Desarrollo de una metodología para el análisis de seguridad en aplicaciones móviles.

Nombre Estudiante

Anton Chavarria García

Máster Universitario en Ciberseguridad y Privacidad

Área de trabajo final

Seguridad empresarial

Nombre Tutor/a de TF

Daniel Brande Hernandez

Profesor/a responsable de la asignatura

Victor Garcia Font

Fecha Entrega

13 de junio de 2023

Universitat Oberta de Catalunya

FICHA DEL TRABAJO FINAL

Título del trabajo:	Desarrollo de una metodología para el análisis de seguridad en aplicaciones móviles.
Nombre del autor:	Anton Chavarria García
Nombre del consultor/a:	Daniel Brande Hernandez
Nombre del PRA:	Victor García Font
Fecha de entrega (mm/aaaa):	06/2023
Titulación o programa:	Máster Universitario en Ciberseguridad y Privacidad
Área del Trabajo Final:	Seguridad empresarial
Idioma del trabajo:	Castellano
Palabras clave	Seguridad, aplicaciones móviles, metodología

Resumen del Trabajo

La finalidad de este TFM es desarrollar una metodología para el análisis de seguridad a aplicaciones para dispositivos móviles, en concreto las desarrolladas en los sistemas operativos Android e iOS.

Para ello se realizará un estudio de la actualidad en respecto a las vulnerabilidades y a la seguridad actual dentro de las aplicaciones móviles, ya que las explotaciones de vulnerabilidades y los ataques de malware están en crecimiento en el ámbito de las aplicaciones móviles. Situación que provoca que los datos sensibles e información que tienen los usuarios estén más expuestos y la utilización de aplicaciones móviles obtenga más riesgos.

Se mostrarán las diferentes metodologías que existen relacionadas con auditorías de aplicaciones móviles, los procesos y pasos a seguir en auditorías móviles y las herramientas más comunes para llevar a cabo dichas auditorías o análisis de seguridad. Basándonos en esta información recopilada se propondrá una metodología a aplicar en los distintos análisis de seguridad en aplicaciones móviles.

Por último, se ejecutará un análisis de una aplicación móvil Android, basándonos en la metodología desarrollada y basándonos en las ya creadas para finalmente exponer los resultados obtenidos.

Abstract

The purpose of this master's thesis is to develop a methodology for the security analysis of mobile applications, specifically those developed on the Android and iOS operating systems.

To achieve this, a study will be conducted on the current state of vulnerabilities and security within mobile applications, as vulnerability exploits and malware attacks are on the rise in the mobile application domain. This situation exposes sensitive data and user information, increasing the risks associated with the use of mobile applications.

Various methodologies related to mobile application audits, the processes, and steps involved in mobile audits and the most common tools for conducting such audits or security analyses will be presented. Based on the gathered information, a methodology will be proposed for conducting security analyses on different mobile applications.

Lastly, an analysis of an Android mobile application will be performed based on the developed methodology and existing methodologies in order to present the obtained results.

Índice

1 Introducción 10

1.1 Contexto y justificación del Trabajo 10
1.2 Objetivos del Trabajo 11
1.3 Impacto en sostenibilidad, ético-social y diversidad 11
1.4 Enfoque y método seguido 13
1.5 Planificación del Trabajo 13
1.6 Breve sumario de productos obtenidos 15
1.7 Breve descripción de los otros capítulos de la memoria 15

2 Estado del arte 17

2.1 Android 17
2.2 IOS 18
2.3 Tipos de aplicaciones que existen en Android e iOS 19
2.4 Aplicaciones de Android / IOS y sus categorías 20
2.5 Principales vulnerabilidades de las aplicaciones móviles 22

M1- Uso inadecuado de la plataforma: 23
M2- Almacenamiento inseguro de datos: 23
M3- Comunicaciones inseguras: 24
M4-Autenticación insegura: 24
M5- Criptografía insuficiente: 25
M6- Autorización insegura: 26
M7- Mala calidad de código: 26
M8- Manipulación de código: 27
M9- Ingeniería inversa: 27
M10- Funcionalidad extraña: 28

2.6 Modelos de seguridad 28
2.7 Metodologías de seguridad para realizar auditorías de seguridad 33
2.7.1 Auditorías de seguridad 37
2.7.2 Tipos de análisis 39
2.8 Procedimiento habitual de un analista en las auditorías 40
2.9 Herramientas y Frameworks 44

3 Caso Práctico 47

3.1 Metodología 47
3.1.1 Aplicación elegida 48
3.1.2 Recopilación de datos e información (Primer procedimiento) 49
3.1.3 Análisis de los diferentes activos y riesgos (Segundo procedimiento) 51

3.1.4 Análisis estático y dinámico de la aplicación (Tercer procedimiento) 57
3.1.4.1 Análisis estático 57
3.1.4.1.1 Análisis estático con MobSF 57
3.1.4.1.2 Detalles del análisis estático 57
3.1.4.2 Análisis dinámico 62
3.1.4.2.1 Análisis dinámico con MobSF 62
3.1.4.2.2 Detalles del análisis dinámico 63
3.1.5 Análisis de los resultados (Cuarto procedimiento) 72
3.1.6 Documentación y resultados (Quinto procedimiento) 76

Conclusiones y trabajos futuros 78
Bibliografía 82
Anexos 85
Anexo A Instalación de la aplicación elegida 85
Anexo B Instalación de MobSF (Análisis estático) 86
Anexo C Instalación de MobSF (Análisis dinámico) 91

Lista de figuras

Figura	Página
Figura 1 Contexto y justificación del trabajo	10
Figura 2 Diagrama de Gantt de la planificación del TFM	15
Figura 3 Android	17
Figura 4 iOS	18
Figura 5 TOP 10 OWASP Mobile Risks	22
Figura 6 Proceso de Autenticación	29
Figura 7 Almacenamiento seguro de los datos	30
Figura 8 Conexión segura(http>https)	30
Figura 9 Autenticación insegura	31
Figura 10 Manipulación de código	32
Figura 11 Funcionalidad extraña	33
Figura 12 MASVS	34
Figura 13 MASTG	35
Figura 14 MASPT	36
Figura 15 MASA	37
Figura 16 Tipos de auditoría	38
Figura 17 Tipos de análisis	40
Figura 18 Definir el alcance	40
Figura 19 Recopilar información	41
Figura 20 Pruebas de seguridad	42
Figura 21 Evaluación de cumplimiento	42
Figura 22 Análisis de rendimiento	43
Figura 23 Documentación de hallazgos	43
Figura 24 Comunicación y presentación	43
Figura 25 Angr	44
Figura 26 Frida	45

Figura 27 MobSF	45
Figura 28 Ghidra	46
Figura 29 Radare2	46
Figura 30 Metodología	47
Figura 31 Aplicación vulnerable(Damn Vulnerable Bank)	48
Figura 32 Login	49
Figura 33 Datos de la aplicación	51
Figura 34 Funcionalidades clave	51
Figura 35 Actividades	54
Figura 36 Riesgos en aplicaciones móviles	56
Figura 37 Resumen del análisis estático con MobSF	57
Figura 38 Actividades, servicios, receptores y proveedores	58
Figura 39 Gráficas de seguridad, riesgo y distribución de seguridad.	61
Figura 40 Hallazgos dentro de la app y con su gravedad	62
Figura 41 Análisis dinámico de la aplicación vulnerable	62
Figura 42 Frida scripts I	65
Figura 43 Frida scripts II	65
Figura 44 Funciones o procesos de análisis utilizados dentro de MobSF.	67
Figura 45 TLS/SSL Pruebas de seguridad	68
Figura 46 Base64 string descodificado I	70
Figura 47 Base64 string descodificado II	70
Figura 48 Base64 string descodificado III	70
Figura 49 Sqlite database “webviewCookies”	71
Figura 50 Sqlite database “webviewWeb_data”	71
Figura 51 Ficheros XML	72
Figura 52 Ficheros extra	72
Figura 53 Documentación de la metodología	78
Figura 54 Conclusiones y trabajos futuros	79

Figura 55 Instalación de la herramienta adb	*85*
Figura 56 Clonación del repositorio aplicación vulnerable	*85*
Figura 57 Instalación de la aplicación en ubuntu	*85*
Figura 58 Repositorio de la aplicación vulnerable	*86*
Figura 59 ZIP de la aplicación	*86*
Figura 60 ZIP de la aplicación descargado	*86*
Figura 61 Ficheros de la aplicación	*86*
Figura 62 apt update	*87*
Figura 63 apt list –upgradable	*87*
Figura 64 apt upgrade	*87*
Figura 65 apt install python3-pip	*88*
Figura 66 apt install openjdk-8-jdk	*88*
Figura 67 apt install git	*88*
Figura 68 Comprobación de requisitos	*88*
Figura 69 Instalación de las dependencias	*89*
Figura 70 Clonación repositorio MobSF	*89*
Figura 71 Cambio de directorio	*89*
Figura 72 Aplicación del setup en Ubuntu	*89*
Figura 73 Instalación completada	*90*
Figura 74 Iniciación del Framework	*90*
Figura 75 Interfaz de MobSF	*90*
Figura 76 Android Studio.exe	*92*
Figura 77 La aplicación vulnerable en Android Studio	*92*
Figura 78 Android Virtual Device (AVD)	*92*
Figura 79 Android Virtual Device creado	*93*
Figura 80 Git	*93*
Figura 81 Python	*93*
Figura 82 JDK	*94*
Figura 83 Visual Studio Build	*94*

Figura 84 C++	94
Figura 85 OpenSSL	95
Figura 86 wkhtmltopdf	95
Figura 87 wkhtmltopdf en el path	95
Figura 88 Clonación del repositorio MobSF	96
Figura 89 Cambio de directorio	96
Figura 90 Completamos el setup	96
Figura 91 Instalación completada	96
Figura 92 Lista de emuladores activos	97
Figura 93 ADB_BINARY en la configuración	97
Figura 94 Ejecución de AVD	97
Figura 95 Aplicación vulnerable en la AVD	98
Figura 96 Iniciación de MobSF	98
Figura 97 Interfaz de MobSF	99
Figura 98 Soporte de MobSF	99
Figura 99 Aplicación vulnerable en MobSF	99
Figura 100 Iniciación del análisis dinámico	99

Lista de tablas

Tablas	*Página*
Tabla I Permisos de la app	*58*
Tabla II Seguridad de la red	*59*
Tabla III Análisis del archivo Manifest	*59-60*
Tabla IV Análisis de código	*61*
Tabla V Análisis Dex class Loader	*68-69*
Tabla VI Binder	*69*
Tabla VII Sqlite Database	*71*

1 Introducción

1.1 Contexto y justificación del Trabajo

El proceso de este trabajo comienza con la revisión de las diferentes metodologías existentes con la idea de proponer una implementación práctica con base en el modelo teórico, es decir, obtener toda la información posible y realizar una implementación con base en las metodologías ya existentes.

Una de las razones claves para realizar este trabajo es el gran impulso y cantidad de nuevas aplicaciones que existen y se crean cada día. En la actualidad la mayoría de las empresas comienzan a desarrollar aplicaciones móviles de sus negocios al conocer el gran tiempo que utilizamos en las diferentes aplicaciones móviles y los diferentes beneficios que conservan. Las empresas utilizan las aplicaciones móviles para convertirse en empresas más accesibles y cercanas a los usuarios. Situación que provoca cada día más riesgos, ya que al tener más aplicaciones creadas la cantidad de vulnerabilidades aumenta, por lo que concluye en que tanto las empresas como los usuarios se van a someter a más riesgos con la utilización de estas aplicaciones. Problema que cada día se engrandece más y por el cual debemos realizar mayores investigaciones debido a que no solo cada vez se crean más aplicaciones, sino que cada día damos datos más importantes o más cruciales, por lo que el impacto en las empresas o usuarios será mayor.

Por todos estos procesos y razones realizaremos este trabajo con la idea de intentar solventar los problemas comentados de la mejor manera posible. Para ello realizaremos una investigación de las diferentes metodologías y vulnerabilidades existentes para poder realizar una implementación práctica de una metodología de análisis de seguridad en aplicaciones móviles lo más completa posible y que intente reducir tanto las vulnerabilidades como los diferentes riesgos existentes.

El resultado final esperado es una implementación de una metodología para el análisis de seguridad en aplicaciones móviles. Además, de un análisis práctico aplicado en una aplicación móvil de prueba con la implementación de la metodología mencionada previamente.

Figura 1 Contexto y justificación del trabajo

1.2 Objetivos del Trabajo

1) Conocer las principales vulnerabilidades que hay sobre las aplicaciones móviles.

En este primer objetivo realizaremos una inmersión en las diferentes vulnerabilidades que existen dentro de las aplicaciones móviles.

2) Modelos de seguridad en Android/iOS.

Estudio y definición de los distintos modelos de seguridad que existen dentro de Android e IOS. También incluiremos los diferentes controles que tienen los sistemas operativos para prevenir y protegerse de las diferentes vulnerabilidades que existen.

3) Analizar distintas metodologías existentes para hacer un análisis de seguridad en aplicaciones móviles.

Efectuar un análisis de las metodologías actuales en seguridad en aplicaciones móviles. Obtendremos toda la información posible para poder ejecutar una implementación a las metodologías creadas para realizar un análisis eficaz.

4) Realizar un marco para hacer auditorías de seguridad y realizar el análisis práctico de un caso propuesto.

Tras obtener toda la información de lo relacionado con la seguridad en aplicaciones móviles, ejecutaremos un marco con el que podamos hacer auditorías de seguridad. Una vez consigamos el marco de seguridad, procederemos con un análisis práctico sobre una aplicación de prueba.

5) Elaborar un informe con las conclusiones a las que se ha llegado.

Llevaré a cabo un informe y una documentación completa tanto de la nueva implementación de metodología desarrollada como de la aplicación de la misma. En la que podamos observar tanto el proceso como los resultados o conclusiones finales obtenidas del trabajo realizado.

1.3 Impacto en sostenibilidad, ético-social y diversidad

Identificación de los impactos positivos y/o negativos del TFM en las tres dimensiones de la CCEG:

Dimensión de sostenibilidad

En relación con la dimensión de sostenibilidad, el resultado obtenido es la metodología. En las diferentes situaciones posibles, si encontramos impactos serán positivos en aspectos de sostenibilidad medioambiental y/o huella ecológica.

En el desarrollo de este TFM se va a utilizar un ordenador, el cual puede llegar a tener ciertas piezas que provienen de países en los cuales se ha dañado el medioambiente para todo el proceso, sin embargo, al utilizar un ordenador ya creado el impacto es nulo en cuanto al medioambiente. Otra de las posibilidades es que al ser una metodología para mejorar la seguridad en aplicaciones móviles, el buen resultado del TFM en perspectiva sostenibilidad aumentaremos la confianza en los diferentes usuarios en el uso de las aplicaciones móviles. Lo que provocará una mayor utilización de las mismas, por lo que contribuiremos en una mayor eficiencia en los diferentes procesos de trabajo y en la reducción del consumo de papel y otros recursos dañinos para el medio ambiente, esta vez sustituidos por aplicaciones móviles. También a la hora de realizar el TFM tendremos un impacto en la sostenibilidad ambiental, ya que se puede reducir la cantidad de tiempo dedicado al transporte y el consumo de energía, tanto en la producción como en el transporte de bienes físicos.

En definitiva, en la realización de este TFM voy a utilizar un ordenador ya creado y el producto final obtenido va a ser la metodología, por lo que en ninguno de los casos voy a crear nada que pueda dañar la sostenibilidad medioambiental y/o huella ecológica.

Dimensión de comportamiento ético y de responsabilidad social (RS)

Con respecto a la dimensión comportamiento ético y de responsabilidad social, el impacto positivo es inmenso debido a que este TFM busca que tanto los datos como la privacidad de las personas que utilizan las distintas aplicaciones móviles estén más seguros, es decir, este TFM está enfocado para encontrar la mayor seguridad posible en la información o en los datos de las personas. Está centrado en mejorar la seguridad en las aplicaciones, lo que provocará una mejora de comportamiento ético profesional de los distintos dueños de las aplicaciones móviles debido a que lanzarán aplicaciones con mayor seguridad en las distintas plataformas. Lo que provocará que los diferentes usuarios que usen estas aplicaciones tendrán mayor seguridad y sus datos o archivos privados estén más protegidos. Los procesos de análisis para obtener las diferentes vulnerabilidades o amenazas en las diferentes aplicaciones se realizan bajo el código ético profesional. Como ingeniero existe la responsabilidad ética de que en caso de que en un futuro cree una aplicación, busque que sea lo más segura posible para no afectar con mis conductas a terceros, por lo que se podría argumentar que este TFM está efectuado bajo una preocupación ética sobre las actuales y no tan actuales aplicaciones móviles.

Dimensión de diversidad, género y derechos humanos

El proceso realizado en este TFM tiene un impacto positivo directo en la legislación debido a que tiene relación directa con los datos, la privacidad y la seguridad de las personas. Sobre todo tiene un gran impacto en la seguridad de los datos debido a que la intención de este TFM es el aumento de la seguridad en las diferentes aplicaciones móviles con la utilización de la metodología creada. Como consecuencia de este

proceso, las aplicaciones tendrán una mayor seguridad y estarán más seguros tanto los datos de los creadores como de los posibles usuarios de las aplicaciones.

1.4 Enfoque y método seguido

La estrategia a seguir en este trabajo será el plantear una implementación práctica de una metodología para el posterior análisis de seguridad en diferentes aplicaciones móviles. Para poder llevar la estrategia a cabo de la mejor manera posible, basaremos esta nueva implementación en metodologías ya existentes. Entre estas metodologías podemos encontrar la de OWASP y dentro de ella su Mobile security testing guide.

En cuanto al método a seguir los separaremos en tres distintas fases. La primera de ellas tratará del análisis del estado del arte actual, de las diferentes metodologías de las cuales aprenderemos todos sus procesos y distintos modelos que existen. Una vez hayamos analizado toda la información de todas las metodologías existentes y toda la información relacionada con el mundo de las aplicaciones móviles y la seguridad de cada una de ellas, pasaremos a la segunda parte del método a seguir.

En esta segunda parte realizaremos el desarrollo de un marco de aplicación de manera práctica con base en todo el conocimiento adquirido y con la búsqueda de realizar una implementación que complemente a las ya existentes en busca de mejorar la seguridad en las aplicaciones móviles.

Para finalizar con el método a seguir de este trabajo realizaré una aplicación práctica sobre una aplicación de prueba para comprobar la eficacia y qué resultados obtenemos de la nueva implementación realizada.

1.5 Planificación del Trabajo

Recursos:
Los recursos necesarios serán una buena investigación sobre el mundo de las aplicaciones móviles y sobre todo centralizado en la seguridad en las aplicaciones móviles. Toda esta investigación se realizará sobre recursos existentes de internet relacionados directamente con este TFM. En cuanto a recursos computacionales, necesitaremos un ordenador con el que ejecutaremos las posteriores pruebas a una aplicación con la utilización de la metodología creada y la documentación necesaria para completar el TFM.

Tareas a realizar:

PEC 1: 1/03/23 - 14/03/23

Tarea 1: Plan de trabajo.
- **Tarea 1.1:** Contexto y justificación de trabajo.
- **Tarea 1.2:** Objetivos del Trabajo.
- **Tarea 1.3:** Impacto en sostenibilidad, ético-social y diversidad.

Tarea 1.4: Enfoque y método seguido.
Tarea 1.5: Planificación del trabajo.
Tarea 1.6: Breve sumario de productos obtenidos.
Tarea 1.7: Breve descripción de los otros capítulos de la memoria.

Tarea 2: Entrega del PEC 1.

PEC 2: 15/03/23 - 11/04/23

Tarea 3: Análisis del estado del arte.
Tarea 3.1: Análisis de amenazas y vulnerabilidades en móviles
Tarea 3.2: Seguridad en android/ios
Tarea 3.3: Metodologías de análisis de seguridad.
Tarea 3.4: Herramientas utilizadas en la seguridad de las aplicaciones móviles.

Tarea 4: Entrega del PEC 2.

PEC 3: 12/04/23 - 09/05/23

Tarea 5: Aplicación de la metodología en un caso práctico.
Tarea 5.1: Desarrollar una metodología para el análisis de seguridad en aplicaciones móviles.
Tarea 5.2: Documentación de la metodología desarrollada.
Tarea 5.3: Aplicación continua de la metodología desarrollada en una aplicación móvil.
Tarea 5.4: Documentación de la aplicación de la metodología.

Tarea 6: Entrega del PEC 3.

PEC 4: 10/05/23 - 13/06/23

Tarea 7: Análisis de resultado y metodología aplicada.
Tarea 7.1: Análisis práctico de seguridad de una aplicación móvil con la metodología desarrollada.
Tarea 7.2: Documentación de la metodología aplicada en una aplicación móvil.

Tarea 8: Entrega de la memoria final y completa.
Tarea 8.1: Revisión y corrección de entregas anteriores.

Diagrama de GANTT :

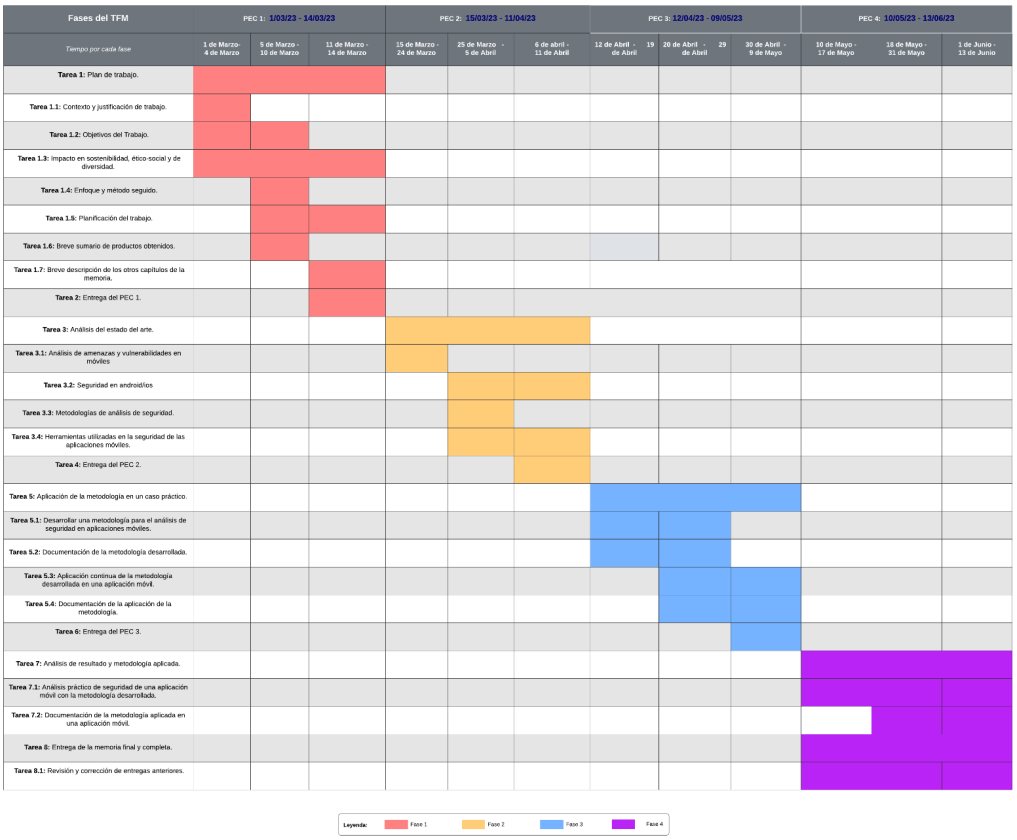


Figura 2 Diagrama de Gantt de la planificación del TFM

1.6 Breve sumario de productos obtenidos

El producto obtenido será una metodología completa para realizar análisis a aplicaciones móviles en búsqueda de encontrar vulnerabilidades y amenazas. Otro de los productos que obtendremos será un análisis completo de una aplicación con la metodología mencionada previamente.

