BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Licenciatura en ciencias de la computación

Auditoría en Aplicaciones móviles Android e iOS

TESIS

PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

JUAN CARLOS GARCIA LEZAMA

ASESORES DE TESIS:

MAYA CARRILLO RUIZ

CARLOS DOCE REYES

Puebla, Pue., México enero 2025

Página dejada en blanco intencionalmente

DEDICATORIA

A mis padres, por su amor, arduo trabajo y sacrificio en todos estos años, gracias a ustedes he logrado llegar hasta aquí y convertirme en lo soy.

A todos los profesores que me han apoyado y han hecho que el trabajo se realice con éxito en especial a aquellos que nos abrieron las puertas y compartieron sus conocimientos.

AGRADECIMIENTOS

A mis padres, esposa y en especial a la empresa MCSec por brindarme la oportunidad de incursionar en el área de ciberseguridad.

CONVENCIONES Y NOMENCLATURAS

`Ancho constante`

El ancho constante en el texto indica fragmentos de código, URL, estructuras de datos, funciones, métodos, constantes, tipos de datos, o expresiones como:

`context = this.getContext()`.

`Ancho constante cursiva`

El ancho constante en cursiva en el cuerpo texto indica comandos y parámetros reemplazables.

Itálico

El texto en cursiva indica anglicismos, significado de siglas o acrónimos, lenguaje coloquial o "jerga informática". Además, se utiliza para denotar recursos interesantes como rutas, nombres de archivos, directorios, extensiones y funciones intrínsecas de la plataforma.

TABLA DE CONTENIDO

1 INTRODUCCIÓN 11

1.1 PLANTEAMIENTO DEL PROBLEMA 12

1.2 JUSTIFICACIÓN 17

1.3 OBJETIVOS 21

2 MARCO TEÓRICO 22

2.1 APLICACIONES MÓVILES 22

2.1.1 Estructura de las aplicaciones 25

2.1.2 Almacenamiento en aplicaciones 34

2.2 CONCEPTOS EN UNA AUDITORÍA DE SEGURIDAD 42

2.2.1 Amenazas y vulnerabilidades 42

2.2.2 Modalidad de evaluación 44

2.2.3 Análisis de Vulnerabilidades 45

2.2.4 Métrica de evaluación de vulnerabilidades 46

2.3 OWASP MOBILE APPLICATION SECURITY 49

2.3.1 Guía de pruebas de seguridad móvil (MSTG) 50

2.3.2 Estándar de Verificación de Seguridad de Aplicaciones Móviles (MASVS) 51

2.3.3 Lista de verificación de seguridad de aplicaciones móviles 56

2.3.4 OWASP Mobile Top 10 60

2.3.5 Fases de una auditoría de seguridad 66

2.4 MARCO LEGAL 70

2.4.1 La Ley de Protección de Datos en México 70

2.4.2 La Ley de Protección de Datos Personales aplicada 72

2.4.3 Obligaciones de las empresas 73

2.4.4 Sanciones 74

2.5 RECURSOS Y CONFIGURACIONES PARA LA AUDITORIA 76

2.5.1 Hardware 77

2.5.2 Software 78

2.5.3 Configuración del entorno Mobexler 86

2.5.4 Configuración del dispositivo Android 89

2.5.5 Configuración del dispositivo iOS 96

3 TRABAJO RELACIONADO 102

4 MÉTODO PROPUESTO 111

5 CASO PRÁCTICO 169

5.1 ANDROID 170

5.2 IOS 200

5.3 RESULTADOS 221

6 CONCLUSIONES Y TRABAJO A FUTURO 230

BIBLIOGRAFÍA 233

ANEXO A - JAILBREAK Y ACCESO ROOT 236

ANEXO B – CONFIGURACIONES DE SEGURIDAD POR EL CIS 239

ANEXO C – RECURSOS Y MATERIAL DE ESTUDIO 243

ANEXO D – GLOSARIO 245

LISTA DE ILUSTRACIONES

FIGURA 1 – CUOTA DE MERCADO DE SISTEMAS OPERATIVOS MÓVILES POR STATCOUNTER [2] 13

FIGURA 2 – DISTRIBUCIÓN DEL TRÁFICO MUNDIAL DE INTERNET EN 2023, SEGÚN DISPOSITIVO DE ACCESO [3] 14

FIGURA 3 – APLICACIONES DISPONIBLES PARA LAS PLATAFORMAS ANDROID E IOS [4] 15

FIGURA 4 – CATEGORÍAS DE AMENAZAS EN DISPOSITIVOS MÓVILES [5] 16

FIGURA 5 – DESCARGAS DE APLICACIONES IOS Y ANDROID EN LOS ÚLTIMOS AÑOS [6] 17

FIGURA 6 – ESTRUCTURA DE UN ARCHIVO APK 26

FIGURA 7 – ESTRUCTURA DE UN ARCHIVO .IPA 32

FIGURA 8 – ESTRUCTURA DE ARCHIVOS DEL ALMACENAMIENTO EN ANDROID 35

FIGURA 9 – IOS SANDBOX 37

FIGURA 10 – ESTRUCTURA DE ARCHIVOS DE LOS CONTENEDORES EN IOS 38

FIGURA 11 – DIAGRAMA DEFINICIÓN DE RIESGO 43

FIGURA 12 – CLASIFICACIONES CUALITATIVAS DE VULNERABILIDADES 48

FIGURA 13 – NIVELES DE SEGURIDAD DEFINIDOS EN MASVS 52

FIGURA 14 – LISTA DE VERIFICACIÓN SECCIÓN - DESING AND THREAT MODELING [19] 57

FIGURA 15 – LISTA DE VERIFICACIÓN SECCIÓN - DATA STORE AND PRIVACY [19] 58

FIGURA 16 – LISTA DE VERIFICACIÓN SECCIÓN – RESILIENCE [19] 59

FIGURA 17 – OWASP TOP 10 DE LOS RIESGOS DE SEGURIDAD MÓVIL 2016 60

FIGURA 18 – OPCIONES DE CONEXIÓN VMWARE FUSION 86

FIGURA 19 – PRUEBA DE CONEXIÓN DISPOSITIVO ANDROID 87

FIGURA 20 – PRUEBA DE CONEXIÓN DISPOSITIVO IOS 88

FIGURA 21 – INSTALAR APLICACIÓN DVIA-V2-SWIFT .IPA 98

FIGURA 22 – METODOLOGÍA PROPUESTA PARA AUDITAR APLICACIONES MÓVILES 111

FIGURA 23 – EJEMPLO DE EJECUCIÓN DEL MÓDULO ATTACKSURFACE DE DROZER 143

FIGURA 24 – IMPLEMENTACIÓN SUPERPOSICIÓN EN SWIFT 148

FIGURA 25 – EJEMPLO DE <INTENT-FILTER> DE UN ENLACE PROFUNDO 149

FIGURA 26 – EJEMPLO DE MANEJO DE PARÁMETROS EN DEEP LINKS ANDROID 152

FIGURA 27 – RESULTADO DE MANEJO DE DEEP LINK EN ANDROID 153

FIGURA 28 – SCRIPT DE FRIDA PARA OBTENER PARÁMETROS DE WKWEBVIEW 166

FIGURA 29 – RESULTADOS ANÁLISIS ESTÁTICO CON MOBSF 170

FIGURA 30 – ACTIVITES EXPUESTAS DE LA APLICACIÓN INSECUREBANKV2 171

FIGURA 31 – ACCESO A LA ACTIVITY POSTLOGIN 172

FIGURA 32 – ANÁLISIS BROADCASTRECEIVER....173
FIGURA 33 – ANÁLISIS BROADCASTRECEIVER....173
FIGURA 34 – ANÁLISIS CÓDIGO FUENTE BROADCASTRECEIVER....174
FIGURA 35 – SMS CON LA NUEVA CONTRASEÑA DEL USUARIO....175
FIGURA 36 – CONTENT PROVIDER EN EL ARCHIVO ANDROIDMANIFEST.XML....176
FIGURA 37 – ANÁLISIS CONTENT PROVIDER CON LA HERRAMIENTA DROZER....177
FIGURA 38 – ANÁLISIS DE URIS ACCESIBLES....177
FIGURA 39 – USUARIOS OBTENIDOS A TRAVÉS DEL CONTENT PROVIDER....178
FIGURA 40 – CÓDIGO FUENTE DE LA CLASE VIEWSTATEMENT....179
FIGURA 41 – ARCHIVO EN ALMACENAMIENTO EXTERNO....180
FIGURA 42 – XSS EN WEBVIEW....181
FIGURA 43 – USO DE SHAREDPREFENCES....182
FIGURA 44 – MÉTODO SAVECREDS() DE LA CLASE DOLOGIN....183
FIGURA 45 – CREDENCIALES DEL USUARIO ALMACENADAS EN EL ARCHIVO MYSHAREDPREFERENCES.XML....183
FIGURA 46 – NOMBRE DE USUARIO DECODIFICADO....184
FIGURA 47 – CLAVE DE CIFRADO Y VECTOR DE INICIALIZACIÓN....184
FIGURA 48 – SCRIPT PARA DESCIFRAR LA CONTRASEÑA....185
FIGURA 49 – BÚSQUEDA DE TEXTO DE LA HERRAMIENTA JADX-GUI....186
FIGURA 50 – ANÁLISIS DE LA CLASE DOLOGIN....187
FIGURA 51 – CREDENCIALES DE USUARIO EN LOGS DE LA CONSOLA DEL DISPOSITIVO....187
FIGURA 52 – DIRECCIÓN IP Y PUERTO DEL PROXY BURP SUITE....188
FIGURA 53 – CONFIGURACIÓN PROXY EN EL DISPOSITIVO ANDROID....189
FIGURA 54 – COMUNICACIÓN ENTRE LA APLICACIÓN Y EL SERVIDOR....190
FIGURA 55 – ATRIBUTO ALLOWBACKUP....191
FIGURA 56 – RESPALDO DE LA APLICACIÓN INSCUREBANKV2....192
FIGURA 57 – DESEMPAQUETADO DEL ARCHIVO DE RESPALDO BACKUP.AB....192
FIGURA 58 – CONTENIDO DEL ARCHIVO DE RESPALDO DATA.TAR....193
FIGURA 59 – MENSAJE DEL SERVIDOR CUANDO UN USUARIO NO EXISTE....194
FIGURA 60 – MENSAJE DEL SERVIDOR CUANDO UN USUARIO EXISTE....195
FIGURA 61 – USO DE LA HERRAMIENTA INTRUDER DE BURP SUITE....195
FIGURA 62 – CONFIGURACIÓN INTRUDER DE BURP SUITE....196
FIGURA 63 – LISTA DE POSIBLES NOMBRES DE USUARIO....197
FIGURA 64 – ENUMERACIÓN DE USUARIOS CON LA HERRAMIENTA INTRUDER....198

Figura 65 – Falta de identificador de sesión 199
Figura 66 – Cambio de contraseña del usuario jack 199
Figura 67 – Contenido del archivo userInfo.plist 201
Figura 68 – Uso de NSUserDefaults 202
Figura 69 – Información almacenada con NSUserDefaults 203
Figura 70 – Autenticación local con touchID 205
Figura 71 – Evasión de la autenticación con touchID 206
Figura 72 – Mensaje de autenticación correcta 207
Figura 73 – Dirección IP y puerto del proxy Burp Suite 208
Figura 74 – Configuración proxy en el dispositivo iOS 209
Figura 75 – Comunicación entre la aplicación DVIA y el servidor 210
Figura 76 – Dirección IP y puerto del proxy Burp Suite 212
Figura 77 – Configuración de proxy en dispositivo iOS 212
Figura 78 – Información del usuario enviada a servicios de terceros 213
Figura 79 – Uso de UIWebView en la aplicación DVIA 215
Figura 80 – XSS en la aplicación DVIA 216
Figura 81 – Inyección de JavaScript con URL de protocolos personalizados 217
Figura 82 – Aplicación muestra información sensible 219
Figura 83 – Capturas autogeneradas que muestran información sensible 220
Figura 84 – Gráfica de clasificación de vulnerabilidades de la aplicación InsecureBankv2 225
Figura 85 – Gráfica de clasificación de vulnerabilidades de la aplicación DVIA-v2 229
Figura 86 – Certificado eMAPT 232

RESUMEN

Debido al creciente uso de los dispositivos móviles y sus aplicaciones, es crucial tomar medidas para asegurar la confidencialidad, integridad y disponibilidad de la información que estas manejan. Constantemente los fallos de seguridad representan un riesgo potencial debido a que las vulnerabilidades pueden resultar en la exposición masiva de información confidencial, acceso no autorizado a cuentas de usuarios, mal funcionamiento de las aplicaciones, entre otras.

Realizar auditorías de seguridad en aplicaciones móviles es una tarea desafiante. Esto se debe a la necesidad de adquirir conocimientos clave, disponer de dispositivos móviles físicos o virtuales, conocer e instalar las herramientas tanto en los dispositivos móviles como en el ordenador. Actualmente no existe un recurso básico simplificado que sirva como punto de partida para llevar a cabo una auditoría de aplicaciones móviles de manera efectiva. Por lo tanto, en este proyecto, se recopilan los elementos esenciales para llevar a cabo una auditoría. Estos elementos incluyen definiciones, métodos de evaluación, tipos de aplicaciones móviles, estructura de un archivo .apk de Android y .ipa de iOS, herramientas de software y hardware, así como métricas para evaluar vulnerabilidades. Además, se describe el proyecto OWASP Mobile Security Project, un recurso fundamental para realizar auditorías en aplicaciones móviles. Con esto se busca que cualquier persona que se adentre en este campo pueda comprender los requisitos fundamentales. Se presenta una metodología sencilla que consta de pasos generales para auditar aplicaciones móviles, sin importar el lenguaje o *framework* de desarrollo. Esto servirá como guía para personas sin experiencia. Además, se pone a prueba llevando cabo una auditoría en dos aplicaciones: una en la plataforma Android y otra en la plataforma iOS. Para hacerlo, se utilizaron las herramientas y recursos mencionados a lo largo de este proyecto, finalmente se documentaron las vulnerabilidades encontradas junto con las recomendaciones para solucionarlas.

1 INTRODUCCIÓN

Los teléfonos móviles y sus aplicaciones se han convertido en una herramienta esencial en nuestra vida diaria. Actualmente, existe una amplia variedad de aplicaciones móviles para diferentes servicios como: finanzas, salud, movilidad, entre otros. Muchas de estas almacenan y procesan información personal como fotos, datos bancarios, claves de acceso y más. Incluso existen empresas que han implementado aplicaciones para manejar información sensible de la propia organización y de sus usuarios por lo que es crucial garantizar la seguridad para prevenir incidentes como el robo o fuga de información.

La auditoría de aplicaciones móviles, también conocida como pruebas de penetración o *pentesting* a aplicaciones móviles, consiste en desglosar exhaustivamente la aplicación con el objetivo de examinar su estructura, estudiar cómo funciona a través del análisis estático, análisis dinámico y exploraciones manuales para identificar los puntos de entrada, los datos que almacena y/o se envían a un servidor remoto y de esta manera identificar vulnerabilidades potenciales que puedan tener un impacto en la confidencialidad, integridad o disponibilidad de la información.

1.1 Planteamiento del problema

"Una cadena es tan fuerte como su eslabón más débil" [1]

Hoy en día los dispositivos móviles son una herramienta que nos ayuda en nuestras actividades diarias. A medida que el número de usuarios aumenta cada año, la preocupación por la seguridad en estos dispositivos y sus aplicaciones se vuelve más crítica que nunca. La seguridad de los dispositivos móviles se refiere a la protección de los dispositivos portátiles, como: teléfonos, tabletas y relojes inteligentes, frente a amenazas y vulnerabilidades que puedan comprometer la información almacenada.

Pueden encontrarse una gran variedad de dispositivos móviles en el mercado de diferentes fabricantes y distintas características, existen diferentes sistemas operativos para estos dispositivos. Los más utilizados en el último año, según la cuota (participación) de mercado, como puede observarse en la Figura 1, entre junio del 2023 y mayo del 2024 de sistemas operativos móviles por StatCounter, un sitio web de análisis de tráfico web, con sede en Dublín, Irlanda son: Android un sistema operativo desarrollado por Google, basado en el *kernel* de *Linux* diseñado para dispositivos móviles, tabletas, relojes inteligentes, automóviles y televisores y en segundo lugar, iOS un sistema operativo móvil de la empresa Apple derivado de macOS, que a su vez está basado en *Darwin*, por lo tanto, es un sistema tipo *Unix*.

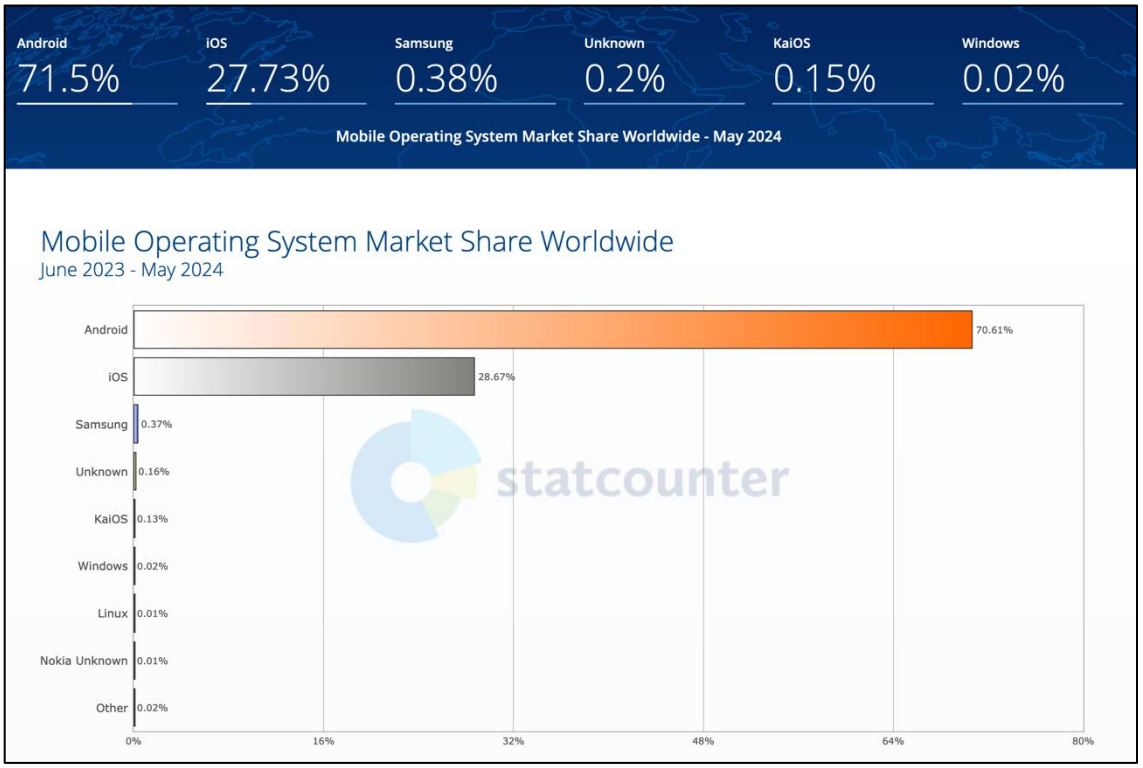


Figura 1 – Cuota de mercado de sistemas operativos móviles por statcounter [2]

De acuerdo con datos recopilados por statista los dispositivos móviles, exceptuando las *tablets*, fueron en último año la principal forma de acceso a Internet, acaparando más del 55% del tráfico mundial. En la figura 2 se puede apreciar lo anterior mencionado.

