MARINA
SECRETARÍA DE MARINA

BUAP

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
(BUAP)

ANÁLISIS, DETECCIÓN Y MITIGACIÓN DE VULNERABILIDADES DE CIBERSEGURIDAD EN UNA APLICACIÓN MÓVIL DE MENSAJERÍA SEGURA

TESIS:
PARA OBTENER EL TÍTULO DE
INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
JEANETTE VARA FORTIS

ASESOR DE TESIS:
M. C. YALÚ GALICIA HERNÁNDEZ
CAP. CORB. C.G.EE. ANTONIO PÉREZ MATUS

VERACRUZ,VER., MÉXICO, Junio 2022

UNINDETEC
Unidad de Investigación y Desarrollo Tecnológico (UNINDETEC)
Veracruz, Ver. Teléfono 01 297 9560110
inidetam@semar.gob.mx

Facultad de Ciencias de la computación

RESUMEN

Las organizaciones deben atender su negocio siendo conscientes del nivel de riesgo tanto de la información como de sus activos. El objetivo primordial de la ciberseguridad, es garantizar que los recursos informáticos de una compañía estén disponibles para cumplir sus propósitos y que no se encuentren alterados por factores externos. Ya que cuando se presentan estos inconvenientes, se puede provocar la pérdida o modificación de la información, lo que directamente, puede representar un daño organizacional y económico.

Es por eso que en el presente proyecto de investigación esta orientado a realizar un análisis de seguridad a una aplicación móvil de mensajería segura en los sistemas operativos iOS y Android, esto con la finalidad reforzar la seguridad en la aplicación en cada etapa de su desarrollo para evitar la posible exposición, robo o manipulación de los datos de los usuarios y de la organización.

La aplicación móvil de mensajería instantánea es utilizada por usuarios de una organización donde se envía información de gran importancia, por lo que se requiere una alta disponibilidad e integridad de los datos. Para poder proteger la información de punto a punto y eludir el robo de información es necesario que se analice el sistema y se elimine las vulnerabilidades existentes.

El análisis de seguridad ayudara a identificar las vulnerabilidades a las que la aplicación se encuentra expuesta, conocer sus niveles de riesgo y tomar medidas para protegerlas de tal forma que los datos no puedan ser manipulados de ninguna manera, dándole a los usuarios confianza total al utilizar la aplicación

Para analizar el sistema mencionado se utilizarán metodologías SAST (*Static Aplication Security Testing*) y DAST (*Dinamyc Aplication Security Testing*) con diversas herramientas, las cuales facilitarán la detección de vulnerabilidades que posteriormente serán comprobadas para asegurar su presencia en el sistema.

Se procederá con mitigaciones para las vulnerabilidades encontradas por las herramientas de análisis de seguridad con el apoyo de la información obtenida por fuentes como OWASP (*Open Web Application Security Project*) y CWE (*Common Weakness Enumeration*), así como de más referencias enfocadas a la seguridad de aplicaciones móviles.

DEDICATORIA

Esta tesis está dedicada a:

Toda persona que se integra por primera vez a la seguridad.

Asimismo a mi padre, madre y hermanas quién son mi principal fuente de aliento.

AGRADECIMIENTOS

Me van a faltar páginas para agradecer a las personas que se han involucrado en la realización de este trabajo, sin embargo merecen reconocimiento especial mi madre y mi padre ellos son mis principales pilares de la vida, ya que con su esfuerzo y dedicación me ayudaron a culminar mi carrera universitaria y me dieron el apoyo suficiente para no decaer cuando todo parecía complicado e imposible.

Asimismo, mi profundo agradecimiento a todas las autoridades y personal que hacen la Dirección de Modelado y Simulación, por confiar en mí, abrirme las puertas y permitirme realizar todo el proceso investigativo dentro de su establecimiento.

De igual forma agradezco a mis hermanas, sobrinos, profesores y amigos que siempre me apoyaron y alentaron durante toda mi carrera.

Índice

Índice de tablas

Índice de figuras

CAPÍTULO I

1. INTRODUCCIÓN

1.1. Planteamiento del problema

1.1.1. Descripción del problema

En la actualidad la seguridad de la información se ha vuelto un punto clave de análisis, puesto que la tecnología ha incrementado de manera exponencial a lo largo del tiempo, lo que ha permitido fijar nuevos horizontes dando como resultado, la aparición de nuevas amenazas tecnológicas que pudieran poner en riesgo la integridad de los datos, tanto información privada como también información de una organización, ocasionando altos impactos negativos en la economía, los negocios, seguridad e infraestructura. Por tal motivo, la ciberseguridad se posiciona como un pilar fundamental en la línea de defensa contra los ataques de seguridad cibernética y seguridad de la información.

El presente proyecto de investigación está orientado a realizar un análisis de seguridad a una aplicación móvil de mensajería segura en los sistemas operativos iOS y Android, esto con el fin reforzar la seguridad en la aplicación en cada etapa de su desarrollo para evitar la posible exposición, robo o manipulación de los datos de los usuarios y de la organización.

La aplicación móvil de mensajería instantánea es utilizada por usuarios de una organización donde se envía información de gran importancia, por lo que se requiere una alta disponibilidad e integridad de los datos. Para poder proteger la información de punto a punto y eludir el robo de información es necesario que se analice el sistema y se eliminen las vulnerabilidades existentes.

El análisis de seguridad ayudará a identificar las vulnerabilidades a las que la aplicación se encuentra expuesta, conocer sus niveles de riesgo y tomar medidas para protegerlas, de tal forma que los datos no puedan ser manipulados de alguna manera, dándole a los usuarios confianza total al utilizar la aplicación.

El presente proyecto de investigación cuenta con un capítulo que incluye el marco teórico y la descripción de fundamentos que anteceden al presente, posteriormente se cuenta con capítulos que darán muestra a un ciclo de análisis, comprobación y solución a los problemas de seguridad encontrados con las

diferentes herramientas de análisis, finalmente se tiene un capítulo de conclusiones y recomendaciones.

A continuación se muestra una breve explicación de lo que cada capítulo presentará:

Capítulo I. “Introducción”. La introducción incluye la problemática principal y el ambiente en que esta se desarrolla, así como la hipótesis a comprobar que resuelve dicha problemática.

Capítulo II. “Fundamentación teórica”. Presenta el marco teórico donde se mencionan los conceptos abordados en el tema de investigación, así como también, los antecedentes de investigaciones anteriores sobre ataques mas comunes, análisis de éstos y soluciones teóricas presentadas por fuentes confiables sobre el tema de seguridad.

Capítulo III. “Herramientas de ciberseguridad”. Se presenta una investigación amplia sobre las herramientas necesarias para realizar un escaneo de vulnerabilidades, así como también el proceso de instalación de cada una de ellas.

Capítulo IV. “Análisis de vulnerabilidades”. Describe las herramientas utilizadas para realizar diversos análisis de seguridad a una aplicación móvil, así como la información brindada por fuentes confiables para la detección de vulnerabilidades.

Capítulo V. “Protección de vulnerabilidades”. Se presentarán recomendaciones para las vulnerabilidades que el sistema presenta actualmente que fueron obtenidas en el capítulo IV.

Capítulo VI. “Resultados y Conclusiones”. En este capítulo final se hablará sobre los resultados durante el análisis así como también se describe la forma en que los resultados obtenidos responden a la pregunta de investigación, con una breve conclusión de lo analizado y aplicado en los capítulos anteriores.

1.1.2. Pregunta de Investigación

¿Qué procedimiento se debe seguir para llevar a cabo un análisis de seguridad a una aplicación de mensajería segura, con el fin de minimizar los riesgos de pérdida, daño o alteración de la información contenida?

1.2. Objetivos

1.2.1. Objetivo general

El presente documento tiene por objetivo realizar un análisis de seguridad para la detección, exploración y mitigación de vulnerabilidades de alto riesgo de una aplicación de mensajería tanto para el entorno iOS como Android, obteniendo así una aplicación móvil que tenga un nivel de seguridad de bajo riesgo.

1.2.2. Objetivos específicos

1. Investigar y seleccionar herramientas y material especializado en temas de seguridad que permitan realizar un análisis de vulnerabilidades a una aplicación móvil.
2. Realizar un análisis estático y dinámico de la aplicación con las herramientas necesarias.
3. Investigar las vulnerabilidades arrojadas por los diferentes análisis aplicados, así como también las formas de mitigación de cada una de ellas.
4. Emitir recomendaciones para mitigar posibles ataques.

1.3. Metodología

Para realizar un análisis seguridad a la aplicación móvil de mensajería segura se tiene dos metodologías SAST(*Static Application Security Testing)* y DAST(*Dynamic Application Security Testing)* que son basadas en soluciones AST*(Application Security Testing* o Pruebas de Seguridad de Aplicaciones) de las cuales se adaptan a las nuevas metodologías de desarrollo.[1]

1.3.1. SAST

El uso de la metodología SAST permite detectar defectos en las primeras etapas del desarrollo, estas herramientas también conocidas como analizadores de código, realizan un análisis directo o un enfoque de evaluación de “caja blanca” del código fuente de la aplicación.

La metodología SAST realiza pruebas exhaustivas con los siguientes puntos[2]:

1. Modelado de amenazas de aplicaciones: Primero se estudia cada detalle de la aplicación y se mapea la información crítica.

2. Enumeración del código de la aplicación: El código fuente se enumera en términos de idiomas, dependencias y estructura.
3. Mapeo y funcionalidad del código de la aplicación: El código de la aplicación se mapea sobre la base de algoritmos, funcionalidad, controles de seguridad y módulos.
4. Control de seguridad y casos de prueba: Ayuda a crear todo los métodos de prueba y la reorganización de patrones contra las vulnerabilidades.
5. Categorías de control de seguridad: Autenticación, controles/autorización de acceso, uso indebido de API, recorrido de ruta, fuga de información confidencial, etc.
6. Descubrimiento de puntos de entrada: Este es el aspecto clave para la codificación segura se identifican todos los posibles puntos de entrada al código fuente de la aplicación y luego se evalúa su impacto en la postura de seguridad.
7. Rastreo de clases, funciones y variables: De esta manera se puede observar el impacto de seguridad de cada uno de estos puntos de entrada en detalle.
8. Detección de vulnerabilidades: Estas vulnerabilidades se clasifican en función de su impacto y gravedad de riesgo.
9. Controles de mitigación: Se formula un plan de mitigación y se proporcionan posibles plantillas de codificación segura y fragmentos como parte de un informe.
10. Informe: Todas las observaciones, hallazgos y conjuntos de pruebas se informan en el documento final.

SAST descubre vulnerabilidades altamente complejas durante las primeras etapas de desarrollo de un software, ayudando a resolverlas rápidamente, por lo que no es posible probar la aplicación en un entorno real.

1.3.2. DAST

La metodología DAST también conocidas como “pruebas de caja negra”, prueban las interfaces expuestas de una aplicación en ejecución en busca de vulnerabilidades y fallas, generalmente en aplicación.

La metodología DAST realiza pruebas exhaustivas con los siguientes puntos[2]:

1. Huella de aplicación: Se usan herramientas y métodos para identificar bloques de IP, hosts, dominios, dominio cruzados y dominios secundarios.
2. Descubrimiento de aplicación: En esta fase, se identifican todos los conjuntos de aplicación funcionales y en vivo.
3. Enumeración y creación de perfiles de aplicación: Se ejecutan varias herramientas en cada aplicación de destino y enumeramos la aplicación completa junto con los puntos de entrada y los atributos para cada uno de los recursos que residen en la aplicación.
4. Modelado de amenazas de aplicaciones: Para este módulo se revisa la aplicación con gran detalle y se mapean las funcionalidades críticas.
5. Control de seguridad y casos de prueba: Sobre la base de creación de perfiles y el modelado de subprocesos, se crea un gran conjunto de posibles casos de uso y casos de abuso el cual se asigna a los controles de seguridad estándar de la industria que se enumeran a continuación para generar posibles escenarios de ataque:
    1. OWASP Top 10.
    2. Clasificación de amenazas WASC.
    3. CWE.
    4. OWASP Top 10 Mobile.
6. Evaluación de la vulnerabilidad: Se realiza una evaluación exhaustiva de la vulnerabilidad utilizando inteligencia humana con la ayuda de herramientas semiautomáticas y automáticas de apoyo.
7. Explotación de vulnerabilidades: Las pruebas de penetración y el intento de explotación de las vulnerabilidades descubiertas se realizan para detectar y determinar la gravedad del posible impacto y la posible exposición al riesgo.
8. Estrategias de mitigación: Se construye un plan de mitigación junto con recomendaciones. En este proceso las estrategias se deben implementar para proteger la aplicación.
9. Informe: Sin falsos positivos donde el informe debe incluir:
    1. Descripción de vulnerabilidades descubiertas.
    2. Clasificación de riesgo y lista de posibles amenazas para cada vulnerabilidad descubierta.

3. Impacto de la vulnerabilidad y el perfil de riesgo según los estándares(OWASP,SANS,CWE/CVE).
4. Evidencia de vulnerabilidades(capturas de pantalla, trafico HTTP, parámetro vulnerable, vector de explotación, resultados de herramientas, pasos de reproducción, etc).
5. Evidencia de explotación de vulnerabilidades(si es necesario).
6. Estrategias de mitigación y enfoques de defensa en profundidad para desarrolladores/ administradores.

DAST permite a los desarrolladores detectar los problemas durante la ejecución del código, pero a diferencia de las pruebas SAST no proporciona información sobre las causas subyacentes de las vulnerabilidades.

Siendo así dos metodologías importantes dentro de la rama de ciberseguridad, es esencial abarcar ambas pruebas dentro del análisis para eludir el mayor número de vulnerabilidades dentro de la aplicación en cada etapa de desarrollo.

CAPÍTULO II

2. FUNDAMENTACIÓN TEÓRICA

2.1. Antecedentes

2.1.1. ISO 27001

ISO 27001 es una norma internacional emitida por la Organización Internacional de Normalización (ISO) y describe cómo gestionar la seguridad de la información de una empresa. La revisión más reciente de esta norma fue publicada en 2013 y ahora su nombre completo es ISO/IEC 27001:2013. La primera revisión se publicó en 2005 y fue desarrollada en base a la norma británica BS 7799-2.[3]

ISO 27001 puede ser implementada en cualquier tipo de organización, con o sin fines de lucro, privada o pública, pequeña o grande. Está redactada por los mejores especialistas del mundo en el tema y proporciona una metodología para implementar la gestión de la seguridad de la información en una organización. También permite que una empresa sea certificada; esto significa que una entidad de certificación independiente confirma que la seguridad de la información ha sido implementada en esa organización en cumplimiento con la norma ISO 27001.

El objetivo general de ISO 27001 es proteger la confidencialidad, integridad y disponibilidad de la información en una empresa. Esto lo hace investigando cuales son los potenciales problemas que podrían afectar la información(es decir la evaluación de riesgos) y luego definiendo lo que es necesario hacer para evitar que estos problemas se produzcan(es decir, mitigación o tratamiento del riesgo).

Por lo tanto, la filosofía principal de la norma ISO 27001 se basa en la gestión de riesgos, investigar dónde están los riesgos y luego tratarlos sistemáticamente.

ISO/IEC 27001 se encuentra conformado por las siguientes características:

1. El establecimiento de un Sistema de Gestión de Seguridad de la Información (SGSI) ISO 27001 notoriamente se direcciona en relación a nuevas medidas de organización que se tendrán que tomar en su organización para empezar a resguardar la información de posibles riesgos.

2. Estas medidas se tendrán que diseñar e instalar en una primera fase, estableciendo una política de los objetivos, de los procesos y de los procedimientos y posteriormente medir el rendimiento, revisando y haciendo una supervisión en las oficinas o áreas en donde se han implementado.
3. En la segunda fase se evaluará y se buscará mejorar con corrección o prevención, según sea el caso, de acuerdo al resultado obtenido de auditorías internas del sistema de gestión y de auditorías de revisión.
4. Las ventajas son muchas debido a que suman seguridad y además confianza en cada una de las áreas de su organización. En resumen, dichos beneficios, mencionamos que implementando ISO 27001 logrará disminuir las amenazas de corrupción de la información, logrará implementar una metodología de gestión de la seguridad eficiente, facilitará continuar con las actividades tras haber padecido un ataque en la seguridad, reforzará a su organización ante eventuales peligros de pérdida de información y mejorará su imagen institucional en el mercado.
5. El estándar UNE-ISO/IEC 27001 Seguridad de la Información tiene afinidad con diversos sistemas de gestión que quizás tenga implementado en su organización como el sistema de gestión de la calidad según el estándar ISO 9001 y el sistema de gestión Medioambiental bajo el estándar ISO 14001, sistema de gestión de servicios de tecnologías de la información ISO 20000. etc.

2.1.2. CWE

CWE (*Common Weakness Enumeration*) es un sistema de categorías para las debilidades y vulnerabilidades del software. Como objetivo es comprender las fallas en el software y crear herramientas automatizadas que se puedan utilizar para identificar, corregir y prevenir esas fallas.[4]

CWE Top 25 es una lista demostrativa de los problemas más comunes e impactantes experimentados en años anteriores. Estas debilidades son peligrosas porque a menudo son fáciles de encontrar, explotar, y pueden permitir a los adversos hacerse cargo completamente de un sistema, robar datos o impedir que una aplicación funcione.

A continuación se muestra la Tabla 1: Top 25 de los principales riesgos de CWE 2021.

#	ATAQUE	DESCRIPCIÓN
1	Escritura fuera de límites.	El software escribe datos después del final, o antes del comienzo. Normalmente, esto puede resultar en corrupción de datos, un bloqueo o ejecución de código.
2	Neutralización incorrecta de la entrada durante la generación de páginas web (*Cross-site Scripting*).	El software no neutraliza o neutraliza incorrectamente la entrada controlable por el usuario antes de que se coloque en la salida que se utiliza como una página web que se sirve a otros usuarios. Una vez que se inyecta el script malicioso, el atacante puede realizar una variedad de actividades maliciosas. El atacante podría transferir información privada, como *cookies* que pueden incluir información de sesión, desde la máquina de la víctima al atacante.
3	Lectura fuera de límites.	El software lee los datos después del final, o antes del comienzo. Normalmente, esto puede permitir a los atacantes leer información confidencial desde otras ubicaciones de memoria o causar un bloqueo. Un bloqueo puede ocurrir cuando el código lee una cantidad variable de datos y asume que existe un centinela para detener la operación de lectura, como un NULL en una cadena.
4	Validación de entrada incorrecta.	El producto recibe entrada o datos, pero no valida o válida incorrectamente que la entrada tiene las propiedades que se requieren para procesar los datos de forma segura y correcta. La validación de entrada es una técnica de uso frecuente para comprobar entradas potencialmente peligrosas con el fin de garantizar que las entradas sean seguras para su procesamiento dentro del código o cuando se comunican con otros componentes. Cuando el software no valida la entrada correctamente, un atacante es capaz de crear la entrada en una forma que no es esperada por el resto de la aplicación.

5	Neutralización incorrecta de elementos especiales utilizados en un comando del sistema operativo(Inyección de comandos del sistema operativo).	El software construye todo o parte de un comando del sistema operativo utilizando la entrada influenciada externamente desde un componente aguas arriba, pero no neutraliza ni neutraliza incorrectamente elementos especiales que podrían modificar el comando del sistema operativo previsto cuando se envía a un componente aguas abajo. Esto podría permitir a los atacantes ejecutar comandos inesperados y peligrosos directamente en el sistema operativo. Esta debilidad puede conducir a una vulnerabilidad en entornos en los que el atacante no tiene acceso directo al sistema operativo, como en aplicaciones web.
6	Neutralización incorrecta de elementos especiales utilizados en un comando SQL(Inyección SQL).	El software construye todo o parte de un comando SQL utilizando entradas influenciadas externamente desde un componente ascendente, pero no neutraliza o neutraliza incorrectamente elementos especiales que podrían modificar el comando SQL deseado cuando se envía a un componente descendente. Sin la eliminación o citación suficiente de la sintaxis SQL en las entradas controlables por el usuario, la consulta SQL generada puede hacer que esas entradas se interpreten como SQL en lugar de datos de usuario normales. Esto se puede usar para alterar la lógica de la consulta para evitar los controles de seguridad o para insertar declaraciones adicionales que modifiquen la base de datos de back-end, posiblemente incluyendo la ejecución de comandos del sistema.
7	Usar después de liberar.	Hacer referencia a la memoria después de que se ha liberado puede hacer que un programa se bloquee, use valores inesperados o ejecute código. El uso de memoria previamente liberada puede tener cualquier número de consecuencias adversas, que van desde la corrupción de datos válidos hasta la ejecución de código arbitrario, dependiendo del momento del

		defecto.
8	Limitación incorrecta de un nombre de ruta a un directorio restringido.	El software utiliza entrada externa para construir una ruta de acceso que tiene la intención de identificar un archivo o directorio que se encuentra debajo de un directorio padre restringido, pero el software no neutraliza correctamente elementos especiales dentro del nombre de ruta que pueden hacer que el nombre de ruta se resuelva en una ubicación que está fuera del directorio restringido.
9	Falsificación de solicitudes entre sitios(CSRF).	La aplicación web no verifica, o no puede, verificar suficientemente si una solicitud bien formada, válida y consistente fue proporcionada intencionalmente por el usuario que presentó la solicitud.
10	Carga sin restricciones de archivos con tipo peligroso.	El software permite al atacante cargar o transferir archivos de tipos peligrosos que se pueden procesar automáticamente dentro del entorno del producto.
11	Autenticación faltante para la función crítica.	El software no realiza ninguna autenticación para la funcionalidad que requiere una identidad de usuario demostrable o consume una cantidad significativa de recursos.
12	Desbordamiento entero o envolvente.	El software realiza un cálculo que puede producir un desbordamiento o envolvente de enteros, cuando la lógica asume que el valor resultante siempre será mayor que el valor original. Esto puede introducir otras debilidades cuando el cálculo se utiliza para la gestión de recursos o el control de ejecución.
13	Deserialización de datos no confiables.	La aplicación deserializa datos no confiables sin verificar suficientemente que los datos resultantes serán válidos. A menudo es conveniente serializar objetos para la comunicación o guardarlos para su uso posterior. Sin embargo, los datos o el código deserializados a menudo se pueden modificar sin usar las funciones de acceso

		proporcionadas si no utiliza criptografía para protegerse. Además, cualquier criptografía seguiría siendo seguridad del lado del cliente, lo que es una suposición de seguridad peligrosa.
14	Autenticación incorrecta.	Cuando un actor afirma tener una identidad determinada, el software no prueba o prueba insuficientemente que la afirmación es correcta.
15	Des-referencia del puntero NULL.	Una desreferencia de puntero NULL se produce cuando la aplicación des-referencia un puntero que espera que sea válido, pero es NULL, normalmente causando un bloqueo o salida. Los problemas de desreferencia del puntero NULL pueden ocurrir a través de una serie de defectos, incluidas las condiciones de la carrera y las omisiones de programación simples.
16	Uso de credenciales codificadas.	El software contiene credenciales codificadas, como una contraseña o clave criptográfica, que utiliza para su propia autenticación entrante, comunicación saliente a componentes externos o cifrado de datos internos. Las credenciales codificadas normalmente crean un agujero significativo que permite a un atacante eludir la autenticación que ha sido configurada por el administrador del software. Este agujero podría ser difícil de detectar para el administrador del sistema. Incluso si se detecta, puede ser difícil de solucionar, por lo que el administrador puede verse obligado a deshabilitar el producto por completo.
17	Restricción incorrecta de operaciones dentro de los límites de un búfer de memoria.	El software realiza operaciones en un búfer de memoria, donde puede leer o escribir en una ubicación de memoria que esté fuera del límite previsto del búfer. Ciertos idiomas permiten el direccionamiento directo de ubicaciones de memoria y no se aseguran automáticamente de que estas ubicaciones sean válidas para el búfer de memoria al que

		se hace referencia.
18	Falta la autorización.	El software no realiza una comprobación de autorización cuando un actor intenta acceder a un recurso o realizar una acción. Se tiene que un usuario tenga una identidad determinada, la autorización es el proceso de determinar si ese usuario puede acceder a un recurso determinado, basado en los privilegios del usuario y cualquier permiso u otras especificaciones de control de acceso que se apliquen al recurso.
19	Permisos predeterminados incorrectos.	Durante la instalación, los permisos de archivo instalado se establecen para permitir que cualquier persona modifique esos archivos.
20	Exposición de información confidencial a un actor no autorizado.	El producto expone información confidencial a un actor que no está explícitamente autorizado a tener acceso a esa información. Algunos tipos de información confidencial incluyen: 1. Información confidencial. 2. Estado y entorno del sistema operativo y paquetes instalados. 3. Estados de red.
21	Credenciales insuficientemente protegidas.	El producto transmite o almacena credenciales de autenticación, pero utiliza un método inseguro que es susceptible a la interceptación y/o recuperación no autorizadas.
22	Asignación de permisos incorrecta para recursos críticos.	El producto especifica permisos para un recurso crítico para la seguridad de una manera que permite que ese recurso sea leído o modificado por actores no deseados. Cuando se otorga a un recurso una configuración de permisos que proporciona acceso a una gama más amplia de actores de lo requerido, podría conducir a la exposición de información confidencial o a la modificación de ese recurso por partes no deseadas. Esto es especialmente peligroso cuando el recurso está relacionado con la

		configuración del programa, la ejecución o los datos confidenciales del usuario.
23	Restricción incorrecta de la referencia de entidad externa XML.	El software procesa un documento XML que puede contener entidades XML con URI que se resuelven en documentos fuera de la esfera de control prevista, haciendo que el producto incruste documentos incorrectos en su salida. Los documentos XML contienen opcionalmente una definición de tipo de documento (DTD), que, entre otras características, permite la definición de entidades XML. Es posible definir una entidad proporcionando una cadena de sustitución en forma de URI. El analizador XML puede acceder al contenido de este URI e incrustarlo de nuevo en el documento XML para su posterior procesamiento.
24	Falsificación de solicitudes del lado del servidor(SSRF).	El servidor web recibe una URL o una solicitud similar de un componente original y recupera el contenido de esta URL, pero no garantiza suficientemente que la solicitud se envíe al destino esperado. Al proporcionar URL a hosts o puertos inesperados, los atacantes pueden hacer parecer que el servidor está enviando la solicitud, posiblemente omitiendo controles de acceso como *firewalls* que impiden que los atacantes accedan directamente a las URL. El servidor se puede utilizar como proxy para realizar análisis de puertos de hosts en redes internas, utilizar otras URL como que pueden acceder a documentos en el sistema (usando file:///), o utilizar otros protocolos como gopher:// o tftp:///, que pueden proporcionar un mayor control sobre el contenido de las solicitudes.
25	Neutralización inadecuada de elementos especiales utilizados en un comando(Inyección de comandos).	El software construye todo o parte de un comando utilizando una entrada influenciada externamente desde un componente ascendente, pero no neutraliza o neutraliza incorrectamente elementos especiales que podrían modificar el

		comando previsto cuando se envía a un componente descendente. Las inyecciones suelen ocurrir cuando: 1. Los datos entran a la aplicación desde una fuente no confiable. 2. Los datos son parte de una cadena que es ejecutada como un comando por la aplicación. 3. Al ejecutar el comando, la aplicación le da privilegios a un atacante.

Tabla 1: Top 25 de los principales riesgos de CWE 2021

2.1.3. OWASP

OWASP (*Open Web Application Security Project*) es una fundación sin fines de lucro enfocada en mejorar la seguridad del software. Opera como una comunidad abierta dedicada a permitir que las organizaciones conciban, desarrollen, adquieran, operen y mantengan aplicaciones en las que se pueda confiar, asegurando el éxito a largo plazo[5]. OWASP es un recurso centralizado destinado a brindar a los desarrolladores y equipos de seguridad los recursos que necesitan para crear y mantener aplicaciones seguras, con el objetivo de clasificar los riesgos de seguridad y proporcionar controles de desarrollo para reducir su impacto o probabilidad de explotación.