1.7 Breve descripción de los otros capítulos de la memoria

PEC 1. Plan de trabajo:

En este primer capítulo determinaré en qué consistirá el trabajo final y explicaré el razonamiento de porque voy a realizar este trabajo. Todo ello realizando una planificación para el posterior desarrollo del trabajo. Dentro de este capítulo contendremos los siguientes apartados: Tarea 1.1: Contexto y justificación de trabajo,

Tarea 1.2: Objetivos del Trabajo, Tarea 1.3: Impacto en sostenibilidad, ético-social y de diversidad, Tarea 1.4: Enfoque y método seguido, Tarea 1.5: Planificación del trabajo, Tarea 1.6: Breve sumario de productos obtenidos y Tarea 1.7: Breve descripción de los otros capítulos de la memoria.

PEC 2. Entrega de seguimiento:

En este segundo capítulo continuaremos con el TFM haciendo hincapié en el apartado del estado del arte, en el cual realizaré una investigación sobre todo el mundo de las aplicaciones móviles, las diferentes seguridades y metodologías que en la actualidad existen. En esta segunda entrega realizaré una documentación sobre la investigación realizada. Dentro de este capítulo contendremos los siguientes apartados: Tarea 3.1: Análisis de amenazas y vulnerabilidades en móviles, Tarea 3.2: Seguridad en android/ios, Tarea 3.3: Metodologías de análisis de seguridad y Tarea 3.4: Herramientas utilizadas en la seguridad de las aplicaciones móviles.

PEC 3. Entrega de seguimiento:

En este tercer capítulo continuaremos realizando los últimos apartados del TFM. En él aplicaremos la metodología diseñada a una aplicación real, de la cual obtendremos resultados de dicha aplicación. Continuaremos realizando los apartados correspondientes a la parte de la redacción de la memoria. Dentro de este capítulo contendremos los siguientes apartados: Tarea 5: Aplicación de la metodología en un caso práctico, Tarea 5.1: Desarrollar una metodología para el análisis de seguridad en aplicaciones móviles, Tarea 5.2: Documentación de la metodología desarrollada, Tarea 5.3: Aplicación continua de la metodología desarrollada en una aplicación móvil, Tarea 5.4: Documentación de la aplicación de la metodología.

PEC 4. Memoria final:

En este último capítulo realizaré la última parte del TFM, en el cual finiquitaré el TFM con los últimos apartados correspondientes y realizando la última versión de los ya realizados. Entregando de manera definitiva la última versión de la memoria del TFM. Dentro de este último capítulo de la memoria contendremos los siguientes apartados: Tarea 7: Análisis de resultado y metodología aplicada, Tarea 7.1: Análisis práctico de seguridad de una aplicación móvil con la metodología desarrollada, Tarea 7.2: Documentación de la metodología aplicada en una aplicación móvil, Tarea 8: Entrega de la memoria final y completa y Tarea 8.1: Revisión y corrección de entregas anteriores.

2 Estado del arte

2.1 Android

Android es un sistema operativo móvil desarrollado por Google y es el sistema operativo más popular del mundo. El sistema operativo se basa en el kernel de Linux y está diseñado para dispositivos móviles con pantalla táctil, como smartphones y tablets.

Android ha sufrido muchos cambios desde su lanzamiento en 2008 y ahora se encuentra en la versión 12.0. Las características de Android [1] incluyen muchas aplicaciones y juegos de Play Store, diseño de interfaz de usuario e integración total con los servicios de Google.

Android también tiene muchas herramientas y recursos para desarrolladores que ayudan a crear aplicaciones de alta calidad. Los desarrolladores pueden usar Android Studio, un entorno de desarrollo integrado (IDE), para producir aplicaciones de Android, utilizando lenguajes de programación como Java y Kotlin.

Otras características importantes de Android incluyen numerosas características de seguridad como el cifrado y la protección del dispositivo, diferentes tipos de acceso, integración total con dispositivos IoT y varias tecnologías de conectividad como WI-FI, Bluetooth y NFC.

En conclusión, Android es una popular plataforma móvil avanzada que proporciona muchas características y herramientas para desarrolladores y una experiencia única de usuario y de personalización para usuarios finales.

Figura 3 Android

2.2 IOS

iOS [2] es el sistema operativo móvil desarrollado por Apple para sus dispositivos móviles, incluidos Iphone, Ipad e Ipod Touch. El sistema operativo se basa en el kernel Darwin del sistema operativo Unix de código abierto, diseñado para los dispositivos móviles de Apple.

Desde su lanzamiento en 2007, iOS ha evolucionado y ahora se encuentra en la versión 15.1. iOS es conocido por su diseño limpio, facilidad de uso y estabilidad. Los usuarios de iOS pueden acceder a una variedad de aplicaciones y juegos a través de App Store, así como a servicios de Apple como Apple Music, Apple TV+ e iCloud.

Apple proporciona a los desarrolladores muchas herramientas y distintos recursos. El entorno de desarrollo integrado (IDE) para aplicaciones iOS y lenguajes de programación como Swift y Objective-C. Los desarrolladores también pueden aprovechar las funciones especiales de hardware y software de Apple, como Touch ID, Face ID, Apple Pay y la plataforma de realidad aumentada ARKit.

Las características de iOS incluyen un enfoque en la privacidad, la seguridad y en un sistema de notificación nativo. También incluye una cámara excelente y una integración completa con otros dispositivos Apple como Apple Watch y AirPods.

En definitiva, iOS es un sistema operativo móvil popular y avanzado, conocido por su vistoso diseño y facilidad de uso, que brinda a los desarrolladores muchas herramientas y recursos. La plataforma también ofrece facilidad de uso con un fuerte enfoque en la privacidad y la seguridad.

Figura 4 iOS

2.3 Tipos de aplicaciones que existen en Android e iOS

En este apartado definiremos los tipos de aplicaciones [3] que existen tanto en Android e iOS.

Aplicaciones nativas

Las aplicaciones nativas se crean específicamente para sistemas operativos específicos, principalmente Android e iOS. Blackberry y Windows son dos sistemas operativos adicionales para aplicaciones móviles. Las aplicaciones nativas generalmente se desarrollan para maximizar todas las herramientas y capacidades del teléfono, debido a que se crean utilizando la interfaz de usuario nativa del dispositivo. Las aplicaciones nativas garantizan un alto nivel de rendimiento y una interfaz de usuario atractiva.

Ventajas:

- Las aplicaciones nativas se crean específicamente para un sistema operativo y brindan la mejor experiencia de usuario.
- Las aplicaciones nativas se crean con gestos únicos, lo que brinda una experiencia de usuario positiva y es beneficiosa para todos los usuarios.

Desventajas:

- Debido al mantenimiento adicional que requieren las aplicaciones nativas, son más caras que otros tipos de aplicaciones.
- Necesita introducir nuevas funciones en un código base diferente.

Aplicaciones web

Las aplicaciones web son las aplicaciones producidas para el uso en navegadores. En su mayoría consisten en conexiones HTML, CSS y Javascript. Chrome, Firefox y otros navegadores pueden ejecutarlo. Dado que tanto las aplicaciones nativas como las webs tienen características y capacidad de respuesta casi idénticas, es fácil confundir la capacidad de respuesta y la utilidad de la una con la otra. Una de las mayores diferencias entre los dos es que las aplicaciones móviles nativas pueden funcionar tanto en línea como fuera de línea sin una conexión a internet activa, sin embargo, las aplicaciones web necesitan una conexión a Internet activa para funcionar. Las principales aplicaciones web incluyen Google Docs, Canva y Gmail.

Ventajas:

- Sencillez en su creación.

- En comparación con otro software, las aplicaciones web consumen menos almacenamiento.
- Todos los dispositivos vienen pre-cargados con aplicaciones web.
- Cualquier tipo de aplicación puede acceder fácilmente a las aplicaciones web.

Desventajas:

- Las aplicaciones web no tienen acceso a los recursos locales.
- Depende de las conexiones/redes de internet.

Aplicaciones Híbridas

Las aplicaciones que son híbridas pueden ejecutarse en muchas plataformas, incluidas Android e iOS. Además, estas se crean combinando aplicaciones nativas y web. Las aplicaciones híbridas se pueden instalar en varios dispositivos porque tienen una base de código común. Por ejemplo, también podemos lanzar la aplicación de Android para iOS. Como alternativa de desarrollo multiplataforma, los desarrolladores tienen más libertad creativa al crear sus aplicaciones porque no están restringidas por los estrictos estándares de diseño de Google o Apple. Entre los ejemplos de aplicaciones híbridas se incluyen Instagram y Uber entre otros. En las aplicaciones híbridas empleamos tecnologías como Flutter/Dart y React Native para su creación.

Ventajas:

- Se puede utilizar en varias plataformas.
- Los navegadores están integrados con él.
- Mantenido por numerosas versiones.
- Es menos costosa que una aplicación nativa gracias al código compartido.

Desventajas:

- Aplicaciones más lentas que las nativas.
- Podría haber algunos problemas de interfaz.
- Hay restricciones sobre el uso de todas las funcionalidades del sistema operativo y del hardware en los programas híbridos.

2.4 Aplicaciones de Android / IOS y sus categorías

Ambos sistemas operativos contienen diferentes aplicaciones [4] con características distintas. Entre todas ellas, las siguientes son las más utilizadas dentro de los dos sistemas operativos:

Aplicaciones de comercio electrónico:

Las aplicaciones de comercio electrónico son una figura de un modelo B2B. Las personas pueden pedir, prestar y vender varios productos, ahorrando tiempo y dinero. Se pueden utilizar para intercambiar productos comerciales en los mercados de internet utilizando software de comercio electrónico. Solo necesitamos realizar transacciones electrónicas utilizando el teléfono para comprar ciertos productos y artículos. Sitios web de comercio electrónico y aplicaciones como Flipkart, Amazon y OLX son algunos ejemplos.

Aplicaciones para la educación:

Las aplicaciones para la educación se utilizan con frecuencia para avanzar en el conocimiento y aumentar la productividad. Las personas pueden volverse más comprometidas y productivas con la ayuda de aplicaciones educativas. Las aplicaciones de aprendizaje son una excelente manera de lograrlo. Por ejemplo, edX, Duolingo, Google Classroom y SoloLearn.

Aplicaciones de redes sociales:

Las aplicaciones de redes sociales permiten que las personas interactúen y se relacionen entre sí. Estas aplicaciones se utilizan principalmente para la diversión y para compartir ciertos archivos o mensajes con conocidos o amigos. Varias personas utilizan las herramientas de las redes sociales para el espíritu empresarial, el marketing y otros fines relacionados con los negocios. Las plataformas de redes sociales incluyen, entre otras, Instagram, Facebook, WhatsApp, YouTube, TikTok y LinkedIn.

Aplicaciones de productividad:

Las herramientas de productividad pueden ayudar a organizar y terminar actividades complicadas, como redactar un correo electrónico o calcular una propina. Los usuarios tienen acceso a todos los archivos almacenados en el servicio de almacenamiento basado en la nube a través de una variedad de dispositivos gracias a la aplicación Google Drive. Las aplicaciones para aumentar la productividad vienen en una variedad de formas y tamaños, y con la habilidad de mejorar el flujo de trabajo de formas únicas.

Aplicaciones para el entretenimiento:

En todo el mundo, muchas personas usan aplicaciones para entretenerse. Incluye novelas, plataformas OTT y otros entretenimientos. Las personas pueden entretenerse en estas plataformas mientras aprenden diversos conocimientos sobre distintos temas. En la actualidad se están utilizando plataformas OTT, que son populares en este momento y cuyo desarrollo también se desea a escala global. Las principales aplicaciones de entretenimiento son Hotstar, Netflix y Amazon Prime Video.

Aplicaciones para las finanzas o banca:

Hace unos años era impensable el utilizar aplicaciones móviles con acceso a los bancos o las diferentes finanzas, sin embargo, en la actualidad la mayoría de las personas utilizan aplicaciones de banca, ya sea para comprobar datos bancarios como para realizar ciertas operaciones desde la aplicación del móvil. También hay casos en los que incluso utilizan las aplicaciones para realizar los diferentes pagos diarios sin necesidad de utilizar la tarjeta de crédito. Entre las diferentes aplicaciones de banca se encuentran la mayoría de los bancos, ya que cada uno de ellos mantiene su propia aplicación móvil.

2.5 Principales vulnerabilidades de las aplicaciones móviles

Para la seguridad del software, la fundación Open Web Application Security Project (OWASP) se encarga de ofrecer análisis y sugerencias de seguridad. OWASP [5] también enumera amenazas [6] de seguridad y vulnerabilidades en aplicaciones móviles.

La lista OWASP Mobile Top 10 de fallos de seguridad de aplicaciones móviles ofrece recomendaciones de mejores prácticas para abordar y reducir estos problemas de seguridad. Esta lista es esencial para priorizar los fallos de seguridad en las aplicaciones móviles y crear las suficientes defensas para evitar los ataques tanto estáticos como dinámicos que se aprovechan de las funcionalidades de las aplicaciones.

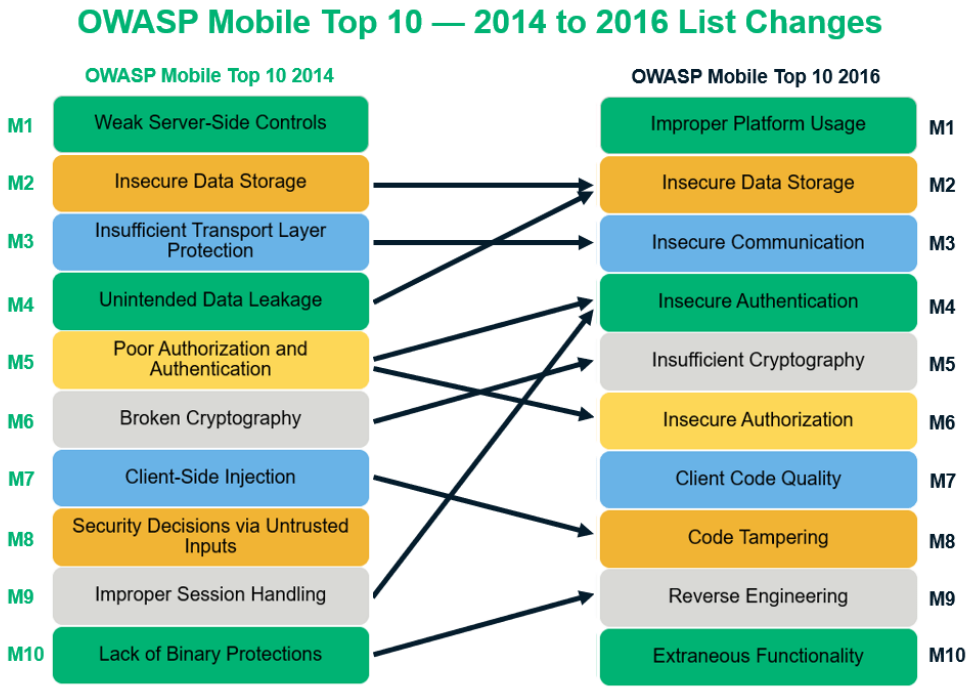


Figura 5 TOP 10 OWASP Mobile Risks

Por lo que los principales riesgos de seguridad dentro de las aplicaciones móviles serán las siguientes:

M1- Uso inadecuado de la plataforma:

Los agentes de amenazas en el contexto de la seguridad de aplicaciones móviles incluyen el mal uso de características de la plataforma o la falta de implementación de controles de seguridad. Esto puede manifestarse en problemas como el uso incorrecto de permisos, características de seguridad o controles del sistema operativo móvil. Estos agentes de amenazas pueden aprovechar vectores de ataque disponibles a través de llamadas de API expuestas, lo que facilita su explotación.

La prevalencia de esta vulnerabilidad es común, ya que ocurre cuando una organización expone servicios web o llamadas de API inseguras que son consumidas por la aplicación móvil. La detección de esta vulnerabilidad requiere un esfuerzo moderado, pero es posible identificarla. Los impactos técnicos y empresariales de esta vulnerabilidad dependen de la vulnerabilidad específica que se esté explotando a través del dispositivo móvil, como Cross-Site Scripting (XSS), que puede tener un impacto moderado tanto técnico como empresarial.

Para prevenir el "uso inadecuado de la plataforma"[7], es importante utilizar prácticas de codificación y configuración seguras en el lado del servidor de la aplicación móvil. Además, se recomienda seguir las directrices de seguridad publicadas por los fabricantes de la plataforma, como iOS, Android o Windows Phone. También es importante cumplir con las mejores prácticas comunes en el desarrollo de aplicaciones móviles y comprender cómo funcionan las protecciones y los modelos de permisos de la plataforma.

M2- Almacenamiento inseguro de datos:

Los agentes de amenaza en este contexto son adversarios que aprovechan dispositivos móviles perdidos o robados, o utilizan malware para acceder a información sensible. Pueden crear malware o modificar aplicaciones legítimas para robar datos personales identificables y otros datos sensibles almacenados en el dispositivo. La explotación de esta vulnerabilidad es relativamente fácil, ya que los adversarios pueden conectarse al dispositivo a través de una computadora y acceder a los directorios de aplicaciones.

La debilidad de seguridad asociada con este riesgo es la prevalencia del almacenamiento inseguro de datos [8]. Los equipos de desarrollo a menudo asumen erróneamente que los usuarios o el malware no podrán acceder a los sistemas de archivos de los dispositivos móviles y a los datos sensibles almacenados en ellos. Sin embargo, los sistemas de archivos son fácilmente accesibles, lo que permite a usuarios maliciosos o al malware inspeccionar los datos almacenados. El uso de

bibliotecas de encriptación deficientes y el enraizamiento del dispositivo también comprometen la protección de los datos.

Esta vulnerabilidad tiene graves impactos técnicos y comerciales. Puede resultar en pérdida de datos, extracción de información sensible mediante malware y aplicaciones modificadas. Además, los riesgos para el negocio incluyen robo de identidad, violaciones de privacidad, fraude, daño a la reputación, violaciones de políticas externas y pérdidas materiales. Para prevenir el almacenamiento inseguro de datos, es importante realizar una modelización de amenazas y comprender cómo las APIs y características como el almacenamiento en caché de URL, el registro y el almacenamiento de datos en el dispositivo pueden comprometer la seguridad de la aplicación y los datos almacenados.

M3- Comunicaciones inseguras:

La comunicación insegura [9] en las aplicaciones móviles puede ser explotada por agentes de amenazas como adversarios en la red local, dispositivos de operador o red comprometidos y malware en el dispositivo. Estos agentes pueden interceptar datos sensibles durante la transmisión a través de redes Wi-Fi, dispositivos de red o mediante el uso de malware. Esta vulnerabilidad es común y puede ser detectada mediante la inspección del tráfico de red y el análisis de la configuración y diseño de la aplicación.

Los impactos técnicos de la comunicación insegura son graves, ya que pueden exponer los datos de los usuarios y llevar al robo de cuentas de usuario. También puede facilitar ataques de phishing y de intermediarios. A nivel empresarial, la comunicación insegura puede provocar violaciones de privacidad, robo de identidad, fraude y daño a la reputación de la organización.

Para prevenir la comunicación insegura, se deben seguir mejores prácticas cómo aplicar SSL/TLS a los canales de transporte, utilizar cifrados sólidos y certificados confiables, verificar la identidad del servidor, alertar a los usuarios sobre certificados inválidos y evitar enviar datos sensibles a través de canales alternativos. Además, es recomendable implementar medidas específicas de cada plataforma, como la validación de certificados en iOS y la eliminación de código que acepte todos los certificados en Android. Estas medidas ayudarán a mitigar el riesgo de comunicación insegura y proteger la confidencialidad e integridad de los datos transmitidos por la aplicación móvil.

M4-Autenticación insegura:

El mal funcionamiento o la falta de esquemas de autenticación en aplicaciones móviles puede ser explotado por agentes de amenazas que utilizan ataques automatizados. Estos agentes pueden eludir o falsificar la autenticación al enviar solicitudes de servicio al servidor backend de la aplicación móvil, a menudo utilizando malware móvil o botnets. La prevalencia de esquemas de autenticación débiles es común en las

aplicaciones móviles debido al factor de forma de entrada y a la necesidad de autenticación sin conexión. Esto permite a los atacantes ejecutar funcionalidades de forma anónima y sin ser detectados, lo que compromete la identificación de usuarios y la capacidad de registrar actividades o detectar fuentes de ataques.

El impacto técnico de una autenticación deficiente es grave, ya que impide identificar a los usuarios y dificulta el registro de actividades, la detección de ataques y la prevención de futuros ataques. Además, los fallos de autenticación pueden revelar deficiencias en la autorización, puesto que no se verifica adecuadamente los permisos de los usuarios que emiten las solicitudes de acción. Esto permite la ejecución anónima de código sensible, resaltando las deficiencias tanto en los controles de autenticación como en los de autorización.

Para prevenir la autenticación insegura [10], es importante evitar patrones de diseño débiles y asegurarse de que los requisitos de autenticación coincidan con los de las aplicaciones web. La autenticación debe realizarse en el servidor siempre que sea posible y los datos almacenados en el dispositivo deben estar encriptados y protegidos contra ataques binarios. Se deben evitar contraseñas almacenadas en el dispositivo, utilizar tokens de autenticación específica del dispositivo y no permitir valores que sean fácilmente falsificables para la autenticación. También se deben reforzar los controles de autenticación y autorización en el lado del servidor, asumiendo que los controles del lado del cliente pueden ser eludidos.

M5- Criptografía insuficiente:

La criptografía insuficiente en aplicaciones móviles presenta amenazas por parte de agentes que tienen acceso físico a datos cifrados de manera incorrecta y malware móvil utilizado por adversarios. Los vectores de ataque son similares a los del OWASP Top Ten, donde cualquier llamada de API expuesta puede ser utilizada como vector de ataque.

Esta debilidad de seguridad es común en aplicaciones móviles y tiene una detectabilidad promedio. Los adversarios deben poder descifrar correctamente el código o los datos sensibles que han sido cifrados débilmente debido a algoritmos de cifrado débiles o fallos en el proceso de cifrado.

El impacto técnico de esta vulnerabilidad es grave, ya que puede resultar en la recuperación no autorizada de información sensible del dispositivo móvil. Además, puede tener diversos impactos comerciales, como violaciones de privacidad, robo de información, robo de código, robo de propiedad intelectual y daño a la reputación de la empresa.

Para prevenir la criptografía insuficiente [11], se deben seguir buenas prácticas como evitar el almacenamiento de datos sensibles en dispositivos móviles siempre que sea posible, utilizar estándares criptográficos sólidos que sean aceptados por la comunidad de seguridad y seguir las pautas de NIST para algoritmos recomendados.

También se deben evitar prácticas como la implementación de algoritmos de cifrado personalizados o el uso de algoritmos inseguros u obsoletos como RC2, MD4, MD5 y SHA1. La gestión adecuada de claves y la protección contra ataques binarios también son aspectos importantes a considerar.

M6- Autorización insegura:

Los agentes de amenaza explotan las vulnerabilidades de autorización mediante ataques automatizados para obtener acceso no autorizado a funcionalidades administrativas. Esto se realiza a través de malware móvil o botnets. La falta de esquemas de autorización adecuados permite que los adversarios ejecuten funciones privilegiadas utilizando cuentas de usuario de menor nivel de privilegios. Esto puede tener un impacto técnico severo, como la destrucción de sistemas o el acceso no autorizado a información sensible. A nivel empresarial, los impactos incluyen daños a la reputación, fraudes y robo de información.

Para prevenir la autorización insegura [12], es relevante verificar los roles y permisos de los usuarios utilizando información del sistema backend y no depender de la información del dispositivo móvil. Además, el código del backend debe verificar de forma independiente los identificadores entrantes y su correspondencia con la identidad para garantizar la autorización adecuada.