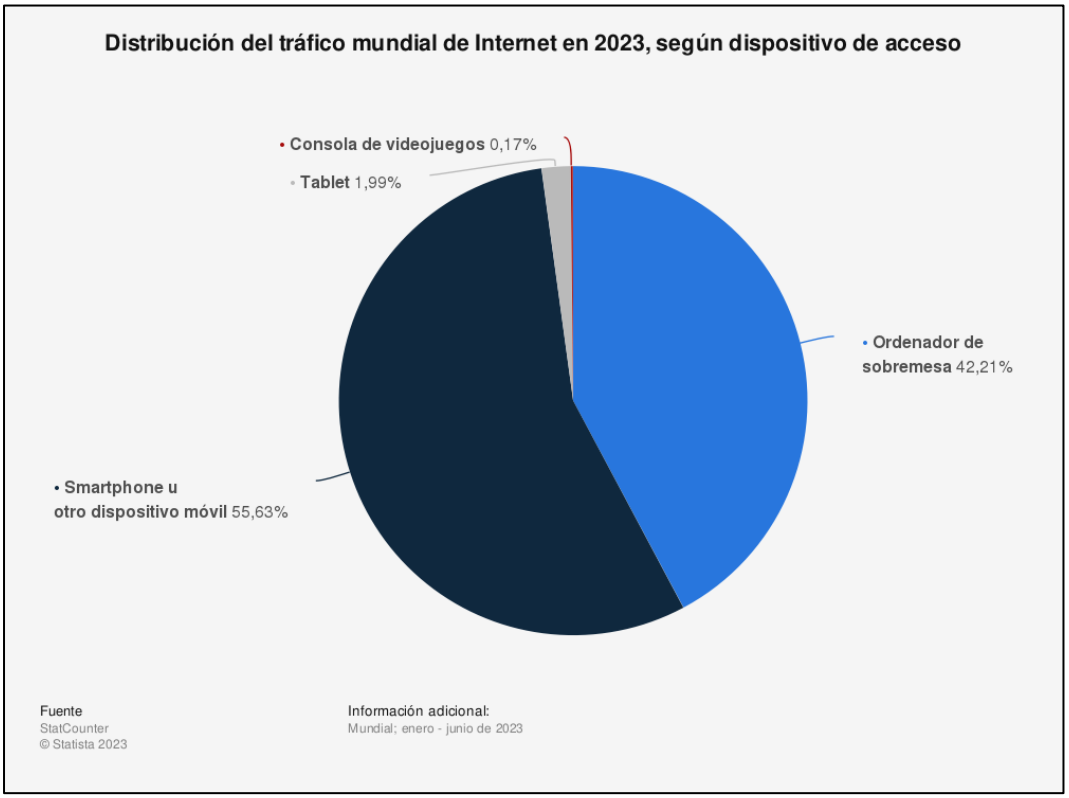


Figura 2 – Distribución del tráfico mundial de Internet en 2023, según dispositivo de acceso [3]

Debido a la variedad de dispositivos, para este proyecto se desarrolló la investigación en aplicaciones móviles para teléfonos inteligentes, mejor conocidos como “Smartphone” con sistema operativo Android e iOS, ya que de acuerdo con las estadísticas son los más usados en el mundo, esta es la misma razón por la que los actores maliciosos se enfocan en atacar las aplicaciones en estas plataformas.

El término “Aplicación móvil”, más comúnmente conocida como aplicación, es un tipo de software diseñado para ejecutarse en dispositivos móviles. Las aplicaciones amplían significativamente la funcionalidad de los teléfonos y enriquecen la experiencia de usuario. Actualmente, hay millones de aplicaciones disponibles en las plataformas de distribución oficiales de cada sistema operativo, *Google Play Store* para Android y *App Store* para iOS. De acuerdo con las estadísticas de *Statista*, una compañía alemana de estadísticas en línea, que traduce los datos recopilados por

los institutos de investigación de mercado y datos del sector económico en diferentes idiomas, al tercer trimestre del 2022 en la *Google Play Store* había 3,553,050 millones de aplicaciones disponibles para la plataforma Android y en la *App Store* 1,642,759 millones de aplicaciones disponibles para la plataforma iOS.

En la Figura 3 se puede observar gráficamente la cantidad de las aplicaciones existentes para cada plataforma.

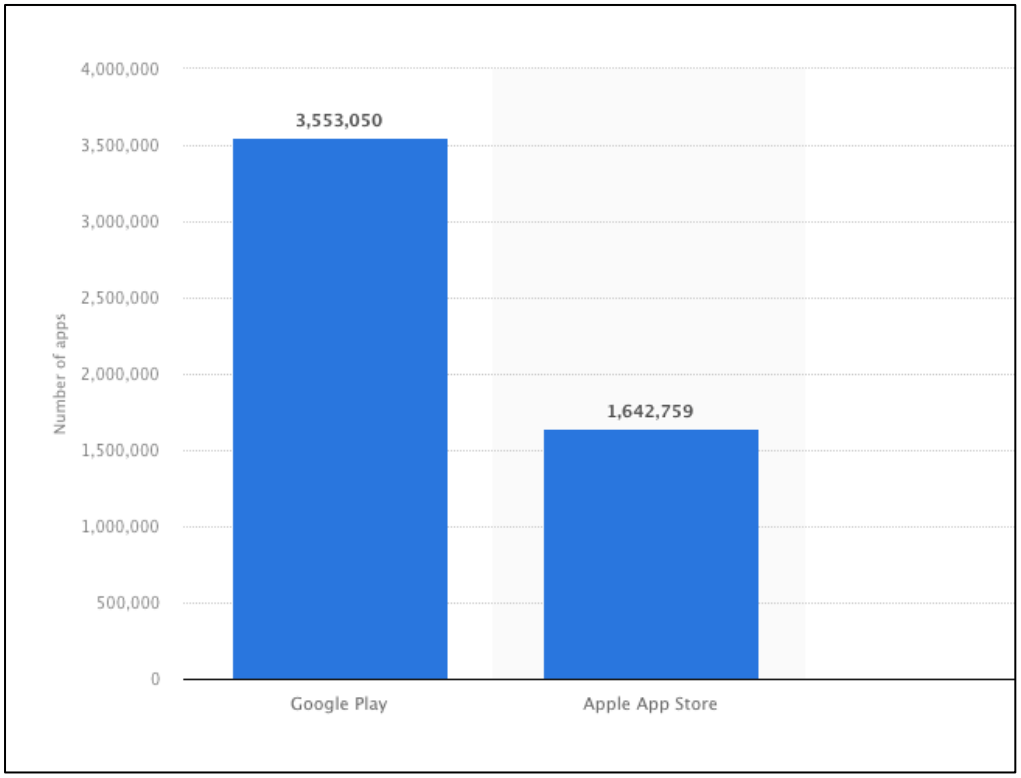


Figura 3 – Aplicaciones disponibles para las plataformas Android e iOS [4]

La funcionalidad proporcionada por los dispositivos móviles ha evolucionado significativamente y continúa avanzando. Inicialmente, los teléfonos móviles fueron diseñados para realizar llamadas telefónicas y los operadores eran el objetivo de los actores malintencionados con el objetivo de realizar llamadas gratuitas, los usuarios y sus datos rara vez eran el objetivo. Una vez se introdujeron los sistemas operativos Android e iOS, el panorama de amenazas cambió drásticamente, a medida que los

usuarios empezaron a usar los teléfonos para operaciones como: banca en línea, comercio electrónico, entre otros.

Por otro lado, las empresas comenzaron a permitir que los empleados utilizaran dispositivos y aplicaciones móviles para acceder al correo electrónico, contactos y redes corporativas.

En el repositorio del *National Institute of Standards and Technology* por sus siglas en inglés (NIST) existe un recurso llamado *Mobile Threat Catalogue* (MTC) que describe, identifica y estructura las amenazas a dispositivos móviles. Se han clasificado un total de 236 amenazas, clasificadas en 12 categorías. En la Figura 4 se puede observar que dentro de las 3 primeras categorías se encuentran las aplicaciones.

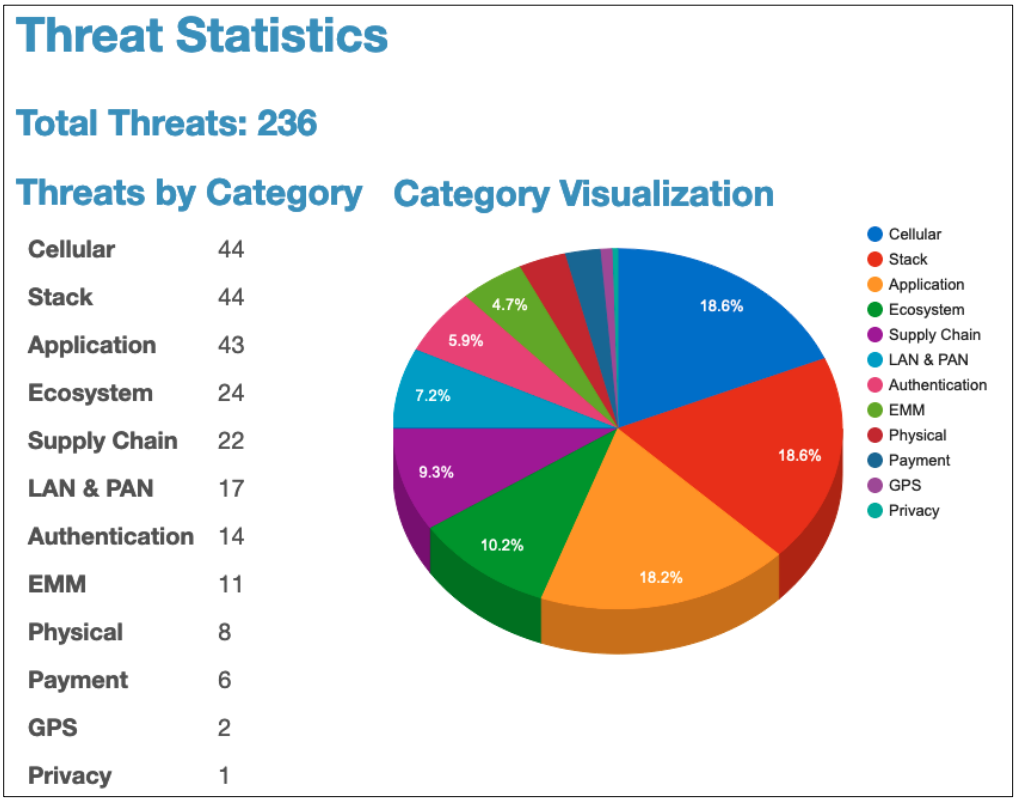


Figura 4 – Categorías de amenazas en dispositivos móviles [5]

Es por ello el interés en las aplicaciones móviles en este proyecto, ya que son una de las principales amenazas, que afecta a los usuarios de los teléfonos inteligentes.

1.2 Justificación

Los desarrolladores en general prestan atención al diseño del software para brindar una experiencia fluida al usuario final. Las personas instalan las aplicaciones móviles y brindan información personal, pero rara vez se detienen a pensar en las implicaciones de seguridad. Durante los últimos años se ha observado el incremento en el uso de los dispositivos móviles y sus aplicaciones, de acuerdo con la empresa statista [6] la cantidad de descargas de aplicaciones móviles en todo el mundo ha aumentado constantemente desde 2016, superando los 200,000 mil millones en 2019. En el segundo semestre del 2022, los usuarios descargaron 255,000 mil millones de aplicaciones móviles en sus dispositivos, un aumento de más del 63 % con respecto a los 140,700 mil millones de descargas de aplicaciones en 2016. A continuación, se presenta la figura representativa de lo anteriormente descrito correspondiente a aplicaciones de las plataformas Android e iOS.

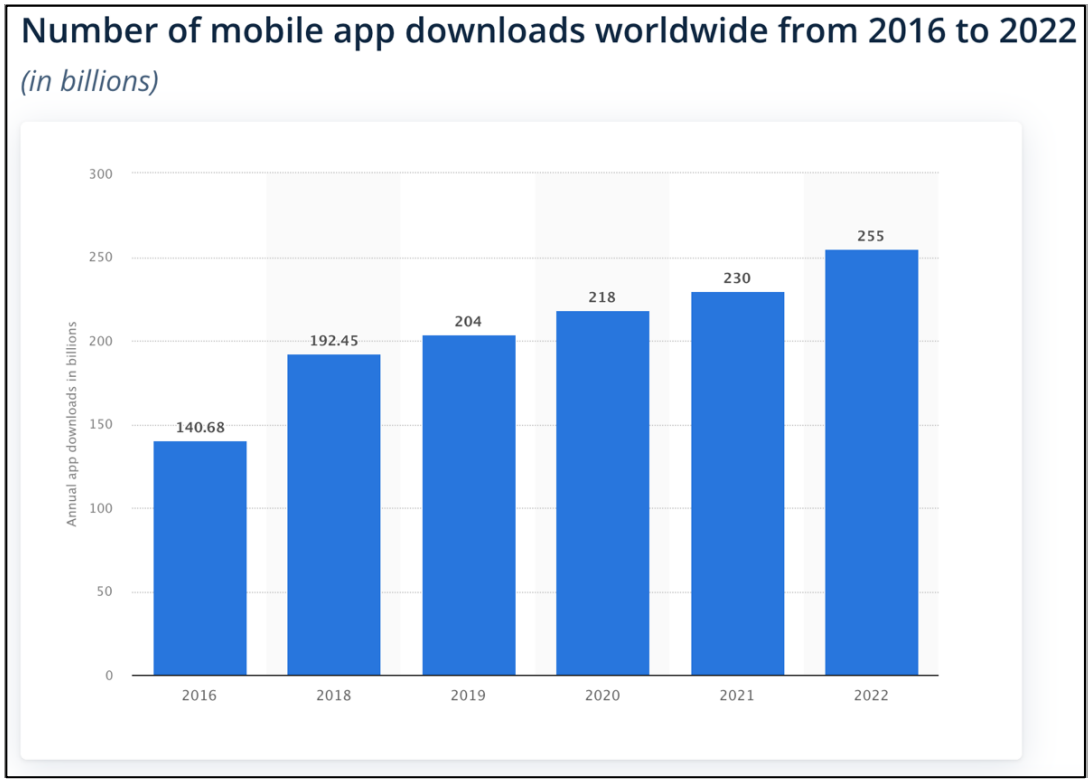


Figura 5 – Descargas de aplicaciones iOS y Android en los últimos años [6]

Con el creciente uso de los dispositivos móviles y sus aplicaciones se ha incrementado las vulnerabilidades en estas y en algunos casos desencadenando fugas de información, afectando principalmente a los usuarios. A continuación, a manera de resumen se describen cuatro casos de vulnerabilidades descubiertas en aplicaciones móviles muy usadas en los últimos años.

Aplicación Amazon Ring

Una vulnerabilidad en la aplicación Ring's Neighbors exponía la ubicación precisa y direcciones de los usuarios que habían publicado en la aplicación [7].

Ring, la empresa de videoporteros y seguridad para el hogar adquirida por Amazon, lanzó *Neighbors* en 2018 como una función independiente. *Neighbors* es una aplicación de vigilancia del vecindario, que permite a los usuarios alertar de forma anónima a los residentes cercanos sobre delitos y problemas de seguridad pública.

Si bien las publicaciones de los usuarios son públicas, la aplicación no muestra nombres ni su ubicación precisa, aunque la mayoría incluye videos tomados por *Ring Doorbells* y cámaras de seguridad. La vulnerabilidad hizo posible recuperar los datos de ubicación de los usuarios que publicaron en la aplicación, incluidos aquellos que denuncian delitos. Sin embargo, los datos expuestos no eran visibles directamente en la interfaz de la aplicación. Más bien, el error estaba al recuperar los datos, incluida la latitud, longitud del usuario y dirección del usuario, de los servidores de Ring.

La aplicación móvil de Slack expone las credenciales de los usuarios

La aplicación *Slack* estaba registrando accidentalmente las credenciales en texto plano en los dispositivos Android, y los clientes afectados fueron notificados por correo electrónico para restablecer sus contraseñas. Se enviaron correos electrónicos a los clientes de *Slack* que se vieron afectados. La compañía informó que esto solo afectó a un pequeño subconjunto de usuarios de Android (menos del 1 %) [8].

El 21 de diciembre de 2020, *Slack* introdujo un error que provocó que algunas versiones de la aplicación para Android registraran las credenciales de usuario en texto sin cifrar en los dispositivos. La organización identificó el problema el 20 de enero de 2021 y lo solucionó el 21 de enero de 2021, para esta fecha lanzaron una actualización de la aplicación de Android y bloquearon el uso de las versiones afectadas.