OWASP se mantiene actualizado sobre los ataques más recurrentes en los sistemas web y móvil cuenta con una lista en donde describe el TOP 10 de las vulnerabilidades más sobresalientes sobre un análisis previo de cientos de organizaciones y miles de aplicaciones, estas guías son principalmente para los desarrolladores creen aplicaciones seguras e incorporen las mejores prácticas de codificación.

A continuación se observa la Tabla 2: Top 10 de los principales riesgos para dispositivos móviles por OWASP 2016.[6]

#	ATAQUE	DESCRIPCIÓN
M1	Uso inadecuado de la plataforma.	Mal uso o falta de uso de las pautas básicas de desarrollo de la plataforma, las funciones de seguridad y las convenciones comunes. Esto podría ser almacenamiento de claves, permisos de la plataforma, uso deficiente de los controles de seguridad que forman parte

		de la plataforma.
M2	Almacenamiento de datos inseguro.	Adversario que ha obtenido un dispositivo móvil perdido/robado; *malware* u otra aplicación reempaquetada que actúa en nombre del adversario y que se ejecuta en el dispositivo móvil.
M3	Comunicación insegura.	Cuando se diseña una aplicación móvil, los datos se intercambian comúnmente de forma cliente-servidor. La transmisión de datos generalmente se realiza a través de un operador de telecomunicaciones y/o por internet. Es posible interceptar datos como un adversario sentado en la red de área local de usuarios a través de una red Wi-Fi, accediendo a la red a través de enrutadores, torres de telefonía móvil, servidores proxy o explotando la aplicación infectada a través de un *malware.*
M4	Autenticación insegura.	Ocurre cuando un dispositivo no reconoce al usuario correctamente y permite que un adversario inicie sesión en la aplicación con las credenciales predeterminadas. Esto suele ocurrir cuando un atacante falsifica o pasa por alto los protocolos de autenticación, que faltan o están mal implementados, e interactúa directamente con el servidor utilizando malware que se encuentra en el dispositivo móvil o redes de bots, por lo que no establece comunicación directa con la aplicación.
M5	Criptografía insuficiente.	Los datos de las aplicaciones móviles se vuelven vulnerables debido a procesos débiles de cifrado/ descifrado. Los piratas informáticos pueden obtener acceso físico al dispositivo móvil, espiar el tráfico de la red o utilizar aplicaciones maliciosas en el dispositivo para acceder a datos cifrados. Utilizando fallas en el proceso de cifrado, su objetivo es descifrar los datos a su forma original para robarlos o cifrarlos mediante un proceso contradictorio y, por lo tanto, inutilizarlos para el usuario legítimo.

M6	Autorización insegura.	Implica que el adversario se aproveche de las vulnerabilidades en el proceso de autorización para iniciar sesión como un usuario legítimo, a diferencia de la autenticación insegura, en la que el adversario intenta eludir el proceso de autenticación iniciando sesión como un usuario anónimo.
M7	Calidad de código fuente.	Los agentes de amenazas incluyen entidades que pueden pasar entradas que no son de confianza a llamadas de método realizadas dentro del código móvil. Estos tipos de problemas no son necesariamente problemas de seguridad en sí mismos, sino que conducen a vulnerabilidades de seguridad.
M8	Manipulación de código.	Un atacante aprovechará la modificación de código a través de formas maliciosas de las aplicaciones alojadas en tiendas de aplicaciones de terceros. El atacante también puede engañar al usuario para que instale la aplicación mediante ataques de *phishing*.
M9	Ingeniería inversa.	Es una ocurrencia comúnmente explotable. Los atacantes tienden a utilizar herramientas de inspección binarias externas comúnmente disponibles.
M10	Funcionalidad extraña.	Un atacante busca comprender la funcionalidad extraña dentro de una aplicación móvil para descubrir una funcionalidad oculta en los sistemas *backend*. El atacante normalmente aprovecha la funcionalidad extraña directamente desde sus propios sistemas sin la participación de los usuarios finales.

Tabla 2: Top 10 de los principales riesgos para dispositivos móviles por OWASP 2016.

NowSecure una compañía experta en software y servicios de pruebas de seguridad de aplicaciones móviles, publicó un artículo sobre las vulnerabilidades de las aplicaciones donde comenta que el 85% de las aplicaciones móviles violan al

menos una o más de las 10 principales áreas de riesgo móviles del Top 10 (OWASP), según las pruebas de referencia de *NowSecure*.

NowSecure descubrió que más de la mitad de las aplicaciones probadas tienen fallas de seguridad que comprometen su capacidad para proteger los datos en tránsito y en reposo. Casi un tercio de las aplicaciones sufrieron problemas de codificación. Las aplicaciones de Android, en particular, presentaban problemas de codificación que podrían exponerlas a ingeniería inversa u otras vulnerabilidades. [7]

A continuación se observa la Figura 1: Top 10 del rango de violación para dispositivos móviles por NowSecure.

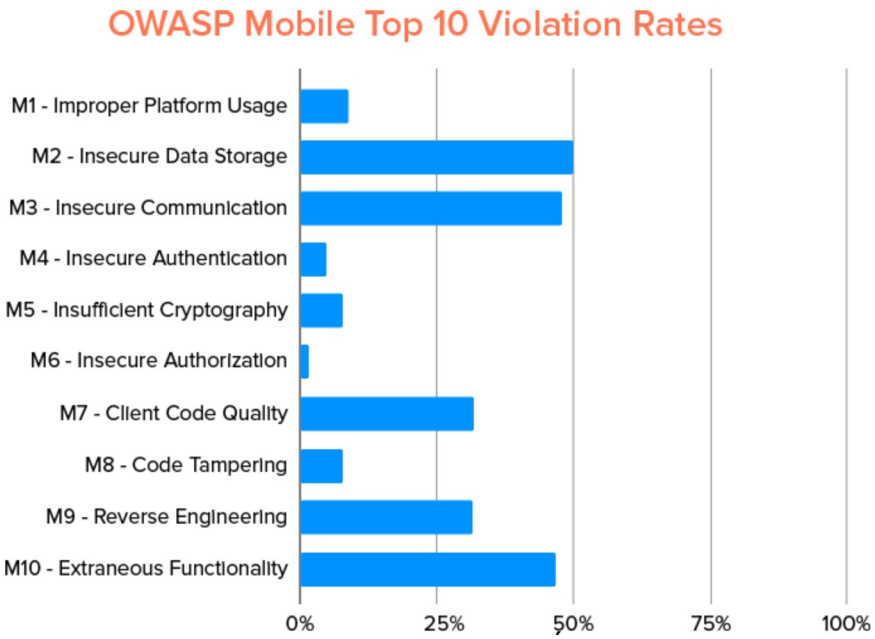

Figura 1: Top 10 del rango de violación para dispositivos móviles por *NowSecure*.

OWASP ha tenido gran impacto sobre el desarrollo de aplicaciones ya que en un mundo real se han presentado vulnerabilidades en aplicaciones más de lo que se podría pensar, a continuación se presenta una serie de ejemplos con errores comunes en aplicaciones:

1. Uso inadecuado de la plataforma(M1 Aplicaciones Citrix Worx): Se descubrió que era posible eludir Touch ID para estas aplicaciones mediante un reinicio de teléfono posteriormente abrir una de las aplicaciones de Citrix Worx iniciar la autenticación pero cancelar el Touch ID, finalmente cerrar la

aplicación y abrirla de nuevo[4]. El problema parecía ser que se recuperaba al pasar Touch ID se almacena incorrectamente. Por lo tanto, la aplicación asume que el usuario estaba correctamente autenticado cuando se canceló el proceso de autenticación y se reinicio la aplicación.[8]

2. Almacenamiento de datos inseguro(M2 Aplicación Tinder): Tinder una aplicación de citas y encuentros, introdujo una función que consiste en mostrar la ubicación exacta de las personas que iniciaban sesión cerca de ti, por lo que fue un problema ya que esta ubicación se enviaba al dispositivo. Una de sus primeras soluciones fue proporcionar sólo la distancia, esta solución obtuvo más vulnerabilidades ya que fue posible falsificar la ubicación y usar la triangulación. Otra de sus posibles soluciones fue enviar estos datos sin precisión. Para mostrar lo mala que fue la vulnerabilidad descubierta, los usuarios crearon una aplicación para mostrar a los usuarios que Tinder la ubicación exacta de las demás personas.[9]

3. Comunicación insegura(M3 Relojes Inteligentes Misafes): Misafes es una empresa destinada a fabricar dispositivos de vigilancia destinados a rastrear a los niños, una de las vulnerabilidades encontradas que llamó la atención a los atacantes son los relojes inteligentes que esta empresa desarrolló, ya que se encontró que la comunicación de estos dispositivos no se encontraba cifrada y no estaba correctamente autenticada. Por lo que los atacantes podrían recuperar coordenadas exactas del GPS en tiempo real de los relojes infantiles, así como también realizar llamadas al infante a través del reloj, crear una llamada de audio unidireccional encubierta espiando al niño, también se logró enviar mensajes de audio y así evitar la lista de llamadas aprobada, y lo mas importante con esta vulnerabilidad es que se logra obtener información personal del niño, como nombre, fecha de nacimiento, sexo, altura, peso incluso hasta recuperar su foto.[10]

4. Autenticación insegura(M4 Grab Android app): Se encontró un bypass de autenticación de dos factores en el endpoint, utilizado por Grab Android App, un atacante fue capaz de eludir 2FA forzando código bruto de 4 dígitos. No había límite de cuántas veces se podía ingresar el código de 4 dígitos enviado por lo que le fue posible obtener acceso a la cuenta con información confidencial como viajes, métodos de pago, pedidos, etc.[11]

5. Criptografía Insuficiente(M5 Aplicación Ola): Appknox analizó la aplicación Ola y descubrió grandes debilidades en la forma en que se utilizaban las claves criptográficas. Descubrieron que la clave criptográfica utilizada era

“PRODKEYPRODKEY12”. La misma clave también se utilizó para cifrar contraseñas, lo que significa que los usuarios de otras cuentas en las que estaban utilizando contraseñas también podrían haber estado en riesgo. Los investigadores pudieron interceptar las solicitudes entre la aplicación y el servidor, solicitar dinero falso y recibir el dinero.[12]

6. Autorización insegura(M6 Viper Smart Start System): Su aplicación móvil desarrollada para este sistema de alarmas contaba con una vulnerabilidad ya que se descubrió que el inicio inteligente Viper no autorizó correctamente a los usuarios. Después de iniciar sesión en el servidor, fue posible cambiar el número de identificación del automóvil y obtener acceso, entre otras cosas, a la ubicación de los automóviles. También fue posible cambiar los datos sobre el coche y abrirlo de forma remota.[13]

7. Calidad del código de cliente(M7 Aplicación WhatsApp): WhatsApp una aplicación de mensajería en donde se descubrió que era posible crear un desbordamiento de búfer enviando series de paquetes especialmente diseñados a WhatsApp al hacer una llamada. Para que esto funcione, la llamada no necesita ser respondida y el adversario puede ejecutar código arbitrario. Se descubrió que la vulnerabilidad se utilizaba para instalar spyware en el teléfono. Este servicio fue vendido por la empresa israelí NSO Group.[14]

8. Manipulación de código(M8 Pokémon Go): Pokémon Go una aplicación de entretenimiento se encontró expuesta a la técnica de ingeniería inversa, lo que fue aplicada y así se logró alimentar datos de geolocalización incorrectos y tiempo para encontrar pokémon raros y hacer que los huevos eclosionen más rápido. Se creó un sitio web que mostraba la ubicación de cada pokémon en un mapa, lo que cambió bastante la dinámica del juego. [15]

9. Funcionalidad extraña(M10 Transferencia de archivos Wifi): La aplicación de transferencia de archivos wifi abre el puerto en el dispositivo Android para permitir conexiones desde el ordenador. Se halló que no existía autenticación como contraseña, cualquiera podría conectarse al dispositivo y tener acceso completo.[16]

2.2. Marco teórico

La ciberseguridad abarca una amplia gama de conceptos y para entrar en contexto se describen algunos términos que se presentarán constantemente en esta tesis:

Vulnerabilidad de software: Defecto de diseño o error de implementación de la que se aprovechan los atacantes. Son puntos débiles que, al ser explotados por una amenaza, afectan la confidencialidad, disponibilidad e integridad de la información de un individuo o empresa .[17]

Seguridad: La seguridad en la informática se puede definir como cualquier medida que impida la ejecución de operaciones no autorizadas sobre un sistema o red informática, cuyos efectos conllevan daños sobre la información, comprometer su confidencialidad, autenticidad o integridad, disminuir el rendimiento de los equipos o bloquear el acceso de usuarios autorizados al sistema [18]. Cabe mencionar que ningún método ni herramienta será capaz de encontrar todas las vulnerabilidades en un sistema y mantenerlo 100% seguro, sin embargo la seguridad trata de mantener protegida la mayor cantidad de debilidades dentro de él.

Hacker: La definición puede variar dependiendo del contexto, define a un desarrollador experto y capaz de resolver problemas de tecnología y seguridad en diferentes contextos.

Ataque: Técnica que se utiliza para realizar un asalto a la seguridad de un sistema derivado de una amenaza, con el fin de eludir y violar las políticas de seguridad de un sistema para explotar sus vulnerabilidades y comprometer su información.[19]

Framework: Conocido como entorno de trabajo definido por un conjunto estandarizado de conceptos, prácticas y criterios, generando un esquema de desarrollo.

Root: *Rooting* es una técnica que permite a los usuarios de dispositivos acceder a un control privilegiado del equipo (lo que se conoce como acceso root, que da nombre a la técnica). Como Android utiliza un kernel Linux (esto es, el elemento software que constituye una fundamental del sistema operativo), el *rooting* proporciona un acceso a los permisos de administrador, que en este entorno se conocen con el nombre de superusuario.[20]

Amenaza: Se puede definir como amenaza a todo elemento o acción capaz de atentar contra la seguridad de la información. Las amenazas surgen a partir de la existencia de vulnerabilidades, es decir que una amenaza sólo puede existir si existe una vulnerabilidad que pueda ser aprovechada, e independientemente de

que se comprometa o no la seguridad de un sistema de información.[21] Las amenazas pueden clasificarse en dos tipos:

1. Intencionales: En caso de que deliberadamente se intente producir un daño (por ejemplo el robo de información aplicando la técnica de *trashing*, la propagación de código malicioso y las técnicas de ingeniería social).
2. No intencionales: En donde se producen acciones u omisiones de acciones que si bien no buscan explotar una vulnerabilidad, ponen en riesgo los activos de información y pueden producir un daño (por ejemplo las amenazas relacionadas con fenómenos naturales).

Jailbreak: Se trata del proceso con el que conseguimos eliminar las limitaciones impuestas por Apple en un dispositivo con iOS. Una vez "liberado", podemos, por ejemplo, instalar aplicaciones de terceros que no estén en la AppStore.

Aplicaciones móviles: Es un tipo de aplicación diseñada para ejecutarse en un dispositivo móvil, que puede ser un teléfono inteligente o una tableta. Se alejan de los sistemas de software integrados, proporciona una funcionalidad aislada y limitada. Por ejemplo, puede ser un juego, una calculadora o un navegador web móvil.

La clave de un software seguro, es el proceso de desarrollo utilizado. En el proceso, es donde se produce el producto que pueda resistir o sostenerse ataques ya anticipados, y recuperarse rápidamente y mitigar el daño causado por los ataques que no pueden ser eliminados o resistidos. Muchos de los defectos relacionados con la seguridad en software se pueden evitar si los desarrolladores estuvieran mejor equipados para reconocer las implicaciones de su diseño y de las posibilidades de implementación.

Para comprender más sobre seguridad de la información que un desarrollador como buena práctica debe tener en cuenta es la triada de CIA[22]:

1. Confidencialidad: La confidencialidad es aproximadamente equivalente a la privacidad. Las medidas emprendidas para garantizar la confidencialidad están diseñadas para evitar que la información confidencial llegue a las personas equivocadas, al tiempo que se garantiza que las personas adecuadas puedan obtenerla: el acceso debe estar restringido a aquellos autorizados para ver los datos en cuestión.

2. Integridad: La integridad implica mantener la consistencia, precisión y confiabilidad de los datos durante todo su ciclo de vida. Los datos no deben modificarse en tránsito, y deben tomarse medidas para garantizar que personas no autorizadas no puedan alterar los datos (por ejemplo, en una violación de la confidencialidad). Estas medidas incluyen permisos de archivos y controles de acceso de usuarios.
3. Disponibilidad: La disponibilidad se garantiza mejor manteniendo rigurosamente todo el hardware, realizando reparaciones de hardware de inmediato cuando sea necesario y manteniendo un entorno de sistema operativo que funcione correctamente y libre de conflictos de software. También es importante mantenerse al día con todas las actualizaciones necesarias del sistema.

Es importante conocer la triada de CIA ya que un hacker buscará defender o atacar alguna de estas tres áreas.

White Box Testing: Es un método de prueba de software donde las pruebas se derivan de la estructura del objeto probado.[23]

Black Box Testing: Es un método de prueba de software en el que las funcionalidades de las aplicaciones de software se prueban sin tener conocimiento de la estructura del código interno, los detalles de implementación y las rutas internas.[24]

CAPÍTULO III

3. HERRAMIENTAS DE CIBERSEGURIDAD

El uso de herramientas destinadas a escáneres de vulnerabilidades reduce los problemas que se pueden detectar durante la fase de desarrollo de software, esta es una fase importante dentro del ciclo de vida del software para emplear tales herramientas, ya que proporciona retroalimentación inmediata al desarrollador sobre los problemas que podrían estar introduciendo en el código durante el desarrollo.

A continuación se muestra la Tabla 3: Herramientas especializadas en el tema de seguridad. Estas herramientas tienen como objetivo realizar pruebas SAST y DAST.

#	HERRAMIENTA	DESCRIPCIÓN	TIPO
1	StaCoAn	Es una gran herramienta que ayuda a realizar análisis de código estático para aplicaciones móviles. Esta herramienta multiplataforma analiza las líneas escritas en un código que contiene claves de API, URL de API, credenciales codificadas, claves de descifrado, errores de codificación, etc. En la actualidad, StaCoAn solo admite archivos APK y los archivos IPA se encuentran en proceso.	Libre
2	MobSF	Es una aplicación para análisis de *malware,* pruebas de penetración, evaluación de seguridad, etc. Puede realizar ambos tipos de análisis: estático y dinámico. Admite binarios de aplicaciones móviles como IPA, APK y APPX, además de códigos fuente	Libre

		comprimidos. Con su analizador dinámico, puede ejecutar evaluaciones de seguridad en tiempo de ejecución, así como pruebas instrumentadas.	
3	Runtime Mobile Security	Esta herramienta ayuda a manipular aplicaciones iOS y Android en tiempo de ejecución, es una multi-herramienta donde se pueden vincular tareas en poco tiempo.	Libre
4	QARK	Esta herramienta está diseñada para buscar varias vulnerabilidades de aplicaciones de Android relacionadas con la seguridad, ya sea en el código fuente o en los APK empaquetados.	Libre
5	MARA	Es una herramienta que reúne las herramientas de análisis e ingeniería inversa de aplicaciones móviles de uso común, para ayudar a probar aplicaciones móviles contra las amenazas de seguridad móvil de OWASP.	Libre
6	Burp Suite	Es una plataforma integrada para realizar pruebas de seguridad. Sus diversas herramientas trabajan juntas a la perfección para apoyar todo el proceso de prueba, desde el mapeo inicial y el análisis de la superficie de ataque de una aplicación, hasta la búsqueda y explotación de vulnerabilidades de seguridad.	Comercial

Tabla 3: Herramientas especializadas en el tema de seguridad

Se presentaron herramientas para la detección de vulnerabilidades especializadas en aplicaciones móviles, sin embargo para la selección de cada una de ellas se requiere que cumplan con la especificación de software libre y/o código abierto. Durante el análisis se requiere evaluar las aplicaciones en los diferentes sistemas operativos como iOS y Android, como se observa en la Tabla 4: Herramientas de seguridad.

HERRAMIENTA	ANÁLISIS ESTÁTICO ANDROID	ANÁLISIS ESTÁTICO iOS	ANÁLISIS DINÁMICO ANDROID	ANÁLISIS DINÁMICO iOS
StaCoAn	✓			
MobSF	✓	✓	✓	
QARK	✓			
RMS			✓	✓
MARA	✓			
Burp Suite			✓	✓

Tabla 4: Herramientas de seguridad

3.1. INSTALACIÓN DE HERRAMIENTAS PARA LA DETECCIÓN DE VULNERABILIDADES

La elección de cada una de las herramientas para realizar un análisis de seguridad se basó en su característica principal que es la compatibilidad con los diferentes sistemas operativos como Android y iOS. Las herramientas seleccionadas son:

1. MobSF.
2. MARA.
3. Runtime Mobile Security.
4. Burp Suite.

Algunas herramientas mencionadas en la Tabla 3: Herramientas

especializadas en el tema de seguridad, no fueron utilizadas por incompatibilidad y versiones obsoletas.

3.1.1. MARA

MARA en su versión 0.2.2 beta es una herramienta de código libre que reúne el análisis de aplicaciones móviles y las herramientas de ingeniería inversa de uso común para ayudar a probar las aplicaciones móviles contra las amenazas de seguridad móvil de OWASP. Diseñada para el análisis estático.[25]

Algunas de sus características son:

1. Realiza ingeniería inversa de un APK, desmontando el código para obtenerlo en código fuente e incluso en código de bytes.
2. Desofuscación de APK.
3. Disponible para Android.
4. Soporta archivos, apk, jar, dex, class.
5. Tiene complementos capaces de escanear vulnerabilidades basado en OWASP Top Mobile Top 10 y la lista de verificación de aplicaciones móviles de OWASP.

Para el uso de la herramienta MARA se necesita una lista de requerimientos los cuales son:

1. Instalar python3-dev
2. Instalar python3+
3. Instalar git
4. Instalar dialog

Para su descarga e instalación se necesita obtener el archivo, desde:

1. La página principal https://github.com/xtiankisutsa/MARA_Framework

La herramienta MARA no cuenta como tal con un proceso de instalación, solo basta con clonar el repositorio y utilizar el ejecutable de la aplicación. Los pasos a seguir para la ejecución de MARA desde terminal son:

1. Iniciar la terminal e ir al directorio donde queremos alojar nuestro archivo como se observa en Figura 2: Comando para situarse en una ruta de destino.

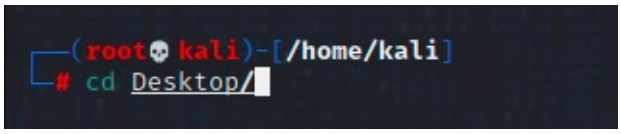

Figura 2: Comando para situarse en una ruta de destino.

2. Clonar desde el repositorio de origen como se observa en Figura 3: Comando para clonar un repositorio con el comando “git clone –-recursive”.

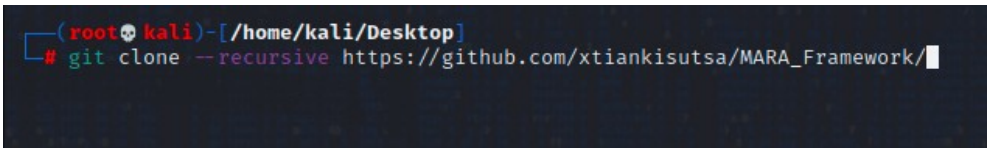

Figura 3: Comando para clonar un repositorio

3. Ir al directorio creado al clonar el repositorio como se observa en la Figura 4: Comando para situarse en la carpeta generada

Figura 4: Comando para situarse en la carpeta generada

4. Con el comando “ls” muestra todos los archivos que contiene el directorio MARA_Framework como se observa en Figura 5: Comando "ls" para ver contenido de un directorio.

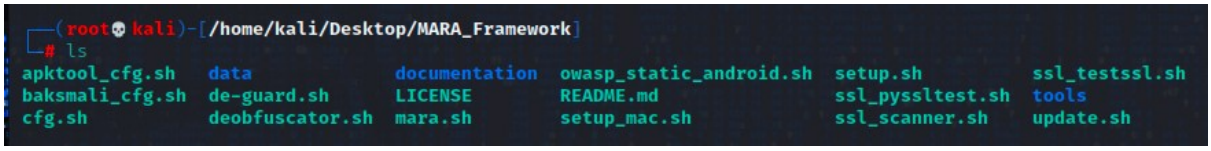

Figura 5: Comando "ls" para ver contenido de un directorio

5. Con el comando “--help” muestra todas las opciones que contiene esta herramienta como se observa en la Figura 6: Comando --help

Figura 6: Comando --help

6. Ejecutar la aplicación a través del archivo “mara.sh” ubicado dentro de la carpeta generada con el comando “./mara.sh [options] <ruta del archivo a analizar>” como se observa en la Figura 7: Comando para ejecutar MARA

Figura 7: Comando para ejecutar MARA

3.1.2. Mobile Security Framework (MobSF)

MobSF en su versión 3.2.4 versión beta es una aplicación automatizada para pruebas móviles (Android / iOS / Windows), realizando prueba de lápiz, análisis de malware y marco de evaluación de seguridad. Apto para realizar análisis estáticos y dinámicos.[26]

Algunas características son:

1. MobSF admite binarios de aplicaciones móviles (APK, XAPK, IPA y APPX).
2. Multiplataforma, puede ser instalado en Windows, Linux/Mac.

3. Tiene complementos capaces de escanear vulnerabilidades basado en OWASP Top Mobile Top 10 y la lista de verificación de aplicaciones móviles de OWASP.

Dicha herramienta se divide como tal en dos partes análisis estático y análisis dinámico donde ambos necesitan una lista de requerimientos los cuales son:

1. Análisis estático:
    1. Mac.
        1. Instalar git.
        2. Instalar python 3.8-3.9.
        3. Después de instalar Python 3.8+, vaya “/Applications/Python3.8/” y ejecútelo “Update Shell Profile.command” primero y luego “install Certificates.command”.
        4. Instalar jdk 8+ o superiores.
        5. Instalar herramientas de línea de comando “xcode-select –install”.
        6. Descargue e instale wkhtmltopdf.
    2. Linux
        1. Instalar git.
        2. Instalar python 3.8-3.9.
        3. Instalar openjk8+.
        4. Instalar las dependencias de python3:
            1. “sudo apt install python3-dev python3-venv python3-pip build-essential libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev”
        5. Descargue e instale wkhtmltopdf.
    3. Windows
        1. Instalar git.
        2. Instalar python 3.8-3.9.
        3. Instalar openjk8+.
        4. Instale las herramientas de compilación de Microsoft Visual C ++.

5. Instalar OpenSSL (no ligero).
6. Descargue e instale wkhtmltopdf agregue la carpeta a una variable de entorno.

2. Análisis dinámico:
    1. Instale un emulador de dispositivos como Genymotion o Genymotion Cloud VM o Android Studio Emulator.

Para su descarga e instalación se necesita obtener el archivo, desde:

1. La página principal https://github.com/MobSF/Mobile-Security-Framework-MobSF
2. Iniciar la terminal e ir al directorio donde queremos alojar nuestro archivo como se observa en la Figura 2: Comando para situarse en una ruta de destino.
3. Clonar desde el repositorio de origen como se observa en la Figura 8: Clonar repositorio MobSF con el comando “git clone”.