M7- Mala calidad de código:

Los agentes de amenaza en el contexto de la "mala calidad de código" [13] se refieren a entidades que pueden aprovechar las vulnerabilidades resultantes de problemas de implementación de código en aplicaciones móviles. Estas vulnerabilidades, como los desbordamientos de búfer y las fugas de memoria, son comunes pero difíciles de detectar mediante revisiones manuales. Los atacantes explotan estas vulnerabilidades al proporcionar entradas cuidadosamente diseñadas a la aplicación para ejecutar código externo o causar denegación de servicio en servidores remotos.

Los impactos técnicos de estas vulnerabilidades suelen ser moderados, pero pueden ser graves si existen desbordamientos de búfer en el dispositivo móvil y los datos de entrada son controlados por terceros. Esto podría llevar a la ejecución de código no autorizado. En términos empresariales, las consecuencias pueden variar desde el robo de información y daño reputacional hasta problemas de rendimiento y arquitectura deficiente en la interfaz.

Para prevenir problemas de mala calidad de código, es recomendable mantener patrones de codificación consistentes, escribir código legible y bien documentado, validar las longitudes de los datos de búfer y utilizar herramientas de análisis estático de terceros para identificar automáticamente vulnerabilidades. Además, se debe dar prioridad a la solución de desbordamientos de búfer y fugas de memoria sobre otros problemas de calidad de código.

M8- Manipulación de código:

El agente de amenaza se enfoca en la manipulación de código en aplicaciones móviles, aprovechando vulnerabilidades en el código de aplicaciones de terceros o engañando a los usuarios mediante phishing. Los ataques implican cambios binarios, modificación de recursos y ejecución de código externo al interceptar API del sistema.

La manipulación de código [14] es una debilidad común y su detección puede ser difícil. Existen medidas de seguridad para detectar y eliminar versiones no autorizadas de aplicaciones móviles, aunque el éxito varía entre organizaciones. Los atacantes utilizan diversas técnicas como parchear binarios, modificar recursos, enganchar métodos y modificar la memoria dinámica.

La manipulación de código puede tener graves consecuencias técnicas, como la aparición de nuevas funciones no autorizadas, robo de identidad o fraude. Desde una perspectiva empresarial, puede causar pérdida de ingresos y daño a la reputación, especialmente en aplicaciones de juegos y similares. Para prevenir la manipulación de código, las aplicaciones móviles deben diseñarse para detectar modificaciones en tiempo de ejecución y reaccionar adecuadamente.

En dispositivos Android, se puede detectar la manipulación de código verificando indicadores de un entorno comprometido, como acceso root. Esto incluye comprobar claves de prueba en build.prop, archivos apk rooteados conocidos y binarios.

M9- Ingeniería inversa:

La ingeniería inversa es el proceso de analizar el código de una aplicación para entender su funcionamiento, a menudo con fines maliciosos. Los atacantes descargan la aplicación y la analizan usando herramientas como IDA Pro y Hopper. La mayoría de las aplicaciones son susceptibles a la ingeniería inversa, especialmente aquellas escritas en lenguajes como Java, .NET, Objective C y Swift.

La ingeniería inversa puede tener impactos técnicos y empresariales significativos. Técnicamente, los atacantes pueden revelar información sobre servidores de backend, robar propiedad intelectual o realizar ataques contra sistemas de backend. Desde el punto de vista empresarial, la ingeniería inversa puede resultar en robo de propiedad intelectual, daño a la reputación de la empresa, robo de identidad o compromiso de los sistemas de backend.

Para prevenir la ingeniería inversa [15], se recomienda utilizar herramientas de ofuscación. Estas herramientas pueden ocultar partes del código, como las tablas de cadenas y los métodos, dificultando el análisis por parte de los atacantes. Sin embargo, es importante evaluar el impacto empresarial potencial de la ingeniería inversa y considerar medidas adicionales de seguridad según sea necesario.

M10- Funcionalidad extraña:

La funcionalidad extraña se refiere a características o código oculto o no intencionado en una aplicación móvil que los atacantes pueden explotar para acceder de manera no autorizada a sistemas backend o realizar acciones de alto privilegio.

Los atacantes pueden descargar la aplicación y examinar sus archivos en busca de interruptores ocultos o código de prueba expuesto por los desarrolladores. La explotabilidad de esta funcionalidad es sencilla. Detectarla puede ser difícil, pero se pueden utilizar herramientas de análisis automatizado. Las debilidades de seguridad incluyen la exposición del funcionamiento del backend y la ejecución de acciones no autorizadas de alto privilegio.

La funcionalidad extraña [16] puede tener impactos graves, como exponer información sensible del backend o permitir acciones no autorizadas de alto privilegio. Estos impactos pueden afectar el funcionamiento y la seguridad de los sistemas.

En el ámbito empresarial, la funcionalidad extraña puede conducir a accesos no autorizados a funciones sensibles, daño a la reputación de la empresa y robo de propiedad intelectual. Estos impactos pueden tener consecuencias financieras y legales significativas.

La mejor manera de prevenir esta vulnerabilidad es realizar una revisión manual del código por parte de expertos en seguridad. Esto implica examinar la configuración de la aplicación, verificar la exclusión del código de prueba en las versiones finales, revisar los puntos de acceso de la API y asegurarse de que las declaraciones de registro no revelen información sensible. La verificación manual es crucial para identificar y eliminar cualquier funcionalidad extraña antes de la distribución de la aplicación.

2.6 Modelos de seguridad

Dada las principales vulnerabilidades relacionadas con los móviles que se detallan en el apartado anterior que son recogidas por la fundación OWASP, desarrollaremos los conocidos controles y modelos de seguridad para mitigar o reducir dichas vulnerabilidades de manera específica, es decir, por cada vulnerabilidad o número del top.

M1: Uso inadecuado de la plataforma

El modelo de seguridad aplicado en esta primera vulnerabilidad es la implementación de buenas prácticas de desarrollos en aplicaciones móviles, como son seguir las directrices y las distintas políticas de seguridad en cada plataforma móvil.

Por lo que es fundamental que los desarrolladores de las aplicaciones móviles sigan las mejores prácticas posibles proporcionadas por los diferentes proveedores de

aplicaciones móviles. Dentro de estas prácticas incluimos el comprender las políticas de permiso y accesorios a funciones críticas y a cómo utilizar las distintas APIs de la plataforma de forma segura y adecuada.

Dentro de esta primera vulnerabilidad utilizaremos los siguientes controles de seguridad: Validación y autenticación de la plataforma, control de restricciones de acceso a ciertas funciones y características de cierta sensibilidad, aplicación de pruebas de seguridad y revisión de código.

Estos controles de seguridad están asociados al modelo mencionado previamente en el cual incluimos la validación y autenticación de la plataforma para asegurar que las aplicaciones móviles se ejecutan en entornos o plataformas confiables. Por lo que será clave el aplicar restricciones de acceso a ciertas funciones y características sensibles para evitar el uso indebido y también el realizar prueba de seguridad o revisar el código de las aplicaciones móviles para detectar posibles vulnerabilidades.

Figura 6 Proceso de Autenticación

M2: Almacenamiento inseguro de datos

El modelo de seguridad a aplicar en esta segunda vulnerabilidad es la utilización de métodos seguros de almacenamiento de datos en los distintos dispositivos móviles, como son los cifrados de datos en reposo y protección de la integridad de los datos almacenados. Este modelo de seguridad se centra en proteger la confidencialidad y la integridad de los datos almacenados en los dispositivos móviles. Lo que provoca que la utilización de los métodos de cifrado sólidos para proteger los datos en reposo y la implementación de mecanismos de protección de datos para detectar y prevenir modificaciones sin autorización sean claves dentro del proceso de este modelo de seguridad.

Para poder garantizar el almacenamiento seguro de los datos, ya sean sensibles o no, incluiremos los controles de cifrado de datos almacenados de manera local en dispositivos móviles, la utilización de almacenes de claves seguros y la protección contra la extracción de datos no autorizados.

Figura 7 Almacenamiento seguro de los datos

M3: Comunicación insegura

En este caso, el modelo de seguridad se enfoca en la protección de la confidencialidad, integridad y autenticidad de los datos que se transmiten entre los dispositivos móviles y los diferentes servidores u otros dispositivos. La aplicación de este modelo se realiza mediante la implementación de canales de comunicación seguros en los que se utilizan protocolos criptográficos idóneos como son los protocolos https, y los mecanismos de autenticación mutua para poder garantizar la identidad de los distintos participantes de la comunicación.

En el apartado de controles de seguridad aplicados entran las conexiones seguras como https en vez de http no cifrado. También incluiremos validaciones adecuadas de certificados de servidor para evitar los ataques de suplantación de identidad que cada vez son más comunes. Finalmente, incluimos controles de seguridad para protegernos contra los ataques intermediarios empleando cifrados de extremo a extremo.

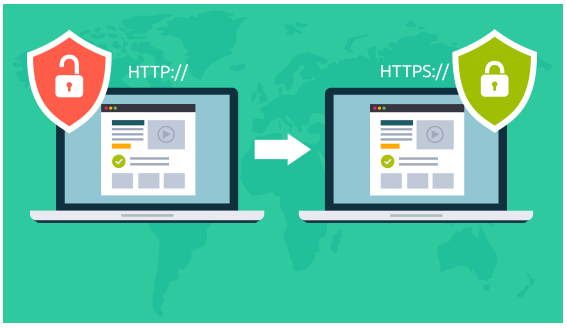


Figura 8 Conexión segura(http>https)

M4: Autenticación insegura

El modelo de seguridad para la autenticación insegura se centra en asegurar que los usuarios verifiquen su autenticidad y que las credenciales de acceso que utilizan están lo más seguras posible. Por lo que aplicaremos métodos de autenticación seguros, como son las contraseñas fuertes, autenticaciones multifactoriales y la correcta gestión de las credenciales de los usuarios.

Los controles de seguridad aplicables a esta vulnerabilidad son la implementación de políticas de contraseñas seguras, como requisitos esenciales, la longitud y complejidad, así como la autenticación multifactorial que añadirá una capa adicional de seguridad. También incluiremos la protección contra ataques de fuerza bruta con la implementación de mecanismos de bloqueo de cuentas o retraso en las respuestas a intentos de autenticación fallidos.

Figura 9 Autenticación insegura

M5: Criptografía insuficiente

El modelo de seguridad para esta vulnerabilidad es la utilización de algoritmos y protocolos criptográficos seguros para proteger la confidencialidad y la integridad de los datos en circulación y en reposo.

Los controles de seguridad asociados a la criptografía insuficiente incorporan la aplicación correcta de los algoritmos criptográficos y la utilización de tamaños de clave adecuados. También es importante llevar a cabo una organización adecuada de las claves criptográficas, incluyendo su creación segura, almacenamiento protegido y la rotación periódica para protegerlas contra ataques de criptoanálisis.

M6: Autorización insegura

El modelo de seguridad aplicado en esta sexta vulnerabilidad trata de la implementación de un sistema de autorización sólido que limite el acceso a ciertas funciones y datos, dependiendo de los roles y permisos de usuario en cada caso de manera específica.

Los controles de seguridad asociados a la autorización insegura incluyen verificar los permisos antes de permitir el acceso a funciones críticas o datos sensibles. También implementaremos mecanismos de protección para evitar la manipulación de los permisos, como son las firmas digitales de los tokens de autorización.

M7: Mala calidad de código

En esta vulnerabilidad, el modelo de seguridad se centra en asegurar que las aplicaciones móviles desarrolladas utilizan prácticas de seguridad de desarrollo de software. Lo que implica en la adopción de buenas prácticas de codificación, revisiones periódicas de código, pruebas de seguridad y certificación de la calidad de código.

En cuanto a los controles de seguridad a utilizar, tenemos la realización de comprobaciones de código estático y dinámico para identificar y corregir posibles vulnerabilidades. Además, se realizarán pruebas de penetración para detectar y solucionar problemas de seguridad y también análisis de seguridad del código fuente para identificar posibles problemas en el código de la aplicación.

M8: Manipulación de código

El modelo de seguridad para esta vulnerabilidad se focaliza en la protección de la integridad del código de la aplicación móvil y en la prevención de modificación no autorizada o manipulación con intencionalidad.

Figura 10 Manipulación de código

Los controles de seguridad asociados a la manipulación de código incluyen la firma digital del código de la aplicación, lo que nos permite verificar su integridad y autenticidad. Asimismo, implementaremos mecanismos para detectar cualquier manipulación de código, como son la detección de las firmas alteradas y la verificación de la integridad del código durante la ejecución del mismo.

M9: Ingeniería inversa

El modelo de seguridad aplicado para la ingeniería inversa tiene su punto de mira en proteger los secretos comerciales o los diferentes algoritmos confidenciales que se encuentran dentro de las aplicaciones móviles con la intención de que no se revelen o se utilicen de forma no autorizada.

Los controles de seguridad en esta vulnerabilidad incluyen la ofuscación del código para dificultar el análisis y la comprensión del funcionamiento interno de las aplicaciones. Igualmente, se utilizan para realizar técnicas de encriptación para proteger las partes críticas dentro de las aplicaciones móviles. Por último, es clave el incluir los mecanismos de detección de análisis de la aplicación para poder identificar posibles intentos de ingeniería inversa.

M10: Funcionalidad extraña

En esta última vulnerabilidad, el modelo de seguridad para la funcionalidad extraña se centra en eliminar características innecesarias aplicadas en las aplicaciones móviles que podrían aumentar la superficie de ataque y debilitar la seguridad de la aplicación.

En cuanto a los controles de seguridad asociados a la funcionalidad extraña, incluimos el análisis requerido para detectar las características innecesarias mencionadas previamente. Igualmente, revisaremos y ajustaremos los permisos y las restricciones de acceso para asegurarnos de que solo los privilegios necesarios están siendo transferidos, evitando de tal manera que la vulnerabilidad se produzca.

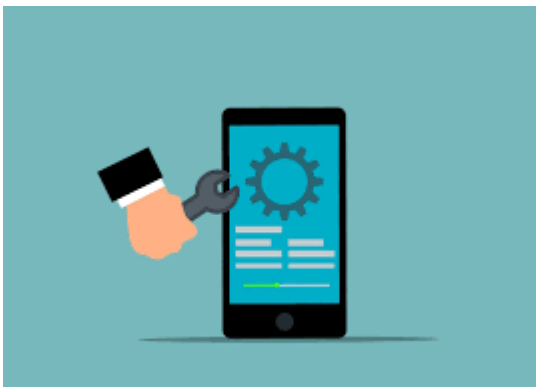


Figura 11 Funcionalidad extraña

2.7 Metodologías de seguridad para realizar auditorías de seguridad

En todas las auditorías que se realizan en la actualidad sobre teléfonos móviles y aplicaciones móviles tienen una metodología [17] a seguir, lo cual permite que tanto el proceso como el resultado final deje satisfechos tanto a los clientes como a las personas encargadas de realizar las auditorías. Al fin y al cabo, las metodologías son el marco de referencia que siguen para efectuar las ejecuciones y prácticas de manera

ordenada y profesional, con el objetivo de que se realicen todas las fases y las etapas definidas de la mejor manera posible. Entre todas esas auditorías las metodologías más comunes y a la vez conocidas son las siguientes:

OWASP Mobile Application Security Verification Standard (MASVS):

MASVS está desarrollada por OWASP y también esta metodología proporciona un conjunto de requisitos y controles de seguridad que deben efectuar las aplicaciones móviles.

Figura 12 MASVS

Esta metodología se divide en tres niveles de verificación de seguridad: Básico, Intermedio y Avanzado, dentro de estos niveles cubrimos áreas como la autenticación, el almacenamiento seguro de datos, la gestión de sesiones, la comunicación segura, entre otros.

El nivel básico se centra en controles de seguridad fundamentales, como la autenticación, la gestión de sesiones, la gestión de errores y la protección de datos.

El nivel medio aborda aspectos más avanzados, como la comunicación segura, la criptografía, las actualizaciones de software, la seguridad de la plataforma y las funciones del sistema operativo.

El nivel avanzado se enfoca en controles de seguridad más sofisticados, como la prevención de ingeniería inversa, la detección de amenazas y la respuesta a incidentes.

OWASP Mobile Security Testing Guide (MASTG):

La metodología MASTG proporciona una guía práctica paso a paso para realizar pruebas de seguridad en aplicaciones móviles. La cual cubre aspectos como la recopilación de información, el análisis de la arquitectura, las pruebas de interacciones de red, las pruebas de almacenamiento local y por último, las pruebas de interacciones de plataforma.

Figura 13 MASTG

Esta metodología se dividirá en tres diferentes fases que son las siguientes:

Fase de planificación:

En esta primera fase se definen los objetivos, el alcance y las restricciones de la auditoría.

Fase de preparación:

En esta segunda fase se recopila información sobre la aplicación móvil y su entorno.

Fase de pruebas:

En esta tercera fase se ejecutan diversas pruebas de seguridad, como pruebas de interacciones de red, pruebas de almacenamiento local, pruebas de interacciones de plataforma y algunas más.

Fase de análisis y reporte:

En esta última fase se evalúa los resultados de las pruebas y se genera un informe detallado con los hallazgos de seguridad y posibles recomendaciones.

Mobile Application Security and Penetration Testing (MASPT):

La metodología MASPT se centra en la realización de pruebas de penetración en aplicaciones móviles. También proporciona un enfoque detallado para identificar y explotar vulnerabilidades en la aplicación, como la inyección de código, la autenticación débil, las fugas de datos y las vulnerabilidades en la comunicación.

Figura 14 MASPT

Fase de recopilación de información:

En la fase de recopilación de información se obtiene información sobre la aplicación móvil y su entorno.

Fase de análisis de la arquitectura:

En la fase de análisis de la arquitectura se examina la estructura de la aplicación, incluidos los componentes cliente y servidor.

Fase de pruebas de penetración:

En la fase de pruebas de penetración se realizan pruebas de seguridad para identificar y explotar vulnerabilidades dentro de la aplicación.

Fase de informe y seguimiento:

En la fase de informe y seguimiento se documentan los hallazgos de seguridad e igualmente se proporcionan recomendaciones para mitigar los riesgos identificados.

Metodología del Mobile Application Security Assessment (MASA):

La metodología MASA se basa en las mejores prácticas y estándares reconocidos en la industria de la seguridad móvil. Dentro de la metodología se proporciona un enfoque estructurado para evaluar la seguridad de las aplicaciones móviles, cubriendo ciertas áreas como la autenticación, la autorización, la seguridad de la comunicación, la protección de datos sensibles y las vulnerabilidades del sistema operativo.

Figura 15 MASA

Fase de planificación:

La fase de planificación sirve para definir los objetivos, el alcance y los recursos necesarios para la auditoría.

Fase de recopilación de información:

En la fase de recopilación de información se obtiene información sobre la aplicación móvil y su entorno.

Fase de análisis de la aplicación:

En la fase de análisis de la aplicación se realiza un análisis estático y dinámico de la aplicación para identificar vulnerabilidades y riesgos de seguridad.

Fase de evaluación de la infraestructura:

La fase de evaluación de la infraestructura sirve para examinar la infraestructura subyacente de la aplicación, como son los servidores y servicios asociados a la aplicación móvil.

Fase de informe y recomendaciones:

La última de las fases de la metodología es la fase de informe y recomendaciones. En ella se documentan los hallazgos de seguridad y se proporcionan recomendaciones para mejorar la seguridad de la aplicación.

2.7.1 Auditorías de seguridad

Las metodologías desarrolladas en el apartado anterior para las auditorías [18] se pueden aplicar en tres tipos diferentes de auditorías. Por lo que cada metodología tendría el mismo proceso desarrollado, sin embargo, dependiendo del tipo de auditoría tendrán mayor o menor cantidad de información y de acceso sobre la infraestructura, sistema o aplicación móvil a auditar.

Existen tres tipos principales de auditorías de seguridad: auditoría de caja negra, auditoría de caja blanca y auditoría de caja gris. Estos términos describen los diferentes niveles de acceso y conocimiento que tiene el auditor sobre el sistema o red que se está evaluando.

Auditoría de caja negra:

En este tipo de auditoría, el auditor tiene un conocimiento limitado o nulo sobre la infraestructura o el sistema que se está evaluando. Se realiza desde la perspectiva de un atacante externo que no tiene información privilegiada. El objetivo es simular un escenario en el que un atacante intenta penetrar en el sistema sin ningún conocimiento interno previo. Esta forma de auditoría proporciona una evaluación realista de las medidas de seguridad externas y es útil para identificar vulnerabilidades que podrían ser aprovechadas por atacantes externos.

Auditoría de caja blanca:

En contraste con la auditoría de caja negra, en este enfoque el auditor tiene un conocimiento completo y detallado del sistema que se está evaluando. Tiene acceso a la infraestructura, la arquitectura de red, el código fuente y otra información relevante. Este tipo de auditoría permite una evaluación exhaustiva de todas las capas de seguridad y es útil para identificar problemas específicos en el diseño, la configuración o la implementación del sistema.

Auditoría de caja gris:

La auditoría de caja gris es un enfoque intermedio entre la auditoría de caja negra y la auditoría de caja blanca. En este caso, el auditor tiene un conocimiento parcial del sistema o la red que se está evaluando. Esto puede incluir cierta información básica sobre la infraestructura o credenciales de usuario limitadas. La auditoría de caja gris es útil cuando se desea realizar una evaluación más centrada en áreas específicas del sistema, manteniendo cierto grado de similitud con las condiciones de un ataque externo.

Figura 16 Tipos de auditoría

2.7.2 Tipos de análisis

En las metodologías y tipos de auditorías [19] desarrollados previamente mencionan en todo momento que en las diferentes fases realizan análisis, en algunos casos utilizan los dos tipos de análisis para aplicaciones móviles y en otros casos utilizan uno en específico. Por lo que en este apartado definiré los dos tipos de análisis que se emplean en las metodologías y auditorías sobre aplicaciones móviles. Los tipos de análisis son los siguientes:

SAST o análisis estático:

La Prueba de Seguridad Estática de Aplicaciones (SAST) es un enfoque utilizado para analizar el código fuente y encontrar vulnerabilidades de seguridad en programas. Mediante el análisis interno del código, SAST detecta problemas como inyección SQL, desbordamientos de búfer y ataques XXE, entre otros. Este enfoque permite a los desarrolladores realizar pruebas desde las etapas iniciales del desarrollo, evitando que se introduzcan fallos de seguridad en fases posteriores y mejorando la seguridad general del programa.

Existen herramientas populares para llevar a cabo la prueba de SAST, como Klocwork y Checkmarx, que son capaces de analizar diferentes lenguajes de programación. Estas herramientas ayudan a identificar vulnerabilidades y errores de seguridad en el código fuente antes de la compilación o ejecución del programa.

La incorporación de SAST en los procesos de integración y despliegue continuo es cada vez más común. Esto permite una detección temprana de problemas de seguridad, automatizando las pruebas y mejorando la conformidad con estándares de seguridad reconocidos, como OWASP Top 10 y SANS Top 25. Además, el uso de SAST puede contribuir al cumplimiento de leyes de protección de datos.

DAST o análisis dinámico:

DAST es un método para evaluar la seguridad de las aplicaciones de software mediante la simulación de acciones de un atacante malintencionado. Escanea las aplicaciones en ejecución en tiempo real y las compara con fuentes de vulnerabilidades conocidas. A diferencia de SAST, que examina el código en reposo, DAST prueba la aplicación en ejecución sin acceder al código fuente.

El análisis dinámico actúa asumiendo que el probador no conoce las funciones internas de la aplicación. Puede detectar vulnerabilidades que solo aparecen durante la ejecución, lo que lo hace valioso para probar la seguridad de las aplicaciones web contra ataques como XSS e inyección SQL.

El análisis dinámico es útil para identificar configuraciones incorrectas, problemas de autenticación y debilidades en la encriptación en servidores, bases de datos y API.

Puede evaluar todo el entorno de TI donde se encuentra la aplicación o los servicios web.

En conclusión, DAST es un método efectivo para realizar pruebas exhaustivas de seguridad en las aplicaciones, especialmente en las etapas avanzadas del desarrollo. Proporciona información sobre vulnerabilidades en tiempo de ejecución y garantiza la implementación de sólidas medidas de seguridad en todo el entorno de TI.