La aplicación para compartir archivos SHAREit vulnerable a la ejecución remota de código

La aplicación Android con más de mil millones de descargas sufría vulnerabilidades sin corregir durante tres meses después de que la firma Check Point informara sobre las vulnerabilidades detectadas [9].

Las vulnerabilidades detectadas afectaban a la versión de Android de la aplicación SHAREit, una aplicación móvil que permite a los usuarios compartir archivos con amigos entre dispositivos personales.

Con una aplicación maliciosa instalada en el dispositivo o a través de un ataque de red *Man-in-the-middle* por sus siglas en inglés (MITM), un atacante podía enviar comandos a la aplicación SHAREit y secuestrar sus funciones legítimas para ejecutar código personalizado, sobrescribir los archivos locales de la aplicación, o instalar aplicaciones de terceros sin el conocimiento del usuario.

Vulnerabilidad en Facebook Messenger permitió a los usuarios de Android espiarse unos a otros

Facebook Messenger para Android permitía a las personas que llamaban, escuchar el entorno de otros usuarios sin permiso, es decir antes de que la persona del otro lado respondiera la llamada [10].

En noviembre del año 2020 el investigador Natalie Silvanovich del proyecto Zero de Google descubrió una vulnerabilidad. Con dicha vulnerabilidad, enviando un tipo especial de mensaje conocido como `SdpUpdate`, la llamada se conectaba al dispositivo de la persona llamada antes de que ésta respondiera.

Normalmente, el dispositivo de la persona que recibe la llamada no transmite audio hasta que el usuario haya dado su consentimiento para aceptar la llamada.

Silvanovich encontró el problema en la versión 284.0.0.16.119 de *Facebook Messenger* para Android. El investigador también proporcionó un código de explotación de prueba de concepto (PoC) basado en Python para reproducir la vulnerabilidad.

Incidentes como los anteriores que se mencionan pueden ocurrir fácilmente cuando los desarrolladores no adoptan prácticas de codificación seguras y las organizaciones no realizan auditorias exhaustivas de seguridad en aplicaciones móviles para descubrir vulnerabilidades y potenciales fugas de información. Es por ello por lo que surge la iniciativa de concentrar en un solo documento los conceptos, herramientas y recursos libres para llevar a cabo auditorias de seguridad a las aplicaciones móviles, que sirva como guía y punto de entrada para aquellos estudiantes y personas que les apasione la ciberseguridad en aplicaciones móviles.

1.3 Objetivos

Objetivo General:

- El objetivo principal de este proyecto es proporcionar una metodología general para el proceso de auditoría de aplicaciones móviles para las plataformas Android e iOS, sin importar el *framework* y lenguaje utilizado durante el desarrollo.

Objetivos específicos:

- Seleccionar y hacer uso de un conjunto de herramientas disponibles para el análisis de vulnerabilidades en aplicaciones móviles para las dos plataformas Android e iOS, que cuenten con soporte por parte de sus desarrolladores.
- Analizar los modelos de seguridad para aplicaciones móviles actuales y los estándares de verificación de seguridad más reconocidos y aceptados.
- Proporcionar las bases del marco legal al que deben ceñirse los desarrolladores de aplicaciones móviles en relación con la protección de datos personales, financieros y de salud, entre otros.
- Realizar el análisis de seguridad de una aplicación en las plataformas Android e iOS para identificar posibles vulnerabilidades de las aplicaciones bajo estudio y evaluar su severidad en función de los resultados obtenidos.
- Presentar a modo de conclusión los resultados alcanzados a lo largo del proyecto y trabajo a futuro.
- Proporcionar una lista de configuraciones de seguridad para reducir la probabilidad de explotación cuando existe una vulnerabilidad y proteger la información del usuario en los dispositivos móviles.

2 MARCO TEÓRICO

Este capítulo, se enfoca en la descripción de conceptos fundamentales para un auditor durante la ejecución de una auditoría de aplicaciones móviles. Algunos de estos conceptos no son exclusivos de aplicaciones móviles y son aplicables a diversas auditorías de seguridad, como las relacionadas con aplicaciones web, API, infraestructura, nube, entre otros. Un ejemplo de esto son las definiciones de: amenazas, vulnerabilidades, métrica evaluación de vulnerabilidades y marco legal.

2.1 Aplicaciones móviles

La seguridad en aplicaciones móviles es esencial sin importar el tipo de aplicación. Dependiendo de cada aplicación pueden encontrarse diferentes tipos de vulnerabilidades, por ejemplo, las aplicaciones nativas pueden enfrentar vulnerabilidades de ejecución de código al acceder a funciones del dispositivo. Las aplicaciones híbridas y progresivas deben considerar vulnerabilidades web comunes, como inyección SQL y ataques XSS, debido a su naturaleza web.

En la actualidad, los sistemas operativos Android e iOS comprenden más del 99% de la cuota de mercado de sistemas operativos móviles según los datos de statcounter tal como se presenta gráficamente en la Figura 1. Además, el acceso a Internet desde dispositivos móviles ha superado el uso de las computadoras de escritorio tal a como se puede observar en Figura 2, lo que hace que los navegadores y las aplicaciones móviles sean el tipo más extendido de aplicaciones compatibles con Internet.

El diseño común de una aplicación es que pueda ejecutarse directamente en la plataforma para la que fue diseñada, sobre el navegador web o bien una combinación de ambas, por lo que es importante conocer el tipo de aplicación como punto de partida al iniciar una auditoría de seguridad de una aplicación móvil.

A continuación, se describen los tipos de aplicaciones y las características de cada uno.

Aplicaciones Nativas

Los sistemas operativos móviles, incluidos Android y iOS, vienen con un kit de desarrollo de software o *Software Development Kit* por sus siglas en inglés (SDK) para desarrollar aplicaciones específicas para el sistema operativo lo que se conoce como aplicación nativa.

Las aplicaciones nativas son aquellas que se construyen utilizando un lenguaje de programación específico, como Objective C/Swift para iOS o Java/Kotlin para Android, además, tienen capacidad de proporcionar el rendimiento más rápido con el mayor grado de fiabilidad. Por lo general, se adhieren a los principios de diseño específicos de la plataforma y debido a su estrecha integración con el sistema operativo, pueden acceder directamente a casi todos los componentes del dispositivo como: cámara, sensores, almacenes de claves respaldadas por hardware, etc.

Una desventaja de las aplicaciones nativas es que están diseñadas específicamente para una única plataforma, lo que implica que, si se necesita disponibilidad en múltiples plataformas, es necesario desarrollar una aplicación independiente para cada una de ellas, lo cual incrementa el esfuerzo, los costos y los tiempos de desarrollo.

Aplicaciones Web

Las aplicaciones web móviles son sitios diseñados para verse y sentirse como una aplicación nativa, pero se ejecutan sobre el navegador web del dispositivo y generalmente se desarrollan en HTML, JavaScript y CSS. Estas aplicaciones tienen una integración limitada con los componentes del dispositivo y en general no tienen buen rendimiento en comparación con las aplicaciones nativas.

Dado que una aplicación web generalmente se dirige a múltiples plataformas, su interfaz de usuario no sigue los principios de diseño de una plataforma específica. La mayor ventaja es la reducción de los costos de desarrollo y mantenimiento asociados con una única base de código, además de permitir a los desarrolladores distribuir actualizaciones sin involucrar a las tiendas de aplicaciones de cada plataforma.

Aplicaciones Híbridas

Las aplicaciones híbridas se ejecutan como una aplicación nativa, pero la mayoría de los procesos se basan en tecnologías web, lo que significa que una parte de la aplicación se ejecuta en un *WebView*. Una capa de abstracción entre la parte web y la nativa, permite el acceso a las capacidades del dispositivo para aplicaciones híbridas a las que no puede acceder una aplicación web pura. Dependiendo del *framework* utilizado para el desarrollo, una base de código puede resultar en múltiples aplicaciones que se dirigen a diferentes plataformas, con una interfaz de usuario muy parecida a la de la plataforma original, para la que se desarrolló la aplicación.

A continuación, se incluye una lista de algunos *frameworks* para desarrollo de aplicaciones híbridas:

- Apache Cordova
- Flutter
- Ionic
- React Native
- Xamarin

Aplicaciones Web Progresivas

Las aplicaciones web progresivas o *Progressive Web Apps* por sus siglas en inglés (PWA) se cargan como páginas web normales, pero difieren de las aplicaciones web habituales en varios aspectos, uno de los más destacados es que es posible trabajar

sin conexión, además, permite el acceso al hardware del dispositivo móvil, que comúnmente solo está disponible para aplicaciones nativas.

Las PWA combinan diferentes estándares abiertos que ofrecen los navegadores para proporcionar los beneficios de una buena experiencia móvil. Aunque son compatibles con Android e iOS, todavía no están disponibles todas las funciones del hardware. Por ejemplo, el *Face ID* o *ARKit* para realidad aumentada en iOS.

2.1.1 Estructura de las aplicaciones

En esta sección se detalla la estructura de los archivos de instalación de aplicaciones móviles para las plataformas Android e iOS. En el caso de Android, se analizan los archivos con extensión .apk (*Android Application Package*), mientras que para iOS se examinan los archivos con extensión .ipa (*iOS AppStore Package*).

Aplicación Android

Un archivo con extensión *.apk* es un paquete de instalación en Android que incluye el código fuente compilado, metadatos, librerías y recursos como imágenes, archivos de configuración de una aplicación. En términos generales, un apk es un archivo Zip que contiene los elementos necesarios para que funcione la aplicación. En la Figura 6 se puede observar la estructura de un archivo *apk*.

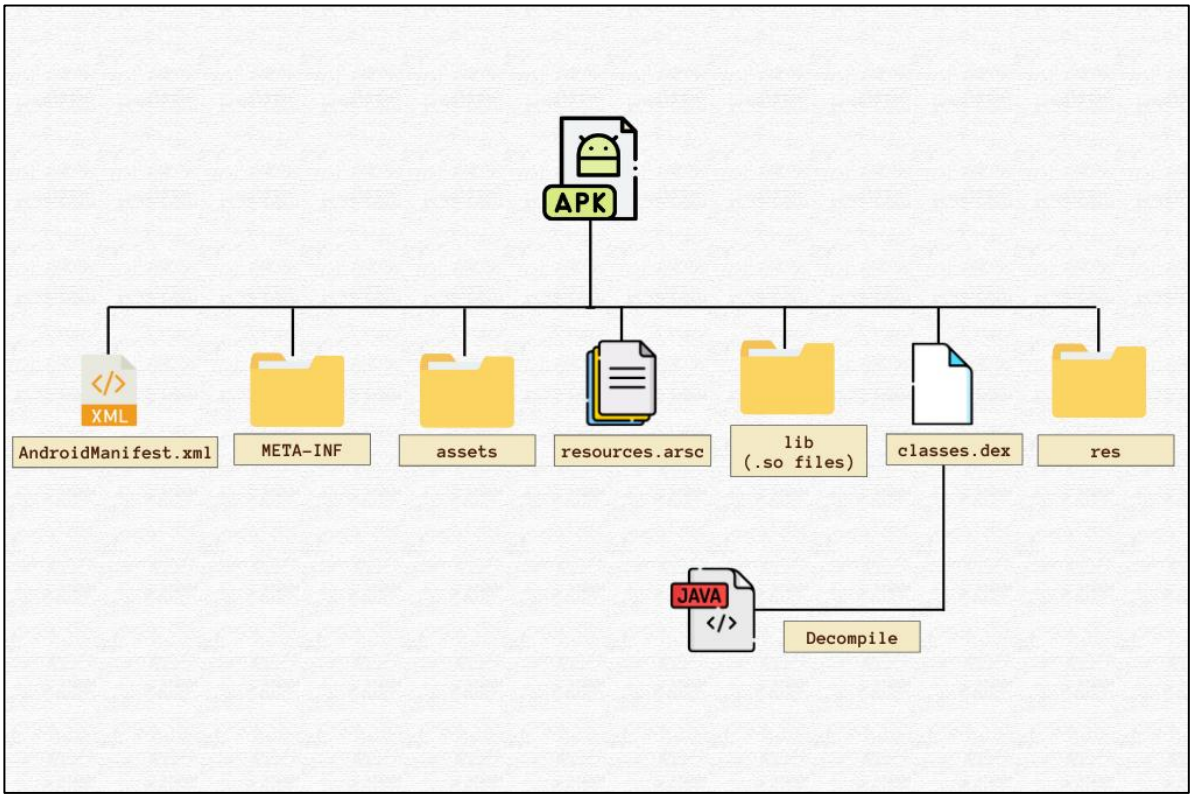


Figura 6 – Estructura de un archivo apk

A continuación, se describen los archivos, directorios y subdirectorios que componen un archivo de instalación apk.

a) El archivo *AndroidManifest.xml* describe información esencial como el nombre del paquete de la aplicación, los permisos necesarios, requerimientos de hardware y software, los componentes como: actividades, servicios, proveedores de contenido y receptores de emisión, entre otros elementos.

b) El directorio *META-INF* contiene información de metadatos relacionada con la firma y la configuración de la aplicación.

   Los siguientes archivos suelen encontrarse dentro de este directorio:

- *CERT.RSA*: Contiene el certificado de firma de la aplicación en formato binario. Este certificado se utiliza para verificar la autenticidad de la firma de la aplicación.
- *CERT.SF*: Archivo de resumen firmado que contiene información de resumen para cada archivo en el APK.
- *MANIFEST.MF*: Archivo de manifiesto firmado que contiene información sobre el manifiesto de la aplicación.
- ARCHIVE_HISTORY: Algunas herramientas de construcción de APK, como ProGuard, pueden generar un archivo llamado "ARCHIVE_HISTORY" que contiene información sobre las versiones anteriores del APK.
- *SIG*: Algunas versiones de Android Studio pueden incluir un directorio "SIG" que contiene información relacionada con la firma de la aplicación.
- *VERITY.KEY*: Puede contener claves necesarias para la verificación de la integridad del archivo.

La firma digital, representada por los archivos de certificado y los archivos de resumen firmado, es utilizada por el sistema operativo Android para verificar que la aplicación no ha sido modificada y que proviene de un origen confiable.

c) El directorio *assets* contiene archivos que la aplicación puede usar durante su ejecución. A diferencia de los recursos en el directorio "*res*", los archivos en este directorio no son procesados automáticamente por las herramientas de compilación de Android y se conservan en su forma original. Algunos de los formatos de archivos que se suelen alojar son .txt, .html, .json, .png, .jpg, .mp4, etc.

d) El archivo *resources.arsc* contiene los recursos XML compilados de la aplicación como: cadenas de texto, colores o estilos, etc. es creado durante el

proceso de compilación y se comprime para ahorrar espacio. Contiene información binaria optimizada para su rápido acceso y lectura por parte de la aplicación en tiempo de ejecución.

e) El directorio *lib* contiene bibliotecas nativas compiladas específicas de la arquitectura del dispositivo. Estas bibliotecas son compiladas en código nativo y se utilizan para proporcionar funcionalidades específicas a nivel del sistema o de hardware que pueden ser requeridas por la aplicación Android.

   Contiene varios subdirectorios correspondientes a diferentes arquitecturas de CPU. Los nombres de estos subdirectorios siguen la convención *lib/<arquitectura>*

   Por ejemplo:

   - *lib/armeabi*: Contiene bibliotecas específicas de la arquitectura *Advanced RISC Machine*, por sus siglas en inglés (ARM).
   - *lib/armeabi-v7a*: Contiene bibliotecas específicas de la arquitectura ARMv7.
   - *lib/arm64-v8a*: Contiene bibliotecas específicas de la arquitectura ARM64.
   - *lib/x86*: Contiene bibliotecas específicas de la arquitectura x86.
   - *lib/x86_64*: Contiene bibliotecas específicas de la arquitectura x86_64.

f) El archivo *classes.dex* contiene el código principal de la aplicación en formato de código de bytes que se ejecuta en la máquina virtual de Android, consiste en los archivos *.class* compilados que se basan en el código Java o Kotlin y se presentan de manera optimizada.

   El tamaño del archivo *classes.dex* depende del tamaño y complejidad del código fuente. Google Play Store y las herramientas de desarrollo de Android pueden aplicar técnicas de optimización, como la división del archivo DEX en

archivos más pequeños (multidex) para mejorar el rendimiento y reducir el tamaño de la aplicación.

g) El directorio *res* contiene todos recursos que no están compilados en el archivo *resources.arsc*. estos archivos son utilizados para definir aspectos visuales y funcionales de la aplicación, como la interfaz de usuario, gráficos, animaciones, valores constantes y más. Algunos de los subdirectorios que contiene son:

- *drawable*: Contiene recursos gráficos, como imágenes, íconos y otros elementos visuales.
- l*ayout*: Archivos XML que definen la disposición de los elementos en las pantallas de la aplicación. Estos archivos describen la estructura de la interfaz de usuario.
- *mipmap*: Contiene íconos de diferentes densidades de píxeles para su uso en dispositivos con diferentes resoluciones de pantalla.
- *values*: Incluye archivos XML que contienen valores constantes utilizados en la aplicación como: colores, dimensiones, estilos y configuraciones.
- *anim*: Contiene archivos XML que definen animaciones utilizadas en la aplicación.
- *animator*: Similar a "anim", este directorio puede contener archivos XML que definen animaciones.
- *xml*: Aloja archivos XML que definen datos estructurados o configuraciones específicas de la aplicación.
- *raw*: Aquí se pueden colocar archivos que no se deben procesar (como archivos de audio o video) y que deben ser accesibles sin procesamiento especial.
- *menu*: Contiene archivos XML que definen menús de la aplicación.

Por otra parte, las aplicaciones Android tienen varios componentes que permiten la comunicación entre otras aplicaciones del dispositivo, descritos a continuación:

Activities

Una *activity* de Android es una pantalla de la interfaz de usuario de la aplicación. Una aplicación de Android puede contener una o más *activities*, lo que significa una o más pantallas.

Broadcast Receiver

Los *Broadcast Receiver* son componentes que permiten que las aplicaciones reciban notificaciones de otras aplicaciones y del propio sistema operativo. Con esto, pueden responder a los eventos ya sea del propio sistema operativo o iniciados por otras aplicaciones. En términos generales, se utilizan para actualizar las interfaces de usuario, iniciar servicios, actualizar el contenido y crear notificaciones.