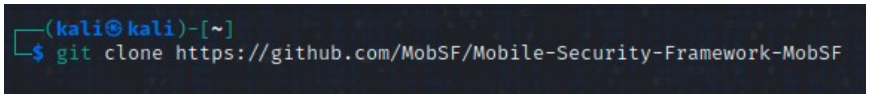

Figura 8: Clonar repositorio MobSF

4. Ir al directorio creado al clonar el repositorio como se observa en la Figura 9: Comando para situarse en la carpeta generada MobSF

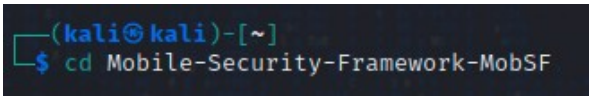

Figura 9: Comando para situarse en la carpeta generada MobSF

5. Ejecutar el comando “./setup.sh” para la configuración de la herramienta como se observa en la Figura 10: Ejecutar la configuración de MobSF

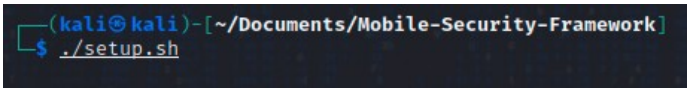

Figura 10: Ejecutar la configuración de MobSF

6. Ejecutar el comando "./run.sh 127.0.0.0:8000". MobSF escuchará "0.0.0.0:8000" si usa el script de ejecución sin argumentos. Como se observa en la Figura 11: Comando para ejecutar MobSF

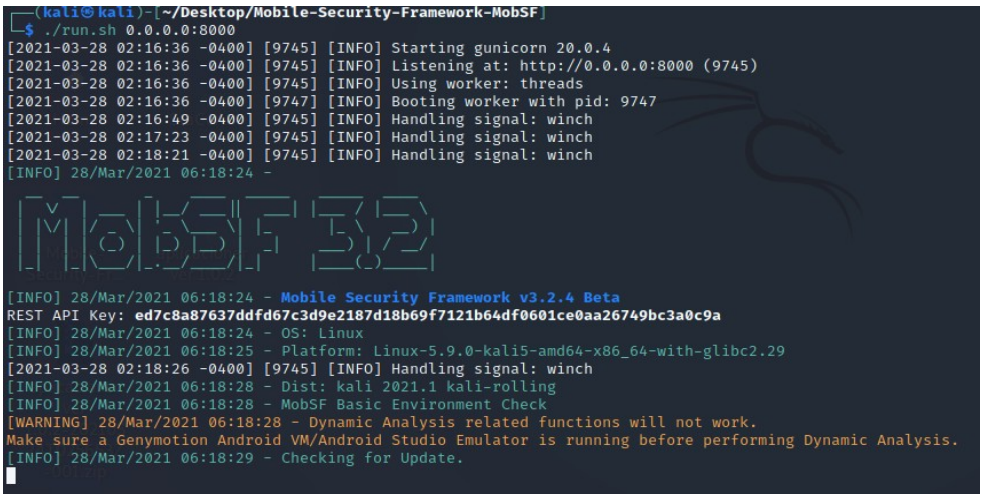

Figura 11: Comando para ejecutar MobSF

7. Abrir en su navegador web, navegue hasta "http://localhost:8000/" para acceder a la interfaz web de MobSF. Finalmente se abrirá una página como se observa en la Figura 12: Ejecución MobSF desde el navegador

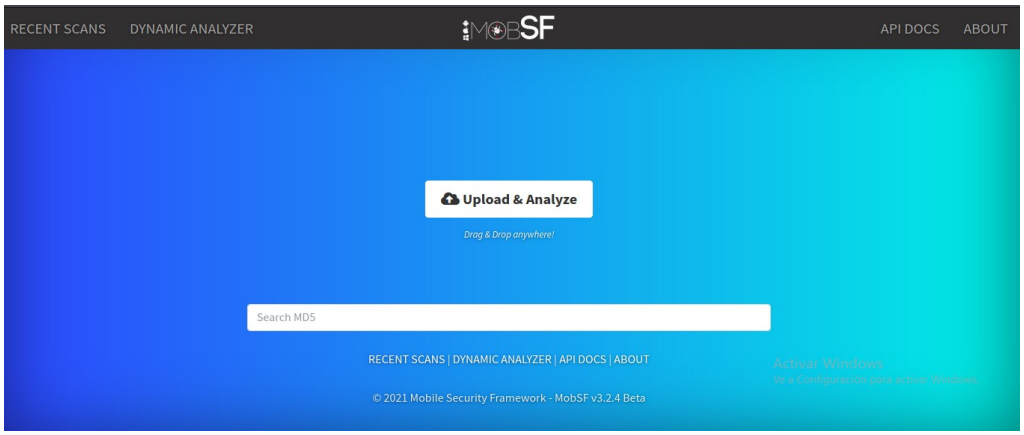

Figura 12: Ejecución MobSF desde el navegador

Además del proceso de instalación de la herramienta MobSF se debe tener una configuración extra para realizar un análisis dinámico. Donde se necesita un emulador de dispositivos, para el uso de la herramienta MobSF se utilizará Genymotion Android.

Para realizar la instalación de Genymotion se necesita algunos requerimientos los cuales son:

1. 400 MB de espacio libre en el disco.
2. 4GB RAM.
3. VirtualBox 6.1.16

Para descargar el emulador es necesario crear una cuenta gratis en el sitio web https://www-v1.genymotion.com/account/create/, se confirma la dirección de correo e iniciamos sesión como se observa en la Figura 13: Iniciar sesión Genymotion

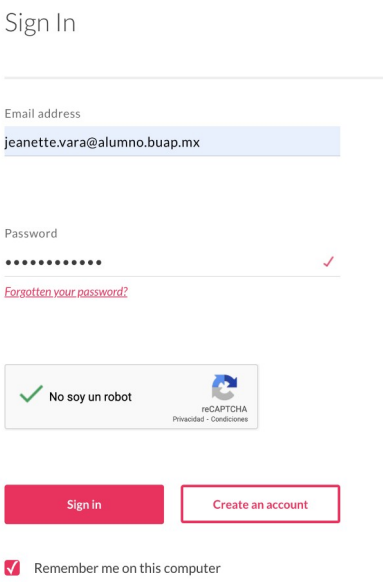

Figura 13: Iniciar sesión Genymotion

Se debe descargar “Genymotion Personal Edition”, posteriormente se dirigirá a la página de descargas donde se elige el sistema operativo deseado, Genymotion tratará de elegir el sistema operativo por defecto y lo mostrará como primera opción.

Para la configuración completa de MobSF de se deben realizar los siguientes pasos:

1. Crear un dispositivo virtual Android dentro de Genymotion. Con el botón “+” alojado en la parte superior derecha mostrará todos los dispositivos

disponibles, como se observa en Figura 14: Creación de dispositivo virtual Genymotion

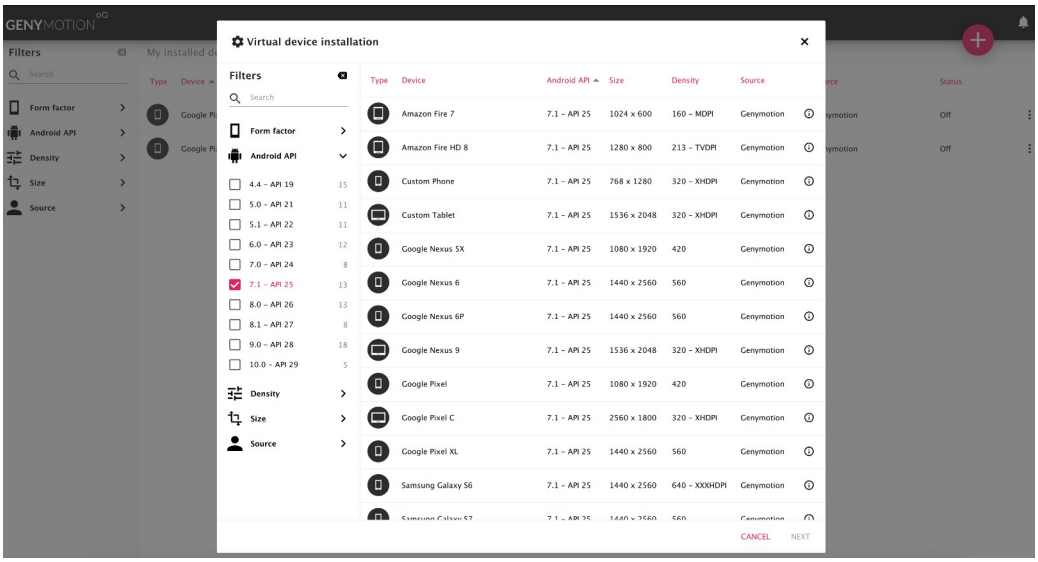

Figura 14: Creación de dispositivo virtual Genymotion

MobSF admite dispositivos Android 7.0 o superior. En la parte superior izquierda de la venta se puede escoger estas características como se observa en Figura 14: Creación de dispositivo virtual Genymotion.

De igual manera es posible modificar sus características de hardware y software. Como se observa en Figura 15: Configuración de dispositivo virtual en Genymotion

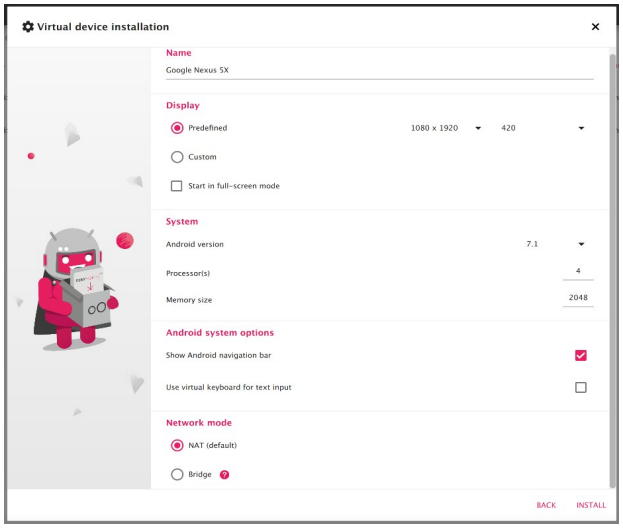

Figura 15: Configuración de dispositivo virtual en Genymotion

2. Ejecutar el dispositivo virtual creado, antes de iniciar MobSF. Como se observa en Figura 16: Ejecutar dispositivo virtual Genymotion

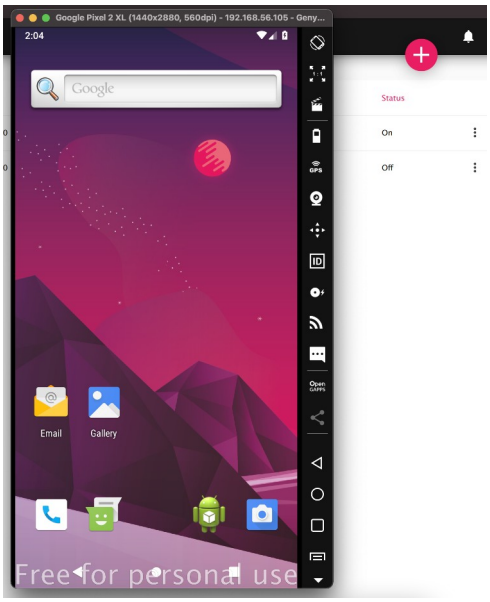

Figura 16: Ejecutar dispositivo virtual Genymotion

3. Ejecutar el comando "./run.sh 127.0.0.0:8000". MobSF escuchará "0.0.0.0:8000" si usa el script de ejecución sin argumentos. Como se observa en Figura 11: Comando para ejecutar MobSF
4. Abrir en su navegador web, navegue hasta "http://localhost:8000/" para acceder a la interfaz web de MobSF. Finalmente se abrirá una página como se observa en la Figura 12: Ejecución MobSF desde el navegador.
5. Dentro de la página de lado superior izquierdo muestra el botón "DYNAMIC ANALYZER". Ingresar al apartado.
6. Haga clic en el botón "MobSFy Android Runtime" y el entorno de ejecución de Android. Como se observa en Figura 17: Configuración de MobSFy

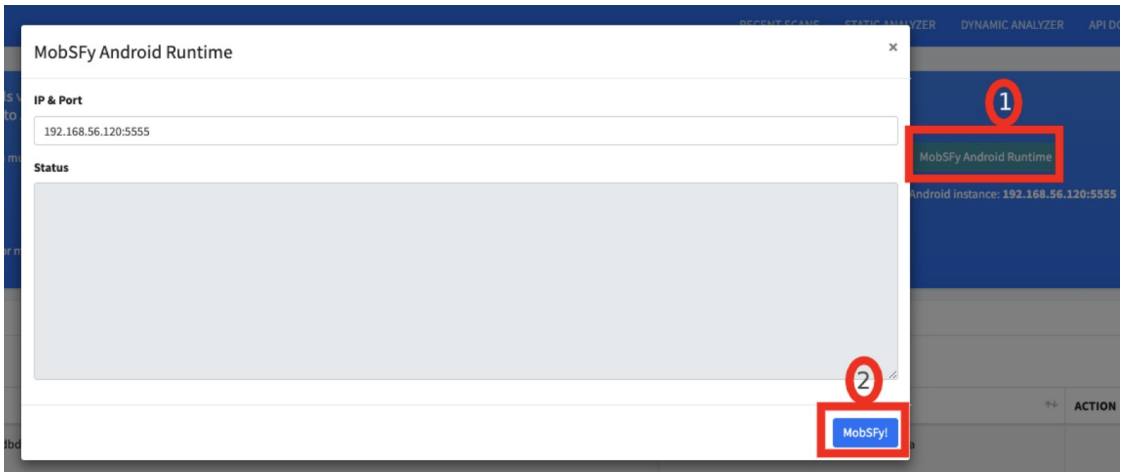

Figura 17: Configuración de MobSFy

7. Ejecutar análisis dinámico con el botón “Start Instrumentation”. Como se observa en la Figura 18: Ejecutar análisis dinámico

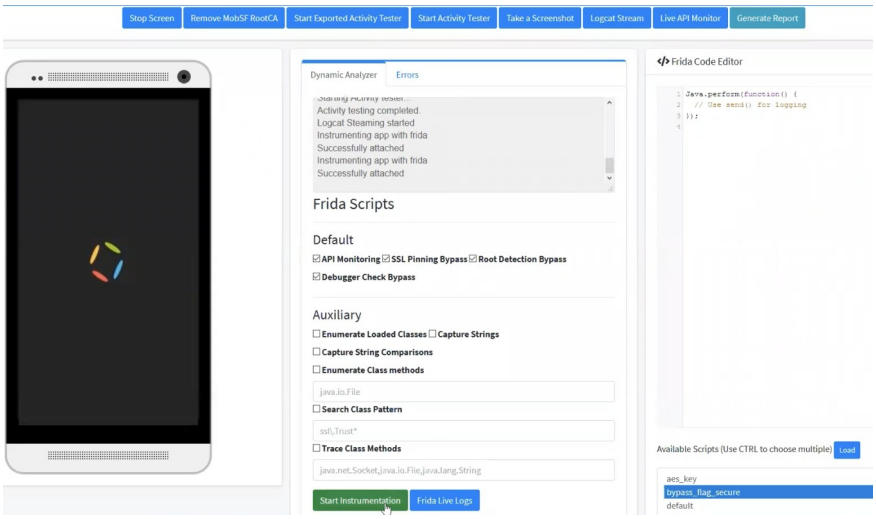

Figura 18: Ejecutar análisis dinámico

3.1.3. Runtime Mobile Security

RMS en su versión 1.5.7 y Frida server 14.2.14 Android x86 en conjunto son una plataforma para la evaluación de código fuente, con su interfaz web de la herramienta RMS ayuda a manipular las aplicaciones de Android e iOS en ejecución.[27]

Algunas características son:

1. Acceso a argumentos de los métodos.

2. Scripts personalizados.
3. Compatible para Android y iOS.

Dicha herramienta necesita algunos requerimientos los cuales son:

1. python 3+
2. NodeJS
3. FRIDA CLI TOOLS
4. Android Studio con la herramienta SDK de Android.

Para su descarga e instalación se deben realizar los siguientes pasos:

1. Instalar frida-server para Android. Para obtener el archivo desde la página oficial. https://github.com/frida/frida/releases
2. Abrir la terminal powershell y ejecutar el comando “npm install rms-runtime-mobile-security”. Como se observa en la Figura 19: Comando para instalar RMS

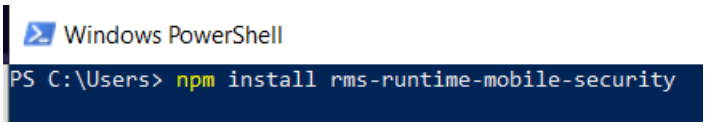

Figura 19: Comando para instalar RMS

3. Iniciar Android Studio para el uso de un adb, está incluido en el paquete de herramientas de la plataforma de Android SDK en el apartado AVD Manager. También es posible descargar este paquete con SDK Manager, donde se instala en android_sdk/platform-tools/. Como se muestra en la Figura 20: Apartado de AVD con Android Studio

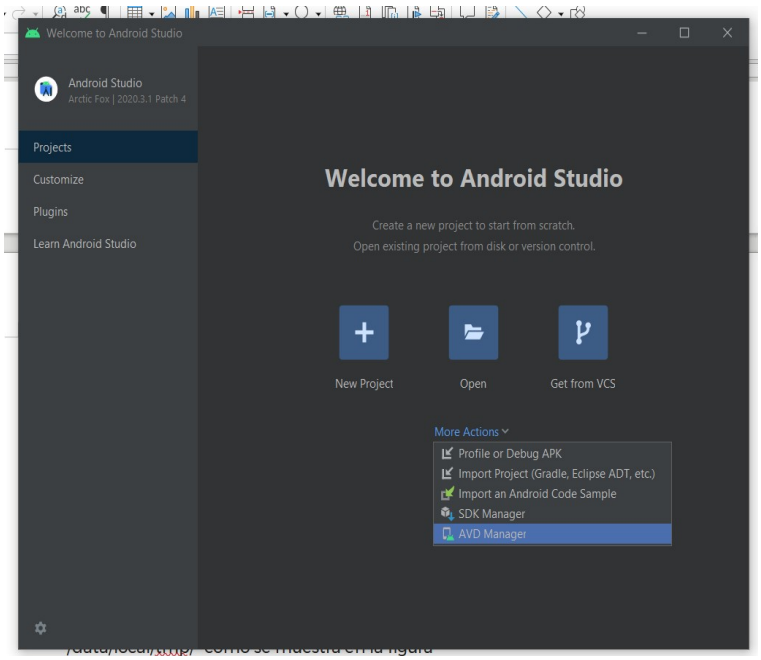

Figura 20: Apartado de AVD con Android Studio

4. Crear un dispositivo virtual dentro del apartado AVD Manager. Como se observa en la Figura 21: Crear un dispositivo virtual(AVD)

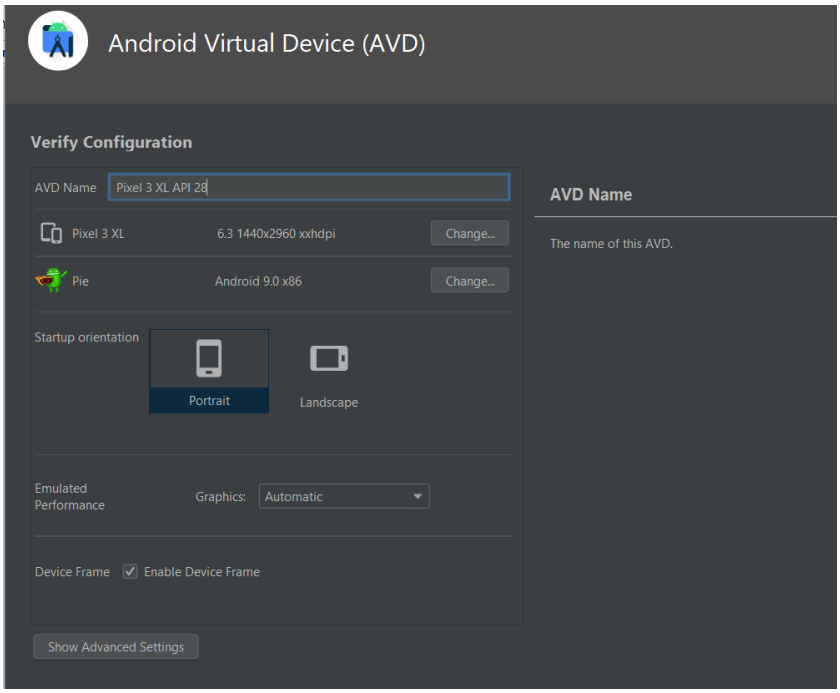

Figura 21: Crear un dispositivo virtual(AVD)

5. Ir a la ruta de destino del adb de Android Studio, “C:\Users\jean\AppData\Local\Android\Sdk\platform-tools” con el comando “cd” desde la terminal. Como se observa en Figura 22: Ruta destino de abd de Android Studio

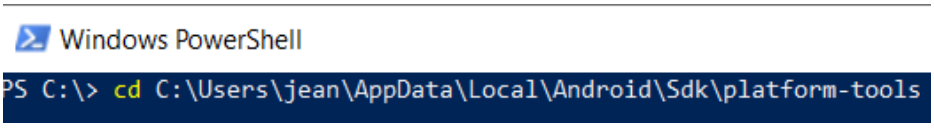

Figura 22: Ruta destino de abd de Android Studio

6. Ejecutar el comando “./adb start-server” para iniciar el servidor adb como se observa en Figura 23: Iniciar servidor adb

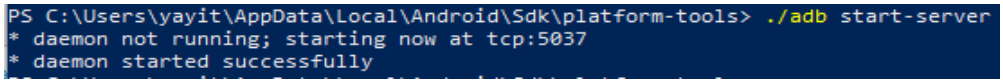

Figura 23: Iniciar servidor adb

7. Mostrar lista de dispositivos AVD con el comando “emulator -list-avds” desde terminal como se observa en Figura 24: Listar dispositivos AVD

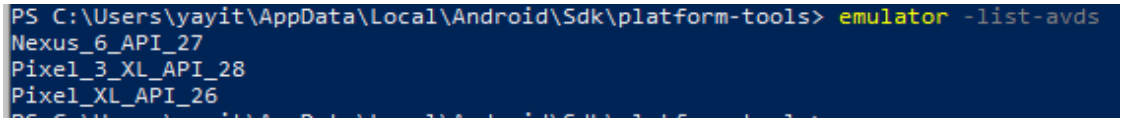

Figura 24: Listar dispositivos AVD

8. Iniciar el emulador desde terminal con el comando “emulator -avd <namedevices> -writable-system -no-snapshot” como se observa en Figura 25: Iniciar el emulador AVD desde terminal

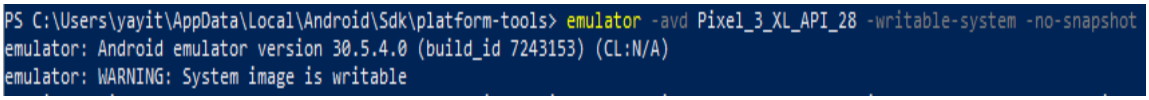

Figura 25: Iniciar el emulador AVD desde terminal

9. Abrir una terminal nueva e iniciar usuario root del adb “./adb root” como se observa en Figura 26: Iniciar usuario root en un adb

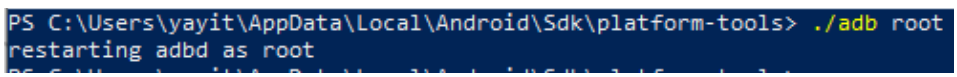

Figura 26: Iniciar usuario root en un adb

10. Guardar el servidor Frida en el dispositivo en la ruta “/data/local/tmp/” con el comando “./adb push <ruta donde se encuentra el servidor Frida> /data/local/tmp/” como se observa Figura 27: Guardar servidor Frida en un adb

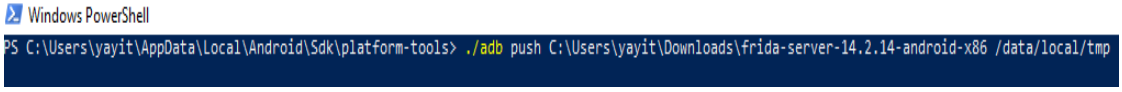

Figura 27: Guardar servidor Frida en un adb

11. Acceder al Shell del adb con el comando “./adb shell” como se observa en Figura 28: Acceder al shell de un adb

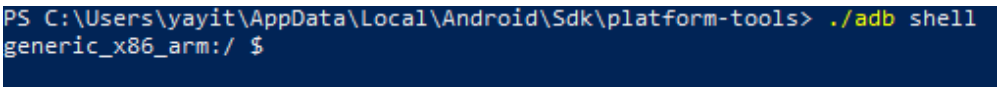

Figura 28: Acceder al shell de un adb

12. Acceder al directorio destino, con el comando “cd /data/local/tmp/” y verificar que el servidor esté dentro del dispositivo con el comando “ls” como se observa en Figura 29: Verificar la ruta del servidor Frida dentro de un adb

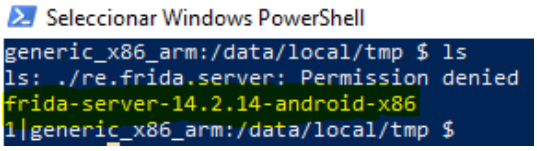

Figura 29: Verificar la ruta del servidor Frida dentro de un adb

13. Dentro de la carpeta local de nuestro dispositivo asignar los permisos necesarios al servidor Frida con el comando “chmod 755 <nombre del servidor>” como se observa en Figura 30: Asignar permisos al servidor Frida

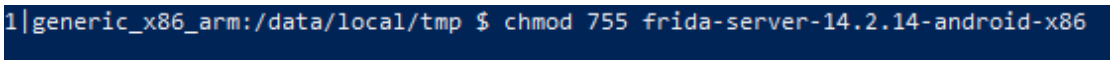

Figura 30: Asignar permisos al servidor Frida

14. Reiniciar el emulador, para ello necesita apagar el dispositivo con el botón que se encuentra ubicada de lado derecho del dispositivo, como se observa en Figura 31: Apagar dispositivo adb, volver a correr el dispositivo como se muestra en el punto 8.

Figura 31: Apagar dispositivo adb

15. Entrar nuevamente al Shell del emulador con usuario root, como se muestra en el punto 9. y posteriormente, entrar a la carpeta temporal del dispositivo como se mostró en el punto 12.
16. Ejecutar el servidor Frida con el comando "./nombre del servidor" como se observa en Figura 32: Ejecutar servidor Frida

```
generic_x86_arm:/data/local/tmp # ./frida-server-14.2.14-android-x86
```

Figura 32: Ejecutar servidor Frida

17. Abrir una terminal nueva, sin cerrar la terminal del emulador, ni la terminal del servidor Frida. En la nueva terminal ver los procesos que está corriendo el servidor Frida con el comando "Frida -ps -u" como se observa en Figura 33: Ver los procesos en ejecución de Frida

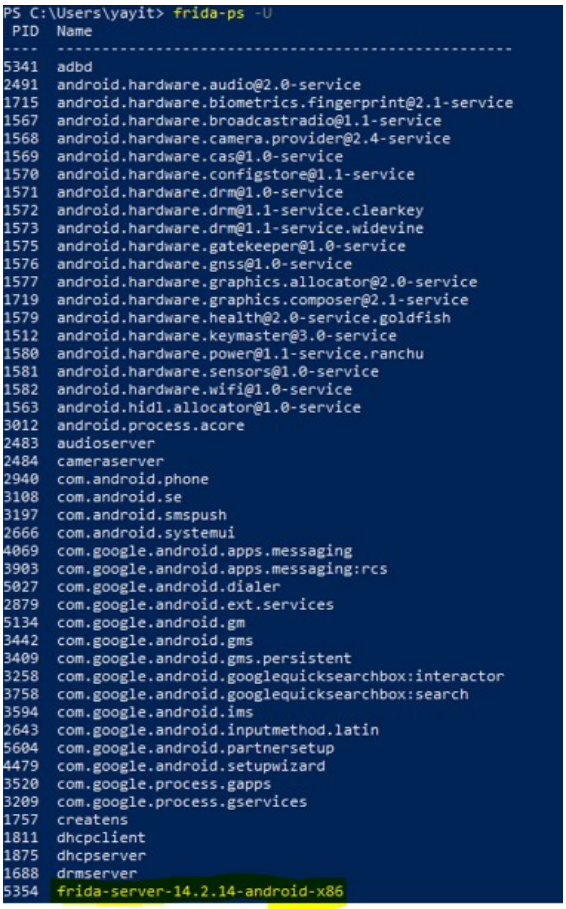

Figura 33: Ver los procesos en ejecución de Frida

18. Verificar si sé está corriendo nuestro servidor como se muestra en la figura, de lo contrario repetir los pasos a partir del punto 3.

