGE
    - Descripción: Permite que la aplicación escriba en el almacenamiento externo del dispositivo.
    - Necesidad: **SI**. La aplicación necesita escribir en el almacenamiento externo del dispositivo en varios puntos de su lógica de funcionamiento, para almacenar datos relativos a usuarios.
    - Nivel de protección: Peligroso
- android.permission.SEND_SMS
    - Permite que la aplicación envíe mensajes SMS.
    - Necesidad: **NO**. La aplicación en ningún momento realiza el envío de un SMS, por lo que este permiso no debería estar activo.
    - Nivel de protección: Peligroso
- android.permission.USE_CREDENTIALS
    - Descripción: Permite que la aplicación use las credenciales del usuario para autenticarse.
    - Necesidad: **SI**. La aplicación necesitará este permiso para llevar a cabo la funcionalidad de autentificación del usuario con servicios externos sin requerir que el usuario vuelva a ingresar sus credenciales.
    - Nivel de protección: Normal

- android.permission.GET_ACCOUNTS
    - Descripción: Permite que la aplicación acceda a la lista de cuentas en el dispositivo.
    - Necesidad: **NO**. La aplicación en ningún momento hace uso de cuentas externas de servicios como Google o Facebook.
    - Nivel de protección: Peligroso

- android.permission.READ_PROFILE
    - Descripción: Permite que la aplicación lea el perfil personal del usuario [64]en el dispositivo.
    - Necesidad: **NO**. La aplicación no necesitará para ningún cometido la información del perfil personal del usuario del dispositivo.
    - Nivel de protección: Peligroso

- android.permission.READ_CONTACTS
    - Descripción: Permite que la aplicación lea los contactos del usuario almacenados en el dispositivo.
    - Necesidad: **NO**. Aunque la app pueda almacenar un número de teléfono al llevar a cabo una transferencia, es introducido por el usuario por lo que no será necesario el acceso a los contactos del dispositivo.
    - Nivel de protección: Peligroso

- android:uses-permission android:name="android.permission.READ_PHONE_STATE"
    - Descripción: Permite que la aplicación acceda al estado del teléfono, incluyendo el número de teléfono, la red de telefonía actual y el estado de las llamadas en curso.
    - Necesidad: **NO**. La aplicación no tiene ninguna funcionalidad que requiera del conocimiento del estado del teléfono.
    - Nivel de protección: Peligroso

- android:uses-permission
  android:name="android.permission.READ_EXTERNAL_STORAGE"
  android:maxSdkVersion="18"
    - Descripción: Permite que la aplicación lea desde el almacenamiento externo del dispositivo.
    - Necesidad: **SI**. Puesto que la aplicación tiene que escribir en el almacenamiento externo del dispositivo en varios puntos, también tendrá que tener permisos de lectura en el mismo.
    - Nivel de protección: Peligroso

- android:uses-permission android:name="android.permission.READ_CALL_LOG"
    - Descripción: Permite que la aplicación lea el registro de llamadas del dispositivo.
    - Necesidad: **NO**. La aplicación en ningún momento necesitará el registro de llamadas para la realización de ninguna de sus funcionalidades.
    - Nivel de protección: Peligroso

[64] Entorno separado que permite gestionar aplicaciones y datos de manera privada y segura, garantizando la separación entre los datos personales y profesionales o de trabajo en un mismo dispositivo.

- android.permission.ACCESS_NETWORK_STATE
    - Descripción: Permite que la aplicación acceda a información sobre las redes, como redes disponibles y estado de las mismas.
    - Necesidad: **SI**. La aplicación utiliza este permiso para determinar si hay una conexión de red disponible antes de intentar realizar operaciones que requieren internet.
    - Nivel de protección: Normal

- android.permission.ACCESS_COARSE_LOCATION
    - Descripción: Permite que la aplicación acceda a la ubicación aproximada del dispositivo basada en la red.
    - Necesidad: **NO**. La aplicación en ningún momento necesitará conocer la ubicación del usuario para llevar a cabo una transferencia.
    - Nivel de protección: Peligroso

La aplicación define 12 permisos en su manifiesto, de los cuales solo 5 resultan indispensables para su funcionamiento. Se definen por lo tanto 7 permisos que no son necesarios y que están permitiéndole el acceso a diversas funcionalidades del dispositivo que pueden a través de las cuales se puede obtener información confidencial y personal de dispositivo y usuario.

Conclusiones

En el análisis anterior de los permisos, también se añade el nivel de protección que cada uno de ellos requiere cuando se encuentran activos. Como puede apreciarse, la definición de la gran mayoría de permisos resulta peligrosa para la seguridad de la aplicación, por lo que se requerirá que solo se utilicen cuando sean totalmente necesarios, y que se tomen medidas de seguridad adicionales en la aplicación cuando se utilicen.

CAPITULO 6.1. AUDITORÍA SOBRE PRIMERA APP: INSECUREBANK

6.2. Auditoría sobre segunda app: RightsApp

La segunda aplicación que se someterá a una auditoría móvil se llama "RightsApp". Este estudio presentará una peculiaridad en comparación con la auditoría anterior: solo disponemos del **código fuente** de la aplicación, ya que nos fue proporcionada sin los archivos de compilación necesarios para construir la APK correspondiente.

Como auditor móvil, me enfrento a una situación poco común, sin embargo, para muchos profesionales en este campo, esta circunstancia puede representar una oportunidad valiosa para llevar a cabo un **análisis estático exhaustivo** del código fuente recibido [46]. Aunque no podremos realizar un análisis dinámico de las funcionalidades de la aplicación, no es motivo de preocupación, puesto que, a través del análisis estático, podremos identificar de manera clara las vulnerabilidades relativas a la privacidad de los usuarios presentes en la aplicación.

Es importante destacar que, al no contar con la APK, no se podrá evaluar el comportamiento de la aplicación ni su diseño de manera directa. Para comprender completamente su funcionamiento y arquitectura, será necesario un análisis minucioso de todas las actividades y componentes definidos en su código fuente. Este enfoque nos permitirá obtener una visión detallada de los posibles problemas de seguridad y calidad del software, asegurando que RightsApp cumpla con los estándares requeridos antes de su despliegue.

6.2.1. [MSTG-STORAGE-1, MSTG-STORAGE-2] Pruebas en almacenamiento local para buscar datos confidenciales

El objetivo de esta prueba es identificar datos potencialmente sensibles que la aplicación pueda almacenar y asegurarse de que se guarden de manera correcta y segura. Para ello, se analizará el código fuente de la aplicación para entender cómo maneja el almacenamiento de la información:

Análisis Estático

Permisos de Almacenamiento:

En primer lugar, es esencial determinar qué **tipo de almacenamiento** utiliza la aplicación y cómo maneja la información sensible. Para ello, se revisarán los permisos de lectura y escritura en el almacenamiento externo definidos en el archivo AndroidManifest.xml.

El manifiesto no incluye los permisos “android.permission.WRITE_EXTERNAL_STORAGE” ni “android.permission.READ_EXTERNAL_STORAGE”, lo que indica que la aplicación no permite la lectura ni la escritura en el almacenamiento local externo.

Uso de Permisos de Ficheros:

Se buscará el uso de ciertos permisos específicos en el código:

- **MODE_WORLD_READABLE**: permite a otras aplicaciones leer archivos en el directorio privado de datos de la aplicación.

- **MODE_WORLD_WRITEABLE**: permite a otras aplicaciones escribir en archivos en el directorio privado de datos de la aplicación.

El análisis del código revela que no se utilizan estos permisos en ningún punto.

Clases y Funciones de Almacenamiento:

Se revisará el uso de las siguientes clases y funciones en el código mediante la búsqueda de sus palabras clave en el mismo:

- Clase **SharedPreferences**: almacena pares clave-valor. Es crucial verificar si se almacenan datos sensibles sin cifrado. En la aplicación, esta clase se menciona 40 veces. Tras un examen detallado, se confirma que los datos almacenados no son personales, sino que corresponden a preferencias de usuario, como el idioma de la aplicación y configuraciones de uso.

- Clase **FileOutputStream**: usa almacenamiento externo o interno para crear una secuencia de salida de archivo. Esta clase no se utiliza en el código fuente, eliminando la preocupación por este modo de almacenamiento.

- Funciones **getExternal**: usan almacenamiento externo. Almacenar datos sensibles en almacenamiento externo sin cifrado puede ser peligroso, pero afortunadamente, la aplicación no utiliza estas funciones, asegurando que no haya riesgos asociados.

- Función **getWritableDatabase**: devuelve una SQLiteDatabase para escritura, lo que podría exponer datos sensibles sin controles de acceso y cifrado adecuados. Esta función no se encuentra en el código.

- Función **getReadableDatabase**: devuelve una SQLiteDatabase para lectura. Sin controles de acceso adecuados, esto podría permitir que cualquier entidad lea datos sensibles. Esta función tampoco se emplea en el código.

Conclusiones

El análisis del código fuente de RightsApp muestra que la aplicación maneja adecuadamente la privacidad y seguridad de los datos. No utiliza permisos que permitirían a otras aplicaciones acceder a su almacenamiento privado ni emplea clases o funciones que podrían poner en riesgo la información sensible. Las preferencias de usuario almacenadas no son confidenciales y no se usa almacenamiento externo ni bases de datos sin los debidos controles de acceso. Por lo tanto, podemos concluir que RightsApp, al no permitir el acceso a datos sensibles a través de almacenamiento externo o permisos inseguros, minimiza los riesgos de filtración de información y asegura que la privacidad del usuario se mantiene intacta a la hora de buscar vulnerabilidades relacionadas con este aspecto.

6.2.2. [MSTG-STORAGE-3] Registros de logs para buscar datos confidenciales

El propósito de esta prueba será revisar cómo la aplicación genera los **registros del sistema** (logs) desde su código fuente, siguiendo un enfoque similar al utilizado en el análisis estático de la auditoría anterior. El objetivo será observar detalladamente qué datos se registran en las funciones encargadas de generar logs.