Content provider

Los *content providers* son componentes que administran el acceso a un repositorio central de datos, forman parte de una aplicación Android y están principalmente orientados a que los usen otras aplicaciones a través de un objeto de tipo cliente. Juntos, los *content providers* y clientes ofrecen una interfaz estándar y uniforme para compartir y acceder a los datos. Este componente tiene dos atributos de permisos separados que se pueden configurar en el archivo *AndroidManifest*, *android:readPermissio*n que restringe quién puede leer del componente y *android:writePermission* restringe quién puede escribir.

Para hacer uso de este componente se requiere de un URI o Localizador de Recursos Uniforme (por sus siglas en inglés, *Uniform Resource Identifier*) de contenido, es un elemento que identifica datos de un proveedor. Los URI de contenido incluyen el

nombre simbólico de todo el proveedor (su autoridad) y un nombre que apunta a una tabla (una ruta de acceso), generalmente tiene una ruta de acceso por cada tabla que expone.

A continuación, se presenta un ejemplo de la estructura de un URI de contenido:

```
content://<autoridad>/<ruta_de_acceso>/<id>
```

`content://`: Este es el esquema utilizado para todas las URIs de contenido en Android.

`<autoridad>`: Este es el nombre del *Content Provider*, que usualmente coincide con el nombre del paquete de la aplicación que lo define. Además, permite al sistema y a las aplicaciones identificar de forma única el *Content Provider*.

`<ruta_de_acceso>`: Esta parte de la URI especifica la tabla, vista, o recurso del *Content Provider* que se está accediendo. Puede ser una ruta simple o una jerarquía de sub-rutas que representan diferentes niveles de datos.

`<id>`: (Opcional) Esta parte es un identificador específico del recurso o registro al que se desea acceder. Si se incluye, se está refiriendo a un elemento individual dentro del *Content Provider*.

WebView

La clase *WebView* es una extensión de la clase *View* de Android que permite mostrar páginas web embebidas en las aplicaciones. En términos generales es una versión limitada del navegador Web debido a que no incluye las funciones del navegador completamente, como controles de navegación o una barra de direcciones. De forma predeterminada, todo lo que hace es mostrar una página web.

Aplicación iOS

Las aplicaciones iOS se distribuyen como un paquete con extensión .ipa, este es un archivo comprimido que contiene el código de la aplicación compilado, los recursos y metadatos, necesarios para definir una aplicación completa. Estos paquetes no son más que un archivo zip que se puede descomprimir, en la figura 7 se puede observar su estructura

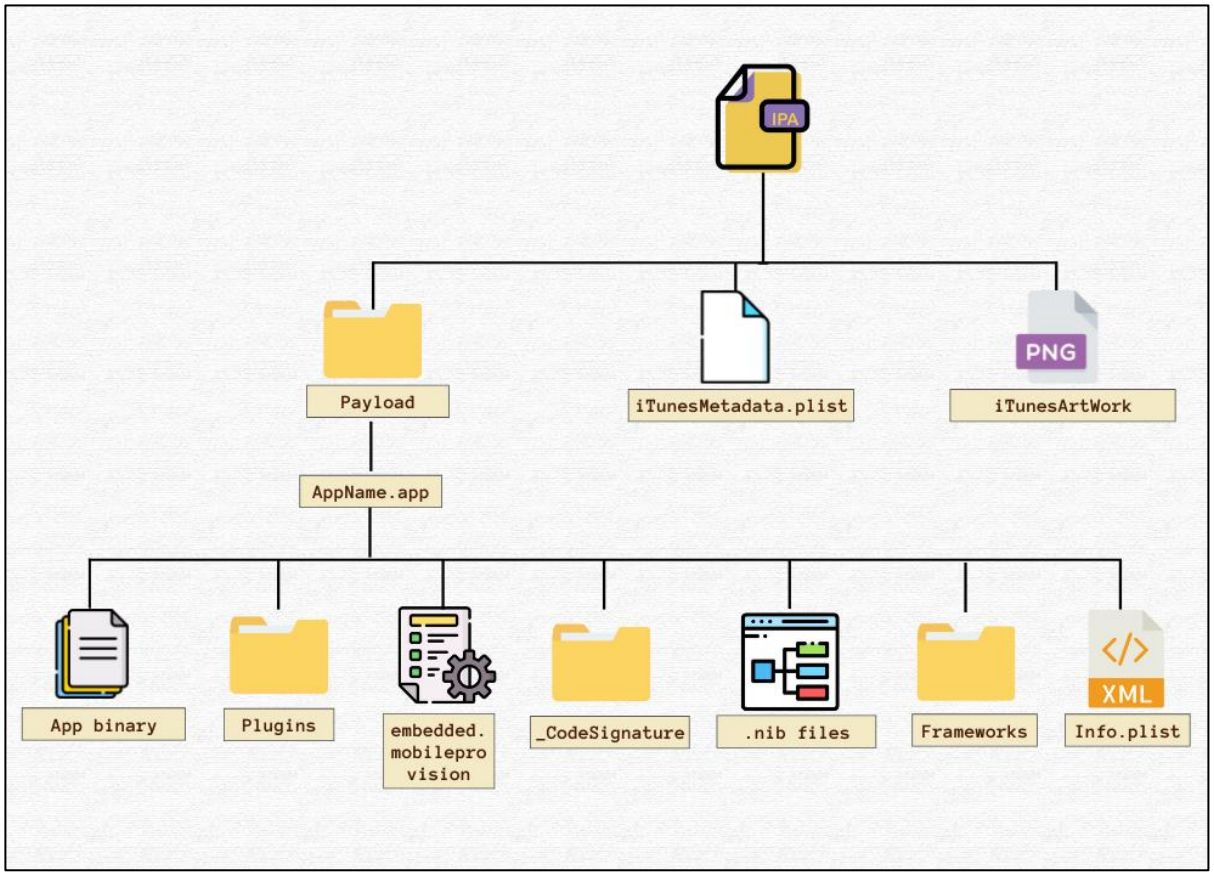


Figura 7 – Estructura de un archivo .ipa

Los archivos, directorios y subdirectorios que componen el paquete de instalación .ipa se describen a continuación:

a) El archivo *iTunesArtWork* es una imagen en formato PNG de 512 x 512 píxeles que se utiliza como icono de la aplicación en iTunes y App Store.

b) El archivo *iTunesMetadata.plist* contiene los metadatos relevantes de la aplicación, incluidos detalles como el nombre del desarrollador, el identificador del paquete y la información de derechos de autor.

c) La carpeta *Payload* contiene todos los datos de la aplicación.

- *Application.app* esta carpeta contiene los siguientes elementos, además imágenes estáticas y otros recursos de aplicación.

    - *App binary* contiene el binario ejecutable de la aplicación.
    - *Plugins* contiene las extensiones de la aplicación, ejecutadas por la aplicación.
    - *Embedded.mobileprovision* este es el archivo de aprovisionamiento original empaquetado por la aplicación, ayuda a los desarrolladores a volver a firmar una aplicación sin requerir del IDE de desarrollo Xcode.
    - *_CodeSignature* este archivo es responsable de verificar la integridad del archivo .app de la aplicación, es decir, que sea exactamente el mismo que fue firmado por el desarrollador.
    - *.nib* los archivos con esta extensión describen la interfaz de usuario de la aplicación y cómo interactúa con su lógica.
    - *Frameworks* en esta carpeta contiene *frameworks* utilizados por la aplicación, cada *framework* residirá en su propia carpeta llamada Frameworks/.framework y contendrá su código y recursos nativos.
    - Info.plist en este archivo se describe la aplicación para el sistema operativo iOS, en este archivo se definen las siguientes propiedades de la aplicación: Nombre para mostrar de la aplicación, icono, versión, identificador único, nombre de archivo ejecutable, versiones de iOS compatible entre otros.

2.1.2 Almacenamiento en aplicaciones

Android proporciona dos tipos de ubicaciones de almacenamiento físico: *almacenamiento interno* y *almacenamiento externo*. En la mayoría de los dispositivos, el almacenamiento interno es más pequeño que el externo. Sin embargo, el almacenamiento interno siempre está disponible en todos los dispositivos.

Almacenamiento Interno: El almacenamiento interno se refiere a la porción de memoria del dispositivo que está reservada para el sistema operativo y las aplicaciones instaladas. Este tipo de almacenamiento es privado para cada aplicación, las aplicaciones no pueden acceder directamente a los archivos de otras aplicaciones en su almacenamiento interno, proporcionando así un nivel adicional de seguridad y privacidad.

Almacenamiento externo: En este directorio se almacenan archivos que se pretenden compartir con otras aplicaciones, como archivos multimedia, entre otros. Se suele confundir palabra "externo" con medios de almacenamiento extraíble como tarjetas SD; sin embargo, este es un directorio que puede considerarse como almacenamiento compartido. Es un sistema de archivos que puede contener una cantidad relativamente grande de datos y que se comparte para todas las aplicaciones del dispositivo. Tradicionalmente, es una tarjeta SD, pero también puede ser el almacenamiento integrado en un dispositivo que es distinto del almacenamiento interno protegido.
En la figura 8 se representan los dos tipos de almacenamiento gráficamente.

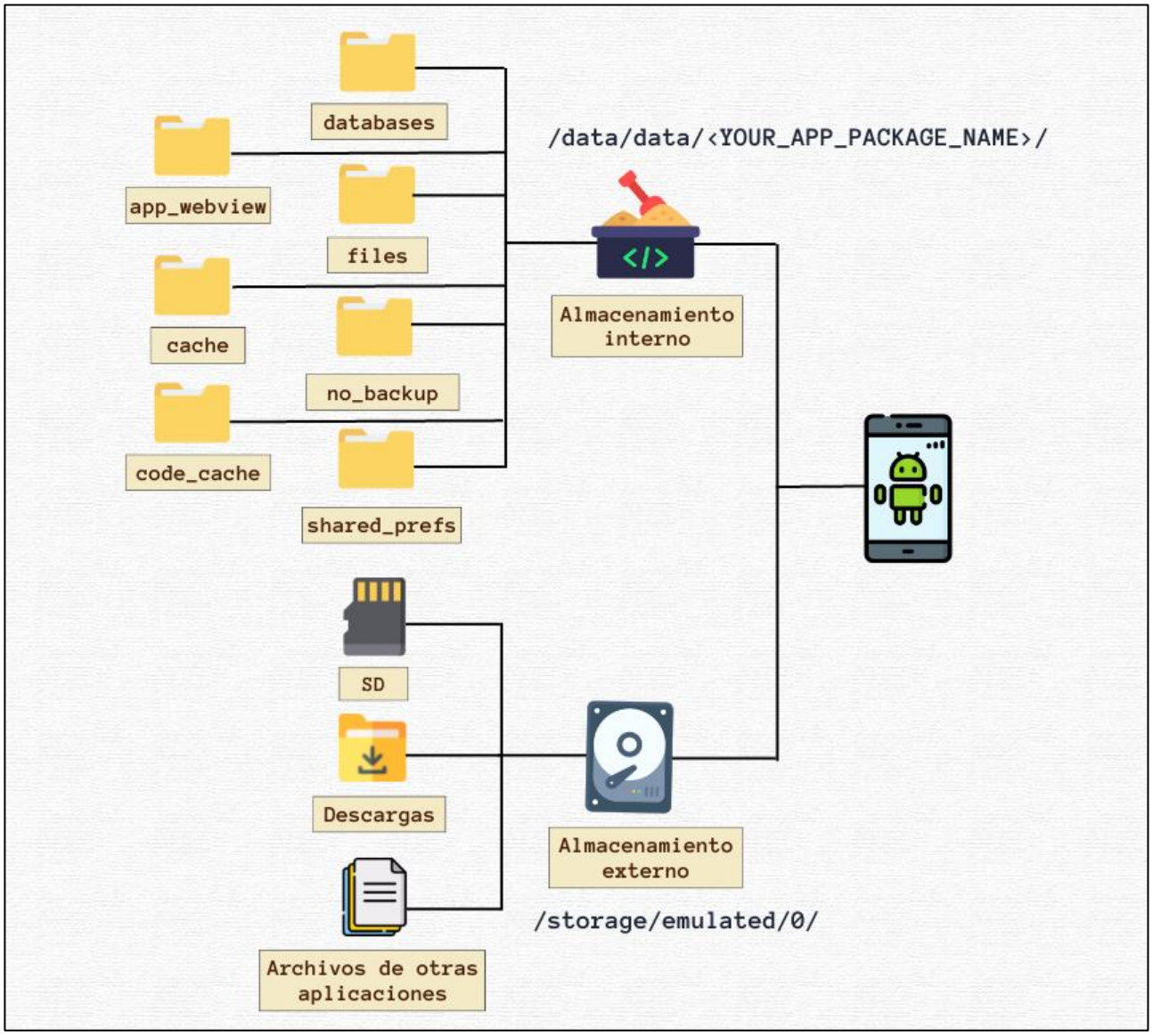


Figura 8 – Estructura de archivos del almacenamiento en Android

Los archivos, directorios y subdirectorios que se alojan en los dos tipos de almacenamiento de una aplicación Android se describen a continuación:

- *App_webview*: Contiene archivos de caché y configuración relacionados con el componente WebView. Entre los que se incluyen cookies, historial de

navegación, caché de recursos web, bases de datos de la aplicación web y otros datos relacionados con el WebView.

- *Databases*: Almacena datos estructurados en una base de datos privada mediante la biblioteca de persistencias como sqlite, realm, etc. Cada aplicación tiene su propio directorio "*databases*", que está protegido para que solo la aplicación tenga acceso a él.
- *Files*: Es un directorio dentro del almacenamiento interno, diseñado para archivos persistentes de la aplicación que no requiere ningún permiso del sistema para leer y escribir archivos en este directorio. Se puede acceder a esta carpeta con el método `filesDir().`
- *no_backup*: Este directorio es utilizado para almacenar archivos que no deben ser incluidos en las copias de seguridad de la aplicación. Estos archivos son excluidos de los procesos de respaldos automáticos del sistema, lo que significa que no se incluirán en los respaldos realizados por el usuario o por las herramientas automáticas.
- *Cache*: En este directorio se almacena la caché de la aplicación dentro del almacenamiento interno para guardar los datos. Al igual que con todo el almacenamiento específico de la aplicación, los archivos almacenados en este directorio se eliminan cuando el usuario desinstala la aplicación.
- *Code_cache*: Este directorio se utiliza para almacenar archivos temporales de cache de código compilado, caché de imágenes o caché de bases de datos. Estos archivos se generan cuando una aplicación se instala o se actualiza, esto ayuda a mejorar el rendimiento de la aplicación al permitir que la aplicación acceda rápidamente a los datos que ha utilizado recientemente.
- *shared_prefs*: se utiliza para almacenar datos en formato clave-valor en archivos XML, lo que permite que la aplicación almacene y recupere información de configuración, preferencias y estado de la aplicación.

En iOS, una aplicación tiene tres directorios contenedores, los cuales están diseñados para ofrecer un entorno seguro para la aplicación. Las interacciones de una aplicación iOS con el sistema de archivos están limitadas a las carpetas y archivos dentro del directorio *sandbox* (caja de arena) que limita el acceso de la aplicación al sistema de archivos. Esto es parte de la seguridad y privacidad implementada por Apple para proteger los datos de una aplicación y prevenir el acceso no autorizado a información sensible, esto se puede visualizar en la figura 9.

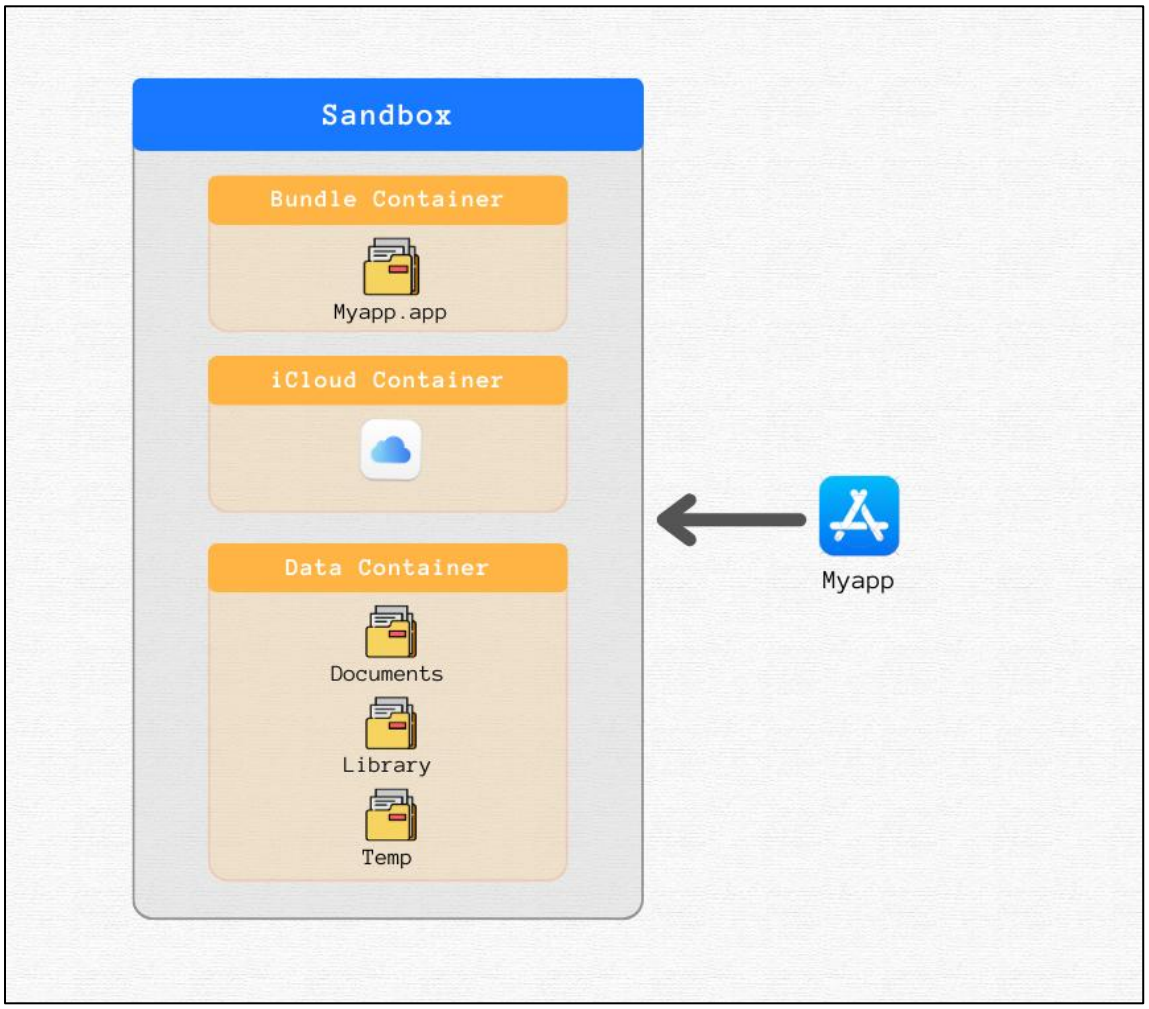


Figura 9 – iOS Sandbox

Durante la instalación de una nueva aplicación, se crean una serie de directorios contenedores dentro del directorio *sandbox*. Cada uno con una función específica. El

directorio *bundle container* contiene el paquete de la aplicación es decir el binario y archivos que componen la aplicación, el directorio *data container* contiene los datos de la aplicación y del usuario, este directorio se divide a su vez en una serie de subdirectorios que la aplicación puede utilizar para clasificar y organizar sus datos. La aplicación también puede solicitar acceso a directorios contenedores adicionales, por ejemplo, iCloud container que proporciona un sistema estructurado de almacenamiento de archivos para las aplicaciones que utilizan iCloud.