19. Correr la herramienta RMS con el comando “rms”, como se observa en Figura 34: Ejecutar Runtime Mobile Security ,de igual forma en caso de no agregar como variable local ejecutar como se observa en la Figura 35: Ejecutar Runtime Mobile Security sin variable local

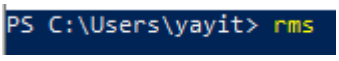

Figura 34: Ejecutar Runtime Mobile Security

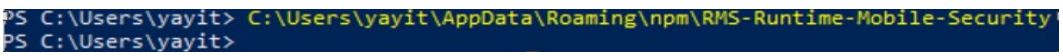

Figura 35: Ejecutar Runtime Mobile Security sin variable local

20. Abrir una pestaña de un navegador e ingresar al localhost en el puerto 5000 o bien con la dirección “http://127.0.0.1:5000” como se observa en Figura 36: Ejecutar RMS desde servidor local, finalmente mostrará la herramienta como se observa en la Figura 37: Interfaz de la herramienta RMS

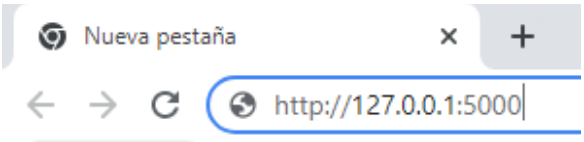

Figura 36: Ejecutar RMS desde servidor local

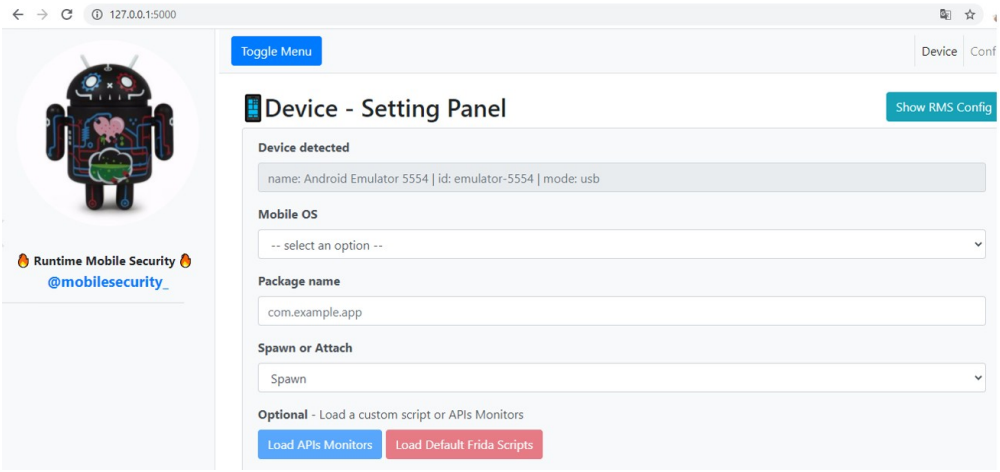

Figura 37: Interfaz de la herramienta RMS

3.1.4. Burp Suite

Burp Suite en su versión Community Edition 2021.12.1 es una plataforma que permite realizar pruebas de seguridad en aplicaciones, está diseñada para soportar numerosas metodologías, realizando diferentes tipos de pruebas.[28]

Algunas de sus características son:

1. Realizar pruebas automatizadas de exploración y escaneo de vulnerabilidades.
2. Escáner de vulnerabilidades para pruebas manuales.
3. Lógica de exploración de vanguardia.
4. Payloads integrados.

5. Interceptar el tráfico de red mediante un Proxy (man-in-the-middle).

Para el uso de esta herramienta se necesita una lista de requerimientos los cuales son:

1. Java Runtime Environment versión 1.7 o superior.
2. Memoria RAM 8GB.
3. Mínimo 2 núcleos de CPU.
4. Arquitectura de 64 bits.
5. Android Studio con la herramienta SDK de Android.

Para su descarga e instalación se necesita obtener el archivo, desde su página principal https://portswigger.net/burp/communitydownload.

Burp Suite no cuenta como tal con un proceso de instalación, solo basta con ejecutar el archivo descargado desde su página principal.

El uso de Burp Suite Community dentro de está investigación va enfocado a un análisis dinámico en un dispositivo móvil, por lo que se necesita una serie de pasos para su configuración, los cuales son:

1. Iniciar Android Studio para el uso de un adb. Que se encuentra en "configure/AVD Manager". Como se observa en Figura 38: Apartado de AVD con Android Studio

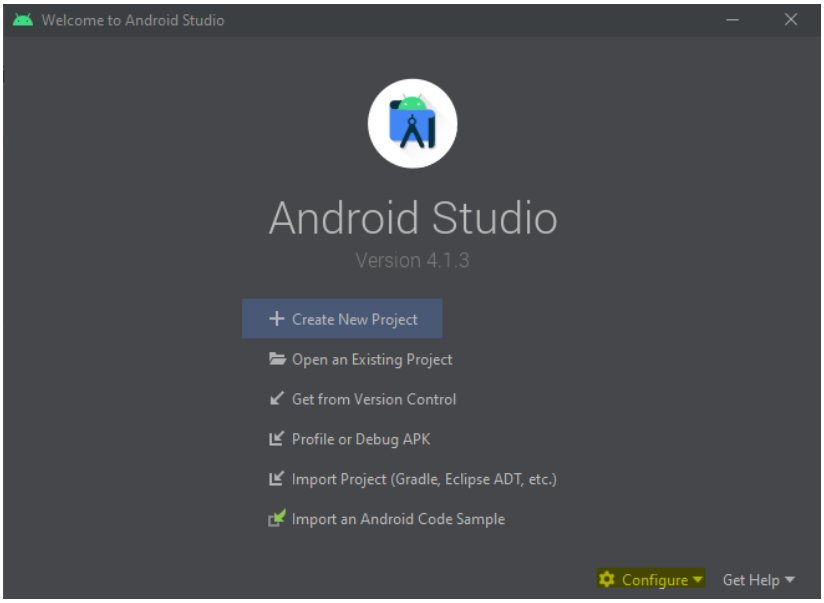

Figura 38: Apartado de AVD con Android Studio

2. Crear un dispositivo virtual dentro del apartado AVD Manager, como se observa en Figura 39: Crear un dispositivo virtual(AVD)

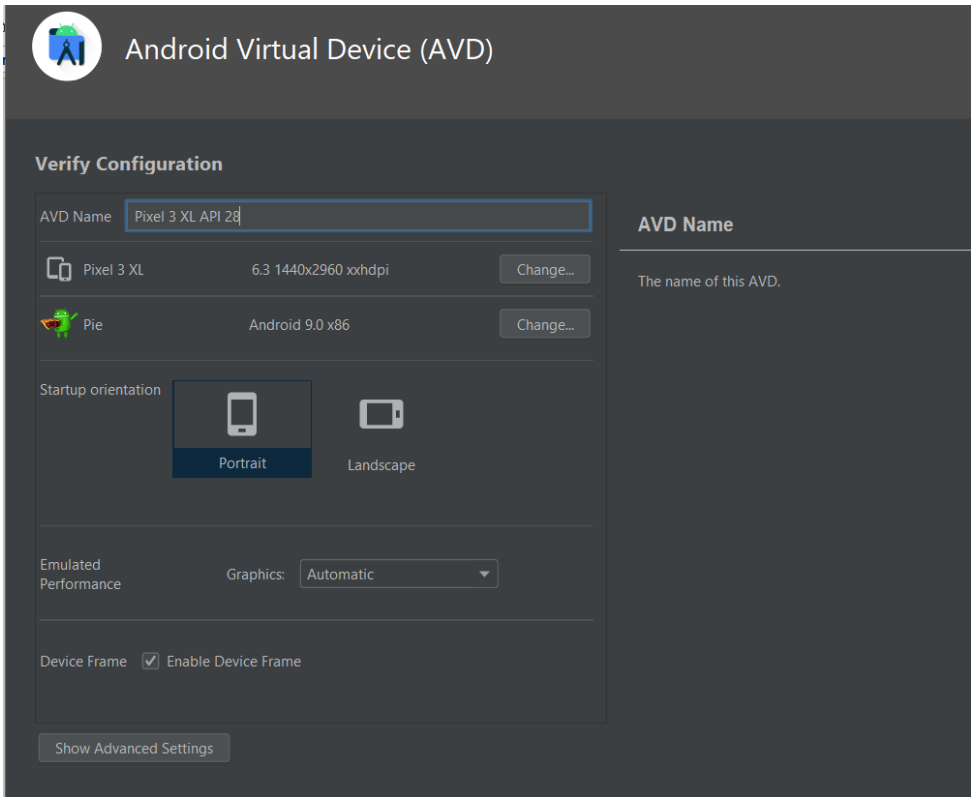

Figura 39: Crear un dispositivo virtual(AVD)

3. Ir a la ruta de destino del adb de Android Studio, “C:\Users\jean\AppData\Local\Android\Sdk\platform-tools” con el comando “cd” desde una terminal *Power Shell* . Como se observa en Figura 40: Ruta destino de abd de Android Studio

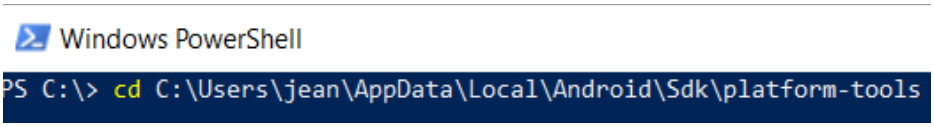

Figura 40: Ruta destino de abd de Android Studio

4. Ejecutar el comando “./adb start-server” para iniciar el servidor adb como se observa en la figura Figura 41: Iniciar servidor adb

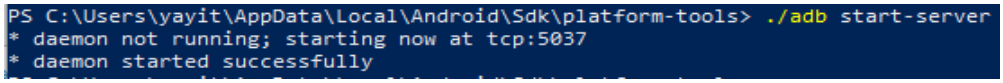

Figura 41: Iniciar servidor adb

5. Mostrar lista de dispositivos AVD con el comando “emulator -list-avds” desde terminal como se observa en la Figura 42: Listar dispositivos AVD

```
PS C:\Users\yayit\AppData\Local\Android\Sdk\platform-tools> emulator -list-avds
Nexus_6_API_27
Pixel_3_XL_API_28
Pixel_XL_API_26
```

Figura 42: Listar dispositivos AVD

6. Iniciar emulador desde terminal con el comando “emulator -avd <namedevices> -writable-system -no-snapshot” como se observa en Figura 43: Iniciar el emulador AVD desde terminal

```
PS C:\Users\yayit\AppData\Local\Android\Sdk\platform-tools> emulator -avd Pixel_3_XL_API_28 -writable-system -no-snapshot
emulator: Android emulator version 30.5.4.0 (build_id 7243153) (CL:N/A)
emulator: WARNING: System image is writable
```

Figura 43: Iniciar el emulador AVD desde terminal

7. Iniciar Burp Suite y agregar nuevo Proxy, para ello ir al apartado “Proxy/Options/Add” dentro de la interfaz de la herramienta, posteriormente cambiar el puerto a “8888” y especificar una dirección ip “192.168.0.15”, como se observa en Figura 44: Agregar nuevo Proxy en Burp Suite

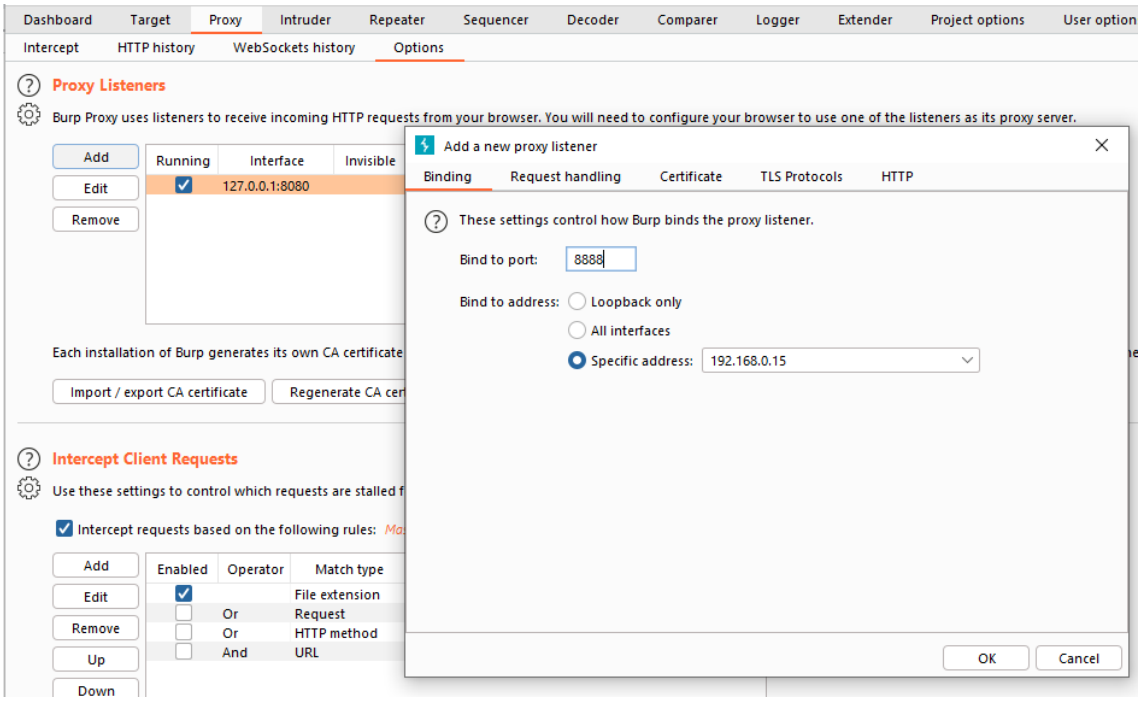

Figura 44: Agregar nuevo Proxy en Burp Suite

8. Deshabilitar Proxy por defecto “127.1.0.0.1:8080”, para evitar interferencias en los paquetes. Solo basta con dar clic en la “palomita” de la dirección ip, como se observa en Figura 45: Deshabilitar Proxy por defecto en Burp Suite

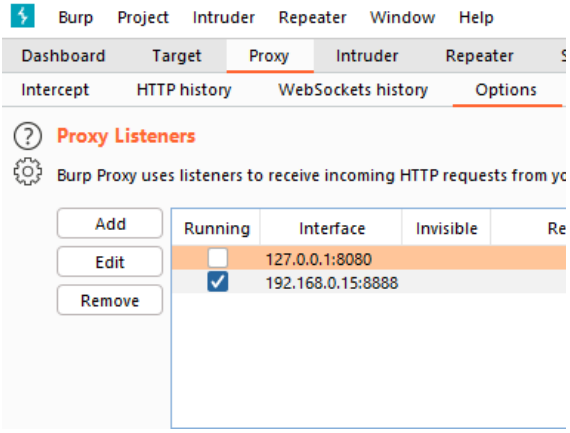

Figura 45: Deshabilitar Proxy por defecto en Burp Suite

9. Exportar certificado CA de Burp Suite desde la interfaz de la herramienta, donde se encuentra disponible en el apartado “Proxy/Options/”, donde basta con dar clic en el botón ”Import/export CA certificate” como se observa en Figura 46: Exportar certificado CA de Burp Suite

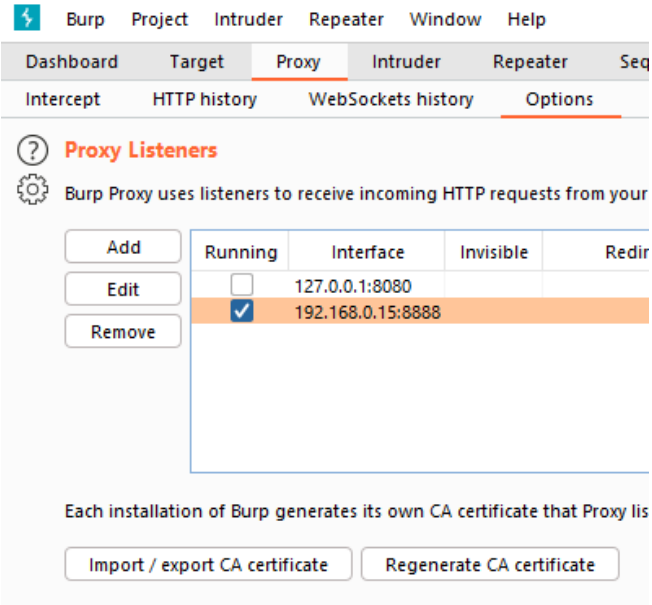

Figura 46: Exportar certificado CA de Burp Suite

10. Guardar el Certificado con el nombre “burp.der” como se observa en Figura 47: Guardar Certificado CA de Burp Suite

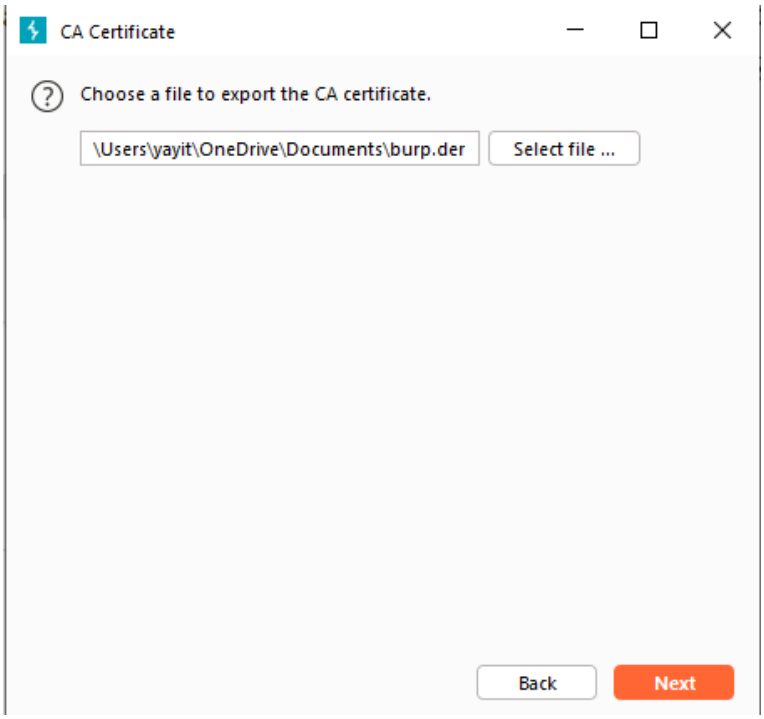

Figura 47: Guardar Certificado CA de Burp Suite

11. Generar una copia del certificado CA descargado con el nombre “burp.cer” como se observa en Figura 48: Generar copia del certificado CA con extensión ".cer"

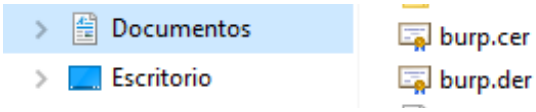

Figura 48: Generar copia del certificado CA con extensión ".cer"

12. Guardar el certificado en el dispositivo en la ruta “/sdcard” con el comando “./adb push <ruta donde se encuentra el certificado> /sdcard/nombre del certificado” como se observa en Figura 49: Guardar certificado Burp Suite en un adb

```
PS C:\Users\yayit\AppData\Local\Android\Sdk\platform-tools> ./adb push C:\Users\yayit\OneDrive\Documents\burp.cer /sdcard/burp.cer
C:\Users\yayit\OneDrive\Documents\burp.cer: 1 file pushed, 0 skipped. 0.0 MB/s (939 bytes in 0.674s)
```

Figura 49: Guardar certificado Burp Suite en un adb

13. Verificar el guardado del certificado CA, entrar al shell del adb con el comando "./adb shell" y posteriormente ingresar a la ruta donde se guardó el certificado con el comando "cd", finalmente listar su contenido como se observa en Figura 50: Verificar el guardado de certificado CA dentro de un adb

```
PS C:\Users\yayit\AppData\Local\Android\Sdk\platform-tools> ./adb shell
generic_x86_arm:/ # cd sdcard
generic_x86_arm:/sdcard # ls
Alarms Android DCIM Download Movies Music Notifications Pictures Podcasts Ringtones burp.cer
generic_x86_arm:/sdcard #
```

Figura 50: Verificar el guardado de certificado CA dentro de un adb

14. Ir a la ubicación del certificado CA previamente instalado en nuestro dispositivo virtual, que se encuentra en "Settings/Security & location/Advanced/Encryption & credentials/Install from SD" como se observa en la Figura 51: Ubicación del certificado CA de Burp Suite dentro de un dispositivo adb

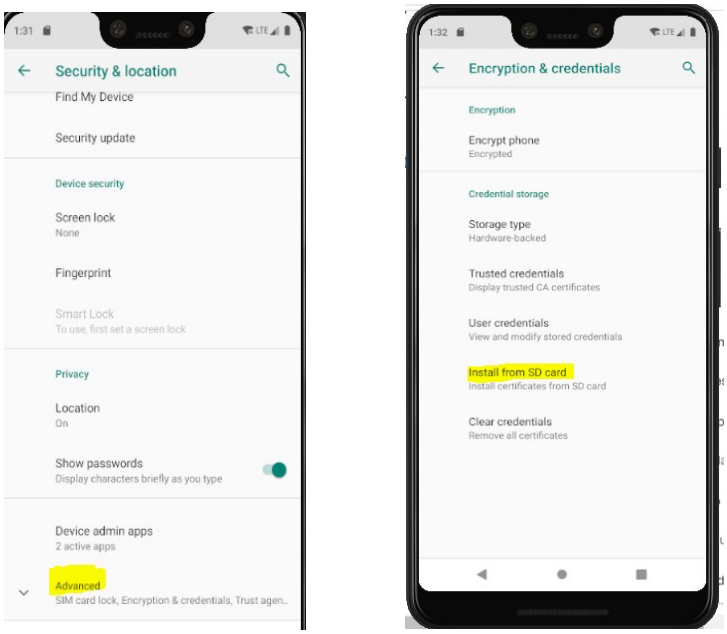

Figura 51: Ubicación del certificado CA de Burp Suite dentro de un dispositivo adb

15. Instalar certificado en el dispositivo adb, para ello dar doble clic al archivo 'burp.cer' e instalar con el nombre “Burp Certificate” finalmente dar clic en 'Ok'. Como se observa en Figura 52: Instalar certificado de Burp Suite en dispositivo móvil virtual

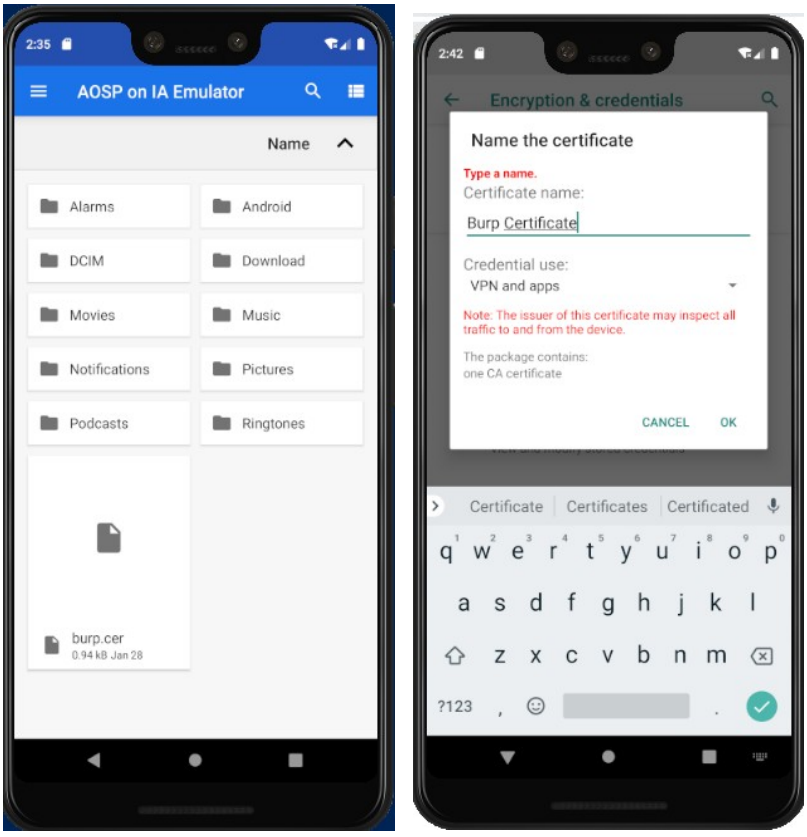

Figura 52: Instalar certificado de Burp Suite en dispositivo móvil virtual

16. Configurar Proxy en el emulador adb, para ello ir a “Settings/Network & Internet/Wi-Fi”, seleccionar la red que actualmente se encuentra conectado. Dentro de “Networds details” dar clic en el símbolo del lápiz que se encuentra en la parte superior derecha. Como se observa en la Figura 53: Detalles de una red en un dispositivo virtual móvil

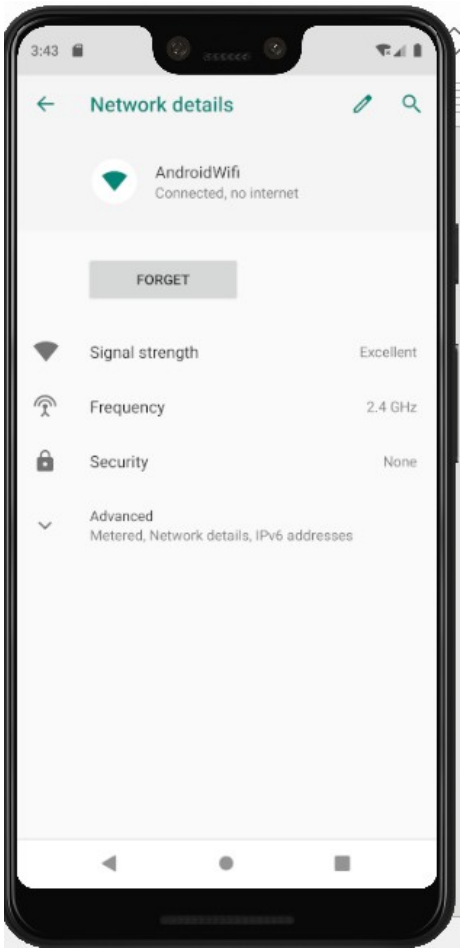

Figura 53: Detalles de una red en un dispositivo virtual móvil

17. Abrir las opciones avanzadas en el apartado “Proxy ” elegir “Manual” como se observa en la Figura 54: Opciones avanzadas de la red en un dispositivo adb finalmente ingresar la dirección IP “192.168.0.15” y el puerto “8888” mismo datos que se configuraron anteriormente en el punto 7. , como se observa en Figura 55: Configurar Proxy en un dispositivo virtual móvil

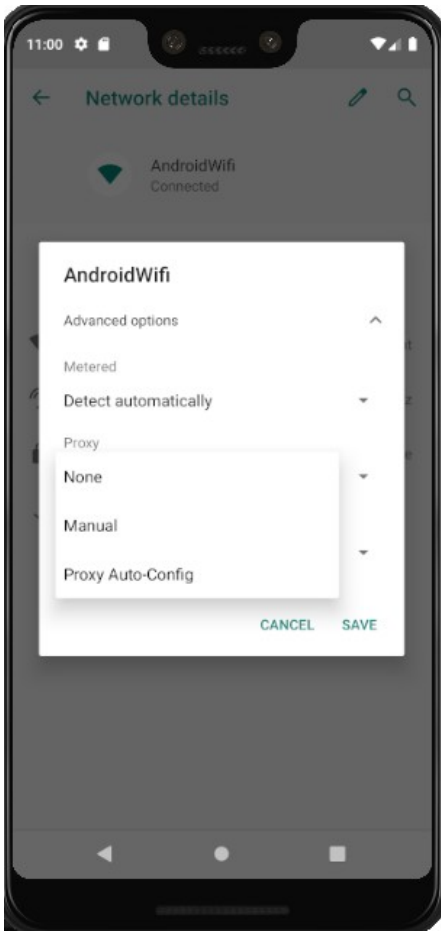

Figura 54: Opciones avanzadas de la red en un dispositivo adb

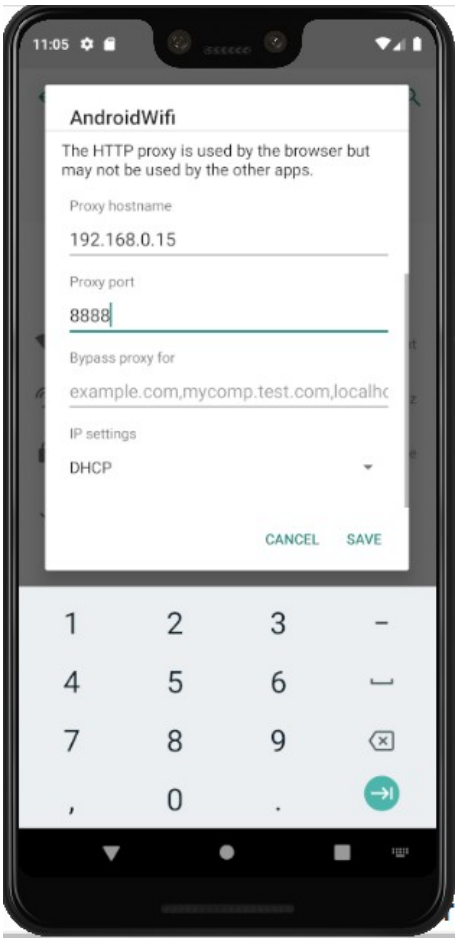

Figura 55: Configurar Proxy en un dispositivo virtual móvil

A continuación en el CAPÍTULO IV se describe las herramientas utilizadas para el análisis de seguridad a una aplicación móvil bajo las metodologías SAST y DAST así como la información brindada por fuentes confiables para la detección de vulnerabilidades.