Análisis Estático

Uso de Clases de Registro:

Las aplicaciones suelen utilizar las clases "**Log**" y "**Logger**" para crear registros. Para identificar su uso, es necesario auditar el código fuente de la aplicación buscando las siguientes palabras clave y funciones:

- Clase android.util.Log: esta clase no se utiliza en ningún momento a lo largo del código.
- Funciones Log.d | Log.e | Log.i | Log.v | Log.w | Log.wtf: estas funciones son comunes para registrar y notificar errores durante la ejecución de la aplicación. El riesgo aquí es que se impriman datos personales en los mensajes de registro. Tras revisar todo el código, se confirma que ninguna de estas funciones se utiliza.

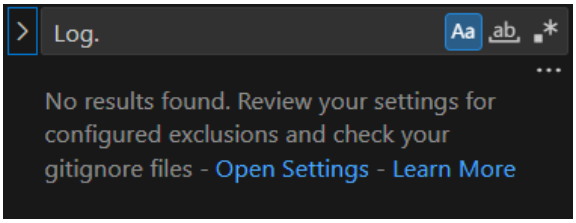


Ilustración 73: Busqueda textual de las funciones "Log."

- Clase **Logger**: utilizada para registrar mensajes que ayudan en la depuración y monitoreo del comportamiento de la aplicación, incluyendo estados de la aplicación, errores, advertencias y otros eventos importantes. En el código fuente de nuestra aplicación, esta clase tampoco se emplea en ningún momento.

Como puede apreciarse, esta aplicación no genera registros de logs en su funcionamiento. Si bien esto implica una **falta de trazabilidad**, también elimina la preocupación de un posible filtrado de datos sensibles a través de logs.

Palabras Clave y Salida del Sistema:

A continuación, investigaremos si la aplicación imprime información de alguna forma mediante los mecanismos de salida del sistema:

- System.out.print | System.err.print: el primer comando se utiliza varias veces en el código fuente. Al investigar cada instancia de su uso, se encuentra que, en la mayoría de los

casos, simplemente sirve para notificar al usuario sobre un error o la finalización de una acción. La única excepción potencialmente preocupante es en la clase "TestQuestionnarie", donde se imprimen las respuestas de un cuestionario realizado por los usuarios. Sin embargo, a simple vista, esta información no parece relevante desde el punto de vista de la privacidad.

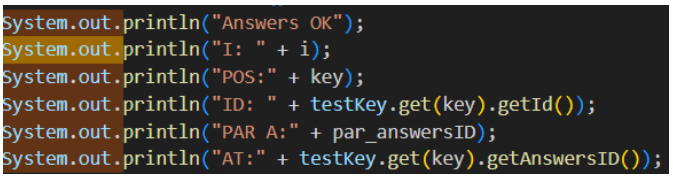


Ilustración 74: Uso de la función "System.out.println" en el código

Conclusiones

El análisis detallado del código fuente revela que la aplicación no utiliza las clases y funciones comunes para la generación de logs, ni imprime información personal significativa a través de los comandos de salida del sistema. Aunque la falta de registros de logs puede afectar la trazabilidad y la capacidad de depuración, también minimiza el riesgo de filtración de datos sensibles. Por lo tanto, se puede afirmar que RightsApp maneja **adecuadamente** la privacidad del dispositivo y de los usuarios en términos de generación de logs. La ausencia de registros de logs que contengan información sensible contribuye a proteger la privacidad del usuario, asegurando que no haya exposición inadvertida de datos a través de estos mecanismos.

6.2.3. [MSTG-STORAGE-8] Pruebas en copias de seguridad para buscar datos confidenciales

El objetivo de esta prueba es verificar que las **copias de seguridad** realizadas por la aplicación no contengan datos sensibles que puedan comprometer la privacidad de los usuarios. Para ello, es necesario investigar cómo la aplicación genera y almacena estas copias de seguridad.

Análisis Estático

Permisos y Configuración de Backups en AndroidManifest:

El primer paso en el análisis del código fuente de la aplicación es revisar la configuración del AndroidManifest para verificar si se permite la realización de copias de seguridad.

- Permisos en AndroidManifest: la configuración en el manifiesto de la aplicación permite el almacenamiento de información en forma de backup. Esto implica que la copia de seguridad se realizará automáticamente.

```
android:allowBackup="true"
```

Ilustración 75: Permisión de backups automáticos en la aplicaicón

A continuación, revisaremos si se han activado otras opciones de backup en el manifiesto:

- Backup Automático y Agente de Respaldo:
    - android:fullBackupOnly: No se encuentra activada esta opción.
    - android:backupAgent: No se ha definido ningún agente de backup en el archivo AndroidManifest.xml.

Almacenamiento en la Nube:

La aplicación no utiliza almacenamiento en la nube, por lo que no es necesario realizar un análisis en ese sentido.

Conclusiones

El análisis estático ha revelado que la aplicación permite la creación automática de copias de seguridad, pero no utiliza agentes de respaldo específicos ni opciones avanzadas de backup. Esto sugiere que las copias de seguridad podrían estar gestionadas de manera básica por el sistema. Al permitir copias de seguridad automáticas, el contenido de las mismas debe ser examinado cuidadosamente para garantizar que no comprometa la privacidad de los usuarios, y asegurarnos de que los datos sensibles no se incluyan en las copias de seguridad o que si se almacenan estén debidamente cifrados.

6.2.4. [MSTG-STORAGE-10] Pruebas en memoria para buscar datos confidenciales

El objetivo de esta prueba es verificar la presencia de información personal en la **memoria interna** del sistema donde se ejecuta la aplicación bajo auditoría. La memoria interna de un dispositivo Android constituye el almacenamiento no volátil utilizado para guardar datos permanentes como aplicaciones, archivos del sistema y datos de usuario. Aunque la memoria interna cuenta con múltiples capas de seguridad, como permisos de acceso y encriptación, diseñadas para evitar accesos no autorizados y proteger datos personales, pueden existir vulnerabilidades que permitan el acceso a información sensible almacenada.

Análisis Estático

El análisis estático del código de la aplicación se centrará en identificar dónde y cómo se almacenan los datos sensibles durante el ciclo normal de ejecución de la app. Esto implicará rastrear las distintas actividades que se ejecutan para llevar a cabo las funcionalidades ofrecidas por la aplicación. Nuestro punto de partida será la actividad inicial definida en el manifiesto de la aplicación, conocida como "SplashScreenActivity".

*Actividad **SplashScreenActivity**:*

La actividad “SplashScreenActivity” es la encargada de llevar a cabo las actividades iniciales cuando se crea, como establecer el tema de la aplicación, copiar una base de datos desde el directorio de recursos a la aplicación usando el método “createDatabase” y acceder a las preferencias compartidas del dispositivo para establecer ciertas configuraciones predeterminadas. Posteriormente, verifica si el usuario ha aceptado los términos y condiciones, y dependiendo de esta verificación y de si se debe mostrar una explicación, inicia las actividades “TermsAndConditions”, “ExplanationActivity” o “RightsAppActivity” con un retraso de tres segundos.

```
Context context = getApplicationContext();
sharedPreferences = context.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVATE);

//Sets first run preferences
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putBoolean(Constants.FIRST_RUN_CRIME_LIST, true);
editor.putBoolean(Constants.FIRST_RUN_BACK_CLUSTER, true);
editor.apply();

//Sets the language stored in Preferences for the app
language = sharedPreferences.getString(Constants.PREF_LANGUAGE, null);
```

Ilustración 76: Almacenamiento de datos de las SharedPreferences en variables

Respecto a las operaciones en memoria que la actividad realiza, se registran operaciones de **lectura** y **escritura** en SharedPreferences, lo que implica almacenamiento persistente de datos sensibles como la aceptación de términos y el idioma preferido. Una potencial vulnerabilidad es la manipulación de estas preferencias, puesto que, si no se protege adecuadamente, un atacante podría cambiar las configuraciones almacenadas. Además, la copia de la base de datos y la actualización de la configuración de recursos también pueden ser puntos de ataque si no se manejan correctamente los permisos y se asegura la integridad de los archivos copiados.

*Actividad **RightsAppActivity**:*

Para continuar con el análisis de las actividades de la aplicación, pasaremos a revisar el funcionamiento de la actividad "RightsAppActivity", puesto que es la encargada de ejecutar la funcionabilidad principal de la app. Esta esta actividad se encarga de proporcionar la interfaz principal para la interacción del usuario con la aplicación, integrando una "**Firebase Analytics**" para rastrear en tiempo real las acciones realizadas por el usuario a lo largo del uso de su aplicación, como hacer clic en varios elementos interactivos, que redirigen a los usuarios a diferentes actividades ("CallActivity", "QuestionnaireActivity", "EntitySearchActivity"). La utilización de la "**Firebase Analytics**"[65] para recopilar datos de usuario, plantea algunas preocupaciones sobre la privacidad, ya que se rastrean y almacenan diversas interacciones del usuario.

Cada una de las actividades a las que se puede acceder desde la interfaz principal son las encargadas de ejecutar las funcionalidades básicas de la aplicación. Procederemos pues a analizar qué es lo que hace cada una de ellas, comenzando por la actividad "CallActivity".