En la figura 10 se muestra una representación de los directorios contenedores de archivos para una aplicación iOS.

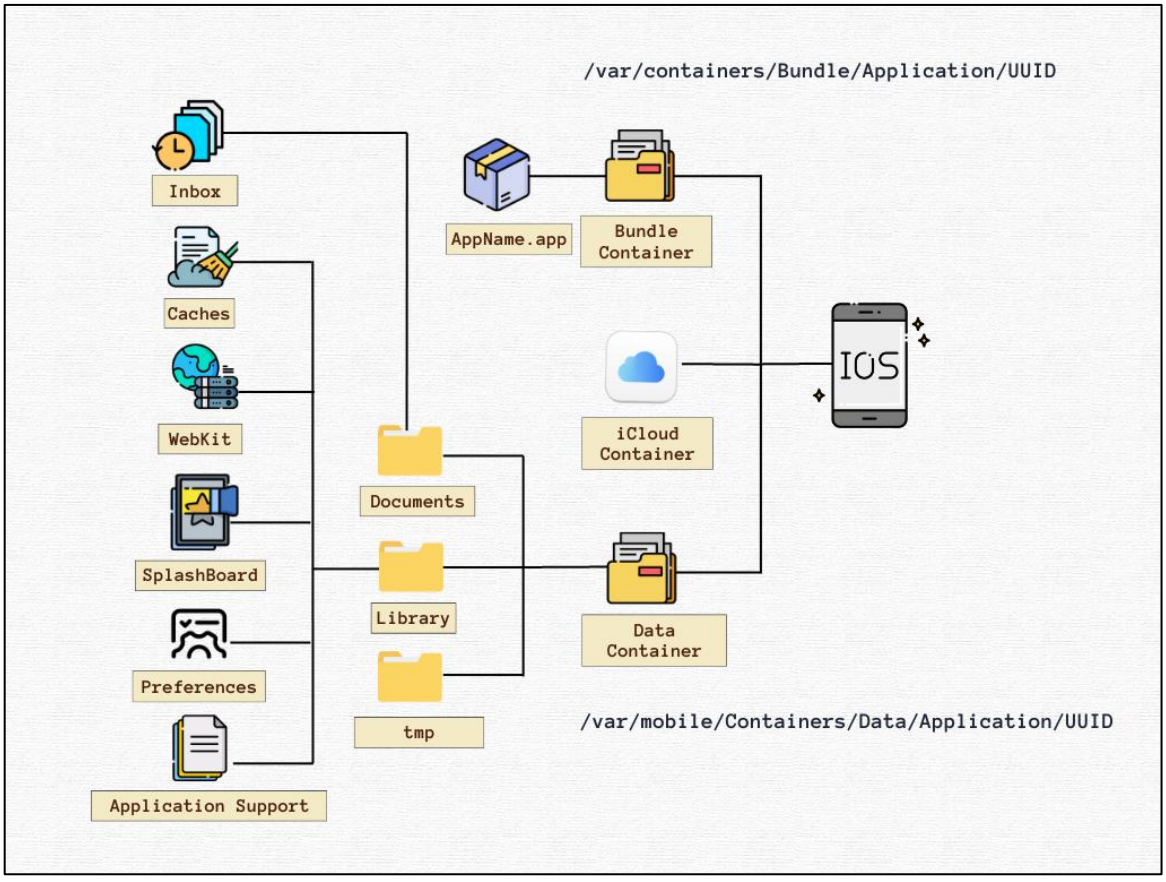


Figura 10 – Estructura de archivos de los contenedores en iOS

Los archivos, directorios y subdirectorios que se alojan en los distintos tipos de contenedores de una aplicación iOS se describen a continuación:

- *AppName.app*: Este es el paquete de la aplicación. Este directorio contiene la aplicación y todos sus recursos. Solo se permite la lectura de los recursos en este directorio. Para evitar manipulaciones, el directorio del paquete se firma al momento de la instalación. Si se logra escribir en este directorio, cambia la firma e impide que la aplicación se ejecute.

  Los archivos de este directorio no son respaldados en iCloud.
- *Documents*: En este directorio se almacenan archivos y documentos que son creados por el usuario y necesitan ser guardados de forma persistente, estos pueden ser accedidos por la aplicación y por el usuario, además, pueden ser respaldados y restaurados a través de la función de sincronización de documentos de iCloud.
    - *Inbox:* Este directorio se utiliza para recibir archivos adjuntos de correo electrónico o archivos enviados desde otras aplicaciones.

      Cuando un usuario selecciona un archivo adjunto en un correo electrónico o envía un archivo desde otra aplicación a través de la opción "*Compartir*", el archivo se guarda temporalmente en la carpeta *Inbox*. La aplicación puede acceder a los archivos en esta carpeta y procesarlos según sea necesario, estos archivos son temporales y solo están disponibles para la aplicación por un tiempo limitado. Por otra parte, es importante que la aplicación elimine los archivos en esta carpeta cuando ya no sean necesarios, ya que si los archivos se acumulan podrían ocupar un espacio significativo en el dispositivo y afectar el rendimiento del sistema.
- *Library*: Es un directorio donde la aplicación puede almacenar archivos y datos que son necesarios para su funcionamiento, pero que no deben ser accesibles para el usuario directamente.

  Esta carpeta solo es accesible para la propia aplicación y no puede ser vista o accedida por otras aplicaciones. Su contenido puede respaldarse y restaurarse

a través de la función de sincronización de documentos de iCloud, excepto por el subdirectorio “*Caches”*.

Dentro de la carpeta "*Library*", una aplicación puede tener los siguientes subdirectorios:

- Caches: contiene archivos temporales que la aplicación utiliza para mejorar el rendimiento. Estos archivos pueden ser eliminados automáticamente por el sistema operativo cuando se necesita liberar espacio de almacenamiento.
- WebKit: contiene archivos y recursos relacionados con el motor de renderizado web utilizado por la aplicación WebKit. Este directorio puede contener subdirectorios como "*LocalStorage*", "*WebsiteData*", "*WebDatabase*" y otros, que contienen datos específicos de la aplicación. Estos subdirectorios pueden contener datos de almacenamiento en caché, bases de datos web, cookies, imágenes y otros archivos descargados desde la web.
- Spashboard: en este directorio se almacenan imágenes de pantalla de inicio personalizadas que se pueden descargar desde un servidor externo y usar en la aplicación, el uso de imágenes personalizadas de pantalla de inicio debe cumplir con las pautas de diseño y rendimiento de Apple. Además, dentro del subdirectorio */Snapshots* el sistema operativo almacena las imágenes de pantalla de inicio en caché en este directorio para acelerar el tiempo de carga de la aplicación.
- Preferences: contiene los archivos de configuración de la aplicación, que pueden incluir información como la configuración del usuario, las preferencias de la aplicación y los datos de autenticación.
- Application Support: contiene datos adicionales que la aplicación necesita para funcionar correctamente, como bases de datos, archivos de respaldo, recursos adicionales y otros datos importantes.

- *Tmp*: En este directorio se almacenan archivos temporales que no necesitan persistir entre lanzamientos de la aplicación. La aplicación debería eliminar los archivos de este directorio cuando ya no los necesite, también el sistema puede purgar este directorio cuando la aplicación no se esté ejecutando.

Cada aplicación obtiene un UUID (Identificador único universal) aleatorio de 128 bits asignado para sus nombres de directorio. Por consiguiente, encontrar la carpeta correcta dentro del directorio /var/mobile/Containers/Data/Application/ simplemente navegando por el sistema de archivos no es una tarea trivial.

2.2 Conceptos en una Auditoría de seguridad

En esta sección se abordan varios aspectos fundamentales de una auditoría de seguridad general, estos se pueden aplicar a aplicaciones web, móvil, API, etc. Entre los conceptos a describir se encuentran la definición de amenazas, vulnerabilidades y riesgos ya que un error común es la confusión entre estos, además, se describe CVSS una métrica de evaluación de vulnerabilidades que no exclusiva para aplicaciones móviles y se usará a lo largo del caso práctico.

2.2.1 Amenazas y vulnerabilidades

A menudo resulta confuso los términos vulnerabilidad, amenaza y riesgo, por lo que es importante tener claro estos conceptos y conocer cómo se relacionan al realizar cualquier tipo de auditoría de seguridad [11].

Una **vulnerabilidad** se define como o fallo en un sistema de información que pone en riesgo la seguridad de la información, permitiendo que un atacante pueda comprometer la integridad, disponibilidad o confidencialidad de esta, por lo que es necesario encontrarlas y mitigarlas lo antes posible.

Una **amenaza** es toda acción que aprovecha una vulnerabilidad para atentar contra la seguridad de un sistema de información. Las amenazas pueden proceder de ataques como: fraude, robo, *malware*, sucesos físicos (incendios, inundaciones) o negligencia y decisiones institucionales como mal manejo de contraseñas. Desde el punto de vista de una organización las amenazas pueden ser tanto internas como externas.

El **riesgo** es la probabilidad de que una amenaza se materialice aprovechando una vulnerabilidad, causando pérdidas o daños en el sistema. Depende de dos factores: la probabilidad de que la amenaza se materialice aprovechando una vulnerabilidad y produciendo un daño a la organización. El producto de estos factores representa el riesgo. En la siguiente figura se observan, de manera simplificada, los elementos que componen el riesgo.

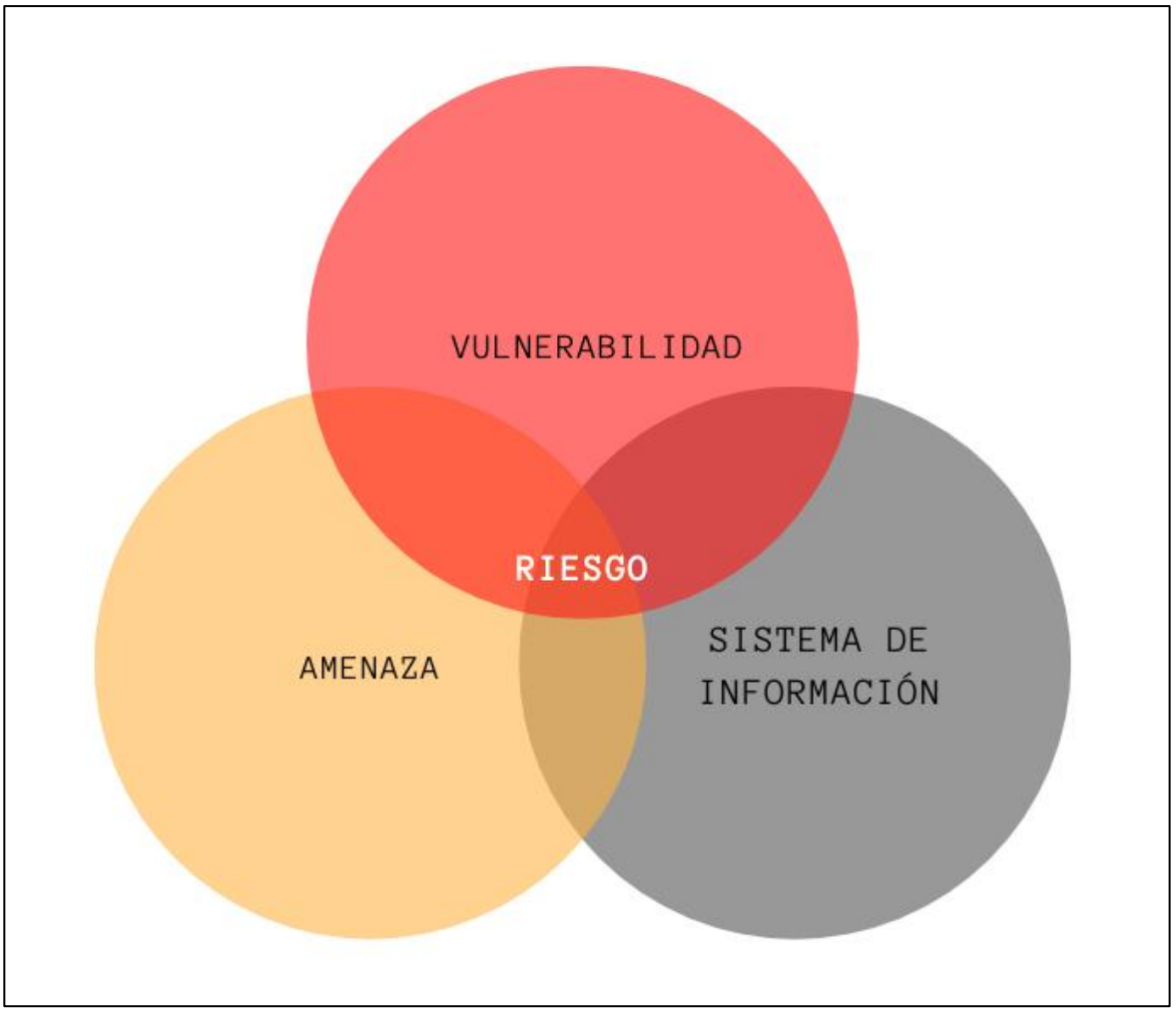


Figura 11 – Diagrama definición de riesgo

2.2.2 Modalidad de evaluación

Las modalidades de auditoría de seguridad se clasifican según el nivel de conocimiento y el acceso otorgado al responsable de ejecutar las pruebas de seguridad. El enfoque va desde las pruebas de caja negra, donde el auditor recibe un conocimiento mínimo del sistema objetivo, hasta las pruebas de caja blanca, donde se le otorga un alto nivel de conocimiento y acceso. Este nivel de conocimientos hace que las diferentes metodologías de prueba.

Caja Negra

Se realiza sin que el auditor tenga información previa sobre la aplicación, este proceso a veces se le conoce como “prueba de conocimiento cero”. El objetivo principal es realizar las pruebas de seguridad como lo haría un atacante real usando sus habilidades para descubrir y usar la información pública disponible [12].

Caja Blanca

Las pruebas de caja blanca a veces llamadas “pruebas de conocimiento completo” el auditor tiene pleno conocimiento de la aplicación como: credenciales, acceso al código fuente, documentación y diagramas. Este enfoque permite agilizar las pruebas debido al conocimiento de la aplicación y construir casos de prueba a medida y granulares [12].

Caja Gris

Las pruebas de caja gris se encuentran entre los dos tipos de pruebas mencionados anteriormente: se proporciona cierta información al auditor (generalmente solo credenciales de acceso) con permisos limitados y el nivel de conocimiento de un usuario [12].

2.2.3 Análisis de Vulnerabilidades

El análisis de vulnerabilidades es el proceso de identificar vulnerabilidades en una aplicación. Aunque esto se puede hacer de forma manual, los escáneres automatizados suelen utilizarse para identificar las principales vulnerabilidades. El análisis estático y dinámico son tipos de análisis de vulnerabilidades en aplicaciones móviles.

Análisis Estático

Las pruebas de seguridad estáticas o *Static Application Security Testing* conocidas por sus siglas en inglés (SAST) consisten en examinar los componentes de una aplicación sin ejecutarlos, analizando el código fuente de forma manual o automática para garantizar la implementación adecuada de los controles de seguridad. Es común utilizar un enfoque híbrido, es decir, revisiones automáticas y manuales. Los escaneos automatizados detectan las vulnerabilidades más comunes y fáciles de detectar, lo que se conoce en el mundo de la ciberseguridad como *low-hanging fruit* o la fruta más baja y las exploraciones manuales permiten identificar vulnerabilidades que afecten la lógica de negocio o que sean difíciles de detectar.

A continuación, se enlistan las tareas que incluye un Análisis Estático para aplicaciones móviles.

- Revisión del proceso de Autenticación
- Revisión de los niveles de Autorización de la aplicación
- Revisión de los métodos de almacenamiento de Información
- Gestión de las comunicaciones Seguras
- Búsqueda de información sensible accesible

Análisis Dinámico

Las pruebas de seguridad dinámicas conocidas por sus siglas en inglés *Dynamic Application Security Testing* (DAST) son las pruebas a una aplicación en tiempo de ejecución. El principal objetivo de análisis dinámico es identificar vulnerabilidades mientras la aplicación se está ejecutando y se realiza tanto en la capa de la plataforma móvil como en los servicios backend y las *API* donde se puede analizar el flujo de las peticiones y respuestas entre la aplicación y el servidor.

El análisis dinámico se usa generalmente para verificar los mecanismos de seguridad que brindan protección suficiente contra los tipos de ataque más frecuentes, como la divulgación de datos en tránsito, problemas de autenticación o autorización y errores de configuración del servidor.