CAPÍTULO IV

4. ANÁLISIS DE VULNERABILIDADES

Para realizar el análisis es necesario contar con las herramientas especializadas e identificar cada vulnerabilidad arrojada por cada una de ellas, comparando falsos positivos y así posteriormente mitigar el mayor número de riesgos, evitando una aplicación insegura.

4.1. PRUEBA CON ANALIZADORES ESTÁTICOS

Para el análisis estático se contemplaron herramientas de código abierto, donde se describen a continuación:

4.1.1. MARA

Para la implementación de la herramienta MARA, es necesario ejecutar el archivo "mara.sh" con sus respectivos parámetros. Como se mostró en el CAPÍTULO III apartado 3.1.1. MARA ver la Figura 7: Comando para ejecutar MARA Los pasos a seguir son los siguientes:

1. Dentro del mismo directorio, se encuentra la carpeta "*data*", donde se almacena el resultado del análisis en una carpeta con el nombre de la aplicación analizada. Como se observa en la Figura 56: Almacenamiento de reporte en herramienta MARA

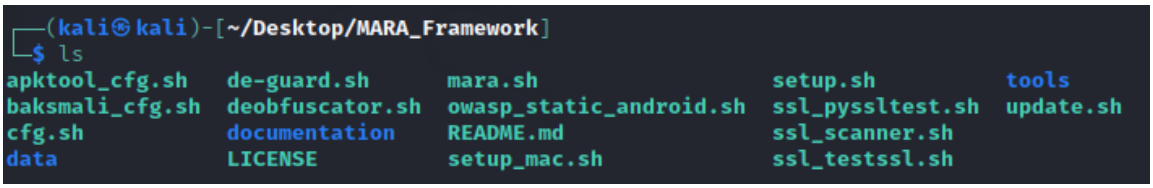

Figura 56: Almacenamiento de reporte en herramienta MARA

2. Una vez identificada la carpeta de la aplicación, se puede observar las siguientes carpetas: "*analysis, certificate, smali, source, unzipped*" donde cada una de ellas contiene reportes e información confidencial del apk, como se observa en la Figura 57: Análisis MARA

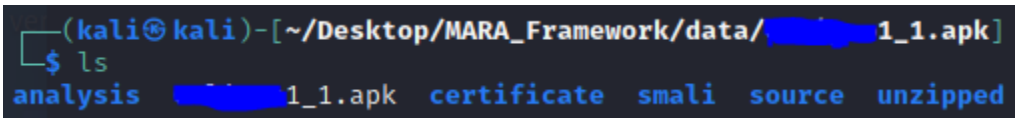

Figura 57: Análisis MARA

3. Los reportes que contienen las vulnerabilidades del apk de nuestro interés se encuentra en la ruta “analysis/static/vulnerabilities” como se observa en Figura 58: Reportes de análisis estático, MARA

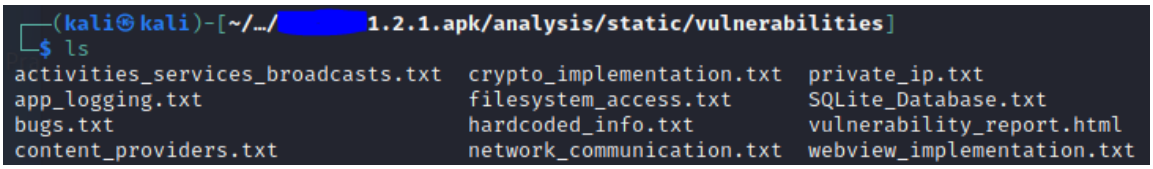

Figura 58: Reportes de análisis estático, MARA

Durante el análisis, con la herramienta MARA devolvió un total de 46 vulnerabilidades. Es importante considerar que no todas son temas de seguridad. MARA clasifica dichas vulnerabilidades de la siguiente forma:

1. **Bugs(46):** Vulnerabilidades de error en la codificación, el cual debe corregirse.
2. **Activities services broadcasts(0):** Registro de actividad de servicios bradcasts.
3. **App Logging(0):** Registros de inicio.
4. **Content providers(0):** Registros de proveedores de contenidos.
5. **Crypo implementation(0):** Registro de implementación de criptografía.
6. **filesystem access(0):** Registro de accesos al sistema de archivos.
7. **hardcoded info(0):** Información de codificación.
8. **network communication(0):** Registros de la comunicación de red.
9. **private ip(0):** Registros de ip privadas.
10. **SQLite Database(0):** Registros de consultas a la base de datos SQLite.
11. **Webview implementation(0):** Implementaciones de vistas web.

MARA divide los resultados de las vulnerabilidades por grados de peligrosidad teniendo en cuenta las siguientes categorías, como se observa en la Tabla 5: Nivel de peligrosidad MARA

COLOR	NIVEL DE PELIGROSIDAD
ROJO	CRÍTICO
NARANJA	ADVERTENCIA
VERDE	BAJO
AZUL	INFORMATIVO

Tabla 5: Nivel de peligrosidad MARA

1. **Crítico(4):** Error con una alta probabilidad de afectar el comportamiento de la aplicación en producción: pérdida de memoria, conexión no cerrada. El código debe corregirse inmediatamente.

2. **Advertencia(2):** Defecto de calidad que puede tener un gran impacto en la productividad del desarrollador: fragmento de código descubierto, bloques duplicados, parámetros no utilizados.

3. **Bajo(0):** Defecto de calidad que puede afectar ligeramente la productividad del desarrollador: las líneas no deben ser demasiado largas, las declaraciones de "cambio" deben tener al menos 3 casos.

4. **Informativo(40):** Ni un error ni un defecto de calidad, solo un hallazgo.

4.1.2. Mobile Security Framework (MobSF)

Una vez realizada la configuración de MobSF, es posible proceder con su implementación sobre la aplicación a analizar, con esta herramienta se analizará tanto Android(.apk) como iOS(.ipa). Para ello, los pasos a seguir son los siguientes:

1. Ejecutar el comando "./run.sh 127.0.0.0:8000". Como se observa en el CAPÍTULO III apartado 3.1.2. Mobile Security Framework (MobSF) ver la Figura 11: Comando para ejecutar MobSF

2. Abrir en su navegador web, navegue hasta "http://localhost:8000/". Como se observa en el CAPÍTULO III apartado 3.1.2. Mobile Security Framework (MobSF) ver la Figura 12: Ejecución MobSF desde el navegador

3. Cargar la aplicación en la herramienta con el botón "*Upload & Analyze*" como se observa en la Figura 59: Cargar la aplicación en MobSF

Figura 59: Cargar la aplicación en MobSF

4. Una vez cargada la aplicación, se visualizará los resultados del análisis estático, como se observa en Figura 60: Interfaz del análisis estático MobSF

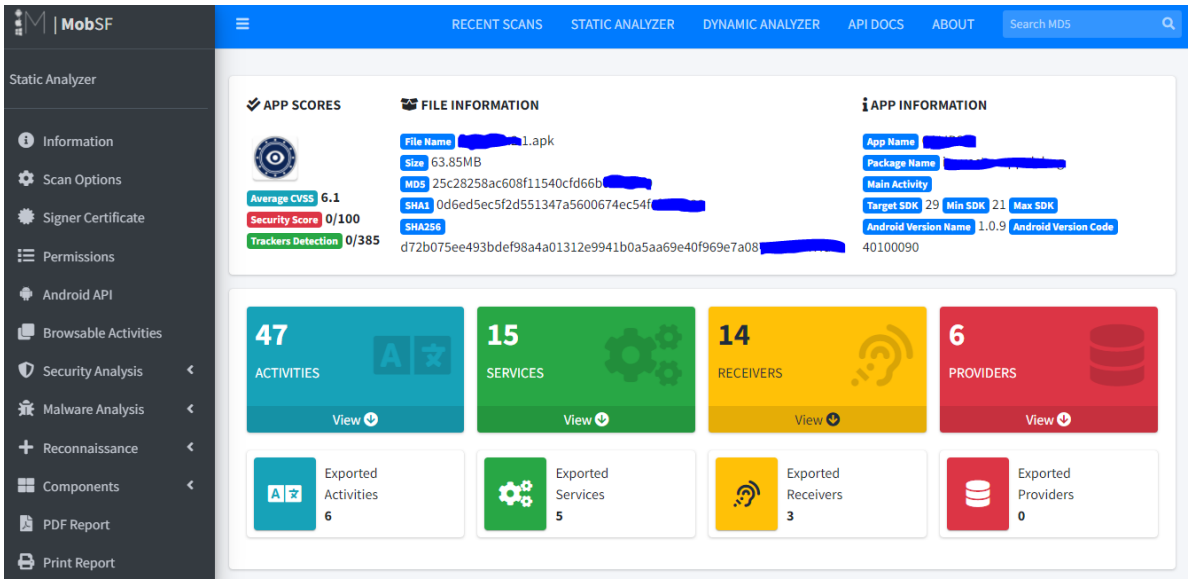

Figura 60: Interfaz del análisis estático MobSF

5. Dentro la misma pantalla en el apartado del panel izquierdo, se encuentra "*PDF Report*", el cual es para descargar el reporte del análisis en formato pdf como se observa en la Figura 61: Guardar reporte estático MobSF

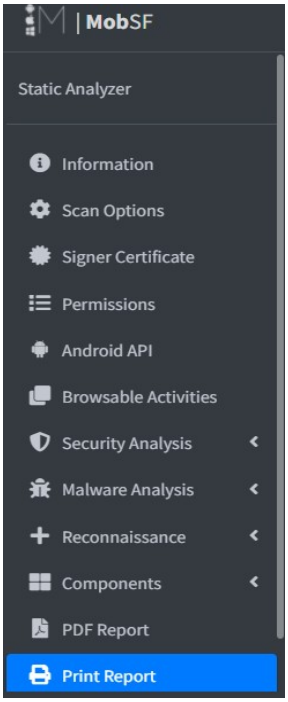

Figura 61: Guardar reporte estático MobSF

Durante el análisis con la herramienta devolvió un total de 69 vulnerabilidades en Android y 20 vulnerabilidades en iOS. Es importante considerar que no todas son temas de seguridad. MobSF clasifica dichas vulnerabilidades de la siguiente forma:

1. **Permissions(Android 38, iOS 7):** Análisis de los permisos asignados en la aplicación.

2. **Security Activities(Android 31, iOS 13):** Registros de actividades de seguridad. Se divide en 5 apartados que son:

    1. **Network Analysis(Android 0, iOS 1):** Análisis de la red, direcciones ip vulnerables.

2. **Manifest Analysis(Android 16):** Análisis del manifiesto, muestra un problema de mantenimiento, el código es confuso y difícil de mantener.

3. **Code Analysis(Android 14, iOS 3):** Vulnerabilidades de error en la codificación, el cual debe corregirse.

4. **Binary Analysis(Android 0, 9 iOS):** Análisis del binario donde muestra fallas de código.

5. **File Analysis(Android 1):** Análisis de archivos vulnerables que contiene la aplicación.

3. **Malware Analysis(Android 0, iOS 0):** Análisis de malware, que muestra el uso de entradas que no son de confianza de una manera que podría crear una vulnerabilidad de seguridad explotable de forma remota.

MobSF divide los resultados de las vulnerabilidades por grados de peligrosidad teniendo en cuenta las siguientes categorías, como se observa en la Tabla 6: Nivel de peligrosidad MobSF

COLOR	NIVEL DE PELIGROSIDAD
ROJO	CRÍTICO
NARANJA	ALTO
AMARILLO	MEDIO
VERDE	BAJO
AZUL	INFORMATIVO

Tabla 6: Nivel de peligrosidad MobSF

1. **Crítico (Android 8, iOS 7):** Error con una alta probabilidad de afectar el comportamiento de la aplicación en producción.

2. **Alto(Android 22, iOS 3):** Error con baja probabilidad de afectar el comportamiento de la aplicación en producción o un problema que representa una falla de seguridad: block catch vacío, inyección SQL. El código debe ser revisado inmediatamente.

3. **Medio(Android 1, iOS 3):** Defecto de calidad que puede tener un gran impacto en la productividad del desarrollador: fragmento de código descubierto, bloques duplicados, parámetros no utilizados.

4. **Bajo(Android 4, iOS 1):** Defecto de calidad que puede afectar ligeramente o no afectar.

5. **Informativo(Android 34, iOS 6):** Ni un error ni un defecto de calidad, solo un hallazgo.

4.2 PRUEBA CON ANALIZADORES DINÁMICOS

Para el análisis dinámico se contemplaron herramientas de código abierto, donde se describen a continuación:

4.2.1 Runtime Mobile Security (RMS)

Una vez realizada la configuración de RMS, es posible proceder con su implementación sobre la aplicación a analizar, con esta herramienta se analizará Android(.apk). Para ello, los pasos a seguir son los siguientes:

1. Iniciar el emulador como se observa en el CAPÍTULO III apartado 3.1.3. Runtime Mobile Security ver la Figura 25: Iniciar el emulador AVD desde terminal

2. Ejecutar el servidor Frida como se observa en el CAPÍTULO III apartado 3.1.3. Runtime Mobile Security ver la Figura 32: Ejecutar servidor Frida

3. Guardar la aplicación a analizar dentro del dispositivo virtual con el comando "./adb install <path del apk>" como se observa en la Figura 62: Instalar aplicación en un adb

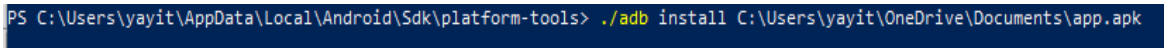

Figura 62: Instalar aplicación en un adb

4. Ejecutar el comando "rms", como se observa en el CAPÍTULO III apartado 3.1.3. Runtime Mobile Security ver la Figura 35: Ejecutar Runtime Mobile Security sin variable local

5. Abrir en su navegador web, navegue hasta "http://127.0.0.1:5000". Como se observa en el CAPÍTULO III apartado 3.1.3. Runtime Mobile Security ver en la Figura 36: Ejecutar RMS desde servidor local

6. Una vez iniciada la herramienta, identificar que la herramienta RMS reconoce el dispositivo móvil en el apartado "*Device detected*" de lo contrario reiniciar el emulador. Ver la Figura 63: Identificar dispositivo virtual en la herramienta RMS

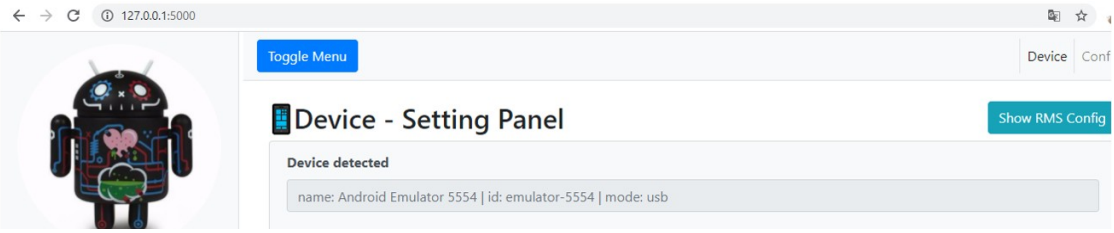

Figura 63: Identificar dispositivo virtual en la herramienta RMS

7. Una vez llenado los datos, iniciar el análisis con el botón "*Start RMS*". Ver la Figura 64: Iniciar análisis con RMS

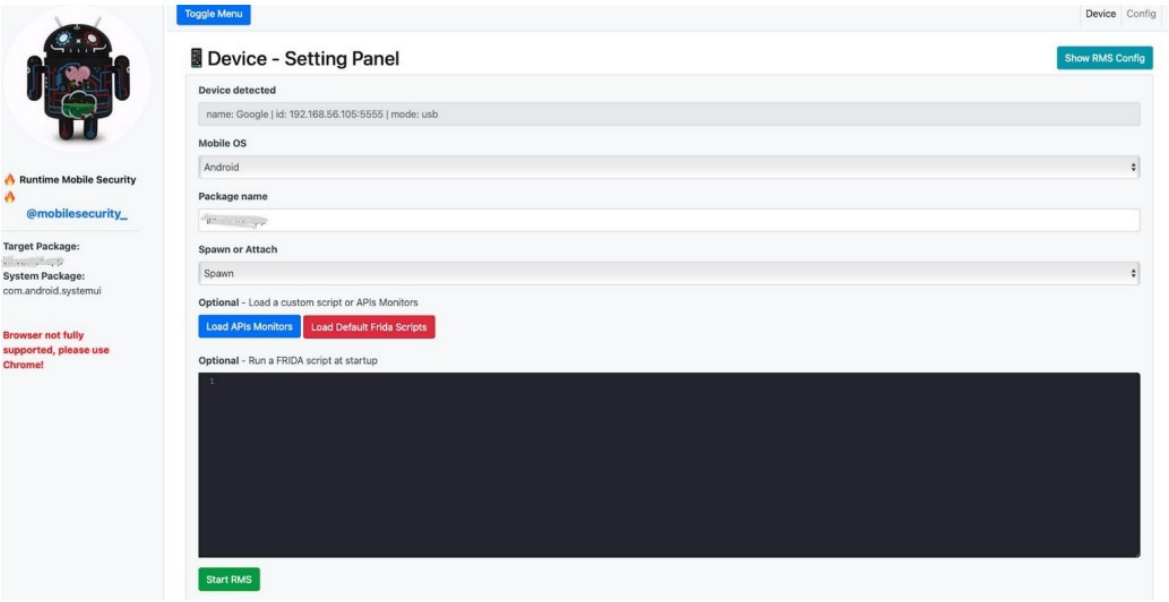

Figura 64: Iniciar análisis con RMS

8. Una vez cargada la aplicación, se visualizará los resultados del análisis dinámico, como se observa en Figura 65: Análisis dinámico con RMS

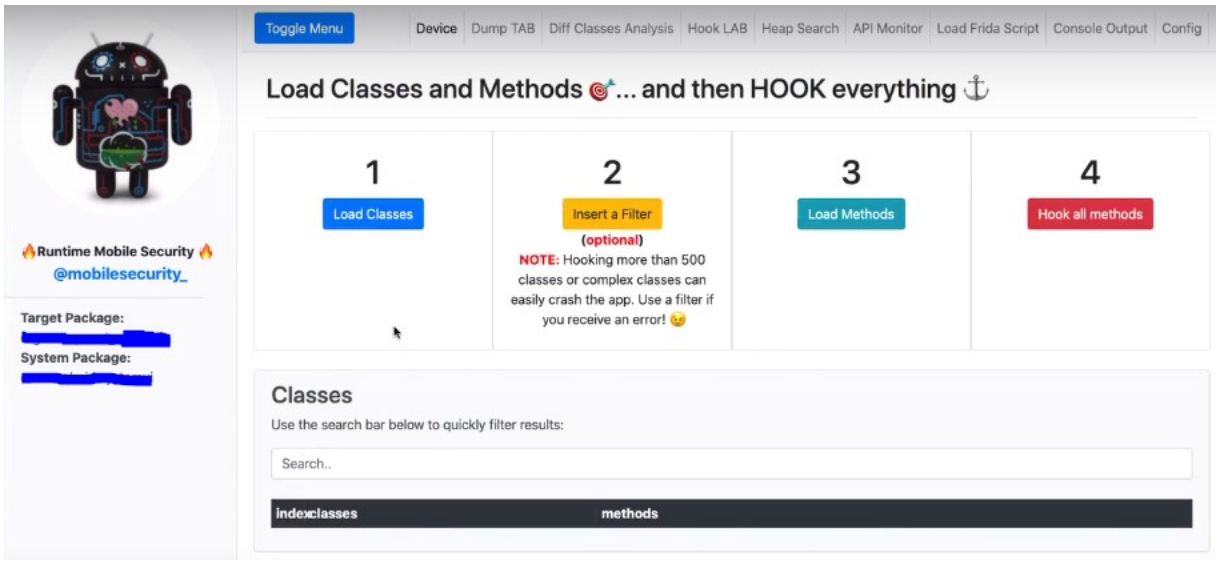

Figura 65: Análisis dinámico con RMS

9. El objetivo de este análisis es identificar funciones, clases y actividades de la aplicación en su ejecución que sean vulnerables. Ver Figura 66: Extracción de clases con RMS y Figura 67: Extracción de funciones y métodos con RMS

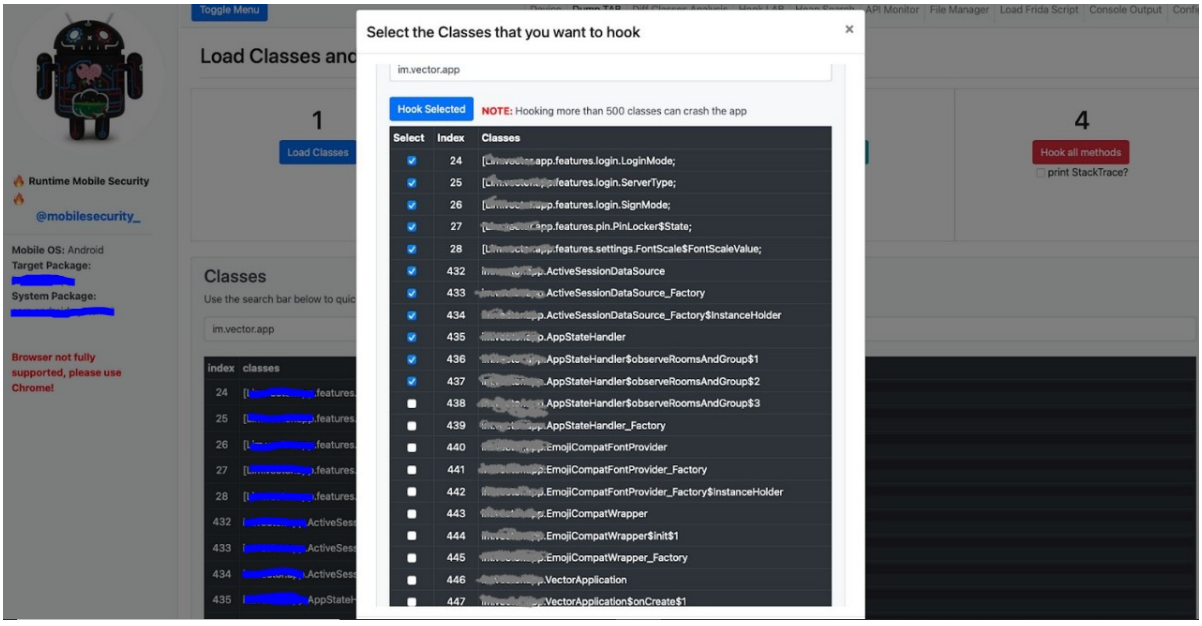

Figura 66: Extracción de clases con RMS

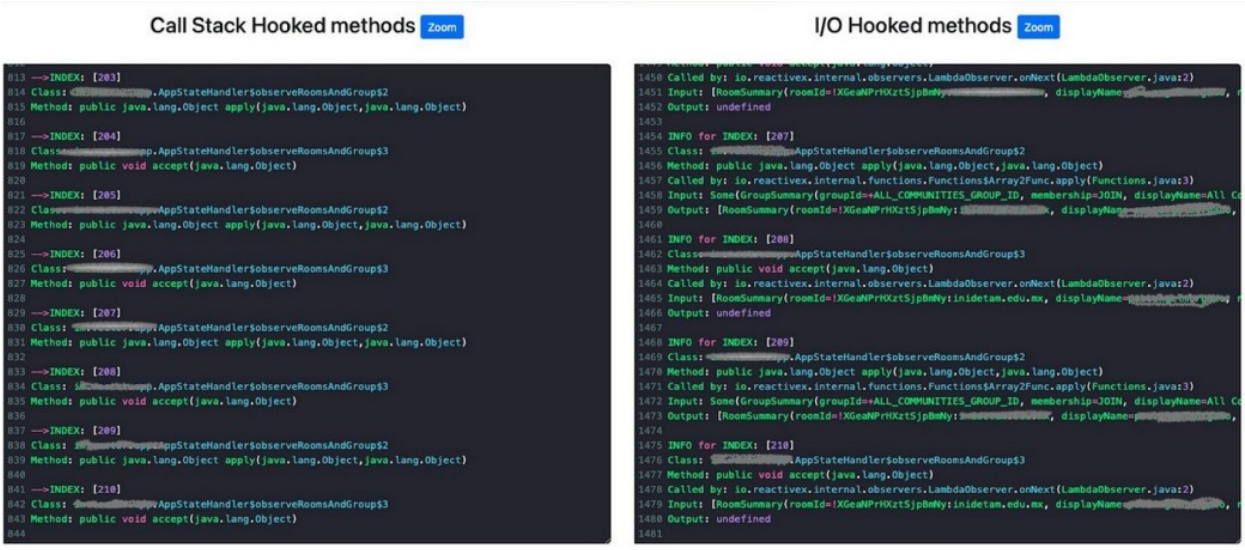

Figura 67: Extracción de funciones y métodos con RMS

Durante el análisis se obtuvieron los siguientes resultados:

1. **Acceso a las clases:** Se encontraron 776 clases usadas en el dispositivo, donde se encuentran registros de inicio de sesión, almacenamiento de datos e información de hardware.