*Actividad **CallActivity**:*

La actividad "CallActivity" está diseñada para manejar la realización de llamadas telefónicas a números de emergencia y otros servicios específicos. Recupera un número de teléfono dado entre varias opciones mediante la función "getResources().getStrings()" y la ID relativa al número a mostrar, y lo muestra en un "TextView" y un "Button". Al pulsar el botón con el número de teléfono seleccionado, se solicitan permisos de llamada si no se han concedido, y si se conceden, inicia una llamada usando un "Intent" con la acción "Intent.ACTION_CALL"

.

```
//Gets the number from the intent
final String phone_number = getIntent().getStringExtra(Constants.PHONE_NUMBER_KEY);
```

Ilustración 77: Almacenamiento de dato de carácter personal en String

```
TextView tv_phone_call = findViewById(R.id.tv_call_phone);
Button btn_phone_call = findViewById(R.id.btn_phone_call);
ImageButton ib_back_phone_call = findViewById(R.id.ib_back_phone_call);

switch (phone_number) {
    case Constants.PHONE_VAW:
        tv_phone_call.setText(getResources().getString(R.string.phone_016));
        break;
    case Constants.PHONE_EMERGENCIES:
        tv_phone_call.setText(getResources().getString(R.string.phone_112b));
        break;
    case Constants.PHONE_SIOVD:
        tv_phone_call.setText(getResources().getString(R.string.phone_siovd));
        break;
    default:
        tv_phone_call.setText(getResources().getString(R.string.phone_call) + " " + phone_number);
        break;
}
```

Ilustración 78: Operaciones en memoria con múltiples variables que parecen almacenar datos personales

Esta actividad maneja **datos sensibles** como números de teléfono y permisos de llamada, por lo que la correcta gestión de la seguridad de la memoria de la aplicación es crucial. Además, el uso de "Intent.ACTION_CALL" en lugar de "Intent.ACTION_DIAL" puede ser riesgoso porque se iniciará la llamada directamente sin la intervención del usuario, lo cual podría explotarse si la aplicación es comprometida.

[65] Servicio de análisis que permite a los desarrolladores recopilar y analizar datos de usuarios y eventos dentro de aplicaciones móviles y web, facilitando la comprensión del comportamiento de los usuarios y el rendimiento de la aplicación.

*Actividad **CuestioonnarieActivity**:*

La siguiente actividad, "QuestionnaireActivity", gestiona un cuestionario dentro de la aplicación. Las operaciones en memoria relativas a esta actividad incluyen el manejo de "SharedPreferences" para almacenar y recuperar datos persistentes, como el estado del cuestionario, el idioma preferido o los datos del cuestionario actual. Es muy importante, por lo tanto, asegurar que estos datos se manejen de manera segura para evitar accesos no autorizados, al igual que con cualquier operación de lectura y escritura en "SharedPreferences" de las cuales debemos asegurar que no puedan ser manipulada por aplicaciones maliciosas.

*Actividad **EntitySearchActivity**:*

La última actividad a la que se puede acceder desde la actividad principal será la "EntitySearchActivity", la cual tiene como objetivo proporcionar una interfaz para que los usuarios busquen entidades basadas en categorías, países y ciudades. La actividad accede a una base de datos para llenar las listas desplegables para la selección, con datos iniciales de categorías, países y ciudades. La actividad también configura varios oyentes ("Listeners"[66]) para manejar las selecciones de los usuarios, actualizando las listas de ciudades en función de la selección de categorías y países. Finalmente, un botón permite a los usuarios iniciar una nueva actividad, "EntitiesListActivity", para mostrar los resultados de la búsqueda si existen entidades coincidentes, de lo contrario, muestra un diálogo de alerta.

Las operaciones en memoria incluyen el **acceso** y la **manipulación** de datos provenientes de la base de datos y el almacenamiento de las preferencias de usuario en las "SharedPreferences". La información relativa a la entidad que el usuario introduce seleccionada mediante parámetros, podría llegar a ser interceptada si no se maneja adecuadamente, comprometiendo la privacidad de los datos del usuario. Por lo tanto, es esencial implementar prácticas de seguridad como cifrado y validación de datos para mitigar estas vulnerabilidades.

```
// Set initial values for cities spinner
cities_list = null;
cities_list = db.getCitiesList(null, null, language, true);
CityModel all_cities_option = new CityModel(
        0,
        getResources().getString(R.string.all_cities),
        0,
        language);
cities_list.add(0, all_cities_option);
```

Ilustración 79: Construcción de modelo para almacenamiento de datos personales del usuario

*Actividad **EntitiesListActivity:***

Como se ha comentado, esta actividad llama a su vez a la actividad "EntitiesListActivity", por lo que también será interesante explorar su funcionamiento. Esta actividad se centra en mostrar una lista de entidades basadas en criterios de búsqueda previamente seleccionados por el usuario. Utiliza "SharedPreferences" para recuperar el idioma preferido del usuario e "Intents" para obtener los criterios de búsqueda almacenados desde la actividad anterior. Además, se accede a

[66] Componente que espera y responde a eventos o cambios en un sistema o aplicación, permitiendo la interacción dinámica y la ejecución de acciones en respuesta a eventos específicos

una base de datos local ("DataBaseHelper"[67]) para obtener las entidades que coinciden con los criterios específicos de categoría, país y ciudad.

```
int coarseLocationPermission = ActivityCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_COARSE_LOCATION);
int fineLocationPermission = ActivityCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_FINE_LOCATION);
```

Ilustración 80: Almacenamiento de información relativa a la localización en un int

En memoria encontramos operaciones que involucran la **gestión de datos** de ubicación mediante "FusedLocationProviderClient" para obtener la ubicación actual del usuario y calcular distancias a las entidades listadas. Esto implica el uso de los permisos de ubicación "ACCESS_COARSE_LOCATION" y "ACCESS_FINE_LOCATION". La lista de entidades se ordena según la distancia al usuario antes de ser mostrada en el "RecyclerView". Una vez, los datos relacionados con la ubicación y la ciudad o país que el usuario busca, pueden considerarse datos sensibles, por lo que será necesario aplicar las medidas de cifrado correspondientes.

Desde esta actividad, descubrimos como es llamada otra de nombre "EntityActivity", la cual muestra los detalles de una entidad específica, más concretamente, su nombre, descripción, dirección, número de teléfono, enlace web y dirección de correo electrónico. Los datos se reciben desde la actividad anterior y se muestran en los respectivos campos de texto, "TextView". La actividad permite al usuario realizar acciones como realizar una llamada telefónica al número proporcionado, abrir la ubicación en Google Maps para navegación y enviar un correo electrónico si se proporciona una dirección de correo electrónico válida.

La actividad accede por lo tanto a **datos sensibles** como el número de teléfono y la dirección de correo electrónico del usuario, además de la ubicación geográfica en forma de coordenadas, por lo que todas las operaciones en memoria que se realicen con ellos deberán seguir las directrices de seguridad pertinentes.

Conclusiones

Como conclusión, la aplicación "RightsApp" presenta un conjunto diverso de actividades que manejan información sensible a lo largo de su ciclo de ejecución. En ella, se realizan operaciones de lectura y escritura en SharedPreferences y acceso a bases de datos locales. Estas actividades gestionan datos como ubicaciones, preferencias de usuario y detalles personales, que son vitales para la funcionalidad de la aplicación, pero también susceptibles a **riesgos de privacidad** si no se manejan adecuadamente. El análisis revela la necesidad de implementar medidas robustas de seguridad, como el cifrado de datos sensibles y la gestión cuidadosa de permisos, para mitigar posibles vulnerabilidades. También resulta crucial asegurar que la información personal almacenada en la memoria interna del dispositivo esté protegida contra accesos no autorizados y posibles manipulaciones, y no sea persistente.

[67] Clase en Android que facilita la gestión de bases de datos SQLite dentro de una aplicación, proporcionando métodos para crear, actualizar y gestionar la estructura y datos de la base de datos de manera eficiente y estructurada.

6.2.5. [MSTG-CRYPTO-2, MSTG-CRYPTO-3, MSTG-CRYPTO-4] Prueba de la configuración de algoritmos estándar criptográficos

En esta prueba, nos enfocaremos en evaluar la implementación y gestión de los **algoritmos criptográficos** en la aplicación, asegurándonos de que las prácticas empleadas cumplan con los estándares actuales de la industria y las mejores prácticas recomendadas. En el contexto de la auditoría de una aplicación móvil, el análisis del uso de algoritmos criptográficos es esencial para garantizar la seguridad y privacidad de los datos manejados.

Análisis Estático

El análisis que realizaremos buscará asegurar que los algoritmos criptográficos utilizados para la encriptación de información personal cumplan con requisitos mínimos de seguridad. Identificaremos todos los puntos de la aplicación en los que se utilizan, definiremos el tipo de algoritmo empleado y evaluaremos su seguridad respecto a los estándares actuales, además de verificar que sean robustos frente a ataques.

Para lograr estos objetivos, se buscarán los siguientes elementos a lo largo del código:

- Clases "Cipher", "Mac", "MessageDigest" y "Signature":
    - La clase "**Cipher**" se utiliza para realizar operaciones de cifrado y descifrado de datos. A lo largo del código fuente, esta clase no se emplea ni una sola vez.
    - La clase "**MessageDigest**" se usa para generar resúmenes de mensaje o "hashes", que son representaciones compactas y únicas de datos. En el código fuente, esta clase no se utiliza en absoluto.
- La clase "**Signature**" se utiliza para verificar y crear firmas digitales, asegurando la autenticidad e integridad de los datos firmados. Al buscar en el código, no encontramos ningún uso de esta clase.
- Instancias "**PrivateKey**", "**PublicKey**" y "**SecretKey**":
    - Ninguna de estas instancias aparece a lo largo del código, lo que deja claro que no se están utilizando claves criptográficas privadas, públicas ni secretas en la implementación.
- Funciones "**getInstance**" y "**generateKey**":
    - La función "getInstance()" se emplea en criptografía para obtener instancias de algoritmos de cifrado, generación de claves y hashing. Esta función proporciona acceso a diferentes algoritmos criptográficos disponibles, permitiendo a los desarrolladores trabajar con ellos de manera consistente. En el código, esta función se emplea una vez, pero su uso se limita a la obtención de la instancia "FirebaseAnalytics" utilizada para seguir la actividad de los usuarios en la aplicación.

- La función "**generateKey**" no se emplea en el código.

Conclusiones

En resumen, el análisis del código fuente revela una falta significativa de implementación de algoritmos criptográficos, lo cual representa una vulnerabilidad crítica en términos de seguridad y privacidad de los datos. La ausencia de uso de clases y funciones esenciales para el cifrado y manejo de claves sugiere que la información personal o sensible podría estar expuesta a riesgos innecesarios. La única función relacionada con "getInstance" está dedicada a "FirebaseAnalytics", lo cual no contribuye a la seguridad criptográfica. Es por lo tanto crucial que la aplicación incorpore prácticas criptográficas sólidas y cumpla con los estándares de seguridad de la industria para proteger la información personal de los usuarios.