2.2.4 Métrica de evaluación de vulnerabilidades

El Sistema de puntuación de vulnerabilidad común o *Common Vulnerability Scoring System* (CVSS), es un *framework* abierto que proporciona una forma de capturar las características principales de una vulnerabilidad y producir una puntuación numérica que refleja su gravedad y la severidad. La puntuación numérica se puede traducir a una representación cualitativa (como baja, media, alta y crítica). Se representa como una cadena, una representación textual comprimida de los valores utilizados para derivar la puntuación mejor conocido como vector [13].

CVSS se encuentra actualmente (noviembre 2022) en la versión 3.1, este proporciona tres grupos de métricas: Base, Temporal y Ambiental. La puntuación base refleja la gravedad de una vulnerabilidad de acuerdo con sus características intrínsecas, que son constantes a lo largo del tiempo, y asume el impacto razonable en el peor de los

casos en diferentes entornos implementados. La métrica temporal ajusta la gravedad base de una vulnerabilidad en función de factores que cambian con el tiempo, como la disponibilidad del código de explotación y la remediación. La métrica ambiental ajusta las métricas base y temporal a un entorno específico, las consideraciones incluyen la presencia de controles de seguridad que pueden mitigar algunas o todas las consecuencias de un ataque exitoso y la importancia relativa de un sistema vulnerable dentro de una infraestructura tecnológica.

La calificación que arroja el vector CVSS sirve para calcular la severidad de las vulnerabilidades descubiertas y como factor en la priorización de las actividades de mitigación de vulnerabilidades.

Los puntajes se pueden asignar a las calificaciones cualitativas definidas en la Figura 12.

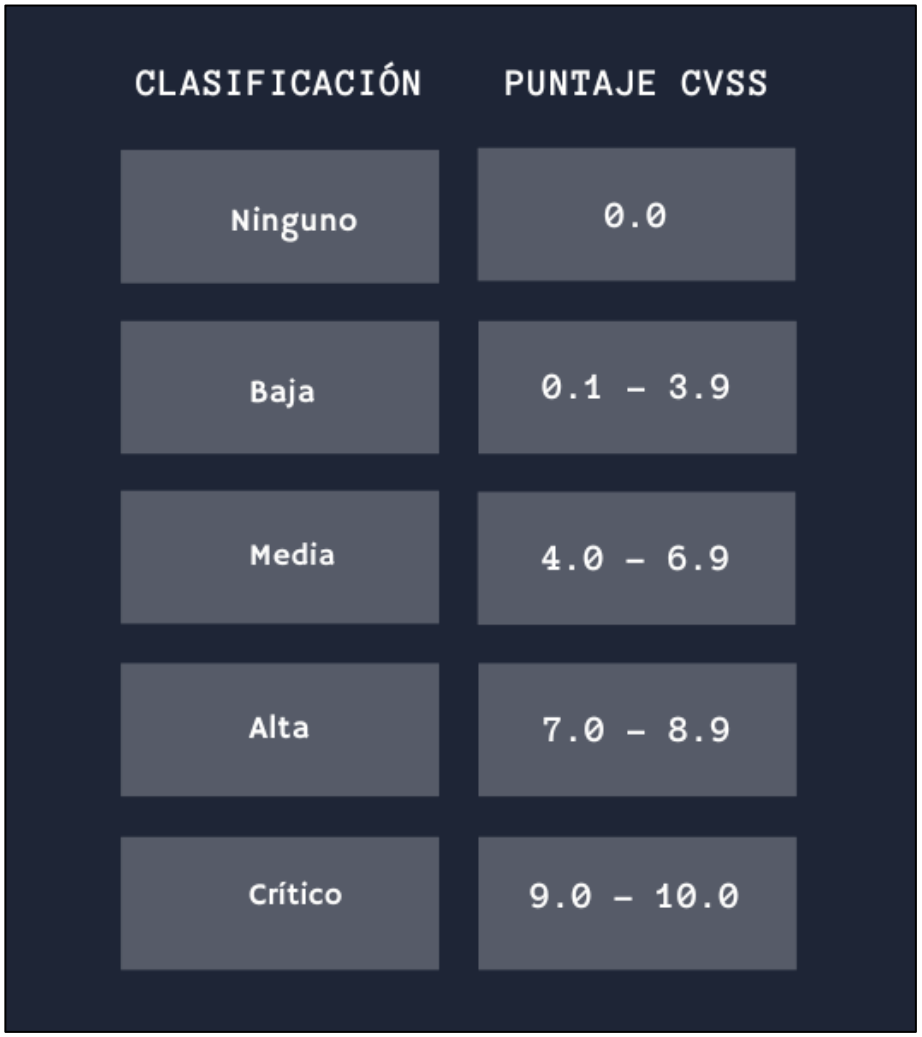


Figura 12 – Clasificaciones cualitativas de vulnerabilidades

CVSS es administrado por FIRST.Org, Inc., una organización sin fines de lucro con sede en EE. UU., Cuya misión es ayudar a los equipos de respuesta a incidentes de seguridad informática en todo el mundo. La documentación oficial se puede consultar en `https://www.first.org/cvss`

2.3 OWASP Mobile Application Security

Con el objetivo de identificar y combatir las causas que hacen que el software sea inseguro, en 2003 nace el proyecto OWASP. Un proyecto de código abierto, con una comunidad formada por empresas, organizaciones educativas e investigadores particulares de todo el mundo. Constituyen una comunidad de seguridad informática que trabaja para crear artículos, metodologías, documentación, herramientas y tecnologías abiertas que pueden ser usadas por cualquiera. Como parte de este proyecto en 2014 surge el OWASP Mobile Security Project, con el objetivo de proporcionar los recursos a desarrolladores y auditores para reforzar la seguridad de las aplicaciones móviles. Los documentos de este proyecto cubren muchos aspectos de la seguridad de las aplicaciones móviles, desde los requisitos de alto nivel hasta los detalles esenciales de implementación y casos de prueba. Con la última actualización de septiembre de 2022 el nombre del proyecto ha cambiado ligeramente de OWASP Mobile Security Project a OWASP Mobile Application Security [14]. A continuación, se listan los recursos que forman este proyecto:

1. *Mobile Security Testing Guide* (MSTG)
    - Es una guía técnica para verificar cada requisito en MASVS, así como las mejores prácticas de seguridad para aplicaciones de los sistemas operativos Android y iOS [12].
2. *Mobile Application Security Verification Standard* (MASVS)
    - Es un estándar que describe los requisitos de seguridad con las asignaciones de los niveles de verificación L1, L2 y R [15].
3. *Mobile Application Security Checklist*

- Lista de verificación de seguridad de aplicaciones móviles se puede utilizar para aplicar los requisitos de MASVS durante la auditoría de seguridad [16].

4. Mobile Application Security Checklist Crackmes

    - Es una colección de aplicaciones vulnerables. Estos desafíos se utilizan como ejemplos en todo el MASTG de OWASP. Por supuesto, también puede resolverlos por diversión y/o práctica.

Un recurso interesante que, si bien no forma parte del proyecto, es la lista OWASP Mobile Top 10 que resume los riesgos más importantes de las aplicaciones móviles [17].

2.3.1 Guía de pruebas de seguridad móvil (MSTG)

Es un manual para probar la seguridad de las aplicaciones. Describe los procesos técnicos para verificar los requisitos enumerados en la lista de verificación MASVS. Además, incluye una lista de casos de prueba, cada uno de los cuales se asigna a un requisito en la lista de verificación y proporciona recomendaciones detalladas y procedimientos de prueba por sistema operativo móvil.

La guía contiene las siguientes secciones principales:

1. La **Guía de pruebas general**: contiene una metodología de prueba de seguridad de aplicaciones móviles y técnicas generales de análisis de vulnerabilidades que se aplican a la seguridad de aplicaciones móviles. También contiene casos de prueba técnicos adicionales que son

independientes del sistema operativo, como autenticación y administración de sesiones, comunicaciones de red y criptografía.

2. La **Guía de pruebas de Android:** cubre las pruebas de seguridad móvil para la plataforma Android, incluidos los conceptos básicos de seguridad, casos de prueba, técnicas de ingeniería inversa, manipulación y prevención.

3. La **Guía de pruebas de iOS** cubre las pruebas de seguridad móvil para la plataforma iOS, incluida una descripción general del sistema operativo iOS, pruebas de seguridad, técnicas de ingeniería inversa, técnicas de manipulación y prevención.

2.3.2 Estándar de Verificación de Seguridad de Aplicaciones Móviles (MASVS)

El estándar de verificación o MASVS es un esfuerzo de la comunidad para establecer un marco de requisitos de seguridad necesarios para diseñar, desarrollar aplicaciones seguras y probar aplicaciones móviles en iOS y Android.

Se puede utilizar para establecer un nivel de confianza en la seguridad de las aplicaciones. Los requisitos se desarrollan con los siguientes objetivos en mente:

- Utilizar como métrica: para proporcionar un estándar de seguridad con el que los desarrolladores y propietarios de aplicaciones puedan comparar las aplicaciones móviles.

- Usar como guía: para brindar orientación durante todas las fases del desarrollo y prueba de aplicaciones móviles.

- Usar como base: para proporcionar una línea de base para la verificación de seguridad de aplicaciones móviles.

Niveles de verificación

MASVS define dos niveles de verificación de seguridad (MASVS-L1 y MASVS-L2), así como un conjunto de requisitos de resistencia contra ingeniería inversa (R). MASVS-L1 describe los requisitos de seguridad genéricos que se recomiendan para todas las aplicaciones móviles, MASVS-L2 define requisitos que deben aplicarse a aplicaciones que manejan datos altamente sensibles. MASVS-R describe controles de protección adicionales que se pueden aplicar si la prevención de amenazas del lado del cliente es un objetivo de diseño.

En la siguiente figura se puede observar gráficamente los niveles anteriormente descritos.

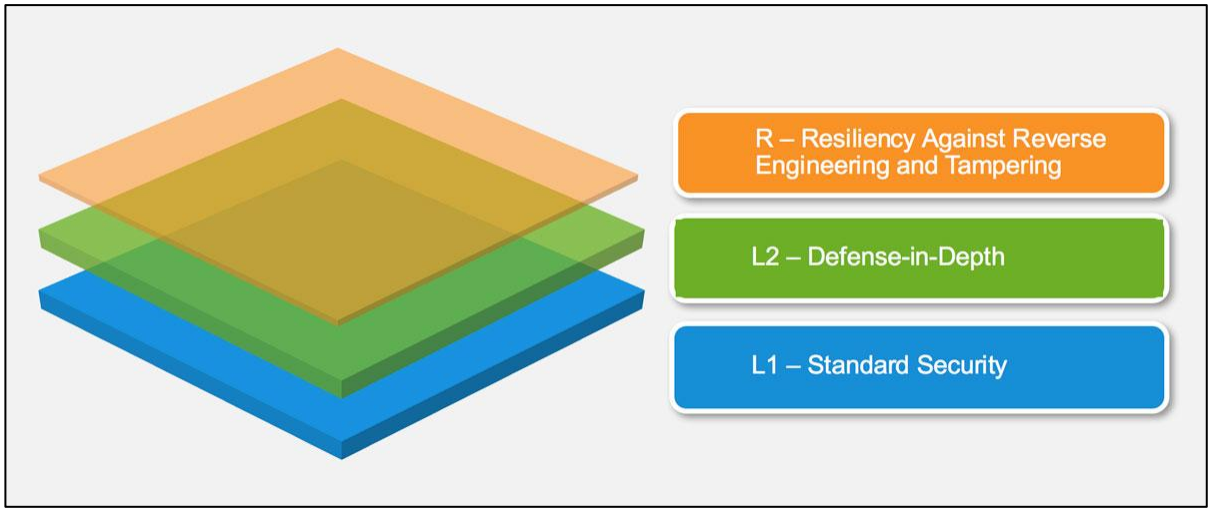


Figura 13 – Niveles de seguridad definidos en MASVS

MASVS-L1: Seguridad estándar

Una aplicación móvil que se adhiere con el nivel MASVS-L1 debería cumplir con los requisitos básicos en términos de calidad del código, manejo de datos sensibles e interacción con el entorno. Debe existir un proceso de prueba para verificar los controles de seguridad.

Este nivel es apropiado para todas las aplicaciones móviles.

MASVS-L2: Defensa en profundidad

MASVS-L2 Describe los controles de seguridad avanzados que van más allá de los requisitos estándar. Para cumplir con MASVS-L2, debe existir un modelado de amenazas y la seguridad debe ser una parte integral de la arquitectura y el diseño de la aplicación.

Este nivel es apropiado para aplicaciones que manejan datos muy sensibles, como aplicaciones de banca móvil.

MASVS-R: Resistencia contra ingeniería inversa y modificaciones

La aplicación tiene un nivel de seguridad alto, es resistente contra ataques del lado del cliente, como manipulación, modificación o ingeniería inversa para extraer código o datos confidenciales. Dicha aplicación aprovecha las características de seguridad del hardware o técnicas de protección de software suficientemente sólidas y verificables.

MASVS-R es recomendable en aplicaciones que manejan datos altamente sensibles y pueden servir como un medio para proteger la propiedad intelectual o proteger una aplicación contra manipulaciones.

¿Qué tipo de verificación elegir?

La implementación de los requisitos del nivel MASVS L2 aumenta la seguridad, mientras que al mismo tiempo aumenta el costo de desarrollo y podría impactar en la

experiencia del usuario final. En general, L2 debe usarse para aplicaciones siempre que tenga sentido desde una perspectiva de riesgo contra costo, es decir, cuando la pérdida potencial causada por una afectación de confidencialidad o integridad es mayor que el costo incurrido por los controles de seguridad adicionales [18].

A continuación, se presenta una serie de ejemplos al elegir el nivel adecuado de seguridad de acuerdo con la aplicación:

MASVS-L1

Todas las aplicaciones móviles deberían cumplir con los requisitos de seguridad descritos en este nivel, ya que en este se enumeran las mejores prácticas de seguridad que se pueden seguir, con un impacto razonable en el costo de desarrollo y la experiencia del usuario.

MASVS-L2

Aplicaciones para el cuidado de la salud: Para el sector de la salud de EE. UU., las consideraciones de cumplimiento incluyen la Ley de Responsabilidad y Portabilidad de Seguros de Salud (HIPAA) Reglas de Privacidad, Seguridad, Notificación de Incumplimiento y Regla . Las aplicaciones financieras deben garantizar el cumplimiento del Estándar de seguridad de datos de la industria de tarjetas de pago (PCI DSS), la Ley Gramm Leach Bliley y la Ley Sarbanes-Oxley (SOX).

MASVS L1+R

Aplicaciones móviles donde la protección de la propiedad intelectual es un objetivo. Los controles de resistencia enumerados en MASVS-R se pueden utilizar para aumentar el esfuerzo necesario para obtener el código fuente original y para impedir la manipulación.

En la industria de los video juegos con una necesidad de evitar modificaciones y trampas, como los juegos competitivos en línea. Hacer trampa es un tema importante en los juegos en línea, ya que una gran cantidad de gente que hace trampa conduce a una base de jugadores descontentos y, en última instancia, puede hacer que el juego falle. MASVS-R proporciona controles básicos contra la manipulación para ayudar a aumentar el esfuerzo de quienes quieren hacer trampa.

MASVS L2+R

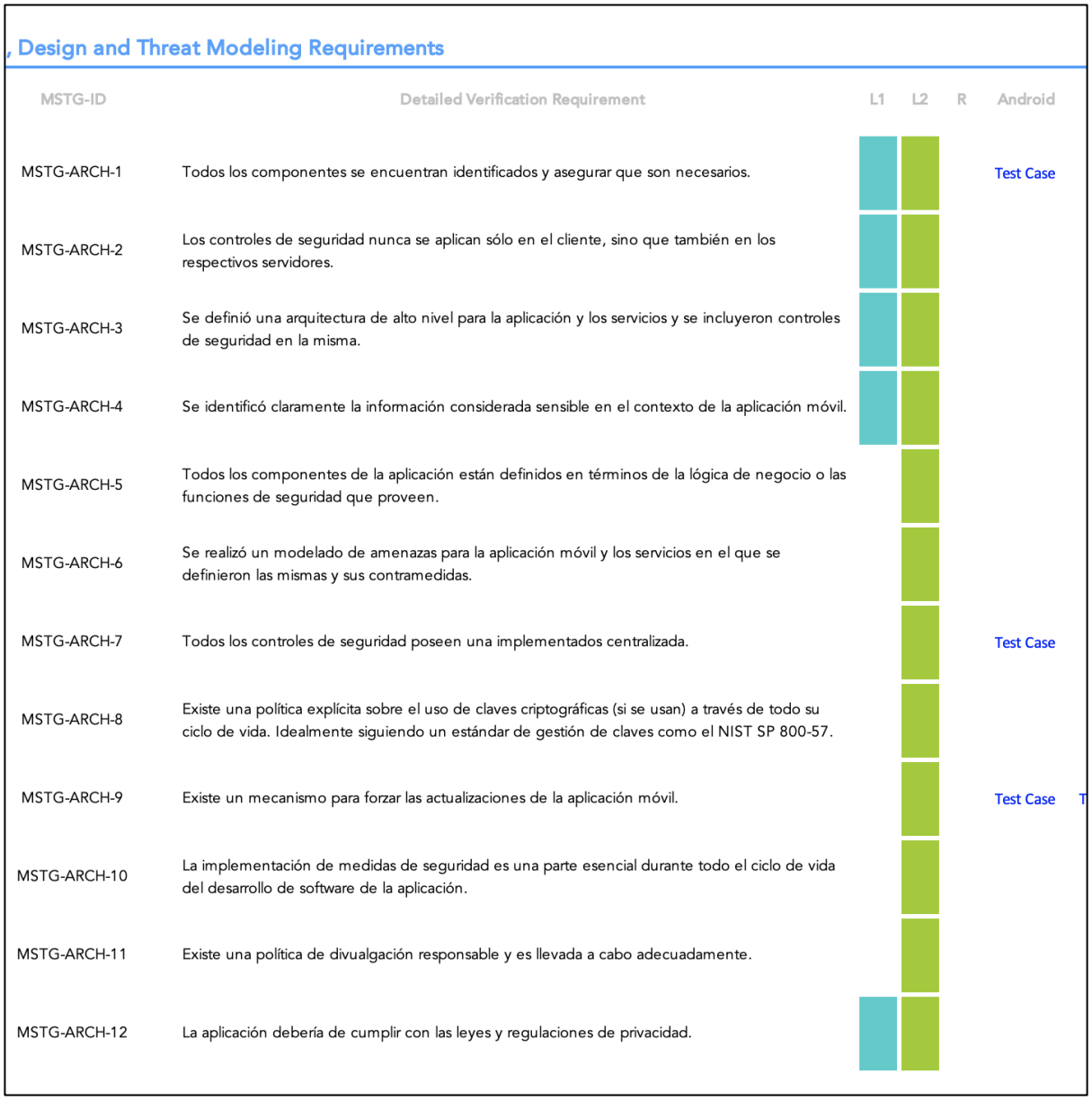
Industria financiera: aplicaciones de banca en línea que permiten al usuario mover fondos, donde las técnicas como la inyección de código y la manipulación en dispositivos comprometidos representan un riesgo.

Todas las aplicaciones móviles que, por diseño, necesitan almacenar datos confidenciales en el dispositivo móvil y, al mismo tiempo, deben admitir una amplia gama de dispositivos y versiones de sistemas operativos. En este caso, los controles de resistencia se pueden utilizar como una medida de defensa en profundidad para aumentar el esfuerzo de los atacantes que buscan extraer los datos confidenciales.

Las aplicaciones que incluyen contenido de pago idealmente deberían usar controles MASVS-L2 y protecciones del lado del servidor para proteger el contenido, sin embargo, puede haber casos en los que no sea posible utilizar la protección del lado del servidor. En esos casos, los controles MASVS-R deben aplicarse adicionalmente para aumentar el esfuerzo de ingeniería inversa y/o manipulación.

2.3.3 Lista de verificación de seguridad de aplicaciones móviles

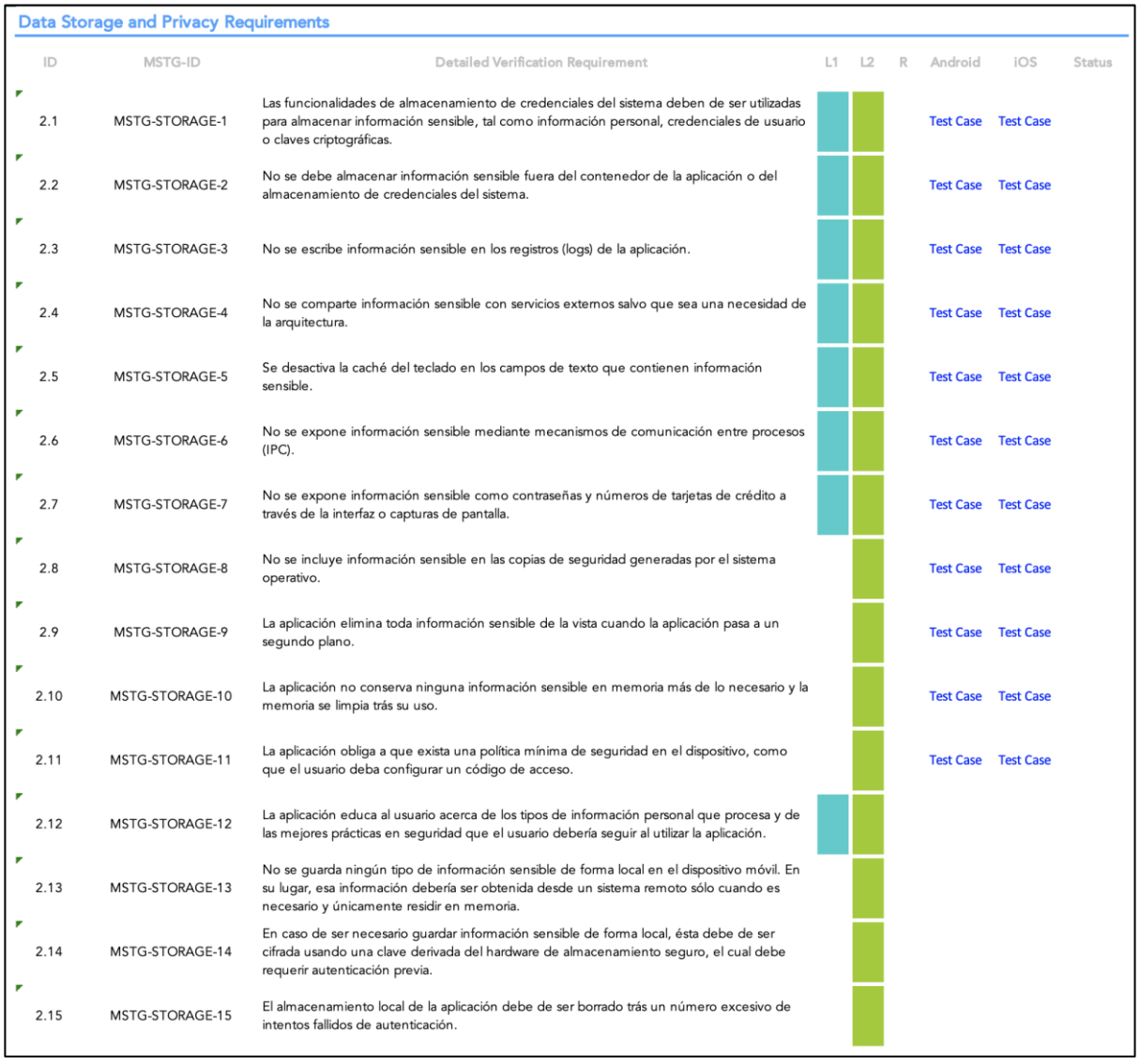
La lista de verificación se puede utilizar para aplicar los requisitos de seguridad durante la auditoría, esta lista se encuentra disponible en diferentes idiomas en la cuenta de GitHub del proyecto OWASP `https://github.com/OWASP/owasp-mastg/releases/latest/download/OWASP_MAS_Checklist.xlsx`. En las siguientes figuras se presenta parte de esta lista.



, Design and Threat Modeling Requirements

MSTG-ID	Detailed Verification Requirement	L1	L2	R	Android
MSTG-ARCH-1	Todos los componentes se encuentran identificados y asegurar que son necesarios.	■	■		Test Case
MSTG-ARCH-2	Los controles de seguridad nunca se aplican sólo en el cliente, sino que también en los respectivos servidores.	■	■		
MSTG-ARCH-3	Se definió una arquitectura de alto nivel para la aplicación y los servicios y se incluyeron controles de seguridad en la misma.	■	■		
MSTG-ARCH-4	Se identificó claramente la información considerada sensible en el contexto de la aplicación móvil.	■	■		
MSTG-ARCH-5	Todos los componentes de la aplicación están definidos en términos de la lógica de negocio o las funciones de seguridad que proveen.		■		
MSTG-ARCH-6	Se realizó un modelado de amenazas para la aplicación móvil y los servicios en el que se definieron las mismas y sus contramedidas.		■		
MSTG-ARCH-7	Todos los controles de seguridad poseen una implementados centralizada.		■		Test Case
MSTG-ARCH-8	Existe una política explícita sobre el uso de claves criptográficas (si se usan) a través de todo su ciclo de vida. Idealmente siguiendo un estándar de gestión de claves como el NIST SP 800-57.		■		
MSTG-ARCH-9	Existe un mecanismo para forzar las actualizaciones de la aplicación móvil.		■		Test Case T
MSTG-ARCH-10	La implementación de medidas de seguridad es una parte esencial durante todo el ciclo de vida del desarrollo de software de la aplicación.		■		
MSTG-ARCH-11	Existe una política de divualgación responsable y es llevada a cabo adecuadamente.		■		
MSTG-ARCH-12	La aplicación debería de cumplir con las leyes y regulaciones de privacidad.	■	■		

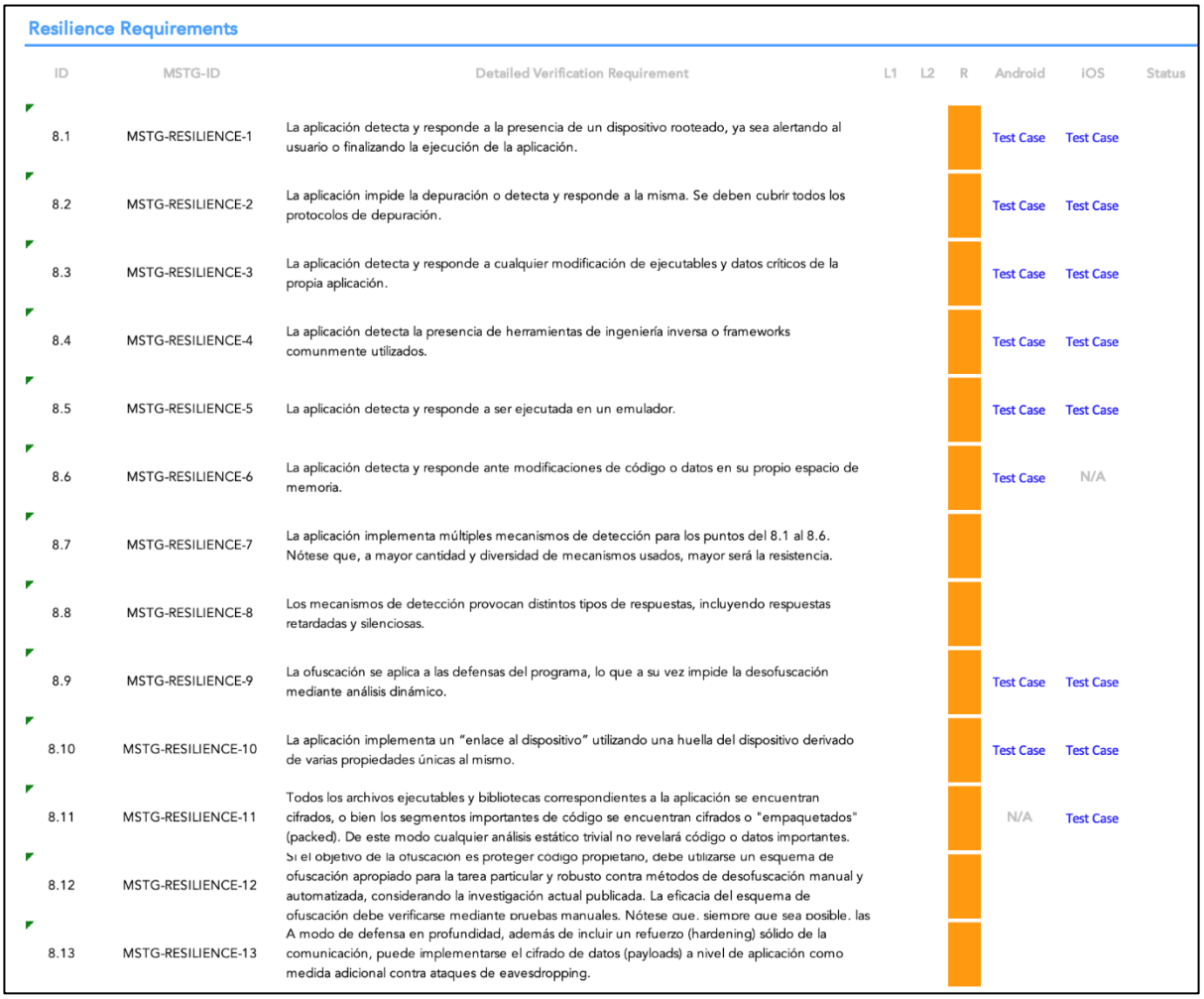
Figura 14 – Lista de verificación sección - Desing and Threat Modeling [19]



Data Storage and Privacy Requirements

ID	MSTG-ID	Detailed Verification Requirement	L1	L2	R	Android	iOS	Status
2.1	MSTG-STORAGE-1	Las funcionalidades de almacenamiento de credenciales del sistema deben de ser utilizadas para almacenar información sensible, tal como información personal, credenciales de usuario o claves criptográficas.	■	■		Test Case	Test Case	
2.2	MSTG-STORAGE-2	No se debe almacenar información sensible fuera del contenedor de la aplicación o del almacenamiento de credenciales del sistema.	■	■		Test Case	Test Case	
2.3	MSTG-STORAGE-3	No se escribe información sensible en los registros (logs) de la aplicación.	■	■		Test Case	Test Case	
2.4	MSTG-STORAGE-4	No se comparte información sensible con servicios externos salvo que sea una necesidad de la arquitectura.	■	■		Test Case	Test Case	
2.5	MSTG-STORAGE-5	Se desactiva la caché del teclado en los campos de texto que contienen información sensible.	■	■		Test Case	Test Case	
2.6	MSTG-STORAGE-6	No se expone información sensible mediante mecanismos de comunicación entre procesos (IPC).	■	■		Test Case	Test Case	
2.7	MSTG-STORAGE-7	No se expone información sensible como contraseñas y números de tarjetas de crédito a través de la interfaz o capturas de pantalla.	■	■		Test Case	Test Case	
2.8	MSTG-STORAGE-8	No se incluye información sensible en las copias de seguridad generadas por el sistema operativo.		■		Test Case	Test Case	
2.9	MSTG-STORAGE-9	La aplicación elimina toda información sensible de la vista cuando la aplicación pasa a un segundo plano.		■		Test Case	Test Case	
2.10	MSTG-STORAGE-10	La aplicación no conserva ninguna información sensible en memoria más de lo necesario y la memoria se limpia trás su uso.		■		Test Case	Test Case	
2.11	MSTG-STORAGE-11	La aplicación obliga a que exista una política mínima de seguridad en el dispositivo, como que el usuario deba configurar un código de acceso.		■		Test Case	Test Case	
2.12	MSTG-STORAGE-12	La aplicación educa al usuario acerca de los tipos de información personal que procesa y de las mejores prácticas en seguridad que el usuario debería seguir al utilizar la aplicación.	■	■				
2.13	MSTG-STORAGE-13	No se guarda ningún tipo de información sensible de forma local en el dispositivo móvil. En su lugar, esa información debería ser obtenida desde un sistema remoto sólo cuando es necesario y únicamente residir en memoria.		■				
2.14	MSTG-STORAGE-14	En caso de ser necesario guardar información sensible de forma local, ésta debe de ser cifrada usando una clave derivada del hardware de almacenamiento seguro, el cual debe requerir autenticación previa.		■				
2.15	MSTG-STORAGE-15	El almacenamiento local de la aplicación debe de ser borrado trás un número excesivo de intentos fallidos de autenticación.		■				

Figura 15 – Lista de verificación sección - Data Store and Privacy [19]



Resilience Requirements

ID	MSTG-ID	Detailed Verification Requirement	L1	L2	R	Android	iOS	Status
8.1	MSTG-RESILIENCE-1	La aplicación detecta y responde a la presencia de un dispositivo rooteado, ya sea alertando al usuario o finalizando la ejecución de la aplicación.				Test Case	Test Case	
8.2	MSTG-RESILIENCE-2	La aplicación impide la depuración o detecta y responde a la misma. Se deben cubrir todos los protocolos de depuración.				Test Case	Test Case	
8.3	MSTG-RESILIENCE-3	La aplicación detecta y responde a cualquier modificación de ejecutables y datos críticos de la propia aplicación.				Test Case	Test Case	
8.4	MSTG-RESILIENCE-4	La aplicación detecta la presencia de herramientas de ingeniería inversa o frameworks comunmente utilizados.				Test Case	Test Case	
8.5	MSTG-RESILIENCE-5	La aplicación detecta y responde a ser ejecutada en un emulador.				Test Case	Test Case	
8.6	MSTG-RESILIENCE-6	La aplicación detecta y responde ante modificaciones de código o datos en su propio espacio de memoria.				Test Case	N/A	
8.7	MSTG-RESILIENCE-7	La aplicación implementa múltiples mecanismos de detección para los puntos del 8.1 al 8.6. Nótese que, a mayor cantidad y diversidad de mecanismos usados, mayor será la resistencia.						
8.8	MSTG-RESILIENCE-8	Los mecanismos de detección provocan distintos tipos de respuestas, incluyendo respuestas retardadas y silenciosas.						
8.9	MSTG-RESILIENCE-9	La ofuscación se aplica a las defensas del programa, lo que a su vez impide la desofuscación mediante análisis dinámico.				Test Case	Test Case	
8.10	MSTG-RESILIENCE-10	La aplicación implementa un "enlace al dispositivo" utilizando una huella del dispositivo derivado de varias propiedades únicas al mismo.				Test Case	Test Case	
8.11	MSTG-RESILIENCE-11	Todos los archivos ejecutables y bibliotecas correspondientes a la aplicación se encuentran cifrados, o bien los segmentos importantes de código se encuentran cifrados o "empaquetados" (packed). De este modo cualquier análisis estático trivial no revelará código o datos importantes.				N/A	Test Case	
8.12	MSTG-RESILIENCE-12	Si el objetivo de la ofuscación es proteger código propietario, debe utilizarse un esquema de ofuscación apropiado para la tarea particular y robusto contra métodos de desofuscación manual y automatizada, considerando la investigación actual publicada. La eficacia del esquema de ofuscación debe verificarse mediante pruebas manuales. Nótese que, siempre que sea posible, las						
8.13	MSTG-RESILIENCE-13	A modo de defensa en profundidad, además de incluir un refuerzo (hardening) sólido de la comunicación, puede implementarse el cifrado de datos (payloads) a nivel de aplicación como medida adicional contra ataques de eavesdropping.						

Figura 16 – Lista de verificación sección – Resilience [19]

2.3.4 OWASP Mobile Top 10

OWASP Mobile Top 10 es una lista de los diez riesgos de seguridad más comunes y críticos en las aplicaciones móviles, en 2015, OWASP realizó una encuesta y se recolectaron datos a nivel mundial, esto ayudó a analizar y recategorizar el Mobile Top 10 presentado inicialmente en 2014 para el año 2016, al a fecha (diciembre 2023) OWASP anunció en marzo de 2023 en su sitio web el inicio del trabajo para actualizar la lista en 2023. También hay un enlace a Slack, para que todos puedan contribuir a la investigación que se está realizando actualmente, disponible en el sitio web: `https://owasp.org/www-project-mobile-top-10/`.

Para este proyecto describiremos la lista publicada en el año 2016. A continuación, se presenta la Figura 17 con la lista de los 10 principales riesgos en aplicaciones móviles.

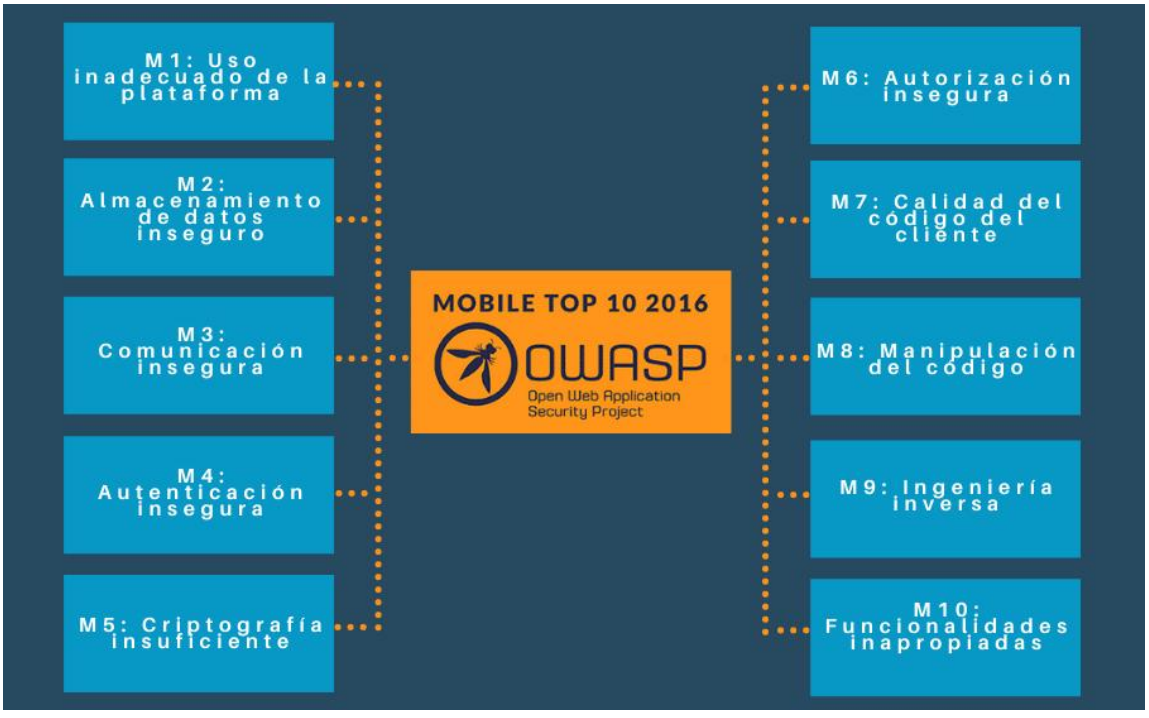


Figura 17 – OWASP Top 10 de los riesgos de seguridad móvil 2016

1. **M1: Uso inadecuado de la plataforma**

   Explotación: Fácil

   Impacto: Severo

   El uso inadecuado de las funciones de la plataforma o falta de uso de los controles de seguridad. Puede incluir *intents* de Android, permisos de plataforma, uso indebido de *TouchID*, *keychain* o algún otro control de seguridad que sea parte del sistema operativo móvil.

   Existen diversas maneras en que las aplicaciones pueden experimentar este riesgo, algunas de ellas son:

   - Violación de pautas de seguridad publicadas por las plataformas Android y iOS.
   - Violación de las prácticas recomendadas, en algunos casos no todas las mejores prácticas de seguridad están documentadas en la guía del fabricante.
   - Uso incorrecto no intencionado, por ejemplo, debido al uso incorrecto de la *API*.

2. **M2: Almacenamiento de datos inseguro**

   Explotación: Fácil

   Impacto: Severo

   Las vulnerabilidades de almacenamiento de datos inseguro se presentan cuando el equipo de desarrollo asume que los usuarios o aplicaciones maliciosas no tendrán acceso al sistema de archivos de un dispositivo y a la información almacenada.

   La explotación podría darse si un usuario malintencionado obtiene un dispositivo móvil perdido o robado y con ayuda de herramientas extrae y lee los archivos del dispositivo que contengan información confidencial.

3. **M3: Comunicación insegura**

   Explotación: Fácil

   Impacto: Severo

   Las aplicaciones móviles con frecuencia se comunican con un servidor remoto, comúnmente se utiliza la arquitectura cliente-servidor. Durante la comunicación entre la aplicación y el servidor, los datos deben atravesar la red del operador del dispositivo móvil e Internet y a menudo no se implementa un canal de comunicación cifrado entre la aplicación y el servidor, esto conlleva el riesgo de exponer los datos del usuario como: credenciales de acceso y datos personales a que puedan ser interceptados por un atacante.

   Los agentes de amenazas pueden aprovechar las vulnerabilidades para interceptar datos confidenciales mientras viajan. Existen los siguientes agentes de amenaza:

   - Un usuario malintencionado que comparte su red local (*Wi-Fi* comprometido o monitoreado).
   - Operador o dispositivos de red (enrutadores, torres de telefonía celular, proxy, etc.).
   - *Malware* en el dispositivo móvil.

4. **M:4 Autenticación insegura**

   Explotación: Fácil

   Impacto: Severo

   Los esquemas de autenticación deficientes o inexistentes permiten que un atacante ejecute funciones de manera anónima dentro de la aplicación móvil o en el servidor con el que se comunica la aplicación.

   Una vez que el atacante comprende cómo el esquema de autenticación es vulnerable, falsifica o evade la autenticación enviando solicitudes al servidor y evitando cualquier interacción directa con la aplicación móvil. Este proceso de generalmente se realiza a través de *malware* en aplicaciones móviles.

Los agentes de amenazas que explotan las vulnerabilidades de autenticación suelen hacerlo a través de ataques automatizados que utilizan herramientas disponibles o personalizadas.

5. **M:5 Criptografía insuficiente**

   Explotación: Fácil

   Impacto: Severo

   Esta vulnerabilidad se presenta cuando se usan algoritmos de cifrado débiles o no se implementa correctamente el proceso de cifrado. Para explotar esta debilidad un atacante debería obtener los datos originales antes de cifrar.

   Este tipo de vulnerabilidad puede tener varios impactos comerciales como lo siguiente:

   - Violaciones de privacidad
   - Robo de información
   - Robo de propiedad intelectual
   - Daño a la reputación

   Los agentes de amenazas incluyen los siguientes: un usuario malintencionado con acceso físico a datos que se hayan cifrado incorrectamente o *malware* en aplicaciones móviles.

6. **M:6 Autorización insegura**

   Explotación: Fácil

   Impacto: Severo

   Los esquemas de autorización deficientes o inexistentes permiten que un atacante ejecute una funcionalidad que no está autorizado debido a los bajos privilegios con los que cuenta originalmente. Los esquemas de autorización son más vulnerables cuando se realiza el proceso de autorización dentro del dispositivo móvil en lugar de hacerlo a través de un servidor remoto.

Los agentes de amenazas incluyen los siguientes: un usuario malintencionado con acceso al dispositivo o *malware* en aplicaciones móviles.

7. **M:7 Calidad del código del cliente**

   Explotación: Difícil

   Impacto: Moderado

   Por lo general, es difícil detectar este tipo de problemas mediante la revisión manual del código. En su lugar, los atacantes utilizarán herramientas de terceros que realizan análisis estáticos. Este tipo de herramientas normalmente identificarán *memory leaks*, desbordamientos de búfer y otros problemas no tan graves que resultan en una mala práctica de programación. Los atacantes con conocimientos y experiencia de lenguajes de bajo nivel pueden explotar eficazmente este tipo de deficiencias. El objetivo principal típico es ejecutar código externo dentro del espacio de direcciones de la aplicación móvil.

8. **M:8 Manipulación del código**

   Explotación: Fácil

   Impacto: Severo

   Existe toda una industria de seguridad construida para detectar y eliminar versiones no autorizadas de aplicaciones móviles dentro de las tiendas de aplicaciones. Esta categoría cubre las modificaciones en el binario del código fuente, la modificación de recursos locales, alterar los métodos y la modificación de memoria dinámica.

   El vector de ataque consiste en alguna de las siguientes acciones para explotar esta categoría:

   - Modificaciones en el binario principal del paquete de la aplicación.
   - Modificaciones en los recursos dentro del paquete de la aplicación.
   - Redirigir o reemplazar las API del sistema para interceptar y ejecutar código malicioso.

9. **M9: Ingeniería inversa**

   Explotación: Fácil

   Impacto: Severo

   Generalmente, la mayoría de las aplicaciones móviles son susceptibles a ingeniería inversa. Algunas aplicaciones son más susceptibles que otras. El código escrito en lenguajes o *frameworks* que permiten la introspección dinámica en tiempo de ejecución como: Java, .NET, Objective C, Swift, están particularmente en riesgo a técnicas de ingeniería inversa y manipulación. Detectar la susceptibilidad a la ingeniería inversa es sencillo. Primero, se descarga la aplicación de la tienda de aplicaciones. Luego, con ayuda de herramientas como IDA PRO, Hopper u otool se analiza el binario de la aplicación. El código será susceptible si es bastante fácil de comprender el flujo de la aplicación, la tabla de cadenas y cualquier pseudocódigo/código fuente generado por estas herramientas.

10. **M:10 Funcionalidades inapropiadas**

    Explotación: Fácil

    Impacto: Severo

    Algunas funciones, archivos o registros de la aplicación que exponen información relacionada con entornos de prueba, demostración o preparación no deben incluirse en entornos de producción. Además, las URL de las API de administración ya sean de ambientes de pruebas o producción no deben incluirse en las compilaciones de producción.

    El vector de ataque consiste en que un atacante descargue la aplicación y la examine, identificando los archivos de registro, archivos de configuración para descubrir una funcionalidad oculta que los desarrolladores dejaron olvidada.

2.3.5 Fases de una auditoría de seguridad

OWASP proporciona una guía detallada para realizar pruebas de seguridad en aplicaciones móviles, conocida como *OWASP Mobile Security Testing Guide* (MSTG), en esta se describen las etapas generales de una auditoria de aplicaciones móviles.

1. **Preparación**

   El nivel de seguridad con el que se probará la aplicación debe decidirse antes de realizar la prueba. Los requisitos de seguridad deben decidirse al inicio del proyecto ya que distintas organizaciones tienen diferentes necesidades de seguridad y recursos disponibles para invertir en actividades de prueba, por lo que se define junto con el cliente o dueño de la aplicación el alcance de las pruebas de seguridad, tipo de pruebas y un contrato, incluida la identificación de los controles de seguridad aplicables, los objetivos de prueba de la organización y los datos confidenciales. De manera más general, la preparación incluye toda la sincronización con el propietario de la aplicación o sistema a auditar, así como la protección legal del auditor.

2. **Recopilación de Información**

La recopilación de inteligencia implica la recopilación de información sobre la arquitectura de la aplicación, los casos de uso empresarial a los que sirve y el contexto en el que opera la aplicación. Dicha información puede clasificarse como "ambiental" o "arquitectónica".

La información ambiental incluye:

- Los objetivos de la organización para la aplicación. La funcionalidad da forma a la interacción de los usuarios con la aplicación y puede hacer que algunas

superficies sean más propensas que otras a ser atacadas por un actor malicioso.

- Partes interesadas e inversores; entender quién está interesado y es responsable de la aplicación.
- Procesos internos, flujos de trabajo y estructuras organizativas. Los procesos internos y flujos de trabajo específicos de la organización pueden crear oportunidades para vulnerabilidades de lógica de negocio.

La información arquitectónica incluye:

- La aplicación móvil: cómo la aplicación accede a los datos y los administra, cómo se comunica con otros recursos y administra las sesiones de los usuarios.
- El sistema operativo: los sistemas operativos y las versiones del sistema operativo en los que se ejecuta la aplicación (incluidas las restricciones de versión de Android o iOS), si se espera que la aplicación se ejecute en dispositivos que tienen controles de administración de dispositivos móviles.
- Red: uso de protocolos de transporte seguros (por ejemplo, TLS), uso de claves seguras y algoritmos criptográficos para proteger el cifrado del tráfico de red, uso de fijación de certificados para verificar el punto final, etc.
- Servicios remotos: los servicios remotos que consume la aplicación y si su compromiso pudiera comprometer al usuario.

3. Mapeo de la aplicación

El mapeo ayuda a obtener una comprensión profunda de la aplicación, sus puntos de entrada, los datos que contiene y las principales vulnerabilidades potenciales. Se basa en información de las fases anteriores; puede complementarse con un escaneo automático y una exploración manual de cada funcionalidad de la aplicación.

Cuando la auditoria de seguridad se realiza en modalidad de caja blanca o caja gris, cualquier documento del proyecto (diagramas de arquitectura, especificaciones funcionales, código, etc.) puede facilitar enormemente el proceso. Si el código fuente está disponible, el uso de herramientas SAST puede ayudar a obtener información valiosa sobre vulnerabilidades. Las herramientas DAST pueden admitir pruebas de caja negra y escanear automáticamente la aplicación: mientras que un evaluador necesitará horas o días, un escáner puede realizar la misma tarea en unos minutos. Sin embargo, es importante recordar que las herramientas automáticas tienen limitaciones y sólo encontrarán aquello para lo que han sido programadas. Por lo tanto, es necesario el análisis manual para aumentar la calidad de los resultados de las herramientas automáticas.

4. Explotación

Las limitaciones de tiempo o financieras limitan muchas auditorias de seguridad al mapeo de aplicaciones a través de escáneres automatizados. Si bien las vulnerabilidades identificadas durante la fase anterior pueden resultar interesantes, es necesario confirmar su relevancia respecto de cinco puntos:

- Daño potencial: el daño que puede resultar de explotar la vulnerabilidad.
- Reproducibilidad: facilidad para reproducir el ataque.
- Explotabilidad: Nivel de facilidad con el que puede ejecutarse el ataque.
- Usuarios afectados: el número de usuarios afectados por el ataque
- Descubribilidad: Nivel de facilidad con el que puede identificarse la vulnerabilidad.

Esta etapa es necesaria para determinar si las vulnerabilidades son reales y verdaderas positivas. Es posible que algunas vulnerabilidades no se puedan explotar y, si las hay, pueden dar lugar a compromisos menores. Otras vulnerabilidades pueden parecer inofensivas a primera vista, pero se consideran muy altas debido a que incurren a cumplimiento de normas de seguridad.

5. Fase de resultados

En esta fase, que es esencial para el propietario de la aplicación, el auditor de seguridad informa las vulnerabilidades. Esto incluye el proceso de explotación a detalle, clasificación de las vulnerabilidades, el impacto si un atacante logra comprometer el objetivo y se describe a qué datos ha podido acceder el auditor de manera ilegítima. Además, se lista una serie de recomendaciones para mitigar las vulnerabilidades identificadas.

2.4 Marco legal

Ante el incremento del uso de Internet para diversas actividades cotidianas, la protección de datos personales en México y en el mundo ha ganado una gran importancia. Muchas empresas recopilan constantemente datos de sus usuarios a través de aplicaciones móviles o web, con la finalidad de procesarlos y mostrar información y contenido relevante.

Las autoridades de todo del mundo han reconocido la importancia de la protección a la privacidad de los usuarios de Internet. Uno de los principales ejemplos, es el Reglamento General de Protección de Datos (GDPR) en Europa. México no se ha quedado atrás, no sólo reconoce la necesidad de proteger los datos personales, sino que reconoce el derecho a la privacidad como un derecho constitucional.

2.4.1 La Ley de Protección de Datos en México

México cuenta con una ley que regula el tratamiento de los datos personales por parte de empresas del sector privado desde el 5 de julio de 2010, esa ley se llama *Ley Federal de Protección de Datos Personales en Posesión de los Particulares* por sus siglas (LFPDPPP) o Ley de Protección de Datos [20].

Su aplicación determina que se evite que los datos personales sean utilizados indebidamente, que se respeten los derechos de los propietarios y que se garantice una expectativa razonable de privacidad. Quienes traten datos personales deben tomar en cuenta las guías y documentos emitidos por el *Instituto Nacional de Transparencia, Acceso a la Información* y *Protección de Datos Personales* (INAI).

Una de ellas, es la Guía para implementar un Sistema de Gestión de Seguridad de Datos Personales.

En el mundo jurídico, los datos personales son aquellos relacionados a la persona natural e identificable. O sea: toda información que permite identificar a un individuo como nombre y apellidos, domicilio, teléfono, historial laboral y académico, sus datos patrimoniales y financieros, su firma, así como sus características físicas, incluyendo datos biométricos como el iris o la huella dactilar, todos ellos se consideran información personal.

La Ley de Protección de Datos permite el tratamiento de datos personales por organizaciones y empresas en determinadas situaciones, pero siempre teniendo como objetivo preservar al usuario. Sin embargo, para eso, es necesario que exista consentimiento o legítimo interés en el uso de los datos en cuestión.

Los datos personales también pueden ser considerados sensibles cuando dan a conocer información íntima de la persona o aquellos que revelan el origen racial, étnico, estado de salud, ideología, opiniones políticas, orientación sexual, y otros.

En México, el tratamiento de este tipo de datos recibe un mayor control a través de la Ley de Protección de Datos, ya que su uso indebido puede originar discriminación o, en otros casos, riesgo para el dueño de los datos. Por eso, es indispensable que las empresas evalúen la necesidad de utilizar esos datos.

2.4.2 La Ley de Protección de Datos Personales aplicada

Están obligadas todas las empresas del sector privado que obtengan, usen, almacenen o transfieran datos personales como parte de sus actividades. Sumadas, estas acciones reciben el nombre de "tratamiento de datos personales".

De acuerdo con la Ley, los principales actores en el tratamiento de los datos personales son el "responsable" y el "encargado". Donde el responsable: es la persona del sector privado que toma las decisiones sobre el tratamiento de datos personales. Y el encargado: es la persona ajena al responsable, que trata los datos personales por cuenta del responsable y de acuerdo con sus instrucciones.

Para aplicar la ley que protege a los datos de los usuarios, son establecidos algunos principios por la Ley de Protección de Datos:

a) **Licitud:** deberá cumplirse la Ley de Protección de