2. **Acceso a un StackTrace:** Se logró mapear todas las llamadas a las funciones que se hacían durante el inicio de cada actividad de la aplicación.

3. **Acceso a métodos:** Se obtuvo una gran parte de los métodos usados en cada actividad de la aplicación.

4. **Acceso a la información de los contactos activos en la aplicación.**

5. **Acceso base de datos:** Se obtuvo consultas a la base de datos como se observa en la **Figura 68: Acceso a consultas de la base de datos con RMS**

Figura 68: Acceso a consultas de la base de datos con RMS

4.2.2 Mobile Security Framework (MobSF)

Una vez realizada la configuración de instalación y ejecución de MobSF para análisis dinámico como se observa en el CAPÍTULO III en el apartado 3.1.2. Mobile Security Framework (MobSF), los pasos a seguir son los siguientes:

1. MobSF cuenta con scripts de Frida, donde ayuda a realizar diversas pruebas. Cargar algunos scripts para el análisis como se observa en Figura 69: Cargar scripts Frida en MobSF

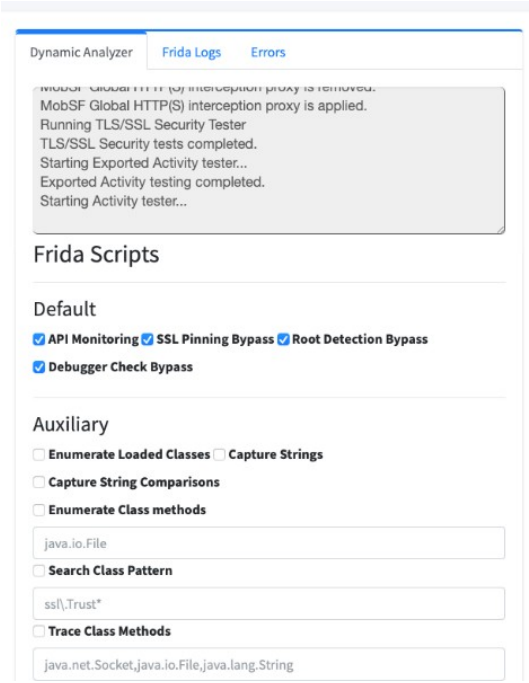

Figura 69: Cargar scripts Frida en MobSF

2. Iniciar el análisis con el botón "*Start Instrumentation*" como se observa en Figura 70: Iniciar análisis con MobSF

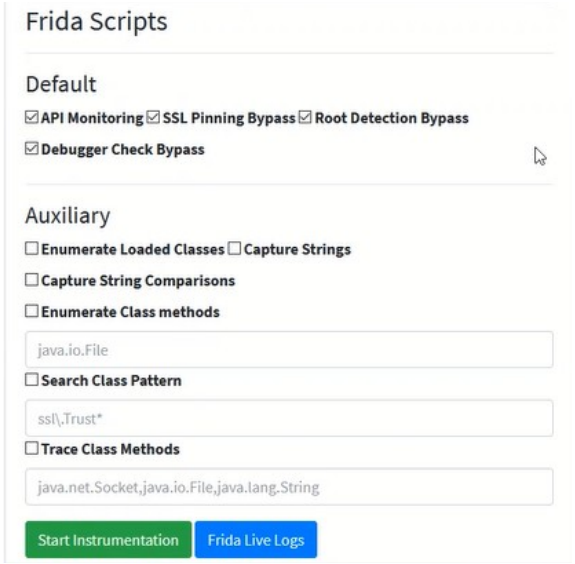

Figura 70: Iniciar análisis con MobSF

3. Dentro de la misma interfaz se encuentra un panel en la parte superior como se observa en la Figura 71: Panel de opciones en MobSF, que contiene más opciones para el análisis dinámico los cuales son:

    1. Remove MobSF RootCA
    2. *Start Explores Activity Tester*
    3. *Start Activity Tester*
    4. *Logcat Stream*
    5. *Live API Monitor*

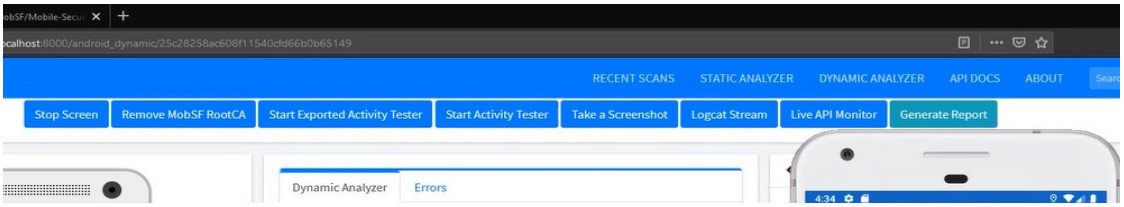

Figura 71: Panel de opciones en MobSF

4. Dentro del mismo panel se encuentra el botón "*Generate Report*", el cual es para generar el reporte del análisis dinámico como se observa en Figura 72: Generar reporte dinámico en MobSF

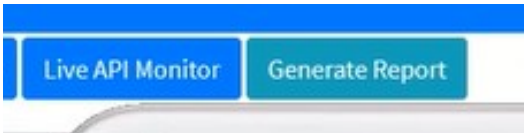

Figura 72: Generar reporte dinámico en MobSF

5. Una vez generado el reporte se visualizará los resultados del análisis como se observa en la Figura 73: Interfaz del análisis dinámico MobSF

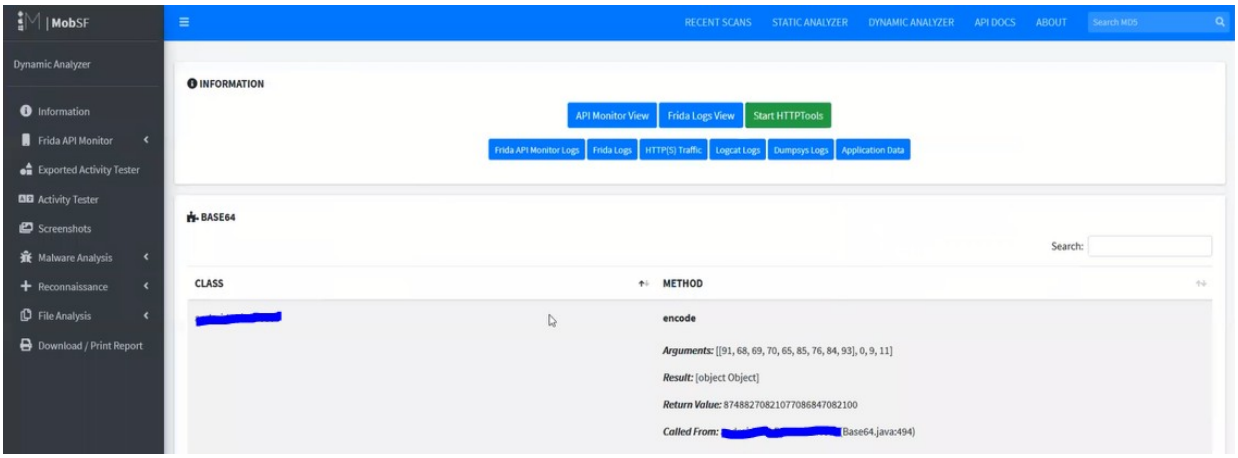

Figura 73: Interfaz del análisis dinámico MobSF

6. Dentro la misma pantalla en el apartado del panel izquierdo, se encuentra "*Donwload/Print Report*", el cual es para descargar el reporte del análisis en formato pdf como se observa en la Figura 74: Guardar reporte dinámico MobSF

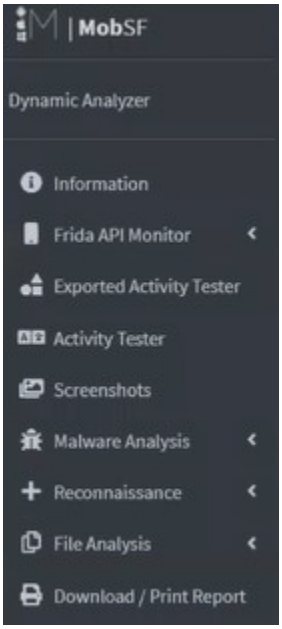

Figura 74: Guardar reporte dinámico MobSF

7. El objetivo de este análisis, es escanear las funciones, clases y actividades de la aplicación en su ejecución. Ver la Figura 75: Extracción de clases con MobSF

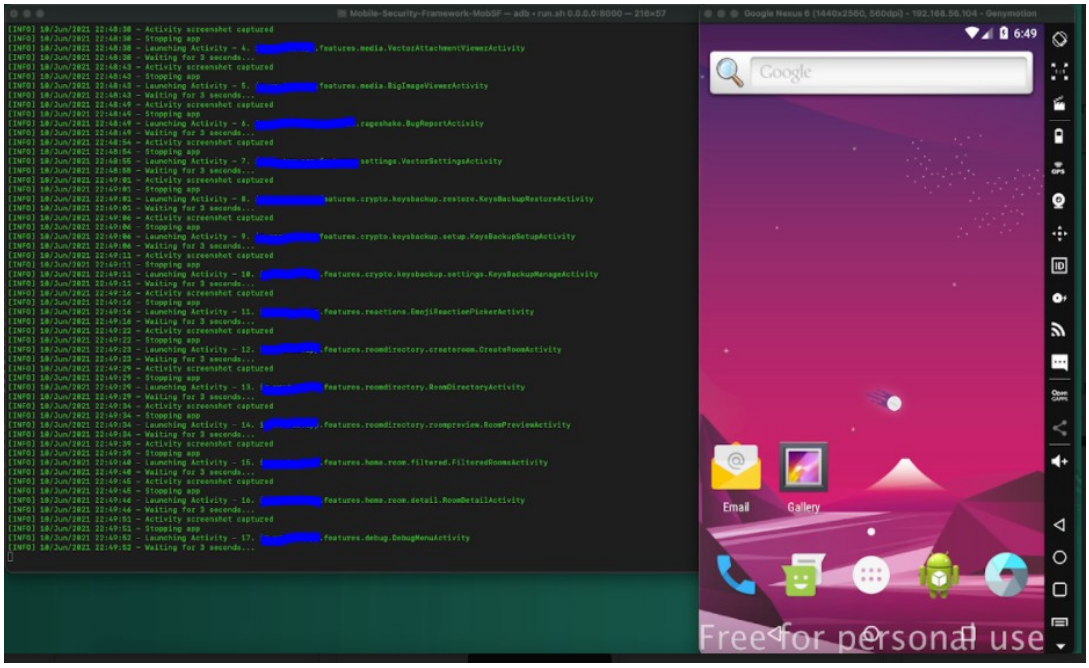

Figura 75: Extracción de clases con MobSF

Durante el análisis se obtuvieron los siguientes resultados:

1. **Extracción de clases:** Se encontraron 78 clases usadas en el dispositivo, donde se encuentran registros de inicio de sesión, almacenamiento de datos.

2. **Dependencias en ejecución:** Se obtuvo una gran parte de los métodos usados en cada actividad de la aplicación y llamadas al sistema.

3. **Archivos XML.**

4.2.3 Burp Suite

Una vez realizada la configuración de Burp Suite para análisis dinámico como se observa en el CAPÍTULO III en el apartado 3.1.4. Burp Suite, es posible proceder con su implementación sobre la aplicación a analizar, los pasos a seguir son los siguientes:

1. Iniciar emulador con aplicación previamente instalada como se observa en la Figura 62: Instalar aplicación en un adb
2. Ejecutar Burp Suite.
3. Realizar un escaneo de la red, en el apartado "*Proxy/Intercept*" dentro de la herramienta de Burp Suite existe un botón "*Intercep is off*", para iniciar es necesario dar clic, como se observa en la Figura 76: Interceptar tráfico con Burp Suite

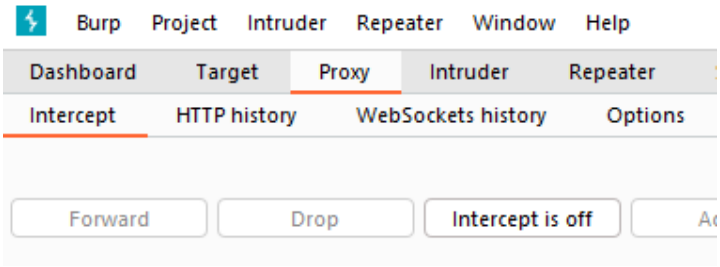

Figura 76: Interceptar tráfico con Burp Suite

4. Una vez instalado la aplicación dentro de nuestro emulador, lo siguiente es interactuar con la aplicación en ejecución ya que de esta manera Burp Suite comienza a interceptar todas las peticiones que va solicitando la aplicación, como se observa en Figura 77: Pruebas de seguridad con Burp Suite

Figura 77: Pruebas de seguridad con Burp Suite

Durante el análisis la aplicación fue sometida a diferentes pruebas de seguridad con ataques comunes, verificando la resistencia a dichos ataques y el nivel de seguridad en el cual la aplicación se encontraba actualmente, para así obtener un reporte final de vulnerabilidades.

Los resultados finales obtenidos durante el análisis son:

1. **Extracción de direcciones ip:** Se obtuvieron las direcciones de los servidores.
2. **Acceso al nombre del servidor:** Se obtuvo nombre y versión del servidor.
3. **Acceso a llamadas de métodos HTTP:** Se obtuvieron métodos http con vulnerabilidades.
4. **Acceso a información de clases y métodos:** Se logró mapear todas las llamadas a las funciones que se hacían durante el inicio de cada actividad de la aplicación.
5. **Acceso a información de contactos:** Se obtuvo registros de todos los contactos.
6. **Detección de criptografía.**
7. **Acceso a información de dispositivos:** Se obtuvo acceso a todos los dispositivos registrados de cada usuario.

4.3. Conclusión

Durante el análisis de las pruebas SAST y DAST se contemplaron herramientas de código abierto, donde se considera cuatro tipos de nivel de peligrosidad, como se observa en la Tabla 7: Nivel de peligrosidad final, dejando así, en aceptación aquellas vulnerabilidades con nivel de riesgo informativo.

COLOR	NIVEL DE PELIGROSIDAD
Marrón	Crítico
Rojo	Alto
Amarillo	Medio
Verde	Bajo

Tabla 7: Nivel de peligrosidad final

1. Marrón: El código debe ser revisado y corregido inmediatamente.
2. Rojo: El código debe revisado.
3. Amarillo: Defecto de calidad que puede tener un impacto en la productividad del desarrollador: fragmento de código descubierto, bloques duplicados, parámetros no utilizados. Estos errores pueden afectar a la vida futura del software.

4. Verde: Defecto de calidad que puede afectar ligeramente la productividad del desarrollador.

Así obteniendo una comparativa general de las 2 herramientas utilizadas para el análisis estático, como se observa en la Figura 78: Nivel de peligrosidad general de las pruebas SAST, Android y Figura 79: Nivel de peligrosidad general de las pruebas SAST MobSF, iOS

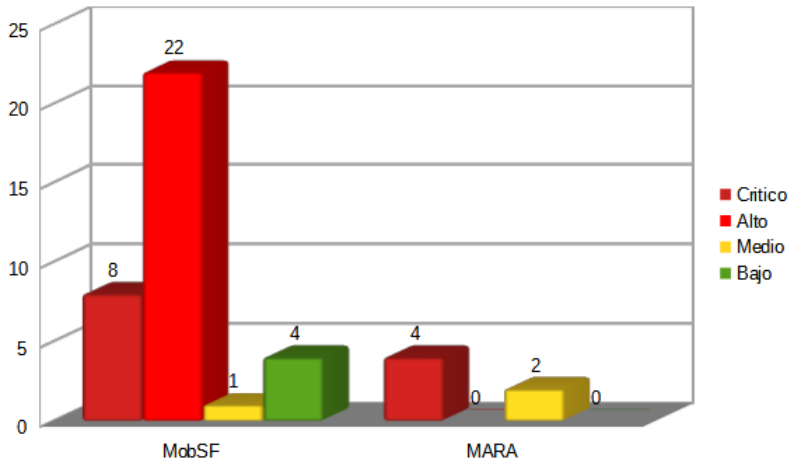

Figura 78: Nivel de peligrosidad general de las pruebas SAST, Android

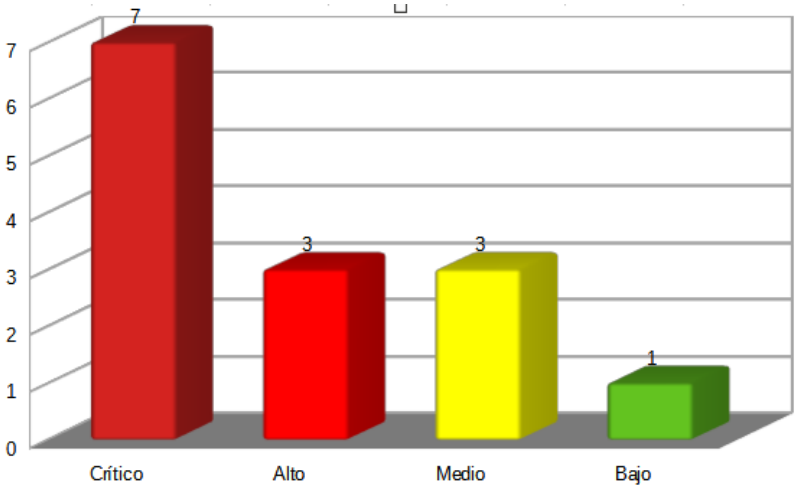

Figura 79: Nivel de peligrosidad general de las pruebas SAST MobSF, iOS

Finalmente se realizó una comparativa de las vulnerabilidades obtenidas por los analizadores utilizados durante las pruebas, esto con el objetivo de disminuir

los resultados obtenidos y así poder visualizar el nivel de prioridad en el que se deben resolver o cubrir dichas vulnerabilidades.

Se concluye con 14 vulnerabilidades para el sistema Android y 10 vulnerabilidades para el sistema iOS. Como se observa en la Figura 80: Total de vulnerabilidades según su nivel de peligrosidad

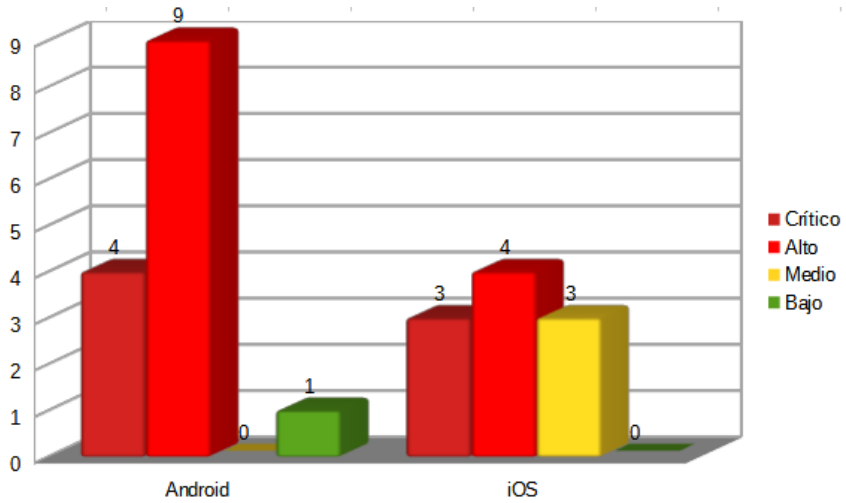

Figura 80: Total de vulnerabilidades según su nivel de peligrosidad

CAPÍTULO V

5. PROTECCIÓN DE VULNERABILIDADES

Los resultados finales de las 24 vulnerabilidades se observa en la Tabla 8: Resultados generales de pruebas SAST Y DAST. Se observa de manera general:

NIVEL	VULNERABILIDAD
CRÍTICO	Anti-Depuración e ingeniería inversa.
	Ofuscamiento.
	Uso de métodos criptográficos para cifrar las consultas a la base de datos SQLite.
	Permisos innecesarios.
	Seguridad de transporte de aplicaciones (ATS). Permite cargas arbitrarias.
ALTO	Neutralización incorrecta de elementos especiales utilizados en un comando SQL ('Inyección SQL').
	Actualización de Criptografía.
	Uso de API(s) potencialmente peligrosa(s).
	Segmentos restringidos.
	Sin control de asignación de memoria.
	Detección de dispositivos con *Jailbroken.*
	Uso de filtros de intención no protegidos.
	Uso de servicios protegidos por un permiso sin antes realizar la comprobación del nivel de protección del permiso.
	Uso de *Broadcast Receiver* (Receptor de Transmisión) protegido por un permiso sin antes realizar la comprobación del nivel de protección del permiso.
	Almacenamiento de texto sin cifrar información confidencial.
	Permisos predeterminados incorrectos.
	Uso de valores insuficientemente aleatorios.
Medio	Uso de función potencialmente peligrosa.
	Símbolos de depuración.
Bajo	Técnicas de detección de dispositivos root.

Tabla 8: Resultados generales de pruebas SAST Y DAST

5.1. MITIGACIÓN DE VULNERABILIDADES

En este apartado se retomarán los puntos expuestos en la Tabla 8: Resultados generales de pruebas SAST Y DAST acerca de las vulnerabilidades encontradas durante las pruebas de seguridad de aplicaciones estáticas(SAST) y las pruebas de seguridad de aplicaciones dinámicas(DAST), así como también se

muestran las soluciones para dichas vulnerabilidades y su clasificación por *OWASP Mobile Top 10.*

1. Anti-depuración e ingeniería inversa.

OWASP Mobile Top 10, M9(Ingeniería inversa): La depuración de una aplicación facilita aplicar ingeniería inversa. Esto permite volcar un seguimiento de pila y acceder a clases auxiliares. Para mitigar esta vulnerabilidad se recomienda lo siguiente:

En el caso de Android, existen dos posibles técnicas de sistemas de depuración:[29]

1. JDWP (*Java Debug Wire Protocol*), que es el protocolo de comunicación de depuración entre un depurador y la máquina virtual Java.
2. Ptrace actúa sobre el kernel nativo de Android Linux el cual se usa para observar y controlar la ejecución de un proceso(*tracee*) de igual manera para examinar y cambiar la memoria y los registros de ese proceso.

En el caso de iOS, existen dos posibles técnicas de sistemas de depuración: [30]

1. PT_DENY_ATTACH es una técnica muy conocida para implementar anti-depuración en iOS. El uso de “ptrace” con “PT_DENY_ATTACH” valida que ningún otro depurador puede adjuntar al proceso protegido. Si un depurador intenta adjuntar, el proceso terminará.
2. Sysctl para detectar un depurador adjunto a un proceso. Sysctl es una función de la API de iOS que puede recuperar información sobre un proceso, especialmente si ha sido depurado.

2. Ofuscamiento.

OWASP Mobile Top 10, M8(Manipulación de código): La aplicación carece de técnicas de ofuscamiento, como cifrado de datos del programa o del código fuente. Esto permite que el código de la aplicación sea alterado por un atacante o en su defecto ocultar código malicioso.[31]

Una de las herramientas más utilizadas para ofuscar código es ProGuard, realiza las siguientes acciones:

1. Elimina variables, clases, métodos y atributos no utilizados.

2. Elimina instrucciones innecesarias.
3. Elimina la información de depuración.
4. Renombra clases, campos y métodos con nombres poco legibles.

Existe otra herramienta para realizar dichas tareas como el compilador R8 donde viene integrado en el complemento de Android para Gradle 3.4.0 o versiones posteriores donde realiza las siguientes tareas en tiempo de compilación: [32]

1. Reducción de código(o eliminación de código obsoleto).
2. Reducción de recursos.
3. Ofuscación(Acorta el nombre de las clases y los miembros, lo que genera archivos DEX de menor tamaño).
4. Optimización.

3. Uso de métodos criptográficos para cifrar las consultas a la base de datos SQLite.

OWASP Mobile Top 10, M7(Calidad del código de cliente): Se detectó que la aplicación hace uso de una base de datos SQLite, donde no se encontró el uso consultas de manera cifrada en sus peticiones.[33]

Se recomienda el uso de los siguientes métodos para para cifrar o descifrar bases de datos:

1. SQLCipher es una biblioteca de código abierto que proporciona cifrado AES de 256 bits transparente y seguro de archivos de base de datos SQLite.
2. SEE(*SQLite Encryption Extension)*, SEE proporciona una manera fácil de crear, leer y escribir archivos de base de datos cifrados. Se puede usar con los enlaces SQLite de Android para agregar capacidad de base de datos encriptada a cualquier aplicación.

4. Permisos innecesarios.

OWASP Mobile Top 10, M4(Autorización insegura): Se detectaron algunos permisos otorgados de un nivel de riesgo crítico dentro de las funciones en la aplicación.[34]

Se recomienda realizar un análisis de los permisos innecesarios y posteriormente deshabilitar los permisos que no están siendo usados en la

aplicación, ya que algunos permisos ponen en riesgo la integridad de los datos de nuestra aplicación.

5. Seguridad de transporte de aplicaciones(ATS). Permite cargas arbitrarias.

OWASP Mobile Top 10, M3(Comunicación insegura): Las restricciones de seguridad de transporte de aplicaciones están deshabilitadas para todas las conexiones de red, significa que se permiten conexiones HTTP no seguras.[35]

App Transport Security(ATS) es un conjunto de comprobaciones de seguridad que el sistema operativo realiza al llevar a cabo conexiones a través de "*NSURLConnection, NSURLSession"* y "CFURL" a nombres de dominios públicos.

Se recomienda que si ATS se encuentra activado se requiere de los siguientes requisitos de seguridad:

1. No permitir conexiones HTTP.
2. El certificado X.509 tiene una huella digital SHA256 y debe estar firmado con al menos una clave RSA de 2048 bits o una clave "*Elliptic-Curve Cryptography(ECC-256)".*
3. La versión de "*Transport Layer Security(TLS)"* debe ser 1.3 o superior y debe admitir "*Perfect Forward Secrecy(PFS)"* mediante el intercambio de claves "*Elliptic Curve Diffie-Hellman Ephemeral(ECDHE)"* y cifrados simétricos AES-256.

Se recomienda que si ATS se encuentra desactivado se pueden configurar excepciones donde se pueden aplicar globalmente o individualmente para cada uno de los dominios definidos como:

1. Permitir conexiones inseguras (HTTP).
2. Utilizar una versión más antigua de TLS.
3. Deshabilitar "Perfect Froward Secrecy(PFS)".
4. Permitir conexiones a dominios locales.