6.2.6. [MSTG-NETWORK-1] Prueba del cifrado de datos en la red

La **comunicación** entre el cliente y el servidor es una de las partes más cruciales para el correcto funcionamiento de una aplicación. A menudo, la información intercambiada entre estos dos componentes incluye datos personales para la aplicación o sensibles para los usuarios. Por esta razón, será esencial comprobar cómo la aplicación lleva a cabo todas estas comunicaciones.

Análisis Estático

La búsqueda de errores en el código durante las **solicitudes de red** es realmente importante, puesto que nos permitirá obtener una visión crítica de cómo se están realizando todas las solicitudes al servidor desde la aplicación. Comenzaremos pues identificando todas las solicitudes de red realizadas desde el código mediante **URLS**. Para ello se realizará la búsqueda textual de las solicitudes "http" y "https" presentes a lo largo del código.

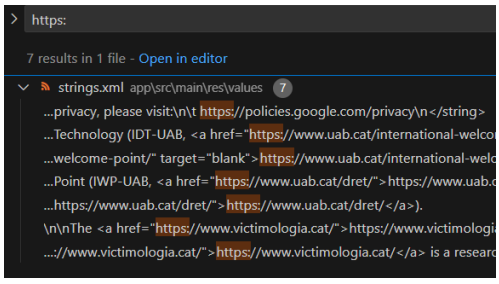


Ilustración 82: Uso de protocolo "https" en el código

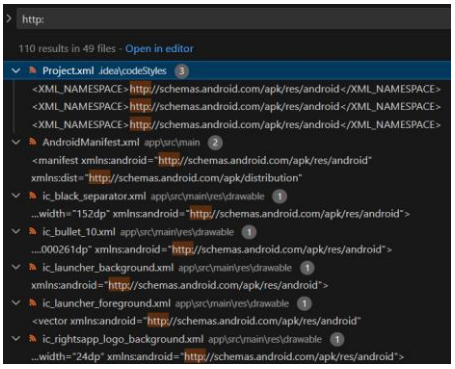


Ilustración 81: Uso del protocolo "https" en el código

El resultado de la búsqueda textual de los dos protocolos no muestra como estos solamente son empleados a modo de urls en los archivos xml de la aplicación. Si analizamos un poco más a qué tipo de sitios hacen las llamadas dichos archivos, podemos observar como el protocolo "**http**" se emplea únicamente para redirección al sitio "http://schemas.android.com/" y sus correspondientes directorios. Por la parte del protocolo "**https**", este se emplea únicamente en el archivo xml "strings.xml", reservado a todas las cadenas de texto empleadas en la interfaz de la aplicación. Las declaraciones de este protocolo se resumen todas en llamadas al sitio "https://www.uab.cat/" y "https://www.victimologia.cat/".

Continuando con el análisis del **método de comunicación** con el servidor, revisaremos si al menos esta se realiza a través de canales seguros como los obtenidos al hacer uso de "**HttpsURLConnection**" o "**SSLSocket**". Sorprendentemente, ninguna de estas dos clases es utilizada para llevar a cabo peticiones en la aplicación.

Para acabar con el análisis estático del cifrado de datos en red, quedaría revisar el AndroidManifest para encontrar como está definido en el mismo la configuración de **seguridad de red**. Buscamos, más específicamente, que el atributo "**android:usesCleartextTraffic**" no

permita tráfico en texto claro, y que la configuración de seguridad de red, "**domain-config cleartextTrafficPermitted**" no permita el tráfico en texto claro.

Ninguna de estos atributos está definido como falso, es más, ni siquiera se encuentran definidos en el manifiesto.

Conclusiones

Este análisis revelaría varias deficiencias críticas en la implementación de seguridad de red en la aplicación, como la falta de uso de canales seguros del tipo "HttpsURLConnection" o "SSLSocket" para las comunicaciones y la ausencia de configuraciones de seguridad en el AndroidManifest, sin embargo, tras el análisis de las funcionalidades de las actividades de la aplicación realizado durante la prueba anterior, podemos afirmar que esta aplicación es completamente autosuficiente y **no necesita de ninguna comunicación** con un servidor, por lo que la seguridad de las comunicaciones no es realmente necesaria. El único punto destacable negativamente sería el empleo de urls de tipo "http" para hacer llamadas a sitios que al utilizar este protocolo resultan poco seguros.

6.2.7. [MSTG-NETWORK-2] Prueba de la configuración TLS

El protocolo *Transport Layer Security* (**TLS**) está diseñado para garantizar la seguridad en la comunicación a través de redes, proporcionando confidencialidad e integridad a los datos transmitidos entre las aplicaciones y sus usuarios. En el contexto de las aplicaciones móviles, el uso adecuado de TLS es esencial para proteger la información sensible de los usuarios contra posibles ataques malintencionados, sin embargo, en nuestra aplicación auditada, esto no será relevante, ya que, como se determinó en la prueba anterior, **no se realiza ninguna comunicación**, ya sea cliente-servidor o de cualquier otro tipo. Por lo tanto, la realización de esta prueba no tiene sentido.

6.2.8. [MSTG-STORAGE-6] Determinar si los datos almacenados han sido expuestos a través de mecanismos de IPC

A lo largo de esta prueba, se realizarán las acciones necesarias para verificar si los datos personales están siendo expuestos durante el uso de los mecanismos IPC. Los mecanismos **IPC** (*Inter-Process Communication*) son métodos que permiten a los procesos en un sistema operativo **comunicarse** y **coordinar** sus acciones, facilitando la interacción entre diversos componentes y servicios del sistema. Su uso inadecuado, puede tener consecuencias graves para la privacidad de los datos de los usuarios, exponiendo datos sensibles, permitiendo la interceptación de comunicaciones y facilitando la fuga de información.

Para asegurarnos de que estos mecanismos se están utilizando correctamente, identificaremos el funcionamiento del proveedor de contenido de la app y verificaremos que ningún tipo de información se comparta a través del mismo.

Análisis Estático

El primer paso para llevar a cabo el análisis estático de la aplicación será revisar el AndroidManifest en busca de todos los elementos "<**provider**>", que se utilizan para gestionar el acceso a conjuntos estructurados de datos y son esenciales para compartir datos entre diferentes aplicaciones de manera controlada y segura.

No encontramos definido **ningún proveedor** de contenido dentro del manifiesto de la aplicación.

Pasaremos ahora a realizar una inspección en el código fuente para comprender el funcionamiento de los posibles proveedores de contenido. Para ello, se hará una búsqueda de las siguientes palabras clave:

- “android.content.ContentProvider”
- “android.database.Cursor”
- “android.database.sqlite”
- “.query`”
- “.update”
- “.delete”

En primer lugar, no encontramos ninguna implementación de la clase “**ContentProvider**”, lo cual tiene sentido debido a que ningún proveedor de contenido fue definido en el manifiesto de la aplicación. Por otro lado, podemos encontrar múltiples usos de la clase “**Cursor**” en el archivo “DataBaseHelper”, donde se utiliza para ejecutar consultas a la base de datos SQLite y para navegar por los resultados obtenidos. Esta clase permite extraer datos de los resultados de las consultas SQL y manejar las filas resultantes. En el contexto del análisis realizado en este apartado, esta clase puede tener algunas implicaciones significativas, ya que puede llegar a exponer datos si se pasan entre diferentes componentes de una aplicación sin el debido control de acceso.

```
Cursor cursor = myDataBase.rawQuery(query, null);
cursor.moveToFirst();
int id_next_question = cursor.getInt(0);

cursor.close();
```

Ilustración 83: Uso de la clase "Cursor" en las operaciones sobre la "DatabaseHelper"

Respecto al resto de instancias que se debían buscar, la clase “sqlite” se emplea en el mismo archivo que la clase “Cursor” anterior para declarar la **base de datos** sobre la que se realizan las distintas operaciones. También se encuentra definida en diversas ocasiones la instancia “query” para poder llevar a cabo las operaciones pertinentes sobre la base de datos definida. A continuación, un ejemplo de cómo se utiliza cada uno de estos elementos en el código:

```
String query = "SELECT * FROM " + DBContract.Answers.TABLE_NAME
        + " WHERE " +  DBContract.Answers.COLUMN_NAME_ID + " IN (" + id_answers[0];

for(int i = 1; i < id_answers.length; i++){
    query = query + "," + id_answers[i];
}
query = query + ") ORDER BY " + DBContract.Answers.COLUMN_NAME_ID + " ASC";

Cursor cursor = myDataBase.rawQuery(query, null);

if(cursor.moveToFirst()){
    // Loop through cursor results if the query has rows
    do {
        switch (language){
            case "es":
                result[index] = cursor.getString(1);
                break;
            case "en":
                result[index] = cursor.getString(2);
                break;
            case "por":
                result[index] = cursor.getString(3);
                break;
            case "it":
                result[index] = cursor.getString(4);
                break;
            default:
                //do default
                break;
        }
        index++;
    } while (cursor.moveToNext());
}
cursor.close();
```

Ilustración 84: Operaciones sobre la base de datos del proveedor de contenido "myDataBase"

El código revela cómo se define en primer lugar una consulta a la base de datos haciendo uso de una instancia “query”, a continuación, se define una instancia de la clase “Cursor” en la que se realiza la consulta definida sobre una base de datos “myDataBase” previamente definida. Por último, se obtienen los resultados almacenados en el cursor y se cierra cuando ya no queden datos en el mismo.