Figura 17 Tipos de análisis

2.8 Procedimiento habitual de un analista en las auditorías

En las auditorías como analista, el trabajo consiste en evaluar la seguridad y la calidad de las diferentes aplicaciones móviles. Por lo que un analista podría enfrentarse a un procedimiento como el siguiente, siguiendo las posteriores actividades:

1. Definir el alcance:

   El analista trabaja en colaboración con el cliente o el equipo interno para comprender los objetivos de la auditoría y establecer los límites del trabajo. Esto implica determinar qué aspectos de la aplicación móvil serán evaluados, como seguridad, rendimiento, usabilidad, entre otros.

Figura 18 Definir el alcance

2. Recopilar información:

   El analista recopila toda la información relevante sobre la aplicación móvil, como documentación técnica, requisitos del sistema, casos de uso, diagramas de arquitectura, entre otros. Esto ayuda a comprender cómo está diseñada la aplicación y cómo se espera que funcione.

Figura 19 Recopilar información

3. Análisis de riesgos:

   El analista realiza un análisis de riesgos para identificar posibles vulnerabilidades y amenazas en la aplicación móvil. Esto puede implicar la revisión de los controles de seguridad existentes, la identificación de posibles puntos débiles y la evaluación de las posibles consecuencias de una brecha de seguridad.

4. Pruebas de seguridad:

   El analista lleva a cabo pruebas de seguridad en la aplicación móvil para identificar posibles vulnerabilidades, como problemas de autenticación, autorización, criptografía débil, entre otros. Esto puede incluir pruebas de penetración, análisis de código estático y dinámico, manipulación de datos y otras técnicas.

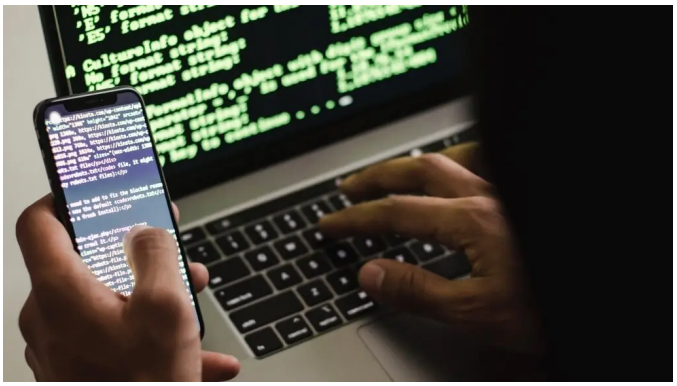


Figura 20 Pruebas de seguridad

5. Evaluación de cumplimiento:

   El analista evalúa si la aplicación móvil cumple con las regulaciones y mejores prácticas de la industria, como las directrices de privacidad y protección de datos. Esto puede implicar revisar políticas de privacidad, permisos de la aplicación, prácticas de recolección y almacenamiento de datos, entre otros aspectos.

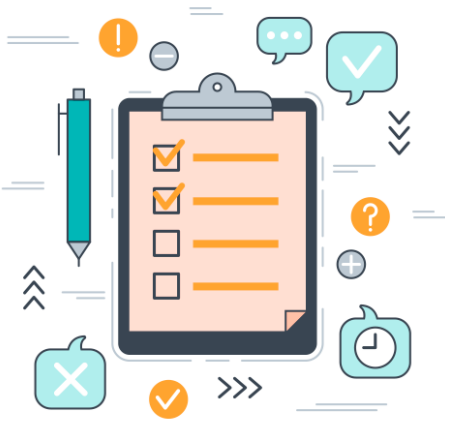


Figura 21 Evaluación de cumplimiento

6. Análisis de rendimiento:

   El analista puede realizar pruebas de rendimiento en la aplicación móvil para evaluar su capacidad de respuesta, estabilidad y escalabilidad. Esto puede implicar simular cargas de usuarios, evaluar tiempos de respuesta, analizar el consumo de recursos, entre otros aspectos.

Figura 22 Análisis de rendimiento

7. Documentación de hallazgos:

   El analista documenta todos los hallazgos y resultados de la auditoría en un informe detallado. Esto incluye las vulnerabilidades encontradas, las recomendaciones de mitigación, los aspectos cumplidos y no cumplidos, y otra información relevante.

Figura 23 Documentación de hallazgos

8. Comunicación y presentación:

   El analista se reúne con el cliente o el equipo interno para comunicar los resultados de la auditoría, discutir los hallazgos y recomendaciones, y responder a cualquier pregunta o inquietud. También puede proporcionar asesoramiento sobre medidas correctivas y prácticas recomendadas para mejorar la seguridad y el rendimiento de la aplicación móvil.

Figura 24 Comunicación y presentación

Estas actividades están definidas de manera general, en cada caso particular el analista debería adaptarse a cada aplicación móvil e incluso el analista podría añadir distintas herramientas y técnicas adicionales en caso de necesitarlas.

2.9 Herramientas y Frameworks

Para realizar diferentes auditorías de seguridad existen varias herramientas que nos facilitan los procesos. Dentro de estas herramientas existen herramientas para el análisis estático y análisis dinámico. Entre estas herramientas algunas las utilizaré en la aplicación de la metodología en los casos prácticos.

Angr

Angr [20] es un framework de Python para analizar binarios. Es útil tanto para el análisis estático como para el análisis simbólico dinámico. Dado un binario y un estado solicitado, Angr intentará llegar a ese estado utilizando métodos formales para encontrar una ruta, así como fuerza bruta. Utilizar Angr para llegar al estado solicitado suele ser mucho más rápido que realizar pasos manuales para depurar y buscar la ruta hacia el estado requerido. Angr utiliza el lenguaje intermedio VEX y cuenta con un cargador para binarios, lo que lo hace perfecto para trabajar con código nativo, como binarios nativos de Android.

Figura 25 Angr

Frida

Frida es una herramienta de instrumentación de código dinámico y de código abierto. El objetivo de Frida es inyectar el motor de JavaScript QuickJS en el proceso que se está instrumentando. Con Frida [21], puedes ejecutar fragmentos de código JavaScript en aplicaciones nativas de Android e iOS, así como en otras plataformas.

Existen varias formas de inyectar código. Por ejemplo, Xposed modifica permanentemente el cargador de aplicaciones de Android para proporcionar puntos de enganche donde se puede ejecutar código personalizado cada vez que se inicia un nuevo proceso. Por otro lado, Frida implementa la inyección de código escribiendo directamente en la memoria del proceso. Cuando se conecta a una aplicación en ejecución:

Frida utiliza ptrace para secuestrar un hilo del proceso en ejecución. Este hilo se utiliza para asignar un bloque de memoria y llenarlo con un mini-arrancador. El arrancador inicia un nuevo hilo, se conecta al servidor de depuración de Frida que se ejecuta en el dispositivo y carga una biblioteca compartida que contiene el agente de Frida. El agente establece un canal de comunicación bidireccional con la herramienta. El hilo secuestrado se reanuda después de ser restaurado a su estado original y la ejecución del proceso continúa como de costumbre.

Frida ofrece tres modos de operación:

Inyectado: Este es el escenario más común cuando frida-server se ejecuta en el dispositivo iOS o Android. Frida-core se expone a través de TCP, escuchando en localhost:27042 de forma predeterminada. Este modo no es posible en dispositivos que no tienen acceso root.

Incrustado: Este modo se utiliza cuando el dispositivo no tiene acceso root. En este caso, es responsabilidad del usuario inyectar la biblioteca frida-gadget insertándose en la aplicación, ya sea manualmente o mediante herramientas de terceros.

Precargado: Este modo permite configurar frida-gadget para que se ejecute de forma autónoma y cargue un script desde el sistema de archivos, especificando una ruta relativa a la ubicación del binario de Gadget.

Figura 26 FRIDA

MobSF

MobSF [22] (Mobile Security Framework) es un marco de prueba de penetración de aplicaciones móviles automatizado y completo, capaz de realizar análisis estáticos y dinámicos. La forma más sencilla de comenzar con MobSF es a través de Docker.

Figura 27 MobSF

Ghidra

Ghidra es un conjunto de herramientas de ingeniería inversa de software de código abierto desarrollado por la Dirección de Investigación de la Agencia de Seguridad Nacional de los Estados Unidos. Ghidra [23] es una herramienta versátil que incluye un desensamblador, un descompilador y un motor de secuencias de comandos incorporado para un uso avanzado.

Figura 28 Ghidra

Radare2

Radare2 es un marco popular de ingeniería inversa de código abierto para desensamblar, depurar, parchear y analizar archivos binarios. Radare2 [24] es un entorno scriptable que admite diversas arquitecturas y formatos de archivo, incluidas aplicaciones de Android e iOS. Para Android, se admiten Dalvik DEX, ELF y Java. Además, radare2 contiene varios scripts útiles que facilitan el análisis de aplicaciones móviles, ya que ofrece desensamblaje de bajo nivel y análisis estático seguro, especialmente cuando las herramientas tradicionales no son suficientes.

Figura 29 Radare2

3 Caso Práctico

En este capítulo se va a realizar el análisis de seguridad a una aplicación móvil y para ello utilizaré la metodología diseñada.

3.1 Metodología

En este primer apartado se definirá la metodología desarrollada para realizar el análisis de seguridad sobre la aplicación móvil del sistema operativo Android.

El objetivo principal de la metodología desarrollada será completar un análisis de seguridad lo más completo posible para identificar y mitigar posibles riesgos y vulnerabilidades en la aplicación móvil. Proceso el cual obtendrá unos resultados y conclusiones que certificarán si la metodología desarrollada ha sido un éxito o no. La metodología será aplicada a una aplicación, sin embargo, la finalidad de la metodología desarrollada es que sirva para cualquier tipo de aplicación móvil y de cualquier sistema operativo.

Figura 30 Metodología

La metodología desarrollada está completada por varios procedimientos a seguir hasta conseguir los resultados o conclusiones mencionados previamente. La metodología desarrollada se podrá utilizar tanto en aplicaciones de iOS como en aplicaciones Android.

El primero de los procedimientos de la metodología estará relacionado con la recopilación de datos e información. En este primer procedimiento recopilaremos la información de la aplicación móvil, dentro de esta información se encuentran datos como sobre qué trata la aplicación, los diferentes apartados o funcionalidades que tiene la aplicación y por último, los diferentes datos que manejan la aplicación.

El segundo de los procedimientos de la metodología tendrá relación directa con los datos obtenidos de la aplicación, ya que en este segundo procedimiento analizaremos los diferentes activos y riesgos que tiene la aplicación. Por lo que determinaremos las

diferentes funcionalidades claves que tiene la aplicación y determinaremos cuán sensibles o críticos son los datos recopilados.

En el tercero de los procedimientos de la metodología realizaremos un análisis estático y otro dinámico a la aplicación. Con estos análisis identificamos las posibles vulnerabilidades, en las cuales están incluidas el estudio del código fuente, las dependencias, configuraciones y APIs utilizadas en la aplicación móvil.

En el cuarto de los procedimientos ejecutaremos un análisis de los resultados obtenidos tras los análisis estáticos y dinámicos. Valorando las diferentes amenazas y vulnerabilidades encontradas y certificando las similitudes o conexiones con los riesgos identificados previamente a la ejecución de los dos tipos de análisis.

En el quinto de los procedimientos de la metodología efectuaremos un informe o documentación en el cual detallaremos las diferentes conclusiones que hemos encontrado en la aplicación, los posibles impactos de las vulnerabilidades identificadas y por último las recomendaciones o contramedidas para corregirlas. También incluiremos una documentación de cómo ha sido todo el proceso de la aplicación de la metodología desarrollada en la aplicación móvil a analizar.

3.1.1 Aplicación elegida

La aplicación vulnerable escogida para la aplicación de la metodología es la siguiente:

Damn Vulnerable Bank [25] es una aplicación diseñada para ser intencionadamente vulnerable. La aplicación brinda una interfaz para someterse a las diferentes pruebas y exámenes de seguridad sobre las aplicaciones de Android. La instalación completa de la aplicación se describe en el *Anexo A Instalación de la aplicación elegida.*

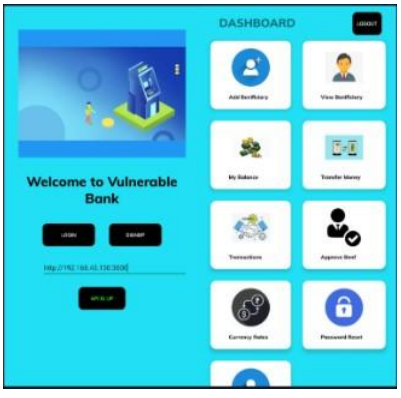


Figura 31 Aplicación vulnerable(Damn Vulnerable Bank)

Repositorio de la aplicación:
https://github.com/rewanthtammana/Damn-Vulnerable-Bank

3.1.2 Recopilación de datos e información (Primer procedimiento)

En este primer procedimiento recopilaremos toda la información relacionada con la aplicación móvil. La recopilación de esta información será clave para comprender al completo la aplicación y sobre la base de esa recopilación realizar el posterior análisis.

El primero de los apartados de este procedimiento será identificar la temática y sobre qué trata la aplicación móvil. En este caso, la aplicación móvil trata sobre un clon cercano a las aplicaciones bancarias que existen en el mundo real. Por lo que la temática tratará sobre el mundo de los bancos, es decir, la banca y las diferentes funcionalidades a las que pueden acceder los usuarios mediante la utilización de los bancos.

El segundo de los apartados de este procedimiento trata de analizar los diferentes apartados y funcionalidades de la aplicación. Las funcionalidades y apartados de la aplicación seleccionada son:

1. Registro de usuarios: Los usuarios tienen la opción de crear una cuenta dentro de la aplicación, en la cual proporcionarán información personal como puede ser el nombre, dirección, número de teléfono y demás.

2. Login: Los usuarios tienen la funcionalidad de iniciar sesión con la cuenta generada utilizando el usuario y la contraseña. Esta funcionalidad puede ser vulnerable a ataques de fuerza bruta, ataques de diccionario o inyección de código.

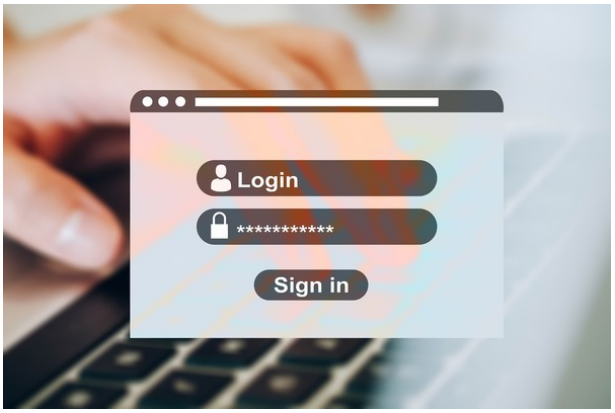


Figura 32 Login

3. Administración de cuentas: Los usuarios tienen completo acceso para administrar su información vital o sensible, en la que incluimos el número de cuenta, saldo, historial de transacciones, íntegros y algunas más.

4. Transferencia de fondos: En esta funcionalidad los usuarios tienen la capacidad de realizar transferencias a otras cuentas bancarias. Dentro de esta

funcionalidad existen vulnerabilidades relacionadas con la autenticación y la autorización.

5. Pagos de facturas: En este apartado, la aplicación permite a los usuarios realizar pagos de facturas empleando fondos de su cuenta bancaria.

6. Administración de perfiles de usuario: La aplicación permite que los usuarios administren sus cuentas a su gusto, por lo que pueden editar datos personales como la dirección de correo electrónico, número de teléfono u otros.

El tercero de los apartados del procedimiento es la exploración de los datos manejados por la aplicación. Los datos manejados por la aplicación son los siguientes:

- Información del usuario:

  En estos datos incluimos los nombres, direcciones, números de teléfono, direcciones de correo electrónico, número de identificación personal(NIP) y contraseñas. Los datos mencionados son críticos y deben estar debidamente protegidos para evitar accesos no autorizados.

- Datos financieros:

  La aplicación maneja información relacionada con las diferentes cuentas bancarias del usuario, como pueden ser números de cuenta, saldos, historial de transacciones y fragmentos de las tarjetas de crédito o débito. Estos datos son sensibles y deben estar protegidos para evitar los posibles fraudes o accesos no autorizados.

- Datos generados por el usuario:

  Los datos utilizados o generados por el usuario, como pueden ser mensajes, comentarios, u otros contenidos que han sido generados por los usuarios. Los datos creados o generados deben ser analizados para corroborar que no inyecten código malicioso, malware o comprometan la seguridad de la aplicación.

- Datos relacionados con transacciones:

  La información o datos relacionados con las transacciones son las compras o pagos, datos de facturación y datos de los productos adquiridos.

Figura 33 Datos de la aplicación

3.1.3 Análisis de los diferentes activos y riesgos (Segundo procedimiento)

Dentro de este segundo procedimiento realizaremos una evaluación específica de las funcionalidades importantes de la aplicación y determinaremos la sensibilidad y cuanto de críticos son los datos recopilados. Con este proceso conseguimos identificar los activos de información más valiosos y que riesgos están vinculados con ellos.

Identificación de las funcionalidades clave:

Las funcionalidades clave dentro de la aplicación vulnerable son las funciones más importantes dentro de la aplicación y por lo cual también las convierte en el objetivo número uno de los atacantes debido a que son las funcionalidades más utilizadas por los usuarios y en las que más datos críticos se utilizan.

Dentro de la aplicación las funcionalidades clave están:

El registro de usuario, el inicio de sesión, la consulta de datos, transferencia de fondos, pago de facturas, historial de transacciones y la administración de perfiles de usuario.

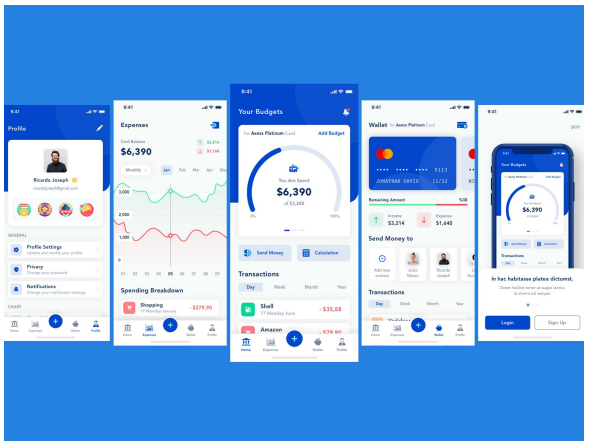


Figura 34 Funcionalidades clave

En el anterior apartado hemos desarrollado las diferentes funcionalidades dentro de la aplicación. En este caso hablaremos sobre todas las actividades que podemos encontrar dentro de la aplicación que estamos analizando con el uso de la metodología.

Las actividades utilizadas en esta aplicación son:

- **La primera actividad *com.app.damnvulnerablebank.Myprofile:***

  Esta primera actividad trata de la representación del perfil del usuario que esté utilizando la aplicación.

- **La segunda actividad *com.app.damnvulnerablebank.CurrencyRates:***

  Esta segunda actividad está relacionada con el manejo de tasas de cambio de divisas dentro de la aplicación vulnerable(damn vulnerable bank)

- **La tercera actividad *com.app.damnvulnerablebank.ResetPassword:***

  En esta tercera actividad está el proceso de restablecimiento de contraseñas dentro de la aplicación.

- **La cuarta actividad *com.app.damnvulnerablebank.ViewBeneficiary:***

  Esta actividad permite a los usuarios ver los beneficiarios asociados a su cuenta dentro de la aplicación.

- **La quinta actividad *com.app.damn vulnerable bank.Approved Beneficiary:***

  En esta actividad aparece la aprobación de los beneficiarios en la aplicación.

- **La sexta actividad *com.app.damnvulnerablebank.PendingBeneficiary:***

  La sexta actividad es la que se encarga de los beneficiarios pendientes de aprobación.

- **La séptima actividad *com.app.damnvulnerablebank.AddBeneficiary:***

  En esta actividad se les permite a los usuarios agregar a nuevos beneficiarios.

- **La octava actividad *com.app.damnvulnerablebank.SendMoney:***

  Esta octava actividad tiene relación con el envío de dinero a otros beneficiarios con el uso de la aplicación.

- **La novena actividad *com.app.damnvulnerablebank.ViewBeneficiaryAdmin:***

  Esta actividad tiene relación con observar los beneficios desde una perspectiva administrativa.

- **La décima actividad *com.app.damnvulnerablebank.GetTransactions:***

  La décima actividad permite a los usuarios obtener información sobre las diferentes transacciones realizadas con la utilización de la aplicación.

- **La undécima actividad *com.app.damnvulnerablebank.ViewBalance:***

  La undécima actividad se utiliza para ver el saldo de la cuenta que tienes dentro de la aplicación.

- **La duodécima actividad *com.app.damnvulnerablebank.Dashboard:***

  Esta actividad es el panel principal de la aplicación.

- **La decimotercera actividad *com.app.damnvulnerablebank.RegisterBank:***

  Esta actividad está relacionada con el registro dentro de la aplicación.

- **La decimocuarta actividad *com.app.damnvulnerablebank.BankLogin:***

  Esta actividad tiene relación con el inicio de sesión en la aplicación móvil.

- **La decimoquinta actividad *com.app.damnvulnerablebank.MainActivity:***

  La decimoquinta actividad representa la actividad principal o pantalla principal de la aplicación.

- **La decimosexta actividad *com.app.damnvulnerablebank.SplashScreen:***

  La decimo sexta actividad es la pantalla de inicio o la representación de la aplicación

- **La decimoséptima actividad *androidx.biometric.DeviceCredentialHandlerActivity:***

  Esta actividad está relacionada con el manejo de la autenticación biométrica.

- **La decimoctava actividad *com.google.firebase.auth.internal.FederatedSignInActivity:***

  Esta penúltima actividad tiene relación con el inicio de sesión federado utilizando Firebase Authentication.

- **La decimonovena actividad *com.google.android.gms.common.api.GoogleApiActivity:***

  Esta última actividad está relacionada con el Google Play API services de Android.

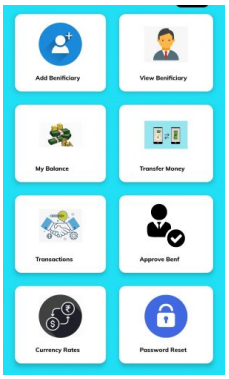


Figura 35 Actividades

Identificación de riesgos:

La identificación de riesgos dentro de la aplicación es el proceso de análisis que tiene como finalidad identificar las posibles amenazas y vulnerabilidades dentro de la aplicación que podrían afectar la seguridad, integridad o disponibilidad de los datos que maneja. Este apartado es clave debido a que permite comprender los posibles peligros a los que se enfrenta la aplicación móvil y tomar medidas para mitigarlos.

Las posibles amenazas [26] o vulnerabilidades encontradas dentro de la aplicación analizada son las siguientes:

En relación con las amenazas:

Desciframiento de datos:

Tras el análisis realizado a la aplicación vulnerable, queda reflejado que existe una amenaza de desciframiento de datos en la que los atacantes pueden ver los datos descifrados de la solicitud, la respuesta de las solicitudes e incluso obtener la capacidad de manipularlos.

Detección del GPU y de root:

Tras identificar que la aplicación tiene posibilidades de sufrir detección del GPU y de root, los riesgos posibles a estas detecciones son los siguientes:

Exploit de escalada de privilegios:

Un exploit de escalada de privilegios es un ataque en el que un usuario malintencionado busca obtener acceso de superusuario o root en un sistema. Si se descubre una vulnerabilidad en el sistema operativo, un atacante podría aprovecharla para elevar sus privilegios y obtener acceso a los permisos de root.

Ataques de fuerza bruta o diccionario:

Los ataques de fuerza bruta o diccionario implican intentar adivinar contraseñas o claves de acceso al sistema mediante la prueba sistemática de todas las combinaciones posibles. Si tienes una contraseña débil o fácil de adivinar, un atacante podría utilizar este tipo de ataque para obtener acceso a tu cuenta de root y, potencialmente, a tu GPU.