Para generar excepciones agregar "NSExceptionDomians" dentro de la llave de "NSAppTransportSecurity"*.*

6. Neutralización incorrecta de elementos especiales utilizados en un comandos SQL (‘Inyección de SQL’).

OWASP Mobile Top 10, M7(Calidad del código de cliente): Una falla de inyección describe una clase de vulnerabilidad de seguridad que ocurre cuando la entrada del usuario se inserta en consultas o comandos de backend. Al inyectar metacaracteres, un atacante puede ejecutar código malicioso que, inadvertidamente, se interpreta como parte del comando o la consulta.[36]

Se recomienda tomar las siguientes precauciones:

1. Establecer el indicador “SQLITE_DBCONFIG_DEFENSIVE”. Esto evita que las sentencias SQL ordinarias corrompan deliberadamente el archivo de base de datos. SQLite debe ser a prueba de ataques que involucran tanto entradas SQL maliciosas como un archivo de base de datos dañado maliciosamente al mismo tiempo. Sin embargo, negar el acceso de un atacante que solo utiliza un script a las entradas de bases de datos corruptas proporciona una capa adicional de defensa.
2. Reducir los límites que SQLite impone a las entradas. Esto puede ayudar a prevenir ataques de denegación de servicio y otros tipos de daños que pueden ocurrir como resultado de entradas inusualmente grandes. Se puede hacer esto en tiempo de compilación usando las opciones de “DSQLITE_MAX” o en tiempo de ejecución usando la interfaz “sqlite3_limit()”.
3. Usar la interfaz “sqlite3_set_authorizer()” para limitar el alcance de SQL que se procesa. Por ejemplo, una aplicación que no necesita cambiar el esquema de la base de datos puede agregar una devolución de llamada “sqlite3_set_authorizer()” que hace que falle cualquier instrucción *CREATE* o *DROP.*
4. Limitar la cantidad máxima de memoria que SQLite asignará utilizando la interfaz “sqlite3_hard_heap_limit64()”. Esto ayuda a prevenir ataques de denegación de servicio. Para averiguar cuánto espacio de pila realmente necesita una aplicación, ejecutarlo contra entradas típicas y luego mida el uso máximo de memoria instantánea con la interfaz “sqlite3_memory highwater()”. Establecer el límite de almacenamiento dinámico en el uso máximo de memoria instantánea observado más algún margen.

7. Actualización de criptografía.

OWASP Mobile Top 10, M5(Criptografía insuficiente): Se detectó el uso de algoritmos y protocolos criptográficos en la aplicación que actualmente presentan debilidades conocidas. Se recomienda evitar el uso de los siguientes algoritmos y protocolos, como:[37]

1. DES, 3DES.
2. RC2.
3. RC4.
4. BLOWFISH.
5. MD4.
6. MD5.
7. SHA1.

Para mejorar la seguridad de nuestros datos se recomienda los siguientes algoritmos:[38]

1. Algoritmos de confidencialidad: AES-GCM-256 o ChaCha20-Poly1305.
2. Algoritmos de integridad: SHA-256, SHA-384, SHA-512, Blake2, la familia SHA-3.
3. Algoritmos de firma digital: RSA (3072 bits y superior), ECDSA con NIST P-384.
4. Algoritmos de establecimiento de claves: RSA (3072 bits y superior), DH (3072 bits o superior), ECDH con NIST P-384.

8. Uso de API(s) potencialmente peligrosa(s).

OWASP Mobile Top 10, M7(Calidad del código de cliente): La aplicación hace uso de API(s) inseguras, como fopen, strlen, memcpy, sscanf.

Se recomienda para cada una de las API(s) lo siguiente:

1. Fopen_s, wfopen tienen mejoras de seguridad. Los archivos que abren “fopen_s” y “wfopen_s” no se pueden compartir. Si necesita que un archivo se pueda compartir, utilice “_fsopen”, “_wfsopen” con la constante de modo de uso compartido adecuada.[39]
2. Strnlen no sustituye a “strlen; strnlen” está diseñado para usarse solo para calcular el tamaño de los datos entrantes que no son de confianza en un

búfer de tamaño conocido “strnlen” calcula la longitud pero no pasa del final del búfer si la cadena no está terminada.[40] Para el uso de “strnlen” es recomendable añadir un tamaño fijo lo cual se puede utilizar:

1. size_t MaxPossibleSize = startOfSharedMemory + sizeOfSharedMemory – input;

   strnlen(input, MaxPossibleSize);

3. Memcpy_s, wmemcpy_s con mejoras de seguridad de la función “mempcy”. La función “memcpy_s” copia “count bytes” de “src a dest” y “wmemcpy_s” copia cuentas con caracteres anchos(dos bytes). Si el origen y el destino se superponen, el comportamiento de “memcpy_s” no está definido. Utilice “memmove_s” para manejar regiones superpuestas. Estas funciones validan sus parámetros.[41]
4. Usar de “sscanf” para evitar desbordamientos, en la función “sscanf” indicar la anchura máxima, también es recomendable delimitar los caracteres admisibles. Se recomienda el uso de "fgets" para hacer la lectura en una cadena de texto y luego extraer la información de ella con "sscanf".[42]

9. Segmentos restringidos.

OWASP Mobile Top 10, M7(Calidad del código de cliente): El binario no cuenta con un segmento restringido que evite la carga dinámica de “dylib” para la inyección de código arbitrario.[43]

Se recomienda el uso de la función “hasRestrictedSegment()” ya que detecta específicamente el segmento “RESTRICT”, cualquier proceso dentro de esta función está configurada para restringir la inserción de biblioteca dinámica. Sin embargo, la detección de “_RESTRICT” se ha eliminado de la nueva versión del código fuente de “dyld”. A partir de iOS 10, este método de protección ha dejado de ser válido.

10. Sin control de asignación de memoria.

OWASP Mobile Top 10, M7(Calidad del código de cliente): El binario hace uso de función “malloc”.

Se recomienda el uso de la función “free()” ya que la memoria reservada por “malloc” se liberará automáticamente al final de cada transacción con la función “free()”, así previniendo fallos de memoria.[44]

11. Detección de dispositivos con *Jailbroken.*

OWASP Mobile Top 10, M9(Ingeniería inversa): La detección de dispositivos con "*jailbreak*" proporciona resistencia contra la ingeniería inversa y cierto tipo de ataques específicos en el lado del cliente. Si bien este tipo de seguridad denominados como "*jailbreak detection*" no son muy efectivos por sí solos, añadir algunos de estos mecanismos dentro de la aplicación sí que pueden ayudar a mejorar el proceso de "*anti-tampering*" implementando en la misma.[45]

Para la detección de esta vulnerabilidad se recomienda una serie de técnicas:

1. Comprobaciones basadas en ficheros, esta técnica consiste en comprobar si existe algún fichero y directorio asociados normalmente a las técnicas de "*jailbreak*".
2. Comprobaciones basadas en permisos de ficheros, esta técnica consiste en comprobar si se crea un fichero fuera del "*sandbox*" de la aplicación. Si el fichero se crea correctamente entonces se podrá inferir que el dispositivo está "*jailbroken*".
3. Verificación de llamada de la *App Store* de *Cydia*, aplicación que se instala por defecto en casi todas las versiones de "*jailbreak"*.

12. Uso de filtros de intención no protegidos.

OWASP Mobile Top 10, M7(Calidad del código de cliente): Se detectaron actividades que comparten con otras aplicaciones en el dispositivo, por lo que se deja accesible para cualquier otra aplicación. La presencia de un "intent-filter" indica que la actividad se está exportando explícitamente.[46]

Para garantizar que tu aplicación sea segura, como buena práctica utilizar un "intent" explícito cuando inicies un "*Service*" y no declarar filtros de "intents" para los servicios. A partir de Android 5.0 (nivel de API 21), el sistema arroja una excepción si llamas a "bindService()" con un "intent" implícito.

13. Uso de servicios protegidos por un permiso sin antes realizar la comprobación del nivel de protección del permiso.

OWASP Mobile Top 10, M7(Calidad del código de cliente): Se detectó el uso de algunos servicios que se comparten con otras aplicaciones en el dispositivo, por lo tanto, lo deja accesible para cualquier otra aplicación en el dispositivo.[47]

Se recomienda restringir las interacciones con los servicios de la aplicación para eso se debe verificar el permiso durante “Context.startService()”, “ContextstopService()” y “Context.bindService()”.

14. Uso de Broadcast Receiver (Receptor de Transmisión) protegido por un permiso sin antes realizar la comprobación del nivel de protección del permiso.

OWASP Mobile Top 10, M3(Comunicación insegura): Se encontró el uso de algunos *Broadcast Receiver* (Receptor de Transmisión) que se comparten con otras aplicaciones en el dispositivo, por lo tanto, lo deja accesible para cualquier otra aplicación en el dispositivo.[48]

Se recomienda verificar el permiso después de que se muestra “Context.sendBroadcast()”, mientras el sistema intenta proporcionar la emisión enviada al receptor en cuestión. De la misma manera se puede suministrar un permiso a “Context.sendBroadcast()” a fin de restringir los receptores de emisión que tienen autorización para recibirla.

15. Almacenamiento de texto sin cifrar información confidencial.

OWASP Mobile Top 10, M9(Ingeniería inversa): Se encontró el almacenamiento de datos confidenciales en la memoria, tales como nombres de usuario, contraseñas, claves, etc. Por lo que si un atacante puede identificar esta información y utilizarla para ataques adicionales como ingeniería social, secuestro de cuenta (si se ha divulgado información de sesión o un token de autenticación) y recopilación de información de aplicaciones con opciones de pago(para atacarlos y abusar de ellos).[49]

Es recomendable evitar el almacenamiento externo de datos privados como los datos están visibles pueden ser tomados por otros usuarios y aplicaciones. También es recomendable añadir un “android:installLocation” con un valor de “internalOnly” al archivo de manifiesto para negar a los usuarios el de poder instalar la aplicación en almacenamiento externo.

16. Permisos predeterminados incorrectos.

OWASP Mobile Top 10, M2(Almacenamiento inseguro de datos): Se detectó el almacenamiento inseguro de datos confidenciales en archivos temporales.[50]

Se recomienda las siguientes técnicas para mitigar dicha vulnerabilidad:

1. Usar “mkstemp()” es una forma razonablemente segura de crear archivos temporales. Intentará crear y abrir un archivo único basado en una plantilla de nombre de archivo proporcionada por el usuario, combinado con una serie de caracteres generados aleatoriamente. Tenga en cuenta que “mkstemp()” es seguro si solo se utiliza el descriptor y el nombre de archivo devuelto no se utiliza en una llamada de función posterior con privilegios adicionales.
2. Usar el método deleteOnExit() para borrar el fichero cuando termine nuestro programa.

17. Uso de valores insuficientemente aleatorios.

OWASP Mobile Top 10, M5(Criptografía insuficiente): La aplicación usa un generador de números aleatorios, donde es fundamentalmente imposible producir números verdaderamente aleatorios en cualquier dispositivo determinista.[51]

Para el uso de valores aleatorios se recomienda el uso de la clase de “SecureRandom()”, ya que proporciona un generador de números aleatorios (RNG) criptográficamente fuerte, produciendo una salida no determinista.

18. Uso de función potencialmente peligrosa.

OWASP Mobile Top 10, M7(Calidad del código de cliente): El binario tiene establecido “*RUNPATH SEARCH PATH*”, que busca recursos críticos mediante una ruta de búsqueda proporcionada externamente que puede apuntar a recursos que no están bajo el control directo de la aplicación En ciertos casos, un atacante puede abusar de esta función, ejecutar sus propios programas, acceder a archivos de datos no autorizados o modificar la configuración de formas inesperadas. [52]

Para evitar esta vulnerabilidad se recomienda eliminar o restringir todas las configuraciones del entorno antes de invocar otros programas. Esto incluye la variable de entorno *“PATH, RPATH, LD_LIBRARY_PATH*”, y otras configuraciones que identifican la ubicación de las bibliotecas de códigos y cualquier ruta de búsqueda específica de la aplicación.

19. Símbolos de depuración.

OWASP Mobile Top 10, M7(Calidad del código de cliente): Los símbolos están disponibles en general, esta configuración garantiza que los símbolos de depuración formen parte del binario distribuido.[53]

Se recomienda eliminar los símbolos de depuración, configurando "*Strip Debug Symbols During Copy*" durante la copia a *YES*, el "*Deployment Postprocessing"* a *YES* y "*Strip Linked Product*" a YES en la configuración de construcción del proyecto.

20. Técnicas de detección de dispositivos root.

OWASP Mobile Top 10, M9(Ingeniería inversa): La aplicación no cuenta con detección de *root.* La detección de dispositivos *root* es recomendable para evitar la ejecución de la aplicación en un dispositivo rooteado, lo que a su vez también bloquea algunas de las herramientas y técnicas de ingeniería inversa.[29]

Algunas técnicas para la detección de dispositivos *root* son:

1. SafetyNet es una API de Android que proporciona un conjunto de servicios y crea perfiles de dispositivos de acuerdo con la información de software y hardware. Luego, este perfil se compara con una lista de modelos de dispositivos aceptados que han pasado las pruebas de compatibilidad con Android.
2. Usar detección programática donde se suele buscar binarios que normalmente se instalan una vez que se ha rooteado un dispositivo.
3. Detección de *Bypassing Root* se realiza una evaluación del comportamiento de la aplicación a nivel de módulos del kernel.

CAPÍTULO VI

6. RESULTADOS Y CONCLUSIONES

6.1. RESULTADOS

Como producto final durante el desarrollo de la presente tesis, se entrega un reporte de seguridad sin falsos positivos de un software de mensajería segura multiplataforma instalable en sistemas operativos móviles(Android y iOS), donde se consideró cuatro tipos de nivel de peligrosidad, como se observa en la Tabla 7: Nivel de peligrosidad final. El reporte de seguridad incluye:

1. Descripción de vulnerabilidades descubiertas.
2. Clasificación de riesgo.
3. Impacto de la vulnerabilidad y el perfil de riesgo según los estándares(OWASP).
4. Estrategias de mitigación y enfoques de defensa en profundidad para desarrolladores y/o administradores.

En la Tabla 9: Reporte de Seguridad Android se muestra el reporte final de la aplicación de mensajería segura para el sistema móvil Android.

No	CASO	DESCRIPCIÓN	INFORMACIÓN	RECOMENDACIÓN
1	Anti-depuración(*Anti-Debugging)* e ingeniería inversa.	La depuración se habilitó en la aplicación (Android: debuggable ="true"), lo que facilita aplicar ingeniería inversa con algún depurador. Esto permite volcar un seguimiento de pila y acceder a clases auxiliares.	OWASP Mobile Top 10: M9(Ingeniería Inversa). Referencia: 1. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md/	En Android, existen dos posibles técnicas de sistemas de depuración: 1. JDWP (*Java Debug Wire Protocol*), que es el protocolo de comunicación de depuración entre un depurador y la máquina virtual Java. 2. Ptrace actúa sobre el kernel nativo de Android Linux el cual se usa para observar y controlar la ejecución de un proceso(tracee) de igual manera para examinar y cambiar la memoria y los registros de ese proceso.
2	Ofuscamiento	La aplicación carece de	OWASP Mobile Top	Habilitar el ofuscamiento en la

	de la aplicación.	técnicas de ofuscamiento, como cifrado de datos del programa o del código fuente. Esto permite que el código de la aplicación sea alterado por un atacante o en su defecto ocultar código malicioso.	10: M8 (manipulación de código). Referencia: 1. https://developer.android.com/studio/build/shrink-code?hl=es%2F 2. https://owasp.org/www-community/controls/Bytecode_obfuscation	aplicación acorta los nombres de las clases, optimiza la aplicación y aplica estrategias más agresivas a fin de reducir aún más el tamaño de la aplicación. Una de las herramientas más utilizadas para ofuscar código es *ProGuard*, realiza las siguientes acciones: 1. Elimina variables, clases, métodos y atributos no utilizados. 2. Elimina instrucciones innecesarias. 3. Elimina la información de depuración. 4. Renombra clases, campos y métodos con nombres poco legibles. Existe otra herramienta para realizar dichas tareas como el compilador R8 donde viene integrado en el complemento de Android para Gradle 3.4.0 o versiones posteriores donde realiza las siguientes tareas en tiempo de compilación: 1. Reducción de código(o eliminación de código obsoleto). 2. Reducción de recursos. 3. Ofuscación(Acorta el nombre de las clases y los miembros, lo que genera archivos DEX de menor tamaño). 4. Optimización.
3	Uso de métodos criptográficos para cifrar las consultas a la base de datos SQLite.	Se detectó que la aplicación hace uso de una base de datos SQLite, donde no se encontró el uso consultas de manera cifrada en sus peticiones.	OWASP Top 10: M7(Calidad del código de cliente). Referencia: 1. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md 2. http://sqlcipher.n	Se recomienda el uso de los siguientes métodos para para cifrar o descifrar bases de datos: 1. SQLCipher: Es una biblioteca de código abierto que proporciona cifrado AES de 256 bits transparente y seguro de archivos de base de datos SQLite. 2. SQLite *Encryption Extension* (SEE) on Android: SEE proporciona una manera fácil de crear, leer y escribir archivos de base de datos cifrados. Se puede usar con los enlaces SQLite de Android para agregar capacidad de base de datos encriptada a cualquier aplicación.

			3. https://sqlite.org/android/doc/trunk/www/see.wiki	
4	Permisos innecesarios	Se detectaron algunos permisos otorgados de un nivel de riesgo crítico dentro de las funciones en la aplicación, se listan los permisos: 1. android.permission.ACCESS_LOCATION. 2. android.permission.ACCESS_FINE_LOCATION. 3. android.permission.CAMERA. 4. android.permission.RED_CONTACTS. 5. android.permission.RED_EXTERNAL_STORAGE. 6. android.permission.RECORD_AUDIO. 7. android.permission.SYSTEM_ALERT_WINDOW. 8. android.permission.WRITE.EXTERNAL_	OWASP Mobile Top 10: M4(Autorización insegura). Referencia: 1. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md	Se recomienda realizar un análisis de los permisos innecesarios y posteriormente deshabilitar los permisos que no están siendo usados en la aplicación, ya que algunos permisos ponen en riesgo la integridad de los datos de nuestra aplicación.
5	Neutralización incorrecta de elementos especiales utilizados en un comando SQL ('Inyección SQL').	Una falla de inyección describe una clase de vulnerabilidad de seguridad que ocurre cuando la entrada del usuario se inserta en consultas o comandos de *backend*. Al inyectar metacaracteres, un atacante puede ejecutar código malicioso que, inadvertidamente, se interpreta como parte del comando o la consulta. Por ejemplo, al manipular una consulta SQL, un atacante podría recuperar registros arbitrarios de la base de datos o manipular el contenido de la base de datos *backend.*	OWASP Top 10: M7(Calidad del código de cliente). Referencia: 1. https://runebook.dev/es/docs/sqlite/security/ 2. http://es.uwenku.com/question/p-yjcfmvax-bq.html/	Las aplicaciones que aceptan entradas SQL no fiables deben tomar las siguientes precauciones: 1. Establezca el indicador “SQLITE_DBCONFIG_DEFENSIVE”. Esto evita que las sentencias SQL ordinarias corrompan deliberadamente el archivo de base de datos. SQLite debe ser a prueba de ataques que involucran tanto entradas SQL maliciosas como un archivo de base de datos dañado maliciosamente al mismo tiempo. Sin embargo, negar el acceso de un atacante que solo utiliza un script a las entradas de bases de datos corruptas proporciona

				una capa adicional de defensa. 2. Reducir los limites que SQLite impone a las entradas. Esto puede ayudar a prevenir ataques de denegación de servicio y otros tipos de daños que pueden ocurrir como resultado de entradas inusualmente grandes. Puede hacer esto en tiempo de compilación usando las opciones de "DSQLITE_MAX" o en tiempo de ejecución usando la interfaz "sqlite3_limit()". 3. Considere usar la interfaz "sqlite3_set_authorizer()" para limitar el alcance de SQL que se procesará. Por ejemplo, una aplicación que no necesita cambiar el esquema de la base de datos puede agregar una devolución de llamada "sqlite3_set_authorizer ()" que hace que falle cualquier instrucción "CREATE" o "DROP". 4. Limite la cantidad máxima de memoria que SQLite asignará utilizando la interfaz "sqlite3_hard_heap_limit64 ()". Esto ayuda a prevenir ataques de denegación de servicio. Para averiguar cuánto espacio de pila realmente necesita una aplicación, ejecútela contra entradas típicas y luego mida el uso máximo de memoria instantánea con la interfaz "sqlite3_memory highwater()". Establezca el límite de almacenamiento dinámico en el uso máximo de memoria instantánea

				observado más algún margen.
6	Actualización de Criptografía.	Se detectó el uso de algoritmos y protocolos criptográficos en la aplicación que actualmente presentan debilidades conocidas, evitar el uso de los siguientes algoritmos y protocolos, como: 1. DES, 3DES. 2. RC2. 3. RC4. 4. BLOWFISH. 5. MD4. 6. MD5. 7. SHA1.	OWASP Mobile Top 10: M5 (Criptografía insuficiente). Referencia: 1. https://github.com/MobSF/owasp-mstg/blob/master/Document/0x04g-Testing-Cryptography.md 2. https://www.keylength.com/	Se recomiendan los siguientes algoritmos: 1. Algoritmos de confidencialidad: "AES-GCM-256 o ChaCha20-Poly1305". 2. Algoritmos de integridad: "SHA-256, SHA-384, SHA-512, Blake2, la familia SHA-3". 3. Algoritmos de firma digital: "RSA (3072 bits y superior), ECDSA con NIST P-384". 4. Algoritmos de establecimiento de claves: "RSA (3072 bits y superior), DH (3072 bits o superior), ECDH con NIST P-384".
7	Uso de filtros de intención no protegidos.	Se detectaron actividades que comparten con otras aplicaciones en el dispositivo, por lo que se deja accesible para cualquier otra aplicación. La presencia de un "*intent-filter*" indica que la actividad se está exportando explícitamente. Se listan las siguientes actividades detectadas: 1. (im.vector.app.gplay.push.fcm.VectorFirebaseMessagingService). 2. (im.vector.app.features.Alias). 3. im.vector.app.features.login.LoginActivity). 4. (im.vector.app.features.link.LinkHandlerActivity). 5. (im.vector.app.features.share.IncomingShareActivity). 6. (im.vector.app.features.permalink.PermalinkHandlerActivity) 7. (androidx.media.session.MediaButtonRe	OWASP Mobile Top 10: M7(Calidad del código de cliente). Referencia: 1. https://developer.android.com/guide/components/intents-filters?hl=es-419	Para garantizar que tu aplicación sea segura, como buena practica utiliza una "*intent*" explícita cuando inicies un "*Service*" y no declares filtros de intents para los servicios. El uso de una "*intent*" implícita para iniciar un servicio es un riesgo de seguridad porque no puedes garantizar qué servicio responderá a la "*intent*", y el usuario no puede ver qué servicio se inicia. A partir de Android 5.0 (nivel de API 21), el sistema arroja una excepción si llamas a "bindService()" con una "*intent*" implícita.

		ceiver).		
8	Uso de servicios protegidos por un permiso sin antes realizar la comprobación del nivel de protección del permiso.	Se detectó el uso de algunos servicios que se comparten con otras aplicaciones en el dispositivo, por lo tanto, lo deja accesible para cualquier otra aplicación en el dispositivo. Los servicios están protegidos por permisos que no están definidos en la aplicación analizada. Como resultado, el nivel de protección de los permisos debe verificarse donde están definidos. Si están configurados como normal o peligroso, una aplicación malintencionada puede solicitar y obtener el permiso e interactuar con el componente. Si están configurados como firma, solo las aplicaciones firmadas con el mismo certificado pueden obtener el permiso. Se listan los siguientes servicios detectados: 1. (im.vector.app.features.call.telecom.VectorConnectionService) protegido por el permiso; android.permission.BIND_TELECOM_CONNECTION_SERVICE [android:exported=true]. 2. (org.jitsi.meet.sdk.ConnectionService) protegido por el permiso; android.permission.BIND_TELECOM_CONNECTION_SERVICE [android:exported=true]. 3. (androidx.sharetarget.ChooserTargetServiceCompat) protegido por el permiso; android.permission.BIND_CHOOSER_TA	OWASP Mobile Top 10: M7(Calidad del código de cliente). Referencia: 1. https://developer.android.com/training/permissions/restrict-interactions?hl=es-419#services/	Los permisos no solo se utilizan para solicitar funcionalidades del sistema. También puedes restringir las maneras en las que otras aplicaciones pueden interactuar con los componentes de nuestra aplicación. Se recomienda restringir las interacciones con los servicios de la aplicación para eso se debe verificar el permiso durante “Context.startService()”, “ContextstopService()” y “Context.bindService()”.