Conclusiones

Tras la realización del análisis, se concluye que la aplicación no presenta implementaciones explícitas de ContentProviders en su AndroidManifest ni en su código fuente. No obstante, el uso intensivo de la clase "Cursor" y operaciones SQL en el archivo "DataBaseHelper" indica una **manipulación directa** de la base de datos SQLite, lo cual podría implicar riesgos potenciales de exposición de datos si no se aplican las medidas adecuadas de control de acceso. La ausencia de ContentProviders **reduce la superficie de ataque** respecto a la exposición de datos a través de IPC. Sin embargo, el análisis revela que la gestión de datos mediante la clase "Cursor" y las operaciones SQL (declaradas a través de instancias "query") deben ser revisadas para asegurar que no existan vulnerabilidades que puedan ser explotadas para acceder a información sensible sin autorización.

6.2.9. [MSTG-PLATFORM-1] Prueba de permisos de aplicaciones

El propósito principal de esta evaluación es analizar los **permisos** que la aplicación ha establecido y determinar su relevancia para el funcionamiento adecuado de la misma. Para realizar esta tarea de manera efectiva, es crucial entender completamente todas las acciones realizadas por la aplicación, de modo que podamos identificar claramente cuáles permisos son imprescindibles y cuáles no.

Análisis Estático

El procedimiento para llevar a cabo el análisis estático de los permisos definidos por la aplicación implicará revisar cada uno de los permisos especificados en el "AndroidManifest" y determinar si son esenciales para el funcionamiento de la aplicación o no. A continuación, se presenta la lista de permisos definidos:

- android.permission.INTERNET
    - Descripción: Permite que la aplicación acceda a Internet.
    - Necesidad: **SI**. La aplicación necesitará conectarse a Internet para poder llevar a cabo las peticiones necesarias al servidor.
    - Nivel de protección: Normal
- android.permission.ACCESS_COARSE_LOCATION
    - Descripción: Permite que la aplicación acceda a la ubicación aproximada del dispositivo basada en la red.
    - Necesidad: **SI**. La aplicación puede requerir conocer la ubicación aproximada del usuario para buscar entidades cercanas.
    - Nivel de protección: Peligroso
- android.permission.ACCESS_FINE_LOCATION
    - Descripción: Permite que la aplicación acceda a la ubicación exacta del dispositivo basada en la red.
    - Necesidad: **SI**. La aplicación puede requerir conocer la ubicación exacta del usuario para buscar entidades cercanas
    - Nivel de protección: Peligroso
- android.permission.CALL_PHONE
    - Descripción: Permite que la aplicación iniciar una llamada telefónica directamente sin ningún paso intermedio.
    - Necesidad: **SI**. La aplicación puede requerir realizar llamadas telefónicas en una de sus actividades.
    - Nivel de protección: Peligroso

La aplicación define 4 permisos en su manifiesto, de los cuales todos son necesarios para su correcto funcionamiento. Como único punto mejorable, quizás solo sería necesario uno de los permisos entre “ACCESS_FINE_LOCATION” y “ACCESS_COARSE_LOCATION”, puesto que resulta algo redundante permitir el acceso a la localización aproximada y exacta.

Conclusiones

Con el análisis realizado sobre los permisos definidos podemos concluir que todos ellos son necesarios para el correcto funcionamiento de la aplicación, y que, aunque la definición algunos de ellos puedan resultar peligrosa para la privacidad de los usuarios, es un riesgo que se debe tomar para que la aplicación pueda realizar todas sus funcionalidades.

CAPITULO 6.2. AUDITORÍA SOBRE SEGUNDA APP: RIGHTSAPP

Capítulo 7

Conclusiones

El proyecto desarrollado y documentado en esta memoria ha sido completado en su totalidad, y, si bien hubiera sido ideal poder realizar alguna prueba más de las definidas por el estándar MASVS v1.5, considero que las auditorías realizadas revelan correctamente el grado de privacidad que las aplicaciones tienen configurada.

En primer lugar, se realizó un estudio de la **normativa actual** en cuanto a la protección de datos personales, considerando principalmente dos con distinto alcance: **RGPD** a nivel europeo y **LOPDGDD2018** a nivel nacional. La primera establece una serie de principios para que una aplicación sea más segura en términos de protección de la información. Paralelamente, se estudiaron las diferentes metodologías y estándares definidos por **OWASP MOBILE**, como **MASVS** o la guía **MSTG**. Finalmente, se realizó un análisis de las vulnerabilidades más comunes según el **OWASP Top 10**, indicando para cada una cómo contrarrestarla, aportando así mayores medidas de seguridad aplicables a un sistema móvil.

En segundo lugar, se **seleccionaron y filtraron las pruebas** definidas en el estándar MASVS en función de si estaban relacionadas con la seguridad o con la privacidad de la aplicación, y en el segundo caso, si permitían acceso a datos personales y sensibles o no. Esto permitió reducir el número de pruebas necesarias para evaluar si una aplicación cumple con la normativa de protección de datos o si se pretende comprobar la seguridad del sistema en cuanto al manejo de cualquier tipo de información.

Con las pruebas escogidas siguiendo el filtro seleccionado, se procedió a redactar el **diseño de las pruebas**, indicando para cada una el objetivo y las acciones a realizar, añadiendo recomendaciones para evitar problemas de seguridad en algunos casos.

Finalmente, se llevaron a cabo las **pruebas** para verificar la explotación de vulnerabilidades que permiten el acceso a datos personales o sensibles. Se documentó cada prueba y sus resultados, que mayormente confirmaron la presencia de vulnerabilidades de acceso a datos en la máquina víctima, indicando que no cumple con los requisitos de seguridad exigidos por las normativas de protección de datos.

La realización de las dos auditorías sobre las aplicaciones seleccionadas, me ha brindado una experiencia muy rica y unos conocimientos que antes no poseía. Desde el estudio de las características de las metodologías para auditar aplicaciones móviles, o el análisis de cada una de las pruebas que las conforman, hasta el diseño de la auditoría a realizar y la ejecución de la misma, han hecho que ahora pueda comprender a la perfección cómo se deben gestionar los datos personales y sensibles de los usuarios.

Poniendo el enfoque sobre las auditorías realizadas, se ha podido comprobar como la primera aplicación, InsecureBank, tiene mal configurados prácticamente todos los puntos en los que se

han buscado sus vulnerabilidades. Además, se ha visto como claramente comparte información personal de los usuarios como su username, password, teléfono o datos bancarios.

Por parte de la segunda app, RightsApp, podemos concluir que está bastante mejor configurada que la anterior, potencialmente solo mostrando datos personales o sensibles de los usuarios en las interacciones con la memoria interna.

7.1. Trabajo futuro

Como trabajo futuro, se propone completar la ejecución del resto de pruebas definidas por MASVS para analizar del todo la exposición de datos que llevan a cabo las aplicaciones.

Entre las principales mejoras está la creación de un paquete de medidas basadas en los principios de seguridad del RGPD, así como las recomendaciones de vulnerabilidades y el diseño de pruebas. También se propone evaluar el nivel de criticidad de las vulnerabilidades según la facilidad con que permiten exponer datos personales.

Finalmente, sería interesante incluir un protocolo de actuación que relacione todas las pruebas que explotan vulnerabilidades que permiten el acceso a datos, indicando el orden de ejecución.

Bibliografía

[1] GDPR.eu. Art. 4 GDPR- Definitions. url: https://gdpr.eu/article 4-definitions/

[2] Boletín Oficial del Estado. Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales. 28 de jun. de 2021. url: https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf

[3] Consejo Europeo de protección de Datos. url: https://european-union.europa.eu/institutions-law-budget/institutions-and-bodies/search-all-eu-institutions-and-bodies/european-data-protection-board-edpb_es

[4] Estándar de Seguridad de Aplicaciones Móviles url: OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation

[5] OEDI ciberdelincuencia en España. url: https://oedi.es/estadisticas/

[6] Introducción a Diagramas de Gantt. url: https://www.openproject.org/es/docs/curso-iniciacion/introduccion-diagrama-gantt/

[7] Ley Orgánica 3/2018 de Protección de Datos Personales y Garantía de Derechos Digitales. url: https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673

[8] Datos personales según RGPD. url: https://commission.europa.eu/law/law-topic/data-protection/reform/what-personal-data_es

[9] Cuando está permitido el procesamiento de datos personales según el RGPD. url: https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_es.htm#inline-nav-10

[10] Datos Sensibles según el RGPD. url: https://commission.europa.eu/law/law-topic/data-protection/reform/rules-business-and-organisations/legal-grounds-processing-data/sensitive-data/what-personal-data-considered-sensitive_es

[11] Directrices de consentimiento según el RGPD. url: https://www.edpb.europa.eu/sites/default/files/files/file1/edpb_guidelines_202005_consent_es.pdf

[12] Auditorías sobre dispositivos móviles. url: https://www.hackbysecurity.com/servicios-empresas/auditoría-informatica/auditoría-de-dispositivos-moviles

[13] Reglamento General de protección de Datos.url: https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32016R0679

[14] OWASP Application Security Verification Standard. url: https://owasp.org/www-project-application-security-verification-standard/

[15] Mobile Security Testing Guide. url: https://mas.owasp.org/MASTG/

[16] Mobile Application Security Verification Standard v1.5. url: https://github.com/OWASP/owasp-masvs/releases/tag/v1.5.0

[17] OWASP MOBILE TOP 10. url: https://owasp.org/www-project-mobile-top-10/

[18] Proceso en auditorías de privacidad. url: https://audit.guru/what-is-a-privacy-audit/#:~:text=Key%20Elements%20of%20a%20Privacy%20Audit,-Privacy%20audits%20typically&text=This%20analysis%20includes%20understanding%20how,practices%20and%20implement%20necessary%20safeguards.

[19] Definición de políticas de privacidad por la AEPD. url: https://www.aepd.es/guias/informe-politicas-de-privacidad-adaptacion-rgpd.pdf

[20] Principales vulnerabilidades de privacidad. url: https://www.tarlogic.com/es/blog/owasp-top-de-10-riesgos-de-privacidad/

[21] Uso de algoritmos criptográficos. url: https://www.incibe.es/sites/default/files/contenidos/politicas/documentos/uso-_tecnicas-criptograficas.pdf

[22] Funcionamiento del protocolo TLS. url: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/tls.html

[23]Mecanismo IPC en apps Android. url: https://proandroiddev.com/ipc-techniques-for-android-45d815ac59be

[24]Funcionamiento de WebViews en apps Android. url: https://www.tutorialspoint.com/android/android_webview_layout.htm

[25] Herramienta LogCat en apps Android. url: https://developer.android.com/tools/logcat?hl=es-419

[26]Data and File Storage overview. url: https://developer.android.com/training/data-storage?hl=es-419

[27]Diferencias entre protocolos HTTP y HTTPS. url: https://www.geeksforgeeks.org/difference-between-http-and-https-2/

[28] Drozer documentation. url: https://labs.withsecure.com/content/dam/labs/docs/mwri-drozer-user-guide-2015-03-23.pdf

[29] Lista de permisos peligrosos. url: https://developer.android.com/reference/android/Manifest.permission

[30] Repositorio de app InsecureBank. url: https://github.com/dineshshetty/Android-InsecureBankv2

[31] Almacenamiento externo en Android. url: https://www.geeksforgeeks.org/external-storage-in-android-with-example/

[32] Documentación de sqlite. url : https://www.sqlite.org/docs.html

[33] Herramienta Base64. url: https://www.base64decode.org/es/

[34] Android Backup Extractor. url : https://github.com/nelenkov/android-backup-extractor

[35] Sistema Android Keystore. url: https://developer.android.com/privacy-and-security/keystore?hl=es-419

[36] OWASP manejo de memoria. url: https://owasp.org/www-project-developer-guide/draft/implementation/dos_donts/memory_management/

[37] Uso de clave pública y clave privada pare encriptación. url: https://www.preveil.com/blog/public-and-private-key/#:~:text=The%20public%20key%20is%20used,data%20passes%20through%20unsecured%20networks.

[38] Herramienta Base63Guru. url: https://base64.guru/converter/decode/hex,

[39] Riesgos en la comunicación cliente-servidor OWASP. url: https://owasp.org/www-project-top-10-client-side-security-risks/

[40] Guía de usuario de WireShark. url: https://www.wireshark.org/docs/wsug_html_chunked/

[41] Proveedores de contenido en apps Android. url: https://developer.android.com/reference/android/content/ContentProvider

[42] GDPR.eu. Art. 6 GDPR- Lawfulness of processing- GDPR.eu. url: https://gdpr.eu/article-6-how-to-process-personal-data-legally/

[43] GDPR.eu. Art. 7 GDPR- Responsability of the controller- GDPR.eu. url: https://gdpr.eu/article-7-how-to-get-consent-to-collect-personal-data/

[44] GDPR.eu. Art. 24 GDPR- Responsability of the controller GDPR.eu. url: https://gdpr.eu/article-24-responsibility-of-the-data-controller/

[45] GDPR.eu. Art. 25 GDPR- Data protection by designd and by default- GDPR.eu. url: https://gdpr.eu/article-25-data-protection-by-design/

[46] RigthsApp github repository. url: https://github.com/jogoco78/RightsApp

[47] Mobsf user guide and github repository. url: https://mobsf.github.io/docs/#/mobsf_docker

// https://github.com/MobSF/Mobile-Security-Framework-MobSF

[48] Genymotion user guide. url: https://docs.genymotion.com/desktop/Get_started/014_Basic_steps/

[49] ADB user guide. url: https://developer.android.com/tools/adb?hl=es-419

[50] CypherChef user guide. url: https://www.csnp.org/post/cyberchef-data-decoding-made-easy

[51] INE: Porcentaje de uso de dispositivos móviles en España. url: https://www.ine.es/prensa/tich_2022.pdf

[52] Trabajo de fin de Grado "*AuditaWebPriv: Auditoría aplicada a la Privacidad en Aplicativos Web*" entregado por **José Francisco Villatoro Meyer**. url: https://www.fi.uva.es/tfg/documentacion/particular/download.php?proy=GI-JUL22-26&file=GI-JUL22-26.pdf

[53] Explicación de modelo en espiral IONOS. url: https://www.ionos.es/startupguide/productividad/modelo-en-espiral/

[54] [MSTG-STORAGE-1, MSTG-STORAGE-2] Pruebas en almacenamiento local para buscar datos confidenciales. url: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0001

[55] [MSTG-STORAGE-3] Registros de logs para buscar datos confidenciales. url: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0003

[56] [MSTG-STORAGE-8] Pruebas en copias de seguridad para buscar datos confidenciales. url: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0009

[57] [MSTG-STORAGE-10] Pruebas en memoria para buscar datos confidenciales. url: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0011

[58] v[MSTG-STORAGE-11] Pruebas en la política de seguridad de acceso al dispositivo. url: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0012

[59] [MSTG-CRYPTO-1] Prueba de criptografía simétrica. url: https://mas.owasp.org/MASTG/tests/android/MASVS-CRYPTO/MASTG-TEST-0013

[60] [MSTG-CRYPTO-2, MSTG-CRYPTO-3, MSTG-CRYPTO-4] Prueba de la configuración de algoritmos estándar criptográficos. url: https://mas.owasp.org/MASTG/tests/android/MASVS-CRYPTO/MASTG-TEST-0014

[61] [MSTG-CRYPTO-5] Prueba los propósitos de las claves.url: https://mas.owasp.org/MASTG/tests/android/MASVS-CRYPTO/MASTG-TEST-0015

[62] [MSTG-CRYPTO-6] Prueba de generación de números aleatorios. url: https://mas.owasp.org/MASTG/tests/android/MASVS-CRYPTO/MASTG-TEST-0016

[63] [MSTG-AUTH-1] Prueba de autenticación biométrica. url: https://mas.owasp.org/MASTG/tests/android/MASVS-AUTH/MASTG-TEST-0017

[64] [MSTG-AUTH-8] Pruebas de confirmación de credenciales. url: https://mas.owasp.org/MASTG/tests/android/MASVS-AUTH/MASTG-TEST-0018

[65] [MSTG-NETWORK-1] Prueba del cifrado de datos en la red. url: https://mas.owasp.org/MASTG/tests/android/MASVS-NETWORK/MASTG-TEST-0019

[66] [MSTG-NETWORK-2] Prueba de la configuración TLS. url: https://mas.owasp.org/MASTG/tests/android/MASVS-NETWORK/MASTG-TEST-0020

[67] [MSTG-NETWORK-3] Verificación de identificación del endpoint de prueba. url: https://mas.owasp.org/MASTG/tests/android/MASVS-NETWORK/MASTG-TEST-0021

[68] [MSTG-NETWORK-4] Prueba de almacenes de certificados personalizados y pinning de certificados url: https://mas.owasp.org/MASTG/tests/android/MASVS-NETWORK/MASTG-TEST-0022

[69] [MSTG-NETWORK-6] Probar el proveedor de seguridad . url: https://mas.owasp.org/MASTG/tests/android/MASVS-NETWORK/MASTG-TEST-0023

[70] [MSTG-STORAGE-6] Determinar si los datos almacenados han sido expuestos a través de mecanismos de IPC .url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0007

[71] [MSTG-STORAGE-7] Comprobación de divulgación de datos confidenciales a través de la interfaz de usuario url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0008

[72] [MSTG-STORAGE-9] Encontrar información confidencial en capturas de pantalla generadas automáticamente url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0035

[73] [MSTG-PLATFORM-1] Prueba de permisos de aplicaciones url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0024

[74] [MSTG-PLATFORM-3] Prueba de enlaces profundos url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0028

[75] [MSTG-PLATFORM-4] Pruebas de implementación vulnerable de PendingIntent y Pruebas de exposición a funcionalidades sensibles mediante IPC url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0030

[76] [MSTG-PLATFORM-5] Prueba de ejecución de JavaScript en WebViews url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0031

[77] [MSTG-PLATFORM-6] Prueba de controladores de protocolo WebView url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0032

[78] [MSTG-PLATFORM-7] Prueba de objetos Java expuestos a través de WebViews url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0033

[79] [MSTG-PLATFORM-9] Pruebas de ataques de superposición (overlay attacks) url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0035

[80] [MSTG-PLATFORM-10] Prueba de limpieza de WebViews url: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0037

[81] [MSTG-CODE-5] Comprobación de debilidades en bibliotecas de terceros urñ: https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0042

[82] [MSTG-CODE-9] Asegúrese de que las funciones de seguridad gratuitas estén activadas url: https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0044

[83] [MSTG-CODE-1] Asegurarse de que la aplicación esté correctamente firmada url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0038

[84] [MSTG-CODE-2] Probar si la aplicación es depurable url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0039

[85] [MSTG-CODE-3] Prueba de símbolos de depuración url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0040

[86] [MSTG-CODE-4] Pruebas de código de depuración y registro de errores detallado url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0041

[87] [MSTG-RESILIENCE-1] Prueba de detección de raíces url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0045

[88] [MSTG-RESILIENCE-2] Prueba de detección antidepuración url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0038

[89] [MSTG-RESILIENCE-3] Prueba de comprobaciones de integridad de archivos url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0047

[90] [MSTG-RESILIENCE-4] Prueba de detección de herramientas de ingeniería inversa url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0048

[91] [MSTG-RESILIENCE-5] Prueba de detección del emulador url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0049

[92] [MSTG-RESILIENCE-6] Prueba de comprobaciones de integridad en tiempo de ejecución url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0050

[93] [MSTG-RESILIENCE-9] Prueba de ofuscación url: https://mas.owasp.org/MASTG/tests/android/MASVS-RESILIENCE/MASTG-TEST-0051

[94] Guía del estándar MASVS v1.5. url: https://github.com/OWASP/owasp-masvs/releases/download/v1.5.0/OWASP_MASVS-v1.5.0-es.pdf

BIBLIOGRAFIA

Capítulo 8
ANEXO I: RGPD

8.1. Artículo 4: Definiciones

1. Datos personales: toda información sobre una persona física identificada o identificable («el interesado»); se considerará persona física identificable toda persona cuya identidad pueda determinarse, directa o indirectamente, en particular mediante un identificador, como por ejemplo un nombre, un número de identificación, datos de localización, un identificador en línea o uno o varios elementos propios de la identidad física, fisiológica, genética, psíquica, económica, cultural o social de dicha persona. [1]

2. Tratamiento: cualquier operación o conjunto de operaciones realizadas sobre datos personales o conjuntos de datos personales, ya sea por procedimientos automatizados o no, como la recogida, registro, organización, estructuración, conservación, adaptación o modificación, extracción, consulta, utilización, comunicación por transmisión, difusión o cualquier otra forma de habilitación de acceso, cotejo o interconexión, limitación, supresión o destrucción.

8.2. Artículo 5: Principios relativos al tratamiento

1. Los datos personales serán:

> *a) Tratados de manera lícita, leal y transparente en relación con el interesado («licitud, lealtad y transparencia»).*
>
> *b) Recogidos con fines determinados, explícitos y legítimos, y no serán tratados ulteriormente de manera incompatible con dichos fines; de acuerdo con el artículo 89, apartado 1, el tratamiento ulterior de los datos personales con fines de archivo en interés público, fines de investigación científica e histórica o fines estadísticos no se considerará incompatible con los fines iniciales («limitación de la finalidad»).*
>
> *c) Adecuados, pertinentes y limitados a lo necesario en relación con los fines para los que son tratados («minimización de datos»); d) exactos y, si fuera necesario, actualizados; se adoptarán todas las medidas razonables para que se supriman o rectifiquen sin dilación los datos personales que sean inexactos con respecto a los fines para los que se tratan («exactitud»).*
>
> *e) Mantenidos de forma que se permita la identificación de los interesados durante no más tiempo del necesario para los fines del tratamiento de los datos personales; los datos personales podrán conservarse durante períodos más largos siempre que se traten exclusivamente con fines de archivo en interés público, fines de investigación científica o histórica o fines estadísticos, de conformidad con el artículo 89, apartado 1, sin perjuicio de la aplicación de las medidas técnicas y organizativas apropiadas que impone*

el presente Reglamento a fin de proteger los derechos y libertades del interesado («limitación del plazo de conservación»).

f) Tratados de tal manera que se garantice una seguridad adecuada de los datos personales, incluida la protección contra el tratamiento no autorizado o ilícito y contra su pérdida, destrucción o daño accidental, mediante la aplicación de medidas técnicas u organizativas apropiadas («integridad y confidencialidad»).

2. El responsable del tratamiento será responsable del cumplimiento de lo dispuesto en el apartado 1 y capaz de demostrarlo («responsabilidad proactiva»). [42]

8.3. Artículo 7: Condiciones para el consentimiento

1. Cuando el tratamiento se base en el consentimiento del interesado, el responsable deberá ser capaz de demostrar que aquel consintió el tratamiento de sus datos personales.

2. Si el consentimiento del interesado se da en el contexto de una declaración escrita que también se refiera a otros asuntos, la solicitud de consentimiento se presentará de tal forma que se distinga claramente de los demás asuntos, de forma inteligible y de fácil acceso y utilizando un lenguaje claro y sencillo. No será vinculante ninguna parte de la declaración que constituya infracción del presente Reglamento.

3. El interesado tendrá derecho a retirar su consentimiento en cualquier momento. La retirada del consentimiento no afectará a la licitud del tratamiento basada en el consentimiento previo a su retirada. Antes de dar su consentimiento, el interesado será informado de ello. Será tan fácil retirar el consentimiento como darlo.

4. Al evaluar si el consentimiento se ha dado libremente, se tendrá en cuenta en la mayor medida posible el hecho de si, entre otras cosas, la ejecución de un contrato, incluida la prestación de un servicio, se supedita al consentimiento al tratamiento de datos personales que no son necesarios para la ejecución de dicho contrato. [43]

8.4. Artículo 24: Responsabilidad del responsable del tratamiento

1. Teniendo en cuenta la naturaleza, el ámbito, el contexto y los fines del tratamiento, así como los riesgos de diversa probabilidad y gravedad para los derechos y libertades de las personas físicas, el responsable del tratamiento aplicará medidas técnicas y organizativas apropiadas a fin de garantizar y poder demostrar que el tratamiento es conforme con el presente Reglamento. Dichas medidas se revisarán y actualizarán cuando sea necesario.

2. Cuando sean proporcionadas en relación con las actividades de tratamiento, entre las medidas mencionadas en el apartado 1 se incluirá la aplicación, por parte del responsable del tratamiento, de las oportunas políticas de protección de datos.

3. La adhesión a códigos de conducta aprobados a tenor del artículo 40 o a un mecanismo de certificación aprobado a tenor del artículo 42 podrán ser utilizados como elementos para demostrar el cumplimiento de las obligaciones por parte del responsable del tratamiento. [44]

8.5. Artículo 25: Protección de datos desde el diseño y por defecto

1. Teniendo en cuenta el estado de la técnica, el coste de la aplicación y la naturaleza, ámbito, contexto y fines del tratamiento, así como los riesgos de diversa probabilidad y gravedad que entraña el tratamiento para los derechos y libertades de las personas físicas, el responsable del tratamiento aplicará, tanto en el momento de determinar los medios de tratamiento como en el

momento del propio tratamiento, medidas técnicas y organizativas apropiadas, como la seudonimización, concebidas para aplicar de forma efectiva los principios de protección de datos, como la minimización de datos, e integrar las garantías necesarias en el tratamiento, a fin de cumplir los requisitos del presente Reglamento y proteger los derechos de los interesados.

2. El responsable del tratamiento aplicará las medidas técnicas y organizativas apropiadas con miras a garantizar que, por defecto, solo sean objeto de tratamiento los datos personales que sean necesarios para cada uno de los fines específicos del tratamiento. Esta obligación se aplicará a la cantidad de datos personales recogidos, a la extensión de su tratamiento, a su plazo de conservación y a su accesibilidad. Tales medidas garantizarán en particular que, por defecto, los datos personales no sean accesibles, sin la intervención de la persona, a un número indeterminado de personas físicas.

3. Podrá utilizarse un mecanismo de certificación aprobado con arreglo al artículo 42 como elemento que acredite el cumplimiento de las obligaciones establecidas en los apartados 1 y 2 del presente artículo. [45]

CAPÍTULO 8. ANEXO I: RGPD

Capítulo 9

ANEXO II: Herramientas empleadas

El presente capítulo tratará de presentar todas las herramientas empleadas para llevar a cabo las pruebas expuestas en el diseño del capítulo anterior y así completar las correspondientes auditorías sobre las apps en estudio. El uso de herramientas a la hora de llevar a cabo auditorías de seguridad resulta fundamental, puesto que su uso facilita enormemente el trabajo realizado, automatizando muchas de las tareas necesarias y haciendo más fáciles otras.

Para llevar a cabo una auditoría móvil será fundamental el uso de un emulador Android o ios que nos permita descargar y ejecutar la aplicación en estudio. Además, otras herramientas tales como MobSF o ABD que automatizan la ejecución de diversas pruebas, tanto estáticas como dinámicas, también resultarán indispensables para nuestro estudio. Por último, también se emplearán herramientas de análisis de código fuente como jadx y de análisis de flujos de comunicaciones como wireshark.

Todas estas herramientas, así como alguna más que se empleará en momentos puntuales, se presentarán en su totalidad a continuación en los apartados de este capítulo.

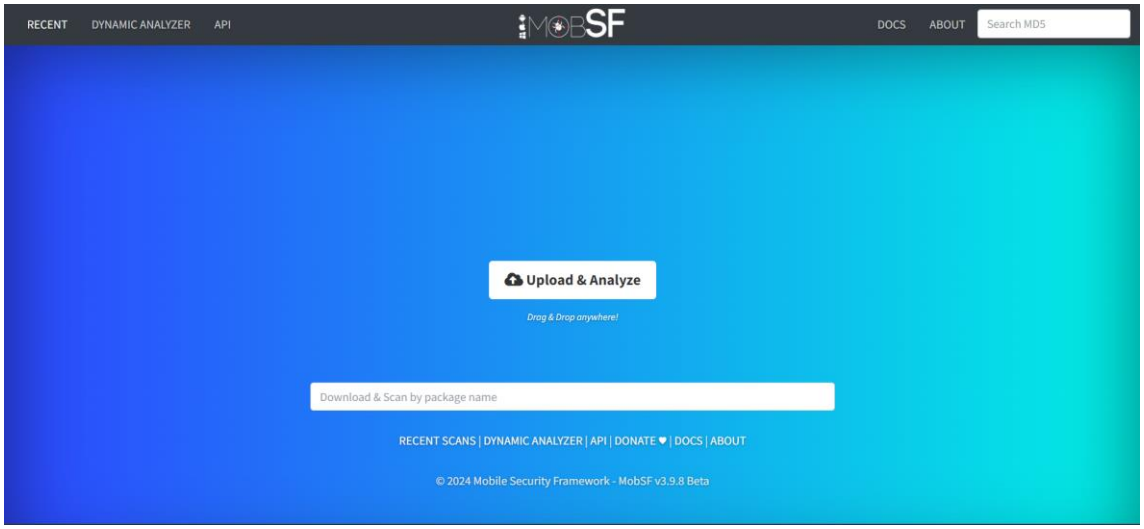
9.1. Mobile Security Framework (MOBSF)

Mobile Security Framework (MobSF) es una herramienta avanzada diseñada para ofrecer una visión general del estado de la seguridad y rendimiento de las aplicaciones móviles en plataformas Android e iOS. Esta herramienta realiza un análisis profundo, identificando vulnerabilidades y riesgos en el código y las bibliotecas de terceros. Además, evalúa el desempeño en términos de uso de recursos y eficiencia de la red. MobSF integra análisis estático y dinámico para proporcionar una visión completa de posibles problemas y comportamientos anómalos, generando informes detallados donde destacan hallazgos clave y recomendaciones, facilitando la mejora de la seguridad y el rendimiento de las apps. [47]

Utilizar Mobsf ofrece múltiples beneficios, como la mejora de la seguridad al identificar y solucionar vulnerabilidades, la optimización del rendimiento de las aplicaciones, el ahorro de tiempo y recursos mediante análisis automatizados, o la seguridad del cumplimiento de normativas y estándares de seguridad. Esta herramienta es esencial para cualquier equipo de desarrollo y seguridad que busca crear aplicaciones móviles seguras, eficientes y de alta calidad.

La configuración a realizar para que la herramienta de análisis estático de MobSF funcione es muy sencilla. En mi caso el servicio MobSF estará funcionando en un contenedor Docker el cual lo ejecutará en la dirección “localhost" de mi equipo. Todo esto se consigue ejecutando el

comando ```docker build -t mobsf ```, para construir el contenedor que contiene el servicio, y ```docker run -it -p 8000:8000 mobsf``` para mapear el puerto 8000 del sistema y correr la herramienta en la dirección "127.0.0.1:800".
Al acceder a esa misma dirección, se encontrará la interfaz principal de MobSF en la cual procederemos a subir el archivo .apk de la app que queremos auditar para proceder con su correspondiente análisis estático.



Para el análisis dinámico de las aplicaciones, será necesaria alguna configuración adicional para establecer la dirección sobre la cual MobSF estudiará el tráfico. Esta dirección se corresponderá con la del emulador en el que se ejecuta la app a auditar y se configurará editando el valor de la variable de sistema "MOSBF_ANALYZER_IDENTIFIER", añadiendo la dirección que queramos en la siguiente línea del fichero "/home/mobsf/.MobSF/config.py":

```
# DYNAMIC ANALYZER SETTINGS
# -------------------------
# =======ANDROID DYNAMIC ANALYSIS SETTINGS===========
ANALYZER_IDENTIFIER = os.getenv('MOBSF_ANALYZER_IDENTIFIER', '')
FRIDA_TIMEOUT = int(os.getenv('MOBSF_FRIDA_TIMEOUT', '4'))
ACTIVITY_TESTER_SLEEP = int(os.getenv('MOBSF_ACTIVITY_TESTER_SLEEP', '4'))
```

Por último, la herramienta Mobsf solo reconoce tres herramientas de emulación para la ejecución de la app en estudio:

- Genymotion Android VM versión 4.1 - 11.0 (x86, upto API 30)
- Android Emulator AVD (non production) version 5.0 - 9.0 (arm, arm64, x86, and x86_64 upto PI 28)
- Corellium Android VM (userdebug builds) version 7.1.2 - 11.0 (arm64 upto API 30)

9.2. Genymotion

Genymotion es una herramienta de emulación que permite crear y ejecutar máquinas virtuales que simulan dispositivos Android, proporcionando un entorno controlado para probar aplicaciones en una variedad de configuraciones tanto de hardware como de software. [48]

La importancia de Genymotion en la realización de auditorías sobre aplicaciones móviles radica en su capacidad para replicar fielmente el comportamiento de dispositivos reales. Esta herramienta puede ser empleada para ejecutar pruebas, detectar vulnerabilidades de seguridad y asegurar que las aplicaciones cumplan con los estándares de privacidad y protección de datos.

Para las auditorías a realizar, se ha creado una máquina virtual corriendo el sistema operativo Android 10 en la que se emulará la ejecución de las aplicaciones, y que tiene las siguientes especificaciones técnicas:

OS image

Android	Android 10.0
Source	Genymotion
Android version	10.0.0
Image version	3.1.0
Beta	No
Architecture	x86
Release date	07/09/2023
Size	442.18 MB

Hardware

Processor(s)	4
Memory size	4096 MB
VM Heap size	512 MB

En cuanto la configuración de red de la app, está se encontrará ligada al adaptador del sistema anfitrión y se ejecutará en la dirección: “192.168.220.105:5555”.

9.3. ADB

Android Debug Bridge (ADB) es una herramienta de línea de comandos versátil que permite la comunicación con un dispositivo Android. ADB facilita una variedad de acciones como la instalación y depuración de aplicaciones, el acceso a una shell de Unix que permite ejecutar varios comandos en un dispositivo y la transferencia de archivos entre dispositivos. En el contexto de una auditoría de seguridad, ADB es esencial para interactuar con el dispositivo de prueba y obtener información detallada sobre el comportamiento de la aplicación en tiempo real. [49]

Para utilizar ADB, primero se debe habilitar la depuración USB en el dispositivo Android o emulador. Luego, mediante el comando adb connect [dirección_ip_del_dispositivo], se establece una conexión entre el sistema anfitrión y el dispositivo. ADB permite ejecutar comandos como adb install [ruta_del_apk] para instalar aplicaciones, adb logcat para obtener registros del sistema y adb shell para acceder a la terminal del dispositivo y ejecutar comandos directamente.

9.4. Drozer

Drozer es una herramienta de seguridad móvil que actúa como un marco de prueba para aplicaciones Android, permitiendo a los evaluadores de seguridad analizar el comportamiento y las vulnerabilidades de las aplicaciones. Drozer permite realizar pruebas de penetración en aplicaciones Android y verificar configuraciones de seguridad del dispositivo.

La utilización de Drozer requiere, en primer lugar, instalar la herramienta en el sistema anfitrión y luego instalar el agente Drozer en el dispositivo Android. El agente puede instalarse usando ADB con el comando adb install [ruta_del_agente_drozer.apk]. Una vez instalado, se inicia el agente en el dispositivo y se conecta mediante el comando drozer console connect. Drozer permite ejecutar módulos que pueden auditar diferentes aspectos de la aplicación, como permisos, actividades expuestas, servicios y receptores, proporcionando un análisis exhaustivo de la seguridad de la aplicación.

9.5. Wireshark

Wireshark es una herramienta de análisis de protocolos de red ampliamente utilizada para capturar y examinar el tráfico de red en tiempo real. En el contexto de auditorías de seguridad móvil, Wireshark permite monitorear y analizar las comunicaciones de red de una aplicación para identificar posibles vulnerabilidades y comportamientos anómalos.

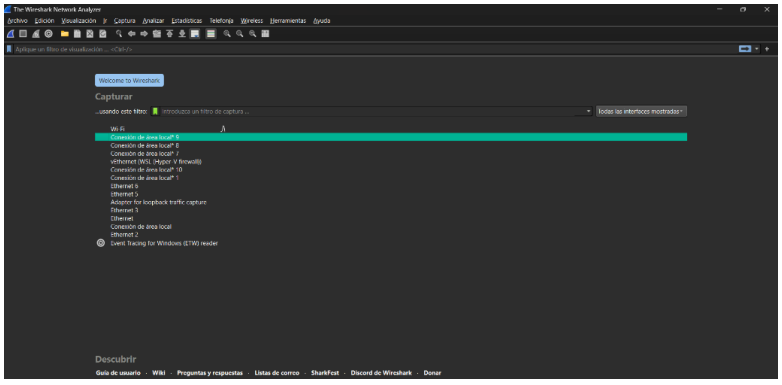


Ilustración 85: Interfaz principal de la herramienta WireShark

Para hacer uso Wireshark, primero se debe configurar el dispositivo o emulador para enrutar su tráfico de red a través del sistema anfitrión donde Wireshark está instalado. Esto puede lograrse configurando un proxy o utilizando herramientas de enrutamiento. Una vez configurado, se inicia la captura de tráfico en Wireshark seleccionando la interfaz de red adecuada. Wireshark permite filtrar y analizar el tráfico capturado, destacando posibles vulnerabilidades como datos sensibles transmitidos sin cifrar, conexiones a servidores no confiables y otros riesgos de seguridad.

9.6. CypherChef

CypherChef es una herramienta de análisis criptográfico que ayuda a evaluar la implementación de mecanismos de cifrado en aplicaciones móviles. CypherChef puede identificar debilidades en el uso de algoritmos de cifrado, configuraciones incorrectas y posibles vulnerabilidades en la gestión de claves. [50]

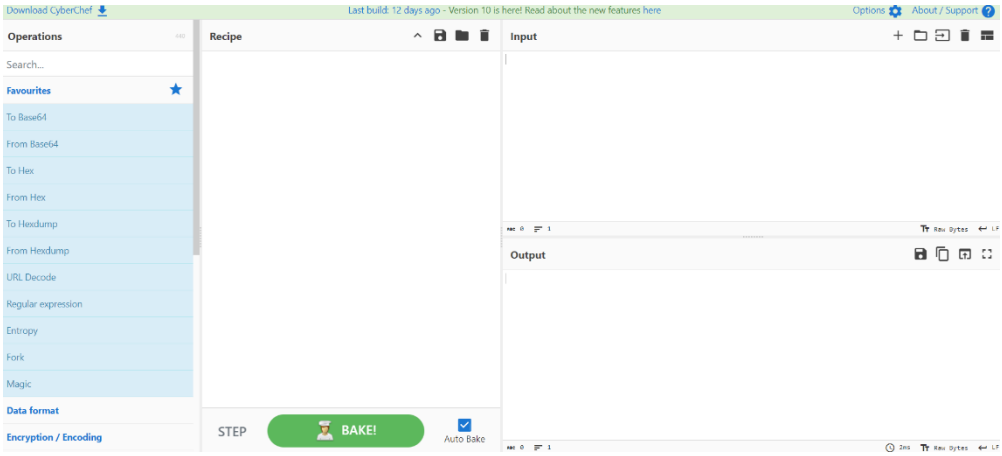


Ilustración 86: Interfaz principal de la herramienta CypherChef

Para utilizar CypherChef en una auditoría de seguridad, primero se debe extraer el código de la aplicación móvil utilizando herramientas como jadx para descompilar el APK. Luego, se analizan las secciones del código relacionadas con el cifrado y la seguridad de datos. CypherChef proporciona una interfaz para evaluar las prácticas criptográficas utilizadas y ofrecer recomendaciones sobre mejoras necesarias para fortalecer la seguridad de la aplicación.

CAPÍTULO 9. ANEXO II: HERRAMIENTAS EMPLEADAS