Inyección de comandos:

La inyección de comandos es una vulnerabilidad común en la que un atacante puede insertar comandos maliciosos dentro de una entrada de datos que se ejecuta en el sistema. Si un programa o script no está debidamente asegurado y no filtra correctamente la entrada de datos, un atacante podría explotar esta vulnerabilidad para ejecutar comandos no autorizados, incluidos aquellos relacionados con el control de la GPU y los permisos de root.

Vulnerabilidades en controladores de GPU:

Los controladores de la GPU son piezas de software complejas y pueden contener vulnerabilidades que podrían ser aprovechadas por atacantes para obtener acceso no autorizado o interrumpir el funcionamiento de la GPU. Estas vulnerabilidades pueden permitir ataques como la ejecución remota de código, la denegación de servicio o el robo de información.

Ataques de intermediario (Man-in-the-Middle):

Un ataque de intermediario implica que un atacante se sitúa entre tu sistema y el servidor al que te estás conectando, interceptando y modificando la comunicación en tiempo real. Si la comunicación entre tu sistema y el servidor no está debidamente cifrada o autenticada, un atacante podría manipular las respuestas del servidor y falsificar la detección de la GPU o los permisos de root.

En relación con las vulnerabilidades:

Vulnerabilidad de la API REST:

Esta vulnerabilidad se refiere a una brecha de seguridad dentro la interfaz API REST. Esta vulnerabilidad permite a los atacantes realizar acciones no autorizadas e incluso acceder a datos sensibles o alterar el funcionamiento de la aplicación a través de la API.

Divulgación de información confidencial:

Esta vulnerabilidad tiene relación con la posibilidad de que información relevante o sensible como pueden ser datos personales, se revelen o se expongan de forma no intencionada. Esto podría suceder debido a las vulnerabilidades encontradas dentro de esta aplicación, como son el acceso no autorizado o errores de implementación.

Actividades exportadas:

Esta vulnerabilidad se refiere a que hay actividades dentro de la aplicación analizada que están declaradas como públicas, ya que existe el acceso desde otras aplicaciones a la aplicación. Esto ocurre debido a que existen actividades que no requieren autenticación o mediante el uso de ellas se pueden realizar acciones maliciosas.

Explotación webview a través de deeplink:

La vulnerabilidad de webview a través de deeplink permite la ejecución de código JavaScript malicioso en el WebView de la aplicación Android mediante el uso de deep links, lo que provoca que puedan existir ataques de cross-site scripting. Por lo que es importante abordar esta vulnerabilidad y corregirla para evitar potenciales ataques a la aplicación móvil.

Figura 36 Riesgos en aplicaciones móviles

3.1.4 Análisis estático y dinámico de la aplicación (Tercer procedimiento)

Realizaremos un análisis estático y dinámico sobre la aplicación vulnerable Damn Vulnerable Bank. El análisis estático y el análisis dinámico los realizaremos con la herramienta MobSF [27].

3.1.4.1 Análisis estático

3.1.4.1.1 Análisis estático con MobSF

El análisis estático lo realizaremos con la herramienta MobSF, descrita en el apartado de Herramientas y Frameworks de la memoria. Con esta herramienta obtendremos un análisis de la aplicación en unos pasos. Una vez tenemos instalada la herramienta de MobSF subiremos o arrastraremos la aplicación al navegador de MobSF. Una vez la herramienta obtiene la apk el análisis estático comienza sobre la aplicación vulnerable "Damn Vulnerable Bank". Los requisitos, instalación, preparación del entorno e iniciación y funcionamiento de la herramienta MobSF se describen en el Anexo B Instalación de MobSF(Análisis estático).

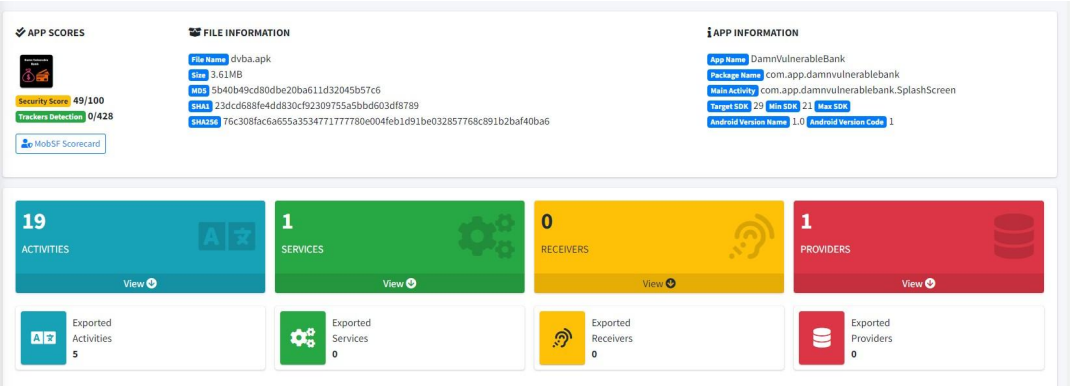


Figura 37 Resumen del análisis estático con MobSF

3.1.4.1.2 Detalles del análisis estático

Tras realizar el análisis estático sobre la aplicación vulnerable "Damn Vulnerable Bank", los detalles e informes relevantes obtenidos son los siguientes:

Información del archivo:

- Nombre del fichero: dvba.apk
- Tamaño de la app: 3.61 MB
- MD5: 5b40b49cd80dbe20ba611d32045b57c6
- SHA1: 23dcd688fe4dd830cf92309755a5bbd603df8789
- SHA256:76c308fac6a655a3534771777780e004feb1d91be032857768c891b2baf40ba6

Información de la App:

- ➔ Nombre de la App: DamnVulnerableBank
- ➔ Nombre del paquete: com.app.damnvulnerablebank
- ➔ Actividad principal: com.app.damnvulnerablebank.SplashScreen
- ➔ Tarjeta SDK: 29 Min
- ➔ SDK: 21
- ➔ Nombre versión de Android: 1.0
- ➔ Código de la versión de Android: 1

La aplicación contiene 19 actividades de las cuales 5 están exportadas, 1 servicio y 1 proveedor.

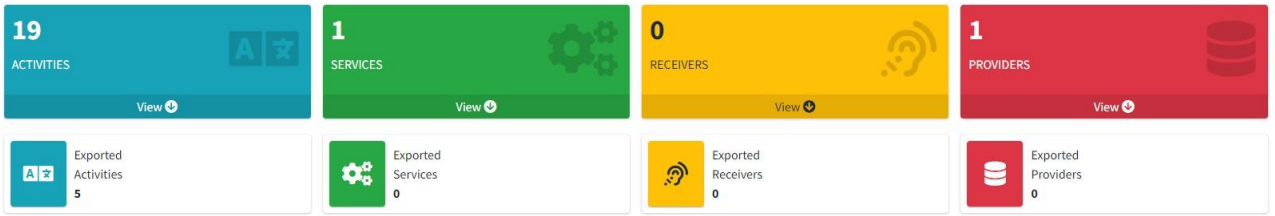


Figura 38 Actividades, servicios, receptores y proveedores

Permisos de la App:

Los permisos detectados tras el análisis estático realizado son los siguientes:

Permiso	Estado	Info	Descripción
android.permission.INTERNET	Normal	Acceso completo a internet.	Permite que una aplicación cree conexiones de red.
android.permission.USE_BIOMETRIC	Normal	Utilización de modalidades biométricas.	Permite que una aplicación utilice modalidades biométricas admitidas por el dispositivo utilizado.
android.permission.USE_FINGERPRINT	Normal	Permite el uso de la huella digital.	La constante empleada está obsoleta a partir del nivel de API 28. Las aplicaciones deben solicitar USE_BIOMETRIC en su lugar.

Tabla I Permisos de la app

Seguridad de la red:

La seguridad de la red detectada tras el análisis estático es el posterior:

n.º	Severidad	Descripción
1	Alta	La configuración base está configurada de manera insegura para permitir tráfico en texto claro a todos los dominios.
2	Alta	La configuración base está configurada para confiar en los certificados instalados por el usuario.
3	Media	La configuración base está configurada para confiar en certificados del sistema.

Tabla II Seguridad de la red

Análisis del archivo manifest:

Se detectan las posteriores vulnerabilidades en el archivo manifest:

n.º	Vulnerabilidad	Severidad	Descripción
1	La aplicación se puede instalar en una versión vulnerable de Android. **[minSdk=21].**	Media	La aplicación se puede instalar en una versión antigua de Android, la cual tiene múltiples vulnerabilidades sin corregir. Se recomienda soportar una versión de Android > 8, API 26 para recibir actualizaciones de seguridad adecuadas.
2	El tráfico en texto claro está habilitado para la aplicación. **[android:usesCleartextTraffic=true].**	Alta	La aplicación tiene la intención de utilizar tráfico de red en texto claro, como HTTP en texto claro, FTP, DownloadManager y MediaPlayer. El valor predeterminado para las aplicaciones que apuntan a API nivel 27 o inferior es el de “true”. Las aplicaciones que apuntan a API nivel 28 o superior tienen como valor predeterminado “false”. La razón principal para evitar el tráfico en texto claro es la falta de confidencialidad, autenticidad y protección contra la manipulación. Con esta vulnerabilidad, un atacante de red puede interceptar los datos transmitidos y también modificarlos sin ser detectado.

3	La aplicación tiene una configuración de seguridad de red llamada. **[android:networkSecurityConfig=@xml/network_security_config]**	Info	La función de Configuración de Seguridad de Red permite que las aplicaciones personalicen su configuración de seguridad de red en un archivo de configuración declarativo seguro sin modificar el código de la aplicación. Estos ajustes se pueden configurar para dominios específicos y para una aplicación específica.
4	Los datos de la aplicación se pueden respaldar. **[android:allowBackup=true]**	Media	Esta bandera permite que cualquiera respalde los datos de la aplicación a través de adb. También permite a los usuarios que han habilitado la depuración USB copiar los datos de la aplicación desde el dispositivo.
5	La actividad (com.app.damnvulnerablebank.CurrencyRates) no está protegida. Existe un intent-filter.	Media	Una actividad está en estado de compartición con otras aplicaciones en el dispositivo, lo que la deja accesible a cualquier otra aplicación en el dispositivo. La presencia de un intent-filter indica que la actividad se exporta explícitamente.
6	La actividad (com.google.firebase.auth.internal.FederatedSignInActivity) está protegida por un permiso, pero se debe verificar el nivel de protección del permiso. Permiso: com.google.firebase.auth.api.gms.permission.LAUNCH_FEDERATED_SIGN_IN **[android:exported=true]**	Media	Una actividad está compartida con otras aplicaciones en el dispositivo, lo que la deja accesible a cualquier otra aplicación en el dispositivo. La actividad está protegida por un permiso que no está definido en la aplicación analizada. Como resultado, comprobaremos el nivel de protección del permiso donde esté definido. Si el resultado se establece en normal o peligroso, una aplicación maliciosa puede solicitar y obtener el permiso e interactuar con el componente. Si se establece en firma, solo las aplicaciones firmadas con el mismo certificado pueden obtener el permiso.

Tabla III Análisis del archivo Manifest

Análisis de código:
Los resultados del análisis de código como parte fundamental del análisis estático:

n.º	Problemas	Severidad	Normas
1	La aplicación registra información. La información sensible nunca debe ser registrada.	Info	**CWE:** CWE-532: Inserción de información sensible en archivo de registro. **OWASP MASVS:** MSTG-STORAGE-3.
2	Esta aplicación puede tener capacidades de detección de root.	Segura	**OWASP MASVS:** MSTG-RESILIENCE-1
3	La aplicación puede leer o escribir en el almacenamiento externo. Cualquier aplicación puede leer los datos escritos en el almacenamiento externo.	Media	**CWE: CWE-276:** Permisos predeterminados incorrectos. **OWASP Top 10**: M2: Almacenamiento inseguro de datos. **OWASP MASVS:** MSTG-STORAGE-2.

Tabla IV Análisis de código

Resumen y puntuación de la app:

La puntuación de la aplicación en el apartado de seguridad es 49/100. La calificación de la app es B (Riesgo Medio).

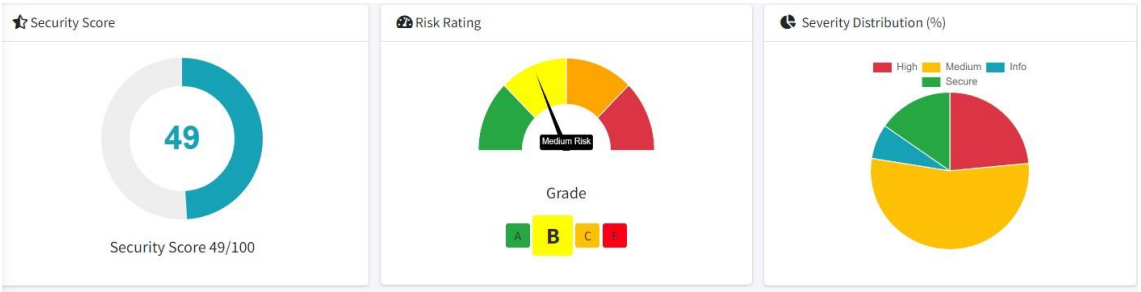


Figura 39 Gráficas de seguridad, riesgo y distribución de seguridad

Los hallazgos en la aplicación según su gravedad son los siguientes:

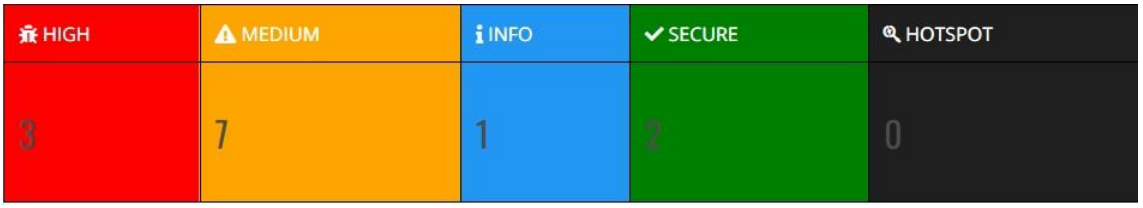


Figura 40 Hallazgos dentro de la app y con su gravedad

La aplicación está firmada con un certificado de firma de código.

3.1.4.2 Análisis dinámico

3.1.4.2.1 Análisis dinámico con MobSF

El análisis dinámico lo realizaremos con la herramienta MobSF. El análisis se efectuará al mismo tiempo que se ejecuta código, es decir, al ser un análisis dinámico, el análisis se realiza en ejecución para encontrar nuevas vulnerabilidades, amenazas o riesgos que no encontramos con el código parado o sin estar en ejecución. También nos permite realizar ataques concretos para probar las diferentes vulnerabilidades de la aplicación y observar cómo responde la aplicación móvil. Los requisitos, instalación, preparación del entorno e iniciación y funcionamiento de la herramienta MobSF se describen en el Anexo C Instalación de MobSF(Análisis dinámico).

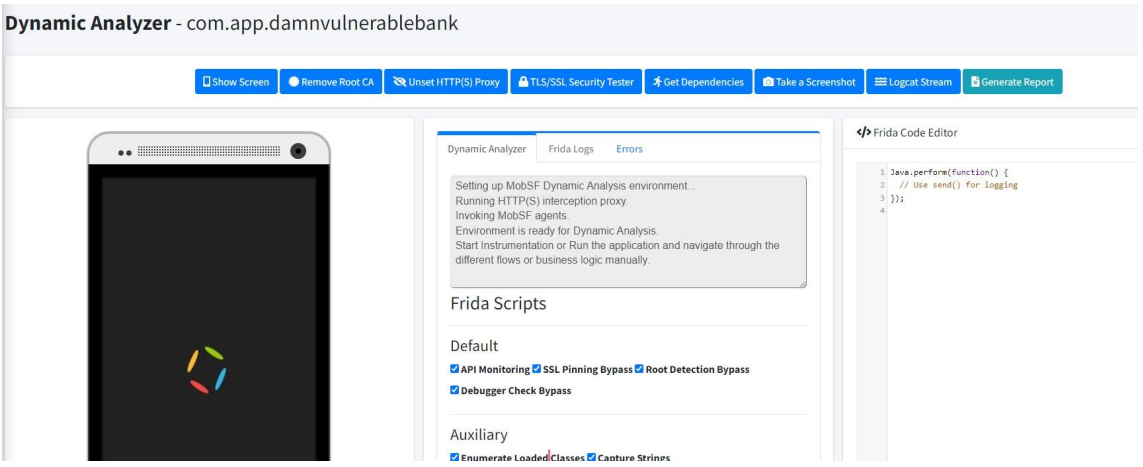


Figura 41 Análisis dinámico de la aplicación vulnerable

3.1.4.2.2 Detalles del análisis dinámico

Tras realizar el análisis dinámico sobre la aplicación vulnerable “Damn Vulnerable Bank”, los detalles e informes relevantes obtenidos son los siguientes:

Los scripts de Frida utilizados o disponibles para realizar el análisis dinámico son los siguientes:

aes_key:

Este primer script nos permite interceptar y registrar la clave AES utilizada en las diferentes operaciones de cifrado dentro de la aplicación analizada.

bypass_flag_secure:

Con este segundo script modificamos la configuración de “Flag_secure” con la idea de conseguir los permisos para poder capturar la pantalla y grabar la aplicación.

bypass_method:

Este tercer script sirve para modificar los diferentes comportamientos de los métodos de seguridad para evitar en ciertos aspectos su funcionamiento o para simplemente evadir su protección.

default:

El cuarto script “default” se emplea para la verificación de que los scripts de Frida están en funcionamiento. La aplicación no tiene impacto en la aplicación analizada, nos sirve para corroborar la utilización de los scripts de Frida.

file_trace:

Con este script interceptamos y registramos las llamadas tanto a las funciones de lectura y de escritura de archivos.

fingeprint_bypass:

Este sexto script se usa para modificar o evitar las técnicas de fingerprinting.

fingeprint_bypass_via_exception_handling:

En este caso, el script sirve para poder manejar las excepciones de detección de fingerprinting.

get_Android_id:

Con este script obtendremos el Android ID de la aplicación analizada.

helper:

Este script se utiliza como ayuda, ya que lo utilizaremos para que nos proporcione funciones y utilidades comunes en otros scripts de Frida.

hook_constructor:

El décimo script empleado o disponible sirve para interceptar o registrar las distintas llamadas efectuadas a los constructores de java.

hook_java_reflection:

Este script está relacionado con el anterior y vale para interceptar y registrar las llamadas realizadas a los métodos de reflexión de Java.

inputstream_dump:

Este script tiene como funcionalidad interceptar y registrar las distintas operaciones ejecutadas en relación con la lectura de datos desde un “InputStream”.

intent_dumper:

Con este script interceptamos y registramos la cantidad de intentos que se envían y se reciben dentro de la aplicación.

jni_hook_by_address:

Este script permite el enganche a funciones JNI mediante el uso de la dirección en la aplicación analizada.

jni_trace:

Este decimoquinto script sirve para interceptar las funciones JNI mencionadas en el anterior script.

keyguard_credential_intent:

Este script intercepta y registra los “Intents” relacionados con la entrada de credenciales de desbloqueo del dispositivo.

tracer_cipher:

Con este script interceptamos y registramos las llamadas a funciones relacionadas con la utilización de cifrados.

tracer_keygenparameterspec:

Este script intercepta y registra las llamadas y configuraciones que se encuentran dentro de la clase “KeyGenParameterSpec”.

tracer_keystore:

En este script interceptamos y registramos las llamadas y operaciones, sin embargo, esta vez de la clase “KeyStore”.

tracer_secretkeyfactory:

En este penúltimo script registramos e interceptamos las llamadas y operaciones de la clase “Secretkeyfactory”.

webview_enable_debugging:

Este último script modifica la configuración del WebView para poder habilitar el modo de depuración y permitir inspeccionar el contenido y el código de JavaScript dentro del WebView.

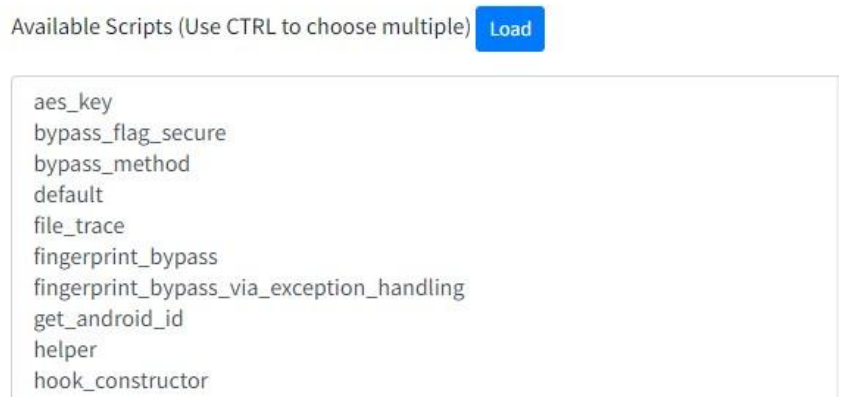


Figura 42 Frida scripts I

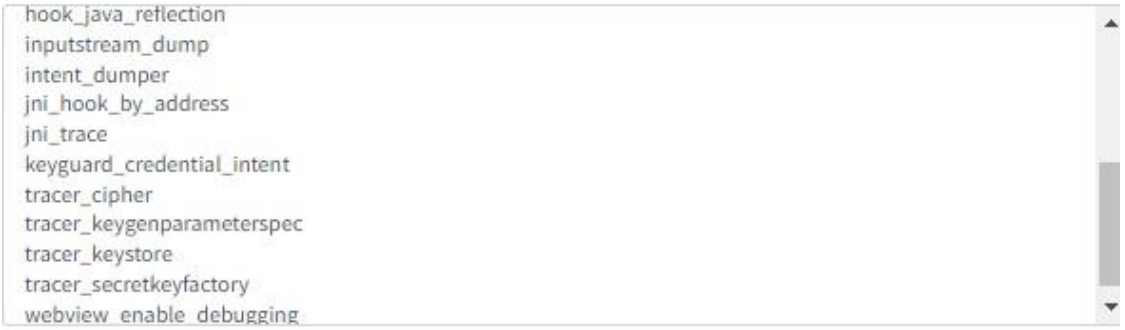


Figura 43 Frida scripts II

Funciones o procesos de análisis:

Show screen:

Con esta función se muestra la captura de pantalla de la aplicación a la cual realizamos el análisis, en la cual podemos visualizar la interfaz de usuarios de la aplicación y verificar su apariencia.

Remove Root CA:

Esta función elimina la autoridad de certificación raíz, una vez ya instalada en la aplicación vulnerable. En cada análisis dinámico, MobSF instala su propia “root CA” para poder realizar, por ejemplo, inspecciones de tráfico TLS/SSL.

Unset Http(s) Proxy:

En este caso la función nos permite desactivar el proxy http configurado dentro del dispositivo. Como en la anterior función, este proceso sirve para desactivar una configuración previa hecha por MobSF. En esta situación, MobSF configura un proxy http para interceptar y analizar el tráfico de red de la aplicación.

TLS/SSL Security Tester:

Esta función nos permite analizar la configuración de seguridad TLS/SSL de la aplicación a analizar.

Get Dependencies:

Con este proceso obtenemos la información sobre las dependencias y bibliotecas utilizadas dentro de la aplicación.

Take a Screenshot:

Esta función sirve para realizar capturas de pantalla durante el proceso del análisis dinámico. Lo que significa que la aplicación es vulnerable en el aspecto de que tiene permitido el poder hacer capturas de pantalla. Proceso que no es habitual en aplicaciones de banca como es el caso.

Logcat Stream:

Este apartado nos permite visualizar el flujo de registros de la aplicación en tiempo real, durante el transcurso del análisis.

Generate Report:

Esta función nos permite generar un informe completo del análisis realizado, en el cual se incluyen todos los procesos y resultados obtenidos.

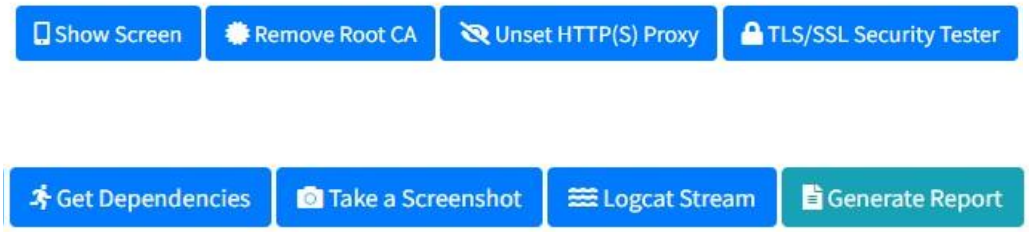


Figura 44 Funciones o procesos de análisis utilizados dentro de MobSF

TLS/SSL Pruebas de seguridad

Prueba de tráfico de texto sin cifrar (Cleartext Traffic Test):

Realizaremos un test para corroborar si está habilitado el tráfico de textos sin cifrado.

Prueba de configuración incorrecta de TLS (TLS Misconfiguration Test):

Habilitaremos el proxy, HTTPS MITM y eliminaremos la CA Root. La ejecución de la aplicación dura 25 segundos.

Con esta prueba descubriremos si las configuraciones inseguras acceden a que las conexiones HTTPS realicen un “bypass” sobre los errores de certificado o los errores de SSL/TLS en WebViews. Lo mismo que significa no tener TLS.

Prueba de transparencia de certificado/fijación TLS (TLS Pinning/Certificate Transparency Test):

Habilitaremos el proxy HTTPS MITM e instalaremos CA Root. La ejecución de la aplicación dura 25 segundos. La prueba evaluará los controles de refuerzo de TLS/SSL de la aplicación y comprobaremos si la aplicación implementa el certificado o la fijación de clave pública o la transparencia del certificado.

Prueba de omisión de transparencia de certificado/fijación TLS (TLS Pinning/Certificate Transparency Bypass Test):

Como en la anterior prueba, habilitaremos el proxy HTTPS MITM e instalaremos CA Root. La diferencia con la anterior prueba es que en este caso realizaremos un bypass de clave pública/certificado o transparencia de certificado. La prueba intenta un “bypass” sobre los certificados o claves públicas o los controles de transparencia de certificados de la aplicación.

TESTS	RESULT
Cleartext Traffic Test	✓
TLS Misconfiguration Test	✓
TLS Pinning/Certificate Transparency Bypass Test	✓
TLS Pinning/Certificate Transparency Test	✓

Figura 45 TLS/SSL Pruebas de seguridad

Dex class loader

En este apartado, se llevó a cabo una búsqueda una clase en un archivo DEX, el cual es un formato de archivo utilizado en el desarrollo de las aplicaciones de Android. También se realizó una búsqueda de autenticación de Firebase y una biblioteca de verificación de Frida.

Los resultados de este apartado del análisis son:

Clase	Método
dalvik.system.BaseDexClassLoader	En este primer caso ejecutaremos una búsqueda de recursos “findResource”. El recurso específico que estaba en búsqueda era el archivo ‘fire-auth.properties’. El resultado de la búsqueda es “Result: jar:file:/data/app/com.app.damnvulnerablebank2CNNLm7-yrgyHi4ax0mjlA==/base.apk!/firebaseauth.properties”. El archivo encontrado es el de “firebase-auth.properties”, el cual está ubicado dentro del archivo APK de la aplicación “com.app.damnvulnerablebank”. El punto de llamada desde donde se realizó la búsqueda del recurso: Called From: java.lang.ClassLoader.getResource(ClassLoader.java:7 93)

dalvik.system.DexClassLoader	Iniciamos la instancia de la clase “DexClassLoder”-> $init. Los argumentos inicializados son “['/data/data/com.app.damnvulnerablebank/frida4567052612148929880.dex', '/data/data/com.app.damnvulnerablebank/cache', None, ''] ”. En cambio, no hemos detectado desde donde se ha realizado la llamada.

Tabla V Análisis Dex class Loader

Binder

En esta parte del análisis utilizaremos la función startActivity en la clase Activity de Android, la cual está relacionada con la navegación entre pantallas de la aplicación. Además, entre los resultados obtenidos se mencionan actividades relacionadas con la visualización de saldos y la autenticación biométrica.

Clase	**Método**
android.app.Activity	Función startActivity. Los argumentos utilizados en esta primera clase son “<instance: android.content.Intent>', None”. El punto de llamada desde donde se ha realizado es “android.app.Activity.startActivity(Activity.java:4759)”.
android.app.Activity	Función startActivity. Los argumentos empleados en esta segunda clase son “<instance: android.content.Intent>”. El punto de llamada en este segundo resultado es “com.app.damnvulnerablebank.SplashScreen$a.run(Unknown Source:15) ”.

Tabla VI Binder

Base64

En el caso de Base64, los resultados muestran operaciones de codificación y decodificación en formato Base64. Entre las operaciones detectadas, algunas de ellas están relacionadas con argumentos y valores de retorno de funciones específicas en Android que trabajan con codificación Base64.

Las cadenas descodificadas de Base64 tras el análisis dinámico son las posteriores:

CALLED	DECODED STRING
a.a.a.a.a.A0(Unknown Source:12)	0⊠0⊠⊠⊠⊠⊠⊠ ∃l}N0 ⊠ *H ⊠⊠⊠⊠01⊠0 ⊠⊠U⊠⊠⊠⊠US1⊠0⊠⊠⊠U⊠⊠⊠ California1⊠0⊠⊠⊠U⊠⊠⊠ Mountain View1⊠0⊠⊠⊠U⊠ ⊠⊠Android1⊠0⊠⊠⊠U⊠⊠⊠⊠Android1⊠0⊠⊠⊠U⊠⊠⊠⊠Andro id1"0 ⊠ *H ⊠ ⊠⊠⊠android@android.com0⊠⊠ 080415233656Z⊠ 350901233656Z01⊠0 ⊠⊠U⊠⊠⊠⊠US1⊠0⊠⊠⊠U⊠⊠⊠ California1⊠0⊠⊠⊠U⊠⊠⊠ Mountain View1⊠0⊠⊠⊠U⊠ ⊠⊠Android1⊠0⊠⊠⊠U⊠⊠⊠⊠Android1⊠0⊠⊠⊠U⊠⊠⊠⊠Andro id1"0 ⊠ *H ⊠ ⊠⊠⊠android@android.com0⊠ 0 ⊠ *H ⊠⊠⊠⊠⊠⊠ 0⊠⊠⊠⊠⊠.⊠ 1Mɜ⊠\=3🡅tì9\,⊠ǧVF=eBk⊠ÜZ9gkə'K⊠"⊠)⊠rm*0⊠o⊠:t5m# }e⊠ QlN⊠⊠|🡅U⊠⊠[uHj⊠O,{ڍ 5"⊠⊠^Uym⊠⊠⊠a⊠Eh!^TlS⊠⊠⊠⊠b🡅⊠aḛ<d_/Uu@p? qQ6pj⊠^⊠

Figura 46 Base64 string descodificado I

a.a.a.a.a.A0(Unknown Source:12)	0⊠C0⊠+⊠⊠⊠⊠⊠ FdJ00 ⊠ *H ⊠⊠⊠⊠0t1⊠0 ⊠⊠U⊠⊠⊠⊠US1⊠0⊠⊠⊠U⊠⊠⊠ California1⊠0⊠⊠⊠U⊠⊠⊠ Mountain View1⊠0⊠⊠⊠U⊠ ⊠⊠Google Inc.1⊠0⊠⊠⊠U⊠⊠⊠⊠Android1⊠0⊠⊠⊠U⊠⊠⊠⊠Android0⊠⊠ 080821231334Z⊠ 360107231334Z0t1⊠0 ⊠⊠U⊠⊠⊠⊠US1⊠0⊠⊠⊠U⊠⊠⊠ California1⊠0⊠⊠⊠U⊠⊠⊠ Mountain View1⊠0⊠⊠⊠U⊠ ⊠⊠Google Inc.1⊠0⊠⊠⊠U⊠⊠⊠⊠Android1⊠0⊠⊠⊠U⊠⊠⊠⊠Android0⊠ 0 ⊠ *H ⊠⊠⊠⊠⊠⊠ 0⊠⊠⊠⊠⊠V.;⊠ o⊠N)⊠V⊔Ẋ⊠⊠Tr@⊠⊠⊠ gbNFVwj⊠=†⊠ $⊠w⊠⊠jG;3` w⊠1E{.XftV[⊠LjYUQ=\'""Ru⊠Jd_qh⊠xWi4y~.v L⊠qT_d⊠D⊠I⊠AW⊠_\⊠U⊠'Q⊠Šo⊠y?h)y⊠⊠k;ĝU*⊠;LX5⊠ D$38rR!^ڏ ⊠⊠[jy⊠⊠⊠00⊠⊠⊠U⊠⊠⊠⊠⊠⊠]!⊠V%⊠ӂkɾ 0⊠⊠U⊠#⊠0⊠}

Figura 47 Base64 string descodificado II

c.c.b.b.e(:3)	[DEFAULT]
c.c.b.b.e(:8)	1:932398433474:android:64f0cff260d1f223f3675b

Figura 48 Base64 string descodificado III

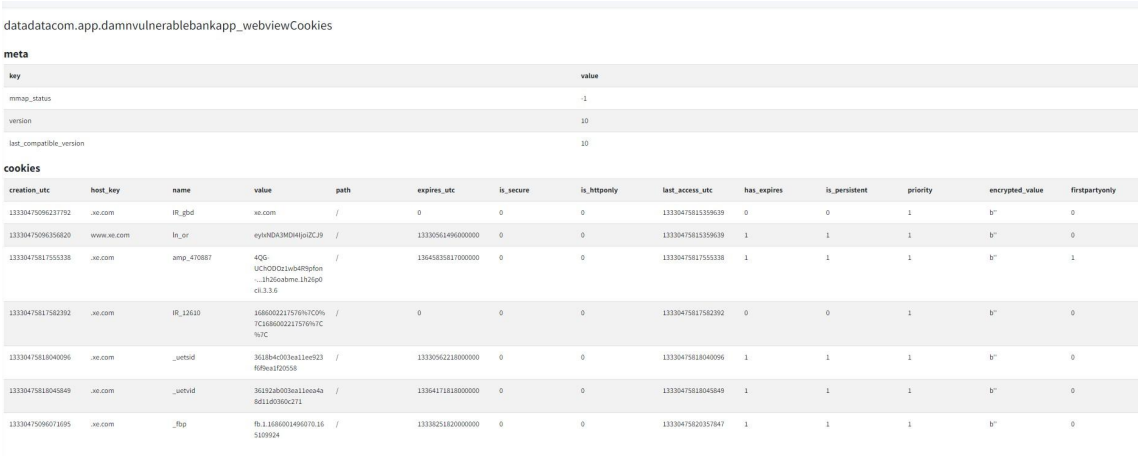
Sqlite database

En el sqlite database se realizó una búsqueda de archivos de base de datos SQLite en la ruta de datos de la aplicación vulnerable. Los resultados indican la presencia de archivos relacionados con cookies y datos de WebView.

Las sqlite database detectadas tras el análisis son:

Archivos
datadatacom.app.damnvulnerablebankapp_webviewCookies
datadatacom.app.damnvulnerablebankapp_webviewWeb_Data

Tabla VII Sqlite Database


datadatacom.app.damnvulnerablebankapp_webviewCookies

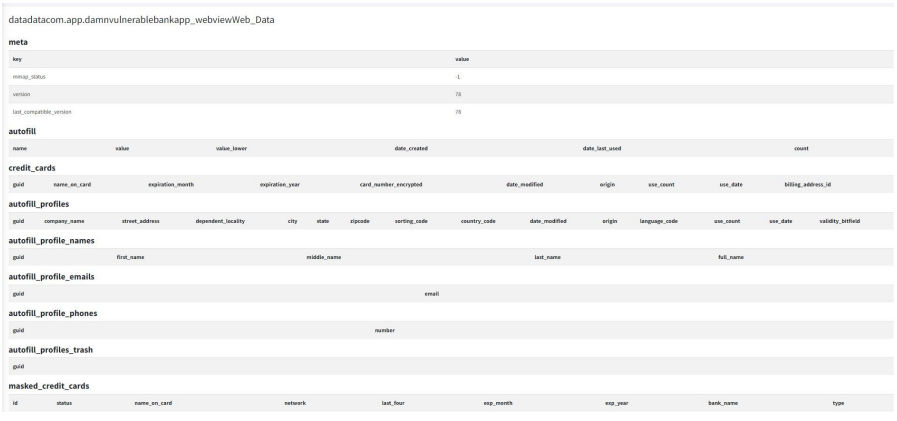
meta

key	value
mmap_status	-1
version	10
last_compatible_version	10

cookies

creation_utc	host_key	name	value	path	expires_utc	is_secure	is_httponly	last_access_utc	has_expires	is_persistent	priority	encrypted_value	firstpartyonly
13330475096237792	.xe.com	IR_gbd	xe.com	/	0	0	0	13330475815359639	0	0	1	b''	0
13330475096356820	www.xe.com	ln_or	eyIxNDA3MDI4IjoiZCJ9	/	13330561496000000	0	0	13330475815359639	1	1	1	b''	0
13330475817555338	.xe.com	amp_470887	4QG-UChODOz1wb4R9pfon-...1h26eabme.1h26p0cii.3.3.6	/	13645835817000000	0	0	13330475817555338	1	1	1	b''	1
13330475817582392	.xe.com	IR_12610	1686002217576%7C0%7C1686002217576%7C%7C	/	0	0	0	13330475817582392	0	0	1	b''	0
13330475818040096	.xe.com	_uetsid	3618b4c003ea11ee923f6f9ea1f20558	/	13330562218000000	0	0	13330475818040096	1	1	1	b''	0
13330475818045849	.xe.com	_uetvid	36192ab003ea11eea3a48d11d0360c271	/	13364171818000000	0	0	13330475818045849	1	1	1	b''	0
13330475096071695	.xe.com	_fbp	fb.1.1686001496070.165109924	/	13338251820000000	0	0	13330475820357847	1	1	1	b''	0

Figura 49 Sqlite database "webviewCookies"


datadatacom.app.damnvulnerablebankapp_webviewWeb_Data

meta

key	value
mmap_status	-1
version	78
last_compatible_version	78

autofill

name	value	value_lower	date_created	date_last_used	count

credit_cards

guid	name_on_card	expiration_month	expiration_year	card_number_encrypted	date_modified	origin	use_count	use_date	billing_address_id

autofill_profiles

guid	company_name	street_address	dependent_locality	city	state	zipcode	sorting_code	country_code	date_modified	origin	language_code	use_count	use_date	validity_bitfield

autofill_profile_names

guid	first_name	middle_name	last_name	full_name

autofill_profile_emails

guid	email

autofill_profile_phones

guid	number

autofill_profiles_trash

guid

masked_credit_cards

id	status	name_on_card	network	last_four	exp_month	exp_year	bank_name	type

Figura 50 Sqlite database "webviewWeb_data"

Xml files

Una vez realizamos el análisis detectamos ficheros de XML de todo tipo, entre ellos, algunos relacionados con WebView y Firebase.

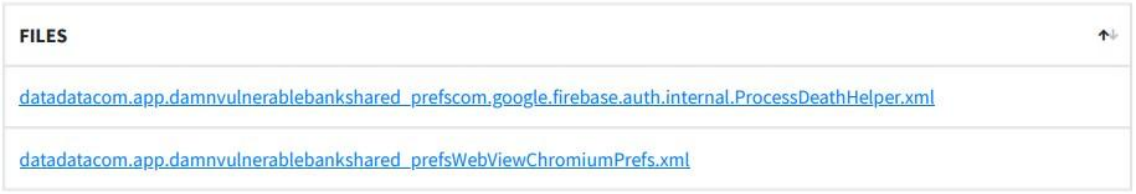


Figura 51 Ficheros XML

Other files

Dentro de los archivos extra del reporte final se mencionan archivos de caché, archivos de bloqueo y archivos de índice, relacionados con WebView y Frida.

Los otros ficheros:

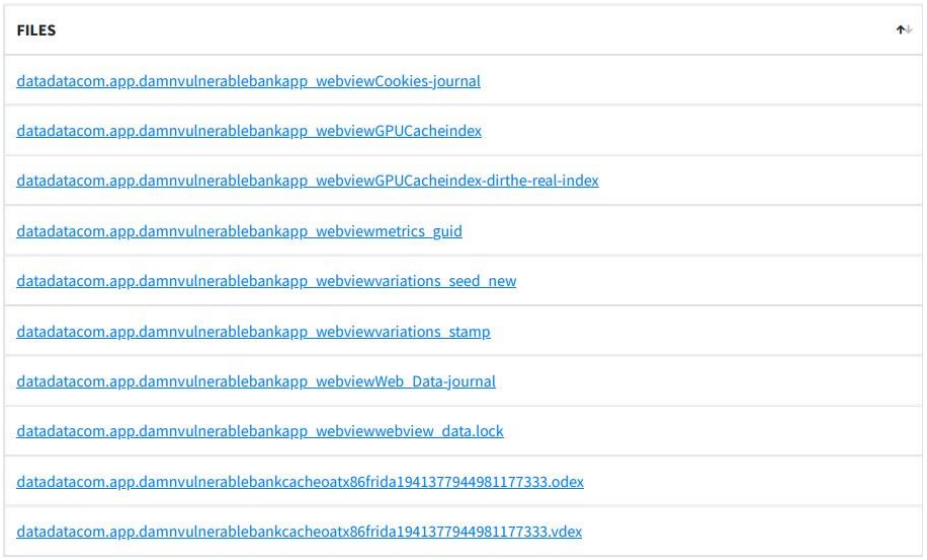


Figura 52 Ficheros extra

3.1.5 Análisis de los resultados (Cuarto procedimiento)

En este apartado analizaremos los resultados obtenidos tanto en el análisis estático como en el análisis dinámico realizado previamente sobre la aplicación vulnerable (Damn Vulnerable Bank).

Análisis estático

Tras descubrir los resultados del análisis estático realizado a la aplicación vulnerable (Damn Vulnerable Bank), el análisis correspondiente es el siguiente:

Actividades Navegables:

Dentro de la aplicación se ha detectado que la actividad CurrencyRates es accesible por aplicaciones externas utilizando ciertos filtros de intenciones, lo cual significa que no está correctamente protegida y es un problema grave de seguridad para la aplicación.

Seguridad de la Red:

La configuración base está configurada de forma insegura para permitir tráfico de texto sin cifrar a todos los dominios. También está configurada para confiar en certificados instalados por el usuario y en certificados del sistema.

Análisis del Manifiesto:

- La aplicación se puede instalar en una versión vulnerable de Android (minSdk=21).
- El tráfico de texto sin cifrar está habilitado para la aplicación, lo que permite la comunicación insegura en la red.
- Los datos de la aplicación pueden ser respaldados, lo que potencialmente expone información sensible.
- La actividad (com.google.firebase.auth.internal.FederatedSignInActivity) está protegida por un permiso, pero se debe verificar el nivel de protección del permiso.

Análisis de Código:

- La aplicación registra información, incluyendo información potencialmente sensible, y tiene capacidades de detección de root.
- La aplicación puede leer/escribir en almacenamiento externo, y cualquier aplicación puede leer los datos escritos en el almacenamiento externo.

Análisis de Biblioteca Compartida Binaria:

- Los objetos compartidos (libtool-checker.so y libfrida-check.so) tienen el bit NX configurado, proporcionando protección de memoria no ejecutable.

- Los objetos compartidos tienen un “stack canary” para detectar desbordamientos de búfer de pila.

- Los objetos compartidos no tienen una ruta de búsqueda en tiempo de ejecución (RPATH) o RUNPATH.

- Los objetos compartidos no tienen funciones fortificadas, que pueden ayudar a prevenir desbordamientos de búfer.

Firebase database, Emails y Secretos codificados

- El análisis indica que la aplicación se comunica con una base de datos Firebase en la siguiente URL: https://damn-vulnerable-bank.firebaseio.com.

- Se han encontrado dos direcciones de correo electrónico expuestas: "u0013android@android.com0" y "u0013android@android.com".

- Por último, se muestran posibles secretos o claves codificadas, incluyendo una URL de base de datos de Firebase y claves de API de Google.

    - "firebase_database_url" : "https://damn-vulnerable-bank.firebaseio.com"
    - "google_api_key". : "AIzaSyBbOHG6DDa6DOcRGEg57mw9nXYXcw6la3c".
    - "google_crash_reporting_api_key" : "AIzaSyBbOHG6DDa6DOcRGEg57mw9nXYXcw6la3c".

Estos últimos detalles analizados de los resultados obtenidos, certifican que los datos son sensibles y de gran importancia, ya que podrían provocar un acceso no autorizado a la base de datos y servicios asociados. Lo que implica que la vulnerabilidad de “Divulgación de información confidencial” está amenazando a la aplicación analizada.

Análisis dinámico

Tras descubrir los resultados del análisis dinámico realizado a la aplicación vulnerable (Damn Vulnerable Bank), el análisis correspondiente es el siguiente:

Pruebas de seguridad TLS/SSL:

En el caso de las pruebas de seguridad TLS/SSL se verifica la configuración de seguridad de las conexiones HTTPS en la aplicación. En los apartados realizados se evaluó si se permitía el tráfico de texto sin cifrar, si existían configuraciones incorrectas de TLS y si se implementa la transparencia de certificados o fijación TLS.

Dex class loader:

En el caso de “Dex class loader” se realizó una búsqueda de clases y recursos en archivos DEX, utilizados en el desarrollo de aplicaciones de Android. En el análisis se encontró el archivo 'firebase-auth.properties' dentro del archivo APK de la aplicación.

Binder:

En el análisis dinámico se analizó la función startActivity en la clase Activity de Android, relacionada con la navegación entre pantallas de la aplicación. En la cual se detectaron actividades relacionadas con la visualización de saldos y la autenticación biométrica.

Base64:

En este caso se identificaron operaciones de codificación y decodificación en formato Base64. En él se encontraron cadenas descodificadas relacionadas con argumentos y valores de retorno de funciones específicas en Android.

SQLite database:

En esta parte del análisis dinámico se buscó la presencia de archivos de base de datos SQLite en la ruta de datos de la aplicación. En el proceso se encontraron archivos relacionados con cookies y datos de WebView.

Archivos XML:

En la búsqueda de archivos se detectaron archivos XML, incluyendo algunos relacionados con WebView y Firebase.

Otros archivos:

Se mencionaron archivos de caché, bloqueo e índice relacionados con WebView y Frida.

En resumen, el análisis dinámico realizado en la aplicación "Damn Vulnerable Bank" reveló varias vulnerabilidades y riesgos potenciales. Las pruebas de seguridad TLS/SSL identificaron configuraciones incorrectas y falta de implementación de certificados, lo que podría permitir ataques MITM y suplantación de identidad. El Dex class loader mostró la posibilidad de acceder a archivos sensibles si no están protegidos adecuadamente. El uso inseguro de startActivity en Binder podría permitir ataques de suplantación de identidad y ejecución de código malicioso. Las operaciones de codificación y decodificación Base64 podrían ser explotadas para inyección de código y acceso no autorizado. La gestión insegura de la base de datos SQLite podría permitir la manipulación de datos y la extracción de información

confidencial. Los archivos XML expuestos podrían sufrir inyecciones de código y modificaciones no autorizadas. Los archivos extra ”Otros archivos” podrían conducir a accesos no autorizados y ataques de denegación de servicio.

3.1.6 Documentación y resultados (Quinto procedimiento)

En este último procedimiento realizaremos una documentación en el cual detallaremos las diferentes conclusiones o resultados que hemos encontrado en la aplicación, los posibles impactos de las vulnerabilidades identificadas y por último las recomendaciones o contramedidas para corregirlas.

Teniendo en cuenta las conclusiones o resultados y procedimientos de la metodología aplicada en la aplicación vulnerable (Damn Vulnerable Bank) para el análisis de seguridad en aplicaciones móviles, detallaremos los siguientes apartados:

Conclusiones:

a. Durante la realización del análisis de la aplicación, se determinaron varias vulnerabilidades de seguridad significativas.

b. En el análisis se encontraron deficiencias en la autenticación y autorización de usuarios, lo que podría permitir accesos no autorizados a información sensible o acciones malintencionadas.

c. También se detectaron vulnerabilidades relacionadas con la gestión inadecuada de sesiones y la protección insuficiente de datos confidenciales.

d. Otra área de preocupación identificada es la falta de validación adecuada de entradas de usuario, lo que podría abrir la puerta a ataques de inyección de código o manipulación de datos.

e. Por último, se hallaron vulnerabilidades de configuración incorrecta y errores de seguridad en componentes de terceros utilizados en varias zonas de la aplicación.

Posibles impactos de las vulnerabilidades identificadas:

- La falta de autenticación y autorización adecuadas podría permitir que usuarios no autorizados accedan a información confidencial, realicen modificaciones no autorizadas o incluso tomen el control del sistema.

- Las vulnerabilidades relacionadas con la gestión de sesiones y la protección insuficiente de datos podrían exponer información confidencial, como credenciales de usuario.

- Las vulnerabilidades de entrada no validada podrían permitir la ejecución de comandos maliciosos, la inyección de código o la manipulación de datos, lo que podría comprometer la integridad y disponibilidad de la aplicación.

- Las vulnerabilidades de configuración incorrecta y los errores de seguridad en componentes de terceros podrían facilitar ataques dirigidos y el acceso no autorizado a recursos o la explotación de vulnerabilidades conocidas en dichos componentes.

Recomendaciones o Contramedidas:

➔ Implementar una autenticación y autorización adecuadas, utilizando prácticas de seguridad recomendadas, como el uso de contraseñas seguras, la aplicación de bloqueos de cuentas tras varios intentos fallidos de inicio de sesión y la asignación de privilegios mínimos necesarios para cada usuario. Además, incluiremos controles de autenticación y autorización en el lado servidor, conociendo que los controles del lado cliente están siendo eludidos.

➔ Mejorar la gestión de sesiones y la protección de datos confidenciales mediante la implementación de mecanismos adecuados de cifrado, el establecimiento de tiempos de expiración de sesiones y la segregación de datos confidenciales del resto del sistema. También incluiremos controles desde el lado servidor para evitar que los datos confidenciales, sensibles o de cualquier tipo sean accesibles.

➔ Mantener un control de acceso de autenticación exclusivo para cada cliente. Asimismo, implementar un modelo de permisos granular que defina qué acciones y que accesos tiene cada usuario para prevenir las operaciones con información de otros clientes.

➔ Agregar la utilización de encriptación de datos, una gestión de claves más óptima y un proceso de destrucción de datos permanente para reducir la posibilidad de almacenamiento de información sensible en el dispositivo.

➔ Realizar una validación exhaustiva de todas las entradas de usuario para evitar ataques de inyección de código o manipulación de datos. Esto puede lograrse mediante el uso de bibliotecas o funciones de validación de entrada confiables.

➔ Mantener una política de actualización regular y monitoreo de los componentes de terceros utilizados en la aplicación. Asegurarse de aplicar parches de seguridad y de configurar adecuadamente estos componentes para mitigar posibles vulnerabilidades conocidas.

➔ Efectuar pruebas periódicas de seguridad, como pruebas de penetración y análisis estático de código, para identificar y abordar nuevas vulnerabilidades y riesgos de seguridad.

➔ Fomentar la concienciación sobre la seguridad entre los desarrolladores y usuarios de la aplicación, proporcionando capacitación y recursos para comprender y aplicar buenas prácticas de seguridad.

Estas recomendaciones y contramedidas ayudarán a mitigar las vulnerabilidades identificadas en la aplicación "Damn Vulnerable Bank" y mejorarán su seguridad en general.

Figura 53 Documentación de la metodología

Conclusiones y trabajos futuros

Las conclusiones extraídas del desarrollo final y del resultado de la realización del trabajo fin de máster superaron las expectativas, lo cual nos permitió alcanzar los objetivos planteados inicialmente, incluso obteniendo resultados adicionales de gran valor. En el proceso del trabajo se logró mitigar de manera efectiva los posibles impactos éticos, sociales, de sostenibilidad y de diversidad, gracias a la implementación de medidas específicas en la recolección y en la utilización de datos, y adoptando prácticas sustentables en la metodología.

En el transcurso de la realización o desarrollo del proyecto se siguió la planificación y la metodología establecidas al comienzo, realizando en algunas ocasiones ajustes o variaciones hasta alcanzar el resultado deseado del trabajo. En las diferentes entregas de seguimiento y reuniones obtuve retroalimentaciones de las cuales me sirvieron para reconducir o realizar cambios necesarios con la intención de optimizar el resultado.

En la realización del trabajo se han cumplido la mayoría de los objetivos impuestos, sin embargo, han quedado variaciones en alguna área en la cual se podría explorar más, lo cual brinda algunas opciones para futuros trabajos. La primera de las posibilidades a futuro sería aplicar la metodología desarrollada en una aplicación vulnerable iOS. La segunda de las opciones sería observar y documentar el impacto con efectos a largo plazo de la metodología desarrollada en diferentes aplicaciones móviles. La última de las opciones sería aplicar la metodología desarrollada para diferentes empresas que creen o desarrollen proyectos a futuro en relación con aplicaciones móviles.

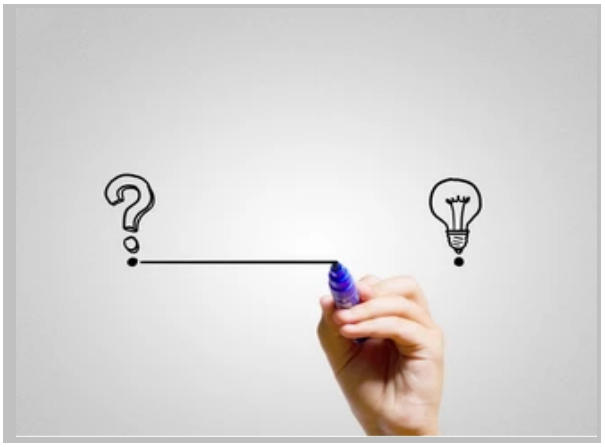


Figura 54 Conclusiones y trabajos futuros

Glosario

OWASP **Acrónimo:** Open Web Application Security Project

Definición: Owasp es una organización dedicada a mejorar la seguridad del software a través de la concienciación, la educación y la promoción de las mejores prácticas de seguridad.

TFM **Acrónimo:** Trabajo Fin de Máster

Definición: TFM es el proyecto o trabajo de investigación final que realizan los estudiantes para terminar de completar los estudios de máster.

CCEG **Acrónimo:** Corporate Governance and Executive Governance

Definición: CCEG es el conjunto de las tres dimensiones de las cuales pueden tener impactos positivos o negativos. Las dimensiones que completan el conjunto son “Dimensión Social”, “Dimensión Ambiental” y Dimensión Económica.

IoT **Acrónimo:** Internet of Things

Definición: IoT se refiere a la red de dispositivos físicos interconectados que están incorporados con sensores, software y tecnología de red para recopilar y compilar datos.

NFC **Acrónimo:** Near Field Communication

Definición: NFC es una tecnología de comunicación inalámbrica de corto alcance que permite el intercambio de datos entre dispositivos cercanos.

API **Acrónimo:** Application Programming Interface

Definición: API es el conjunto de reglas y protocolos que permite la comunicación entre diferentes componentes de software.

JNI **Acrónimo:** Java Native Interface

Definición: Java Native Interface es un framework de programación el cual nos permite que un programa escrito exclusivamente en java pueda interactuar con programas escritos en diferentes lenguajes.

Java

Definición: Java es un lenguaje de programación popular y ampliamente utilizado. Java fue desarrollado por Sun Microsystems y se caracteriza por ser orientado a objetos y tener una amplia plataforma de desarrollo.

Kotlin

Definición: Kotlin es un lenguaje desarrollado por JetBrains. Kotlin es compatible con Java y se emplea principalmente para desarrollar aplicaciones de Android.

Bibliografía

[1] Todo sobre Android

https://www.tutorialspoint.com/android/android_overview.htm

[2] Todo sobre iOS

https://www.techtarget.com/searchmobilecomputing/definition/iOS

[3] Tipos de aplicaciones

https://www.futurespace.es/tipos-de-aplicaciones-moviles/

[4] Categorías de aplicaciones Android/IOS

https://www.solbyte.com/blog/categorias-de-aplicaciones-moviles-que-existen/

[5] OWASP Top 10 riesgos móviles

https://owasp.org/www-project-mobile-top-10/

[6] Las principales amenazas en un futuro en las aplicaciones móviles.

https://latam.kaspersky.com/resource-center/threats/top-seven-mobile-security-threats-smart-phones-tablets-and-mobile-internet-devices-what-the-future-has-in-store

[7] OWASP M1

https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage

[8] OWASP M2

https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

[9] OWASP M3

https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

[10] OWASP M4

https://owasp.org/www-project-mobile-top-10/2016-risks/m4-insecure-authentication

[11] OWASP M5

https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

[12] OWASP M6

https://owasp.org/www-project-mobile-top-10/2016-risks/m6-insecure-authorization

[13] OWASP M7

https://owasp.org/www-project-mobile-top-10/2016-risks/m7-client-code-quality

[14] OWASP M8

https://owasp.org/www-project-mobile-top-10/2016-risks/m8-code-tampering

[15] OWASP M9

https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering

[16] OWASP M10

https://owasp.org/www-project-mobile-top-10/2016-risks/m10-extraneous-functionality

[17] Metodología de seguridad

https://github.com/OWASP/owasp-mastg/releases

[18] Auditorías de seguridad

https://www.vaadata.com/blog/black-grey-white-box-penetration-test-3-options/

[19] SAST y DAST

https://circleci.com/blog/sast-vs-dast-when-to-use-them/#:~:text=SAST%20scans%20the%20application%20code,stimulates%20an%20outside%20attacker's%20perspective

[20] Angr

https://angr.io/

[21] Frida

https://frida.re/

[22] MobSF

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

[23] Ghidra

https://github.com/NationalSecurityAgency/ghidra/releases

[24] Radare

https://rada.re/n/

[25] Damn Vulnerable Bank (Aplicación vulnerable)

https://github.com/rewanthtammana/Damn-Vulnerable-Bank

[26] Principales amenazas

https://www.incibe.es/incibe-cert/blog/auditorias-de-seguridad-de-apps-android

[27] Documentación MobSF

https://mobsf.github.io/docs/#/

[28] Guía de la aplicación vulnerable

https://rewanthtammana.com/damn-vulnerable-bank/index.html

Anexos

Los anexos detallados posteriormente han sido ejecutados siendo administrador o teniendo permisos o accesos con privilegios dentro de ambos entornos.

Anexo A Instalación de la aplicación elegida

Instalación de la aplicación vulnerable (Ubuntu) [28]

Instalamos la herramienta adb para utilizarla de puente entre la consola del equipo y el teléfono o emulador en el que se visualizará la aplicación vulnerable. El comando para instalar adb es el siguiente: “sudo apt install adb”

```
root@user1-VirtualBox:/home/user1# sudo apt install adb
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  android-libadb android-libbase android-libboringssl android-libcrypto-utils
  android-libcutils android-liblog android-sdk-platform-tools-common
The following NEW packages will be installed:
```

Figura 55 Instalación de la herramienta adb

Clonamos el repositorio de la aplicación vulnerable con el comando “git clone https://github.com/rewanthtammana/Damn-Vulnerable-Bank.git”. Es necesario tener instalado Git previamente.

```
root@user1-VirtualBox:/home/user1# git clone https://github.com/rewanthtammana/D
amn-Vulnerable-Bank.git
Cloning into 'Damn-Vulnerable-Bank'...
remote: Enumerating objects: 1126, done.
remote: Counting objects: 100% (48/48), done.
remote: Compressing objects: 100% (10/10), done.
```

Figura 56 Clonación del repositorio aplicación vulnerable

Realizamos el último paso de la instalación ejecutando el comando “adb install dvba.apk”, con el cual realizaremos la instalación de la apk descargada.

```
root@user1-VirtualBox:/home/user1/Damn-Vulnerable-Bank# adb install dvba.apk
```

Figura 57 Instalación de la aplicación en ubuntu

Instalación de la aplicación vulnerable (Windows) [28]

Accedemos a la página de la aplicación vulnerable “https://github.com/rewanthtammana/Damn-Vulnerable-Bank” y descargamos el zip de la aplicación.

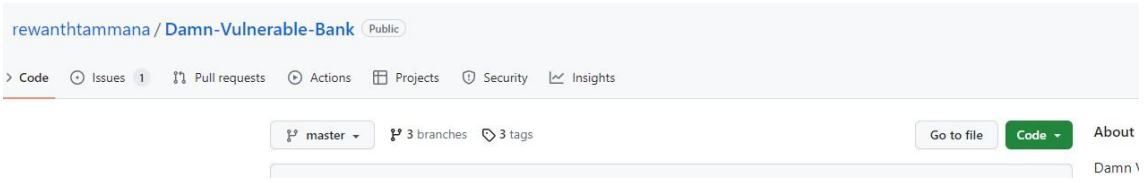


Figura 58 Repositorio de la aplicación vulnerable

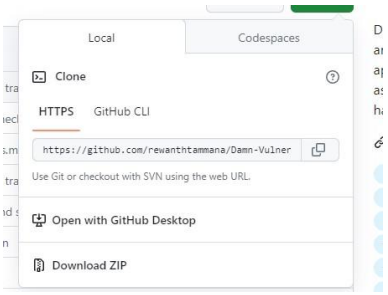


Figura 59 ZIP de la aplicación

Tras tener descargado el zip, lo extraemos y ya tendremos la aplicación descargada y lista para utilizar.

Figura 60 ,ZIP de la aplicación descargado

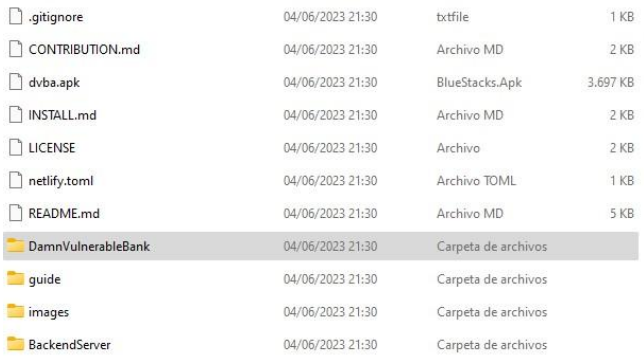


Figura 61 Ficheros de la aplicación

Anexo B Instalación de MobSF (Análisis estático) [27]

- Requisitos previos:
    - Instalar Git.
    - Instalar Python.
    - Instalar JDK 8+.

- Instalar estas dependencias: python3-dev python3-venv python3-pip build-essential libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev wkhtmltopdf.

- Instalación de la herramienta MobSF.
- Iniciación y funcionamiento.

Requisitos previos

El primero de los pasos será actualizar la lista de paquetes disponibles dentro de los repositorios de Ubuntu. El comando utilizado es “apt update”.

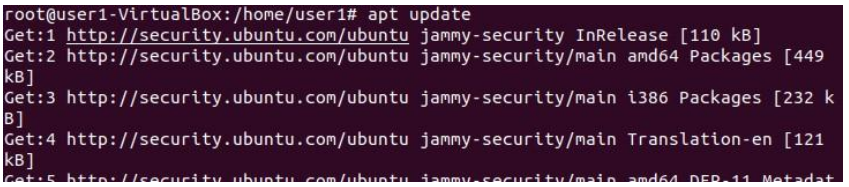


Figura 62 apt update

El segundo de los pasos será mostrar una lista de los paquetes que tienen actualizaciones disponibles y verificar que paquetes están listos para ser actualizados. El comando utilizado es “apt list - -upgradable”.

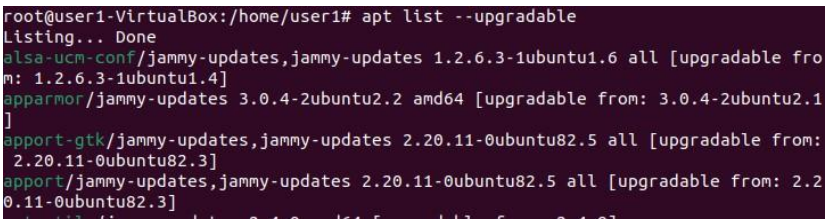


Figura 63 apt list –upgradable

El tercero de los pasos será actualizar los paquetes instalados en el sistema con las últimas versiones disponibles. El comando empleado es “apt upgrade”.

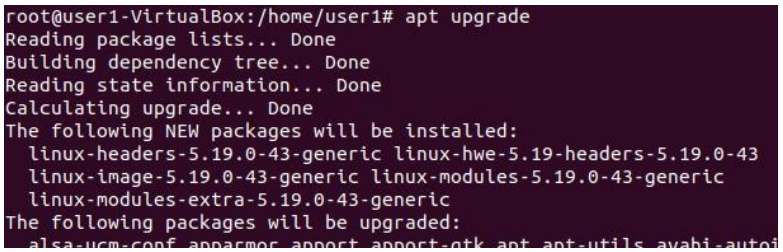


Figura 64 apt upgrade

Instalación de Pip con el comando “apt install python3-pip”

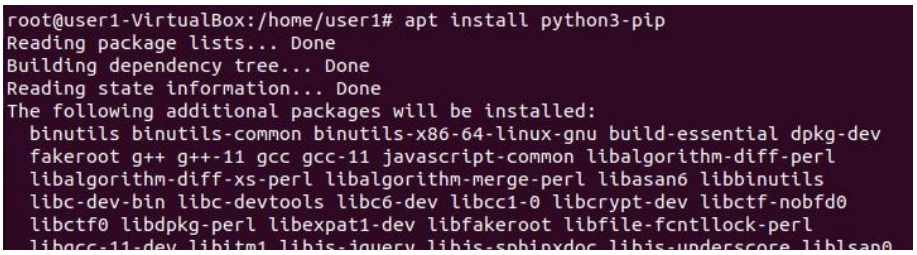


Figura 65 apt install python3-pip

Instalación de Jdk 8+ con el comando “apt install openjdk-8-jdk”

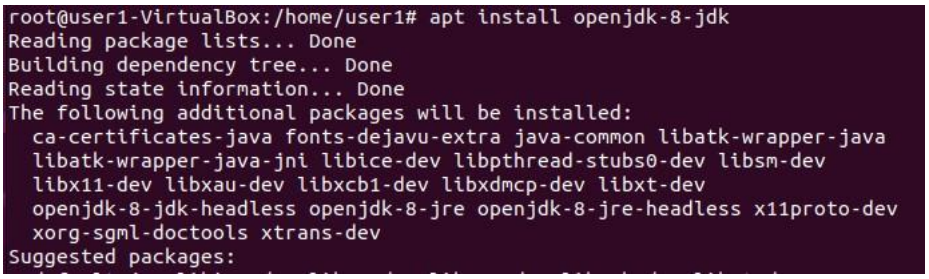


Figura 66 apt install openjdk-8-jdk

Instalación de Git con el comando “apt install git”

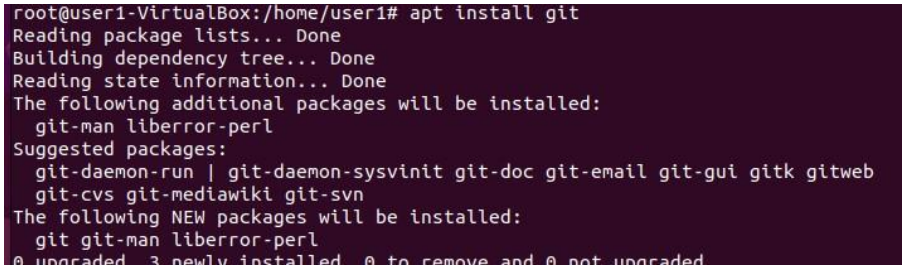


Figura 67 apt install git

Comprobación que tanto Python, Pip, Jdk 8+ y Git están instalados con la última versión disponible. Comandos utilizados: “python3 -V, git - -version, pip3 -V y java -version”

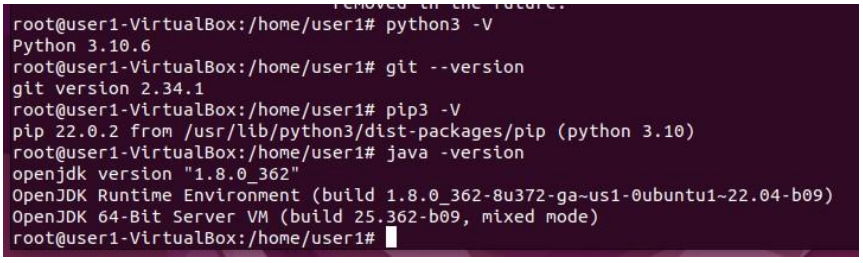


Figura 68 Comprobación de requisitos

Instalación de las últimas dependencias necesarias para la utilización de MobSF. Comando utilizado “sudo apt install python3-dev python3-venv python3-pip build-essential libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev wkhtmltopdf”.

Figura 69 Instalación de las dependencias

Instalación de la herramienta MobSF

Clonamos el repositorio de MobSF de github con el comando “git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git”.

Figura 70 Clonación repositorio MobSF

Accedemos al directorio del repositorio clonado con el comando “cd Mobile-Security-Framework-MobSF”.

Figura 71 Cambio de directorio

Ejecutamos el script de configuración de MobSF en el que se preparan tanto el entorno como la configuración de las dependencias correspondientes. Comando ejecutado “./setup.sh”.

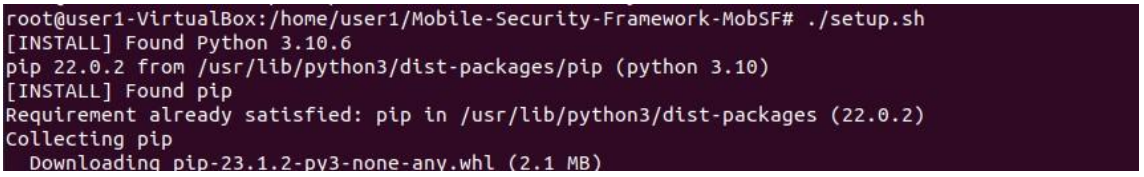


Figura 72 Aplicación del setup en Ubuntu

En la siguiente imagen observamos que ya tenemos el entorno de MobSF preparado para su utilización y que la instalación de MobSF se ha completado.

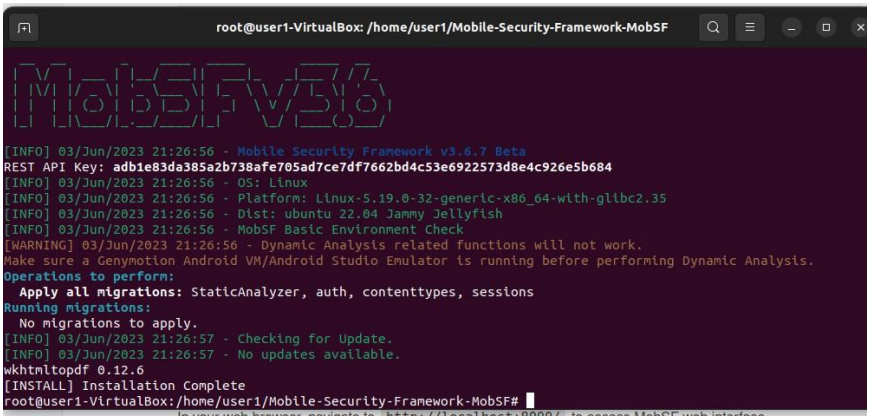


Figura 73 Instalación completada

Iniciación y funcionamiento

Inicializamos MobsF y lo ponemos en funcionamiento con el comando “./run.sh”.

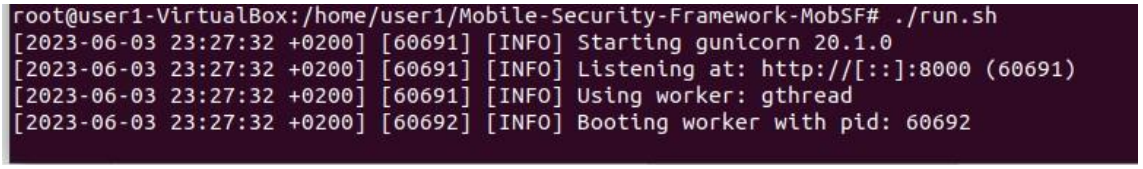


Figura 74 Iniciación del Framework

Una vez inicializado tendremos varias URL para poder acceder a la interfaz de usuario web de MobSF.

Las direcciones son:

- ☐ http://localhost:8000/
- ☐ http://0.0.0.0:8000/

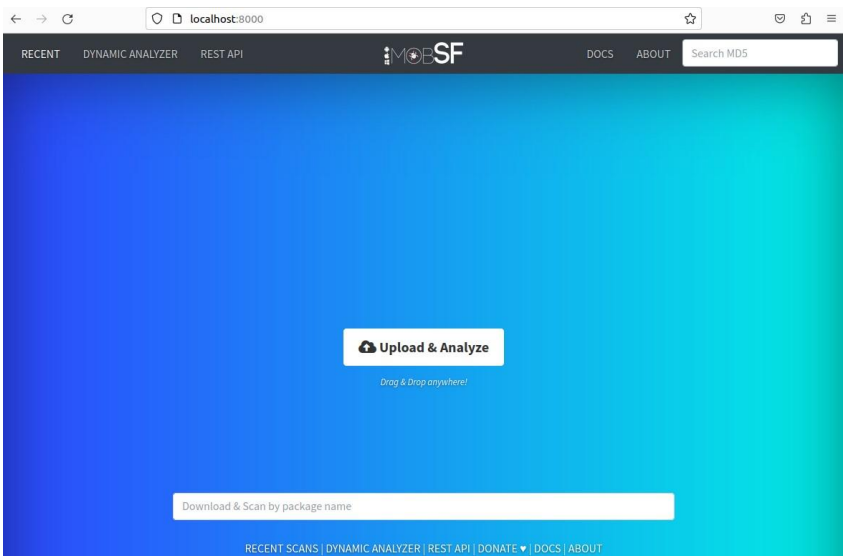


Figura 75 Interfaz de MobSF

Una vez accedido a la interfaz de usuario solo necesitaremos cargar o subir la apk a analizar y ya comenzará el análisis estático mediante la utilización de la herramienta MobSF.

Anexo C Instalación de MobSF (Análisis dinámico)

- Requisitos previos:
    - Instalar emulador (Necesario para realizar el análisis dinámico (Genymotion, Genymotion Cloud VM o Android Studio Emulator)). El análisis dinámico no funciona con un setup dentro de la máquina virtual, por lo que la ejecución del análisis dinámico no podremos realizarlo en el mismo "setup" que el análisis estático, ya que está realizado en una máquina virtual de linux. Esta vez utilizaremos el equipo Windows principal.
    - Instalar Git
    - Instalar Python 3.8-3.9
    - Instalar JDK 8+
    - Instalar las herramientas de compilación de Microsoft Visual C++
    - Instalar OpenSSL (no ligero)
    - Descargar e instalar wkhtmltopdf
    - Agregar la carpeta que contiene wkhtmltopdfel binario a la variable de entorno PATH.
- Instalación de la herramienta MobSF
- Entorno previo a la inicialización
- Iniciación y funcionamiento.

Requisitos previos

Instalación del emulador:

En este caso instalaremos Android Studio para utilizarlo como emulador. Entramos en la página oficial de Android Studio y descargamos el .exe. Tras la descarga, seguimos los pasos del instalador para completar la instalación.

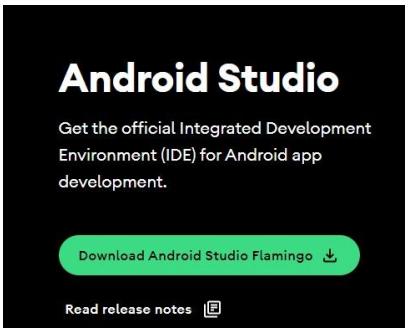


Figura 76 Android Studio.exe

Imagen de la aplicación abierta dentro de Android Studio.

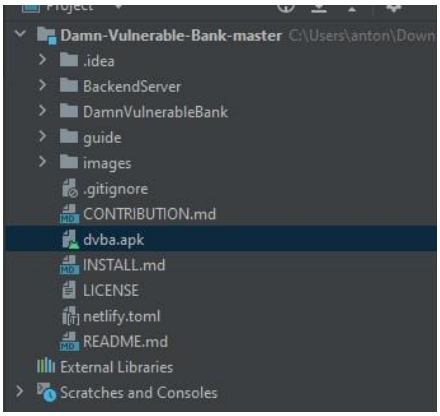


Figura 77 La aplicación vulnerable en Android Studio

Una vez tenemos instalado Android Studio crearemos el “AVD”, es decir, el dispositivo virtual de Android para utilizarlo como emulador. La versión de Android y de aplicación elegida para realizar el análisis dinámico es el siguiente:

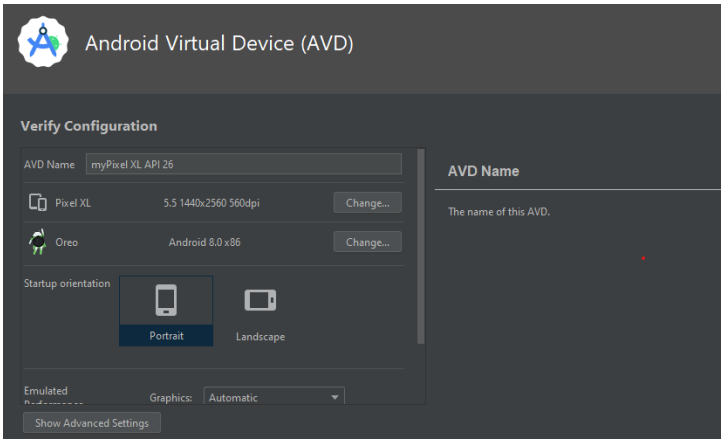


Figura 78 Android Virtual Device (AVD)

Android Virtual Device creado:

Figura 79 Android Virtual Device creado

Instalación de Git

Instalamos el ejecutable de Git desde la página oficial y realizamos los pasos para completar la instalación del software.

Figura 80 Git

Instalación de Python 3.8-3.9

Accedemos a la página de Python y en este caso no debemos instalar la versión más actual, sino que para poder realizar el análisis dinámico con MobSF es necesario el tener el python instalado entre las versiones de 3.8-3.9. En este apartado también es importante el actualizar el path en caso de tener más versiones de Python e incluir o asignar como prioridad el python con versiones de 3.8 a 3.9.

Figura 81 Python

Instalación de JDK 8+

Instalamos el ejecutable de JDK ya sea la versión 8 o superior y realizamos la instalación.

Figura 82 JDK

Instalación de las herramientas de compilación de Microsoft Visual C++

En este apartado instalaremos el Visual Studio Build Tools 2019, en el cual una vez instalado deberemos instalar la dependencia de Desarrollo para C ++ para poder completar los requerimientos de MobSF en Windows.

Figura 83 Visual Studio Build

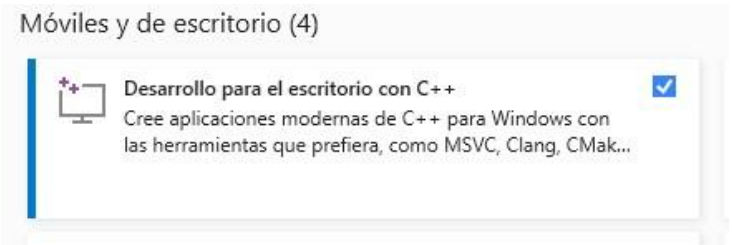


Figura 84 C++

Instalación OpenSSL (no ligero)

Instalamos el ejecutable no ligero y seguimos el proceso del instalador hasta completar la instalación.

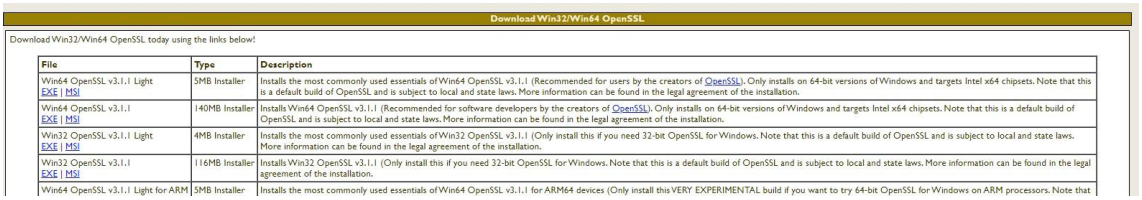


Figura 85 OpenSSL

Instalación wkhtmltopdf y agregar la carpeta al path

Instalamos el ejecutable de wkhtmltopdf con el cual seguiremos el proceso de instalación. Una vez completada la instalación deberemos añadir la ruta en el path para terminar de completar finalmente los requisitos para poder utilizar MobSF en Windows.

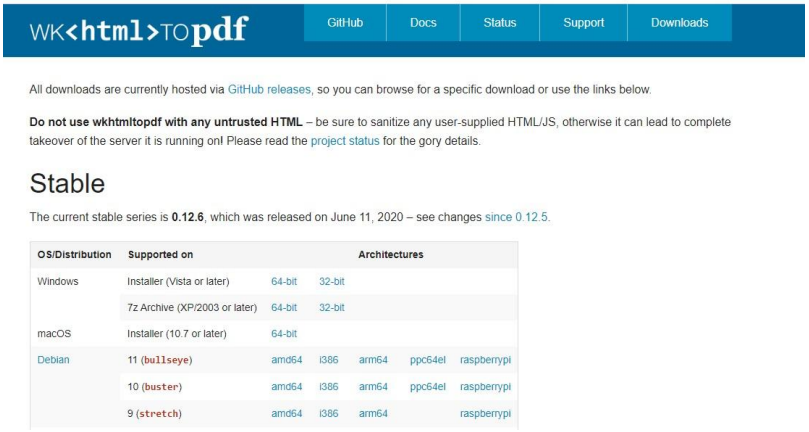


Figura 86 wkhtmltopdf

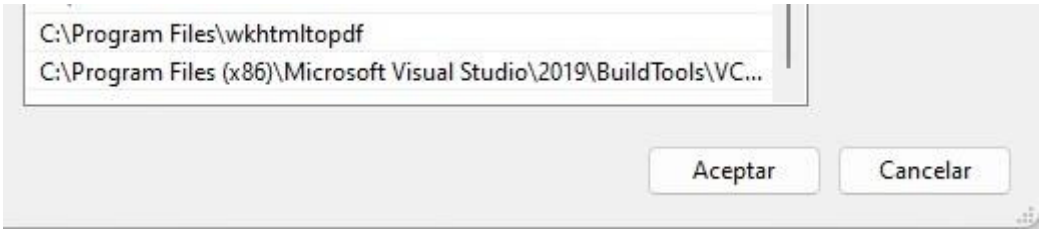


Figura 87 wkhtmltopdf en el path

Instalación de la herramienta MobSF

Nos ubicamos en el escritorio con el comando “cd Desktop” y ejecutamos el comando “git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git” para clonar el repositorio de MobSF.

```
C:\Users\anton>cd Desktop

C:\Users\anton\Desktop>git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git
```

Figura 88 Clonación del repositorio MobSF

Una vez clonado el repositorio de Github de MobSF, nos ubicamos dentro de la carpeta del Framework y ejecutamos el comando “setup.bat” para terminar de completar la instalación de la herramienta MobSF. Con este comando realizamos una comprobación de que todos los requisitos previos se han instalado correctamente y con la versión correcta. Una vez realizada todas las comprobaciones se realiza la instalación final de MobSF.

```
C:\Users\anton\Desktop>cd Mobile-Security-Framework-MobSF

C:\Users\anton\Desktop\Mobile-Security-Framework-MobSF>
```

Figura 89 Cambio de directorio

```
C:\Users\anton\Desktop\Mobile-Security-Framework-MobSF>setup.bat
[INSTALL] Checking for Python version 3.8+
[INSTALL] Found Python 3.9.13
```

Figura 90 Completamos el setup

```
 __  __       _    ____  _____       _____  __
|  \/  | ___ | |__/ ___||  ___|_   _|___ / / /_
| |\/| |/ _ \| '_ \___ \| |_  \ \ / / |_ \| '_ \
| |  | | (_) | |_) |__) |  _|  \ V / ___) | (_) |
|_|  |_|\___/|_.__/____/|_|     \_/ |____(_)___/

[INFO] 05/Jun/2023 18:13:34 - Mobile Security Framework v3.6.6 Beta
REST API Key: f8bf1194d576884c6f3de5eb84693acea949b47d40e72cb2dcee9e714dcbc733
[INFO] 05/Jun/2023 18:13:34 - OS: Windows
[INFO] 05/Jun/2023 18:13:34 - Platform: Windows-10-10.0.22621-SP0
[INFO] 05/Jun/2023 18:13:34 - MobSF Basic Environment Check
[WARNING] 05/Jun/2023 18:13:34 - Dynamic Analysis related functions will not work.
Make sure a Genymotion Android VM/Android Studio Emulator is running before performing Dynamic Analysis.
Operations to perform:
  Apply all migrations: StaticAnalyzer, auth, contenttypes, sessions
Running migrations:
  No migrations to apply.
[INFO] 05/Jun/2023 18:13:34 - Checking for Update.
[INFO] 05/Jun/2023 18:13:35 - No updates available.
Download and Install wkhtmltopdf for PDF Report Generation - https://wkhtmltopdf.org/downloads.html
[INSTALL] Installation Complete
```

Figura 91 Instalación completada

Entorno previo a la inicialización

Tras tener la configuración del emulador y la de MobSF realizada, solo nos quedará realizar los últimos pasos en la configuración del emulador antes de comenzar el análisis dinámico.

El primero de los pasos será ejecutar un comando para comprobar que emulador está en activo, en este caso detectaremos que está en activo el “myPixel_XL_API_26”, el cual es el emulador instalado en Android Studio previamente. El comando utilizado es “emulator -list-avds”.

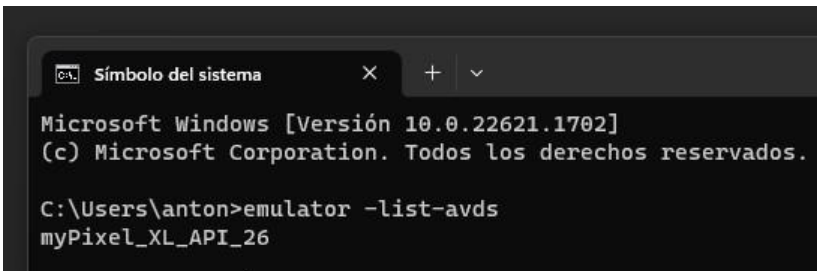


Figura 92 Lista de emuladores activos

Estableceremos un nuevo ADB_BINARY al que existía en el config.py de MobSF. Incluiremos la siguiente nueva ruta:
'C:/Users/anton/AppData/Local/Android/sdk/platform-tools/adb.exe'”

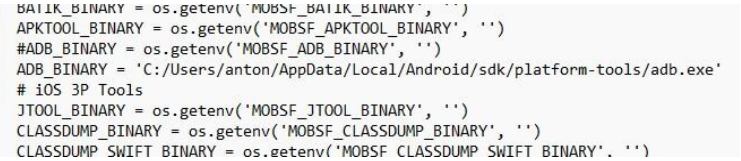


Figura 93 ADB_BINARY en la configuración

Ejecutamos el emulador con el móvil configurado, el cual utilizaremos para realizar el análisis dinámico. El comando empleado es “emulator -avd myPixel_XL_API_26 -writable-system -no-snapshot ”.

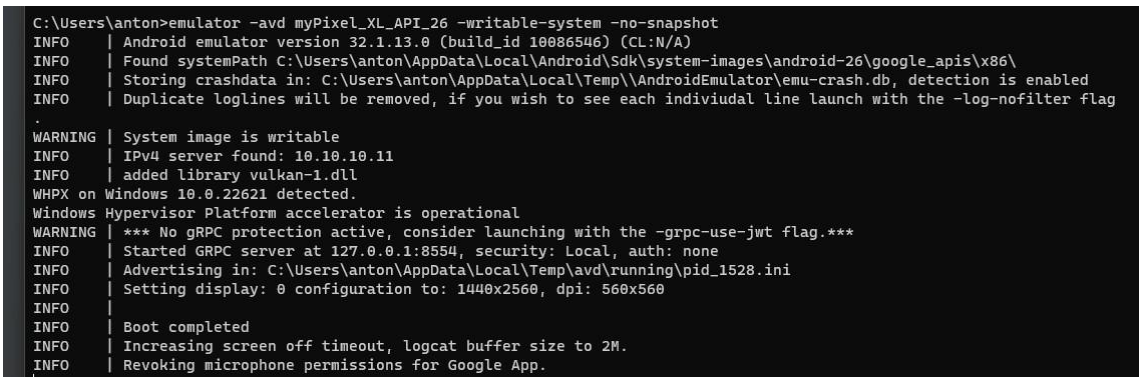


Figura 94 Ejecución de AVD

La aplicación vulnerable “Damn Vulnerable Bank” dentro del emulador Pixel XL implementado. Es el icono negro llamado “Damn Vulnerable bank” que aparece en el centro y en la parte superior..

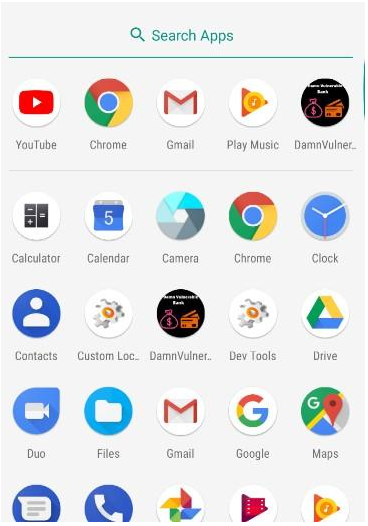


Figura 95 Aplicación vulnerable en la AVD

Iniciación y funcionamiento

Tras tener instalado el entorno, requisitos e instalación, solo nos queda realizar la inicialización de la herramienta MobSF. El comando para inicializar la herramienta MobSF es el de “run.bat 127.0.0.1:8000”, con el cual iniciamos la herramienta y en la dirección “http: 127.0.0.1:8000” o “http:localhost:8000” aparecera la interfaz de la herramienta MobSF.

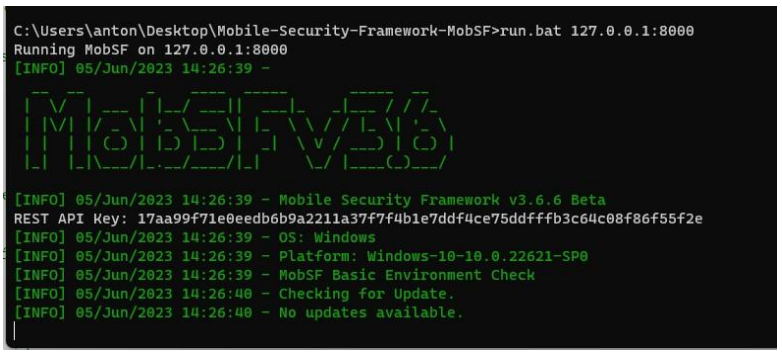


Figura 96 Iniciación de MobSF

Interfaz de MobSF una vez inicializado:

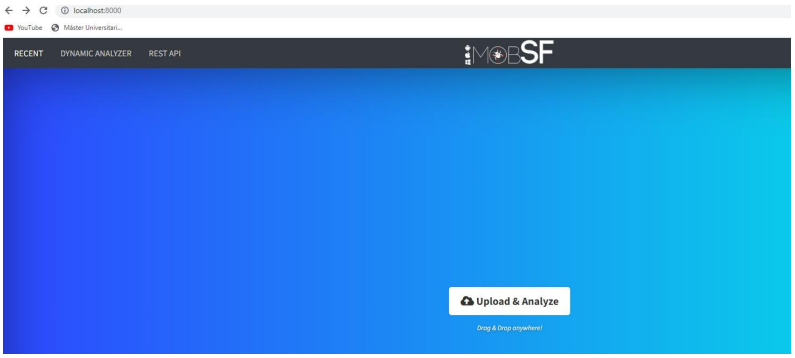


Figura 97 Interfaz de MobSF

Entramos en el apartado de análisis dinámico y detectamos el emulador configurado y realizado, por lo que solo nos quedará acceder al emulador al mismo tiempo que realizamos el análisis dinámico.

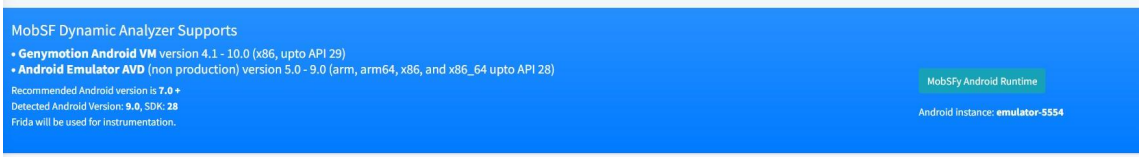


Figura 98 Soporte de MobSF

Aplicación vulnerable “Damn Vulnerable Bank” detectada para poder analizarla.

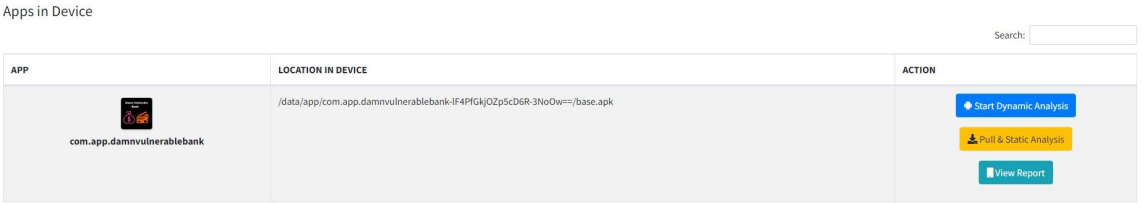


Figura 99 Aplicación vulnerable en MobSF

Interfaz del análisis dinámico de MobSF con la aplicación vulnerable implementada:

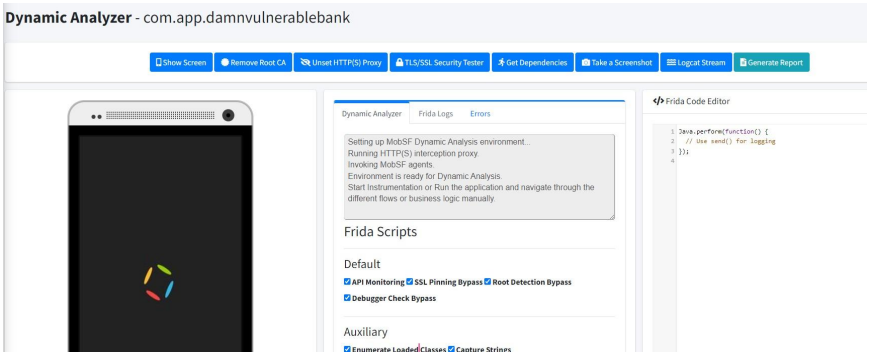


Figura 100 Iniciación del análisis dinámico