		RGET_SERVICE [android:exported=true]. 4. (androidx.work.impl.background.systemjob.SystemJobService) protegido por el permiso; android.permission.BIND_JOB_SERVICE [android:exported=true].		
9	Uso de *Broadcast Receiver* (Receptor de Transmisión) protegido por un permiso sin antes realizar la comprobación del nivel de protección del permiso.	Se encontró el uso de algunos *Broadcast Receiver* (Receptor de Transmisión) que se comparten con otras aplicaciones en el dispositivo, por lo tanto, lo deja accesible para cualquier otra aplicación en el dispositivo. Están protegidos por un permiso que no está definido en la aplicación analizada. Como resultado, el nivel de protección del permiso debe verificarse donde está definido. Si está configurado como normal o peligroso, una aplicación malintencionada puede solicitar y obtener el permiso e interactuar con el componente. Si está configurado como firma, solo las aplicaciones firmadas con el mismo certificado pueden obtener el permiso. Se listan los siguientes *Broadcast Receiver* detectados: 1. (com.google.firebase.iid.FirebaseInstanceIdReceiver) protegido por el permiso; com.google.android.c2dm.permission.SEND [android:exported=true]. 2. (androidx.work.impl.diagnostics.DiagnosticsReceiver) protegido por el permiso;	OWASP Mobile Top 10: M7(Calidad del código de cliente). Refrencia: 1. https://developer.android.com/training/permissions/restrict-interactions?hl=es-419#services/	Se recomienda verificar el permiso después de que se muestra “Context.sendBroadcast()”, mientras el sistema intenta proporcionar la emisión enviada al receptor en cuestión. De la misma manera se puede suministrar un permiso a “Context.sendBroadcast()” a fin de restringir los receptores de emisión que tienen autorización para recibirla.

		android.permission.DUMP [android:exported=true].		
10	Almacenamiento de texto sin cifrar información confidencial.	Se encontró el almacenamiento de datos confidenciales en la memoria, tales como nombres de usuario, contraseñas, claves, etc. Por lo que si un atacante puede identificar esta información y utilizarla para ataques adicionales como ingeniería social, secuestro de cuenta (si se ha divulgado información de sesión o un *token* de autenticación) y recopilación de información de aplicaciones con opciones de pago(para atacarlos y abusar de ellos).	OWASP Mobile Top 10: M9(Ingenieria inversa). Referencia: 1. https://code.tutsplus.com/es/tutorials/storing-data-securely-on-android--cms-30558/ 2. https://github.com/MobSF/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#checking-memory-for-sensitive-data-mstg-storage-10/	Es recomendable evitar el almacenamiento externo de datos privados como los datos están visibles pueden ser tomados por otros usuarios y aplicaciones. De hecho, para evitarlo y hacerlo más difícil para la gente al copiar el binario de la aplicación y datos, se les puede negar a los usuarios el de poder instalar la aplicación en almacenamiento externo. Añadiendo un “android:installLocation” con un valor de “internalOnly” al archivo de manifiesto será lograrlo.
11	Permisos predeterminados incorrectos.	Se detecto el almacenamiento inseguro de datos dentro de la aplicación, como datos confidenciales en archivos temporales.	OWASP Mobile Top 10: M2(Almacenamiento Inseguro de Datos). Referencia: 1. https://owasp.org/www-community/vulnerabilities/Insecure_T	Asegúrese de que se utilizan nombres impredecibles para archivos temporales y que los archivos se crean en un directorio seguro con los permisos adecuados. El uso de “mkstemp()” es una forma razonablemente segura de crear archivos temporales. Intentará crear y abrir un archivo único basado en una plantilla de nombre de archivo proporcionada por el usuario, combinado con una serie de caracteres generados aleatoriamente. Tenga en cuenta que “mkstemp()” es seguro si solo se

			emporary_File 2. https://docs.oracle.com/javase/7/docs/api/java/nio/file/Files.html#createTempFile(java.lang.String,%20java.lang.String,%20java.nio.file.attribute.FileAttribute	utiliza el descriptor y el nombre de archivo devuelto no se utiliza en una llamada de función posterior con privilegios adicionales. El uso de "mkstemp()" no elimina por completo las condiciones de la carrera, pero proporciona una mejor protección que otros métodos. Java nos ofrece un conjunto de métodos muy extensos en la clase "File" y uno de ello es "createTempFile()". El método "createTempFile()" nos crea un fichero temporal en el directorio que le digamos o en el directorio temporal del sistema. Es muy importante saber que cuando estemos trabajando con archivos temporales en Java, estos por defecto no se borran. Para borrar el fichero cuando termine nuestro programa debemos llamar el método "deleteOnExit()".
12	Uso de valores insuficiente-mente aleatorios.	La aplicación usa un generador de números aleatorios, donde es fundamentalmente imposible producir números verdaderamente aleatorios en cualquier dispositivo determinista.	OWASP Mobile Top 10: M5(Criptografía insuficiente). Referencia: 1. owasp-mstg/0x04g-Testing-Cryptography.md at master · MobSF/owasp-mstg · GitHub 2. https://developer.android.com/reference/kotlin/java/security/SecureRandom	Los "RNG" criptográficamente seguros generan números aleatorios que pasan las pruebas estadísticas de aleatoriedad y son resistentes a los ataques de predicción (por ejemplo, es estadísticamente inviable predecir el siguiente número producido). La clase de "SecureRandom()" proporciona un generador de números aleatorios (RNG) criptográficamente fuerte, produciendo una salida no determinista.
13	Permisos	La aplicación puede leer /	OWASP Mobile Top	Si aún se requiere que los datos

	predeterminados incorrectos.	escribir en almacenamiento externo. Cualquier aplicación puede leer datos escritos en almacenamiento externo.	10: M2(Almacenamiento inseguro de datos). Referencia: 1. owasp-mstg/0x05d-Testing-Data-Storage.md at master · MobSF/owasp-mstg · GitHub	confidenciales se almacenen localmente, deben cifrarse con una clave derivada del almacenamiento respaldado por hardware que requiere autenticación.
14	Técnicas de detección de dispositivos root.	La detección de *root* es hacer que la ejecución de la aplicación en un dispositivo rooteado sea un poco más difícil, lo que a su vez bloquea algunas de las herramientas y técnicas de ingeniería inversa.	OWASP Mobile Top 10: M9 (Ingeniería inversa). Referencia: 1. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md/	Como la mayoría de las otras defensas, la detección de *root* no es muy efectiva por sí misma, pero la implementación de múltiples comprobaciones de *root* que se encuentran dispersas por toda la aplicación puede mejorar la efectividad del esquema global anti-manipulación. Algunas técnicas para la detección de dispositivos *root* son: 1. SafetyNet: Es una API de Android que proporciona un conjunto de servicios y crea perfiles de dispositivos de acuerdo con la información de software y hardware. Luego, este perfil se compara con una lista de modelos de dispositivos aceptados que han pasado las pruebas de compatibilidad con Android. 2. Detección programática: Suele buscar binarios que normalmente se instalan una vez que se ha rooteado un dispositivo. 3. Detección de *Bypassing Root.* Realiza una evaluación del comportamiento de la aplicación a nivel de módulos del kernel.

Tabla 9: Reporte de Seguridad Android

En la Tabla 10: Reporte de Seguridad iOS se muestra el reporte final de la aplicación de mensajería segura para el sistema móvil iOS.

No	CASO	DESCRIPCIÓN	INFORMACIÓN	RECOMENDACIÓN
1	Anti-depuración (Anti-Debugging) e ingeniería inversa.	La depuración en una aplicación facilita aplicar ingeniería inversa con algún depurador. Esto permite volcar un seguimiento de pila y acceder a clases auxiliares.	OWASP Mobile Top 10: M9(Ingeniería Inversa). Referencia: 1. https://www.iamin.dev/ios%20security/iOS-security-ptrace-deny-attach/ 2. https://knight.sc/debugging/2019/06/03/debugging-apple-binaries-that-use-pt-deny-attach.html 3.	En iOS hay dos posibles sistemas de depuración: 1. El uso de la bandera "PT_DENY_ATTACH" es una técnica muy conocida para implementar anti-depuración en iOS. El uso de *ptrace* con "PT_DENY_ATTACH" valida que ningún otro depurador puede adjuntar al proceso protegido. Si un depurador intenta adjuntar, el proceso terminará. 2. En iOS, es posible usar Sysctl para detectar un depurador adjunto a un proceso. Sysctl es una función de la API de iOS que puede recuperar información sobre un proceso, especialmente si ha sido depurado.
2	Seguridad de transporte de aplicaciones (ATS) Permite cargas arbitrarias.	Las restricciones de seguridad de transporte de aplicaciones están deshabilitadas para todas las conexiones de red. La desactivación de ATS significa que se permiten conexiones HTTP no seguras. Las conexiones HTTPS también están permitidas y aún están sujetas a la evaluación de confianza del servidor predeterminada. Sin embargo, las verificaciones de seguridad extendidas, como requerir una versión mínima del protocolo *Transport Layer Security* (TLS), están deshabilitadas.	OWASP Mobile Top 10:M3(Comunicación Insegura). Referencia: 1. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06g-Testing-Network-Communic	*App Transport Security*(ATS) es un conjunto de comprobaciones de seguridad que el sistema operativo realiza al llevar a cabo conexiones a través de "NSURLConnection", "NSURLSession" y "CFURL" a nombres de dominios públicos. Si ATS se encuentra activado requiere de los siguientes requisitos de seguridad: 1. No permite conexiones HTTP. 2. El certificado X.509 tiene una huella digital SHA256 y debe estar firmado con al menos una clave RSA de

		Esta configuración no se aplica a los dominios enumerados en NSExceptionDomains.	ation.md 2. https://www.nowsecure.com/blog/2017/08/31/security-analysts-guide-nsapptransportsecurity-nsallowsarbitraryloads-app-transport-security-ats-exceptions/	2048 bits o una clave Elliptic-Curve Cryptography(ECC-256). 3. La versión de *Transport Layer Security*(TLS) debe ser 1.3 o superior y debe admitir *Perfect Forward Secrecy*(PFS) mediante el intercambio de claves *Elliptic Curve Diffie-Hellman Ephemeral*(ECDHE) y cifrados simétricos AES-256. Si ATS se encuentra desactivado se pueden configurar excepciones donde se pueden aplicar globalmente o individualmente para cada uno de los dominios definidos como: 1. Permitir conexiones insegura(HTTP). 2. Utilizar una versión mas antigua de TLS. 3. Deshabilitar *Perfect Froward Secrecy*(PFS). 4. Permitir conexiones a dominios locales. Para generar excepciones solo es agregar "NSExceptionDomians" dentro de la llave de "NSAppTransportSecurity".
3	Permisos innecesarios	Se detectaron permisos de la aplicación que no están siendo usados dentro de las funciones en la aplicación, se listan los permisos: 1. NSCalendarsUSage Description. 2. NSCameraUsageDescription. 3. NSLocationWhenInUseUsageDescription. 4. NSMicrophoneUsageDescription. 5. NSPhotoLibraryUsageDescription. 6. NSSiriUsageDescrip	OWASP Mobile Top 10: M4(Autorización insegura). Referencia: 1. https://developer.apple.com/documentation/bundleresources/information_property_list/protected_	Se recomienda realizar un de análisis de los permisos innecesarios y posteriormente deshabilitar los permisos que no están siendo usados en la aplicación, ya que algunos permisos ponen en riesgo la integridad de los datos de nuestra aplicación.

		tion.	resources	
4	Uso de API (s) potencialmente peligrosa (s).	La aplicación hace uso de API(s) inseguras, puede contener los siguientes API(s) inseguras fopen, strlen, memcpy, sscanf.	MobSF: IPA Binary Code Analysis; 1. OWASP Mobile Top 10: M7(Calidad del código del cliente). Referencia: 1. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/fopen-s-wfopen-s?view=msvc-160 2. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/strlen-wcslen-mbslen-mbslen-l-mbstrlen-mbstrlen-l?view=msvc-160 3. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/memcpy-s-wmemcpy-s?view=msvc-160 4. https://stackoverflow.com/questions/6510858/sscanf-get-	Se tiene las siguientes recomendaciones: 1. Antes de acceder al fichero file a través de la función “fopen”, se comprueba que el usuario tiene permisos suficientes para escribir con “access()”. Si un usuario pudiera cambiar el fichero file después de la llamada a “access()”, el atacante podría llegar a escribir sobre un fichero del cual no tiene permisos, ya que ambas funciones, “access” y “fopen”, trabajan sobre nombres de ficheros en lugar de manejadores de ficheros. Existen las versiones “fopen_s”, “_wfopen_s” tienen mejoras de seguridad. Los archivos que abren “fopen_s” y “_wfopen_s” no se pueden compartir. Si necesita que un archivo se pueda compartir, utilice “_fsopen”, “_wfsopen” con la constante de modo de uso compartido adecuada. La función “fopen_s” abre el archivo especificado por nombre de archivo. “_wfopen_s” es una versión de caracteres anchos de “fopen_s”; los argumentos de “_wfopen_s” son cadenas de caracteres anchos. “_wfopen_s” y “fopen_s” se comportan de manera idéntica de otra manera. 2. En el caso de la función de “strlen” se encuentra la función “strnlen”, que no sustituye a “strlen”; “strnlen” está diseñado para usarse solo para calcular el tamaño

			first-and-last-token-in-a-string 5. https://www.ibm.com/docs/en/aix/7.1?topic=s-scanf-fscanf-sscanf-wsscanf-subroutine	de los datos entrantes que no son de confianza en un búfer de tamaño conocido "strnlen" calcula la longitud pero no pasa del final del búfer si la cadena no está terminada. Para el uso de "strnlen" es recomendable añadir un tamaño fijo lo cual se puede utilizar "size_t MaxPossibleSize = startOfSharedMemory + sizeOfSharedMemory – input"; "strnlen(input, MaxPossibleSize)". 3. Existen las siguientes versiones "memcpy_s", "wmemcpy_s" con mejoras de seguridad de la función "mempcy". La función "memcpy_s" copia "count" bytes de "src" a "dest" y "wmemcpy_s" copia cuentas con caracteres anchos (dos bytes). Si el origen y el destino se superponen, el comportamiento de "memcpy_s" no está definido. Utilice "memmove_s" para manejar regiones superpuestas. Estas funciones validan sus parámetros. Si "count" es distinto de cero y "dest" o "src" es un puntero nulo, o "destSize" es menor que "count", estas funciones invocan el controlador de parámetro no válido, como se describe en Validación de parámetros. 4. Se recomienda que para evitar desbordamientos en la función "sscanf", se debería indicar la anchura máxima, también es recomendable delimitar los caracteres admisibles. Expertos recomiendan no

				usar directamente "gets" ni "scanf", ni siquiera para leer datos numéricos, sino hacer la lectura en una cadena de texto con "fgets" y luego extraer la información de ella con "sscanf".
5	Segmentos restringidos	El binario no tiene un segmento restringido que evite la carga dinámica de *dylib* para la inyección de código arbitrario.	OWASP Mobile Top 10: M7(Calidad del código del cliente). Referencia: 1. https://programmerclick.com/article/65211929285/	La función "hasRestrictedSegment()" detecta específicamente el segmento "RESTRICT", cualquier proceso dentro de esta función está configurada para restringir la inserción de biblioteca dinámica. Sin embargo, la detección de "__RESTRICT" se ha eliminado de la nueva versión del código fuente de dyld. A partir de iOS 10, este método de protección ha dejado de ser válido.
6	Sin control Asignación de memoria	El binario hace uso de función *malloc.*	OWASP Mobile Top 10: M7(Calidad del código del cliente). Referencia: 1. https://es.stackoverflow.com/questions/46915/c%C3%B3mo-funciona-se-usa-la-funci%C3%B3n-free-en-c 2. https://www.tutorialesprogramacionya.com/cya/detalleconcepto.php?punto=37&codigo=37&inicio=30	La memoria reservada por "*malloc"* se liberará automáticamente al final de cada transacción con la función "free()", así previniendo fallos de memoria.
7	Detección de dispositivos con *jailbroken*	La detección de dispositivos con *jailbreak* proporciona resistencia contra la ingeniería inversa y cierto	OWASP Mobile Top 10: M9(Ingeniería Inversa).	Se recomienda una serie de técnicas: 1. Comprobaciones basadas

		tipo de ataques específicos en el lado del cliente. Si bien este tipo de seguridad denominados como "*jailbreak detection*" no son muy efectivos por si solos, añadir algunos de estos mecanismos dentro de la aplicación si que pueden ayudar a mejorar el proceso de *anti-tampering* implementando en la misma.	Referencia: 1. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md	en ficheros, esta técnica consiste en comprobar si existe algún fichero y directorio asociados normalmente a las técnicas de *jailbreak*. 2. Comprobaciones basadas en permisos de ficheros, esta técnica consiste en comprobar si se crea un fichero fuera del *sandbox* de la aplicación. Si el fichero se crea correctamente entonces se podrá inferir que el dispositivo está *jailbroken*. 3. Verificación de llamada de la *App Store de Cydia*, aplicación que se instala por defecto en casi todas las versiones de *jailbreak*.
8	Criptografía	Existe criptografía insuficiente en el binario.	OWASP Mobile Top 10: M5(Criptografia insuficiente). Referencia: 1. https://github.com/MobSF/owasp-mstg/blob/master/Document/0x04g-Testing-Cryptography.md 2. https://www.keylength.com/ 3. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81286/5/fgarcialainezTFM0618memoria.p	Se recomiendan los siguientes algoritmos: 1. Algoritmos de confidencialidad: AES-GCM-256 o ChaCha20-Poly1305. 2. Algoritmos de integridad: SHA-256, SHA-384, SHA-512, Blake2, la familia SHA-3. 3. Algoritmos de firma digital: RSA (3072 bits y superior), ECDSA con NIST P-384. 4. Algoritmos de establecimiento de claves: RSA (3072 bits y superior), DH (3072 bits o superior), ECDH con NIST P-384.

			df	
9	Uso de función potencialmente peligrosa.	El binario tiene establecido "*RUNPATH SEARCH PATH*", que busca recursos críticos mediante una ruta de búsqueda proporcionada externamente que puede apuntar a recursos que no están bajo el control directo de la aplicación En ciertos casos, un atacante puede abusar de esta función, ejecutar sus propios programas, acceder a archivos de datos no autorizados o modificar la configuración de formas inesperadas.	OWASP Mobile Top 10: M7(Calidad del código del cliente). Referencia: 1. https://cwe.mitre.org/data/definitions/426.html	Cuando una biblioteca binaria o compartida transfiere datos debe establecerse "RPATH" o "RUNPATH" es si está vinculada a bibliotecas compartidas privadas en el mismo paquete. Para evitar esta vulnerabilidad se recomienda eliminar o restringir todas las configuraciones del entorno antes de invocar otros programas. Esto incluye la variable de entorno "PATH", "RPATH", "LD_LIBRARY_PATH", y otras configuraciones que identifican la ubicación de las bibliotecas de códigos y cualquier ruta de búsqueda específica de la aplicación.
10	Símbolos de depuración.	Los símbolos están disponibles en general, esta configuración garantiza que los símbolos de depuración formen parte del binario distribuido.	OWASP Mobile Top 10: M7(Calidad del código del cliente). Referencia: 1. https://developer.apple.com/forums/thread/663920 2. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81286/5/fgarcialainezTFM0618memoria.pdf	Se recomienda eliminar los símbolos de depuración, configurando "*Strip Debug Symbols During Copy*" durante la copia a *YES*, el "*Deployment Postprocessing"* a *YES* y "*Strip Linked Product*" a YES en la configuración de construcción del proyecto.

Tabla 10: Reporte de Seguridad iOS

Donde como resultado final se encontró un total de 10 vulnerabilidades para el sistema iOS y 14 vulnerabilidades para el sistema Android con una clasificación de nivel de riesgo: crítico, alto, medio y bajo.

6.2. CONCLUSIONES

Se cumplió el objetivo general propuesto en la presente tesis, ya que como se observa en el CAPÍTULO IV se realizo el análisis de seguridad estático y dinámico a un sistema de mensajería segura, disponible en los sistemas iOS y Android, durante el análisis se encontraron vulnerabilidades de diferentes niveles de riesgo, como críticos, altos, medios, bajos e informativos los cuales fueron detectadas por las herramientas especializadas en análisis de seguridad de aplicaciones móviles, conforme se realizaban dichos análisis, se realizaron pruebas para verificar que los resultados arrojados por las herramientas no fueran un falso positivo, una vez teniendo los resultados finales se emitieron recomendaciones para eliminar la mayor cantidad de fallas que pudieran provocar vulnerabilidades en el sistema, teniendo un análisis final de nivel satisfactorio.

Cabe mencionar que la aplicación móvil cuenta con diversos filtros de seguridad recomendados por el organismo OWASP, sin embargo, se menciona que un sistema no puede ser 100% seguro, ya que día con día surgen nuevos tipos de ataques.

Así como también en el CAPÍTULO IV se observa que no solo se trata de un sistema seguro, si no, que la parte humana también debe ser consciente del peligro al que puede exponer al sistema, provocando errores que puedan poner en peligro tanto la confidencialidad de su información personal, como la de la aplicación.

REFERENCIAS

[1]: Pruebas de Seguridad de Aplicaciones (AST) SAST, DAST e IAST. (2021, 23 mayo). GB Advisors. https://www.gb-advisors.com/es/pruebas-de-seguridad-de-aplicaciones-ast-sast-dast-e-iast/

[2]: Blueinfy. (s. f.).Welcome to Blueinfy. Recuperado 24 de agosto de 2021, de https://www.blueinfy.com

[3]: ISO/IEC 27001:2013. (2020, 16 diciembre). ISO. https://www.iso.org/standard/54534.html

[4]: CWE - Common Weakness Enumeration. (s. f.). CWE. Recuperado 23 de julio de 2021, de https://cwe.mitre.org/index.html

[5]: OWASP Foundation | Open Source Foundation for Application Security. (s. f.). OWASP. Recuperado 21 de julio de 2021, de https://owasp.org

[6]: OWASP Mobile Top 10. (2004). Mobile Security. https://owasp.org/www-project-mobile-top-10/

[7]: A decade in, how safe are your iOS and Android apps? (s. f.). NowSecure, Inc. Recuperado 18 de marzo de 2021, de https://www.nowsecure.com/blog/2018/07/11/a-decade-in-how-safe-areyour-ios-and-android-apps/

[8]: Conk3r. (2016, 31 octubre). Bypassing Apple's Touch ID. . . but from certain apps!, https://medium.com/@c0nk3r/bypassing-apples-touch-id-but-from-certain-apps-e7c6fbe3d848

[9]: Simonds, W., & Simonds, W. (2014, 11 abril). Tinder app vulnerability: how

sharing location data harms your privacy. Online Privacy | Abine. https://www.abine.com/blog/2014/tinder-app-vulnerability/

[10]: Vaas, L. (2018, 16 noviembre). Hacking MiSafes' smartwatches for kids is child's play. Naked Security. https://nakedsecurity.sophos.com/2018/11/16/hacking-misafes-smartwatches-for-kids-is-childs-play/

[11]: Grab disclosed on HackerOne: Two-factor authentication bypass on. . . (2017, 12 septiembre). HackerOne. https://hackerone.com/reports/202425

[12]: Appknox, T. (2020, 11 noviembre).Major Bug in Ola App can Make you Either Rich or Poor!Appknox. https://inc42.com/buzz/major-bug-in-ola-app-can-make-you-either-rich-or-poor/

[13]: Stykas, V. (2018, 31 mayo). Remote smart car hacking with just a phone.- Vangelis Stykas. Medium. https://medium.com/@evstykas/remote-smart-car-hacking-with-just-a-phone-2fe7ca682162

[14]: Khandelwal, S. (2019, 14 mayo).Hackers Used WhatsApp 0-Day Flaw to Secretly Install Spyware On Phones. The Hacker News. https://thehackernews.com/2019/05/hack-whatsapp-vulnerability.html

[15]: Sandoval, K. (2018, 2 febrero).How Pokémon Go Fans Hacked 'Em All: And How to Prevent Similar Reverse-Engineering. Nordic APIs. https://nordicapis.com/how-pokemon-go-fans-hacked-em-all-and-how-to-prevent-similar-reverse-engineering/

[16]: Greenberg, A. (2017, 28 abril).An Obscure App Flaw Creates Backdoors In Millions of Smartphones. Wired. https://www.wired.com/2017/04/obscure-app-flaw-creates-backdoors-millions-smartphones/

[17]: Gisbert Vercher, Belén (2015). Administración y auditoría de los servicios web. Elearning.

[18]: Gómez Vieites, Álvaro (2014). Enciclopedia de la Seguridad Informática. Madrid, España: RA-MA, S.A.

[19]: Stanllings, William (2004). Fundamentos de Seguridad en Redes, Aplicaciones y Estándares, Madrid, España: Pearson Educación, S.A.

[20]: Rooting en Android - Panda Security. (s. f.). Panda Security. Recuperado 21 de julio de 2021, de https://www.pandasecurity.com/es/security-info/root/

[21]: Amenazas a la Seguridad de la Información | Departamento de Seguridad Informática. (s. f.). Departamento de Seguridad informatica. Recuperado 21 de julio de 2021, de http://www.seguridadinformatica.unlu.edu.ar/?q=node/12

[22]: Tríada CID. (2019, 29 agosto). OnTek. https://www.ontek.net/que-es-triada-cid/

[23]: Boris Beizer. Software Testing Techniques. Van Nostrand Reinhold, New York, 2nd edition, 1990.

[24]: Boris Beizer. Black-Box Testing: Techniques for Functional Testing of Software and Systems. John Wiley & Sons, Inc., 1995.

[25]: Kisutsa, C. (2019, 26 julio). GitHub - xtiankisutsa/MARA_Framework: MARA is a Mobile Application Reverse engineering and Analysis Framework. It is a toolkit that puts together commonly used mobile application reverse engineering and analysis tools to assist in testing mobile applications against the OWASP mobile security threats. GitHub. https://github.com/xtiankisutsa/MARA_Framework

[26]: Abraham, A. (2021, 27 junio).MobSF/Mobile-Security-Framework-MobSF. GitHub. https://github.com/MobSF/Mobile-Security-Framework-MobSF

[27]: MobileSecurity. (2021, 23 septiembre). GitHub - m0bilesecurity/RMS-Runtime-Mobile-Security: Runtime Mobile Security (RMS) 📱🔥 - is a powerful web interface that helps you to manipulate Android and iOS Apps at Runtime. GitHub. https://github.com/m0bilesecurity/RMS-Runtime-Mobile-Security

[28]: Burp Suite - Application Security Testing Software. (s. f.). PortSwigger. https://portswigger.net/burp

[29]: owasp-mstg/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md at master · OWASP/owasp-mstg. (2019, 25 enero). GitHub. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md/

[30]: owasp-mstg/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md at master · OWASP/owasp-mstg. (2019, 25 enero). GitHub. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md

[31]: Parrend, P. (2018, 7 marzo). Bytecode Obfuscation Control | OWASP Foundation. OWASP. https://owasp.org/www-community/controls/Bytecode_obfuscation

[32]: ANDROID STUDIO. (s. f.). Shrink, obfuscate, and optimize your app |. Android Developers. https://developer.android.com/studio/build/shrink-code?hl=es%2F

[33]: owasp-mstg/0x05d-Testing-Data-Storage.md at master · OWASP/owasp-mstg. (2022, 8 febrero). GitHub. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md

[34]: owasp-mstg/0x05h-Testing-Platform-Interaction.md at master · OWASP/owasp-mstg. (2022, 18 febrero). GitHub. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md

[35]: owasp-mstg/0x06g-Testing-Network-Communication.md at master · OWASP/owasp-mstg. (2022, 25 enero). GitHub. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06g-Testing-Network-Communication.md

[36]: ¿Es sqlite3_bind_text suficiente para evitar la inyección de SQL en el iPhone - Excelente biblioteca. (2009, 23 mayo). uwenku. http://es.uwenku.com/question/p-yjcfmvax-bq.html/

[37]: M. (2020, 12 junio). owasp-mstg/0x04g-Testing-Cryptography.md at master · MobSF/owasp-mstg. GitHub. https://github.com/MobSF/owasp-mstg/blob/master/Document/0x04g-Testing-Cryptography.md

[38]: Giry, D. (2020, 24 mayo). Keylength - Cryptographic Key Length Recommendation. BlueKrypt. https://www.keylength.com/

[39]: fopen_s, _wfopen_s. (2021, 8 marzo). Microsoft Docs. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/fopen-s-wfopen-s?view=msvc-160

[40]: strlen, wcslen, _mbslen, _mbslen_l, _mbstrlen, _mbstrlen_l. (2021, 8 marzo). Microsoft Docs. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/strlen-wcslen-mbslen-mbslen-l-mbstrlen-mbstrlen-l?view=msvc-160

[41]: memcpy_s, wmemcpy_s. (2021, 8 marzo). Microsoft Docs. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/memcpy-s-wmemcpy-s?view=msvc-160

[42]: sscanf: get first and last token in a string. (2011, 28 junio). Stack Overflow. https://stackoverflow.com/questions/6510858/sscanf-get-first-and-last-token-in-a-string

[43]: Protección de inyección de código de aplicación iOS - programador clic. (s. f.). Programador clic. https://programmerclick.com/article/65211929285/

[44]: ¿Cómo funciona/se usa la función free en C? (2017, 31 enero). Stack Overflow en español. https://es.stackoverflow.com/questions/46915/c%C3%B3mo-funciona-se-usa-la-funci%C3%B3n-free-en-c

[45]: owasp-mstg/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md at master · OWASP/owasp-mstg. (2022, 25 enero). GitHub. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md

[46]: Intents y filtros de intents | Desarrolladores de Android |. (s. f.). Android Developers. https://developer.android.com/guide/components/intents-filters?hl=es-419

[47]: Restringe las interacciones con otras apps | Desarrolladores de Android |. (s. f.). Android Developers. https://developer.android.com/training/permissions/restrict-interactions?hl=es-419#services/

[48]: Restringe las interacciones con otras apps | Desarrolladores de Android |. (s. f.-b). Android Developers. https://developer.android.com/training/permissions/restrict-interactions?hl=es-419#services/

[49]: owasp-mstg/0x05d-Testing-Data-Storage.md at master · MobSF/owasp-mstg. (2020, 15 noviembre). GitHub OWASP. https://github.com/MobSF/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#checking-memory-for-sensitive-data-mstg-storage-10/

[50]: Insecure Temporary File | OWASP Foundation. (s. f.). OWASP. https://owasp.org/www-community/vulnerabilities/Insecure_Temporary_File

[51]: SecureRandom |. (s. f.). Android Developers. https://developer.android.com/reference/kotlin/java/security/SecureRandom

[52]: CWE - CWE-426: Untrusted Search Path (4.7). (2022, 28 abril). CWE. Recuperado 1 de mayo de 2022, de https://cwe.mitre.org/data/definitions/426.html

[53]: Apple. (2021). iOS static library debugging symbols not included in the app's dSYM file when archiving. Developer Forums. https://developer.apple.com/forums/thread/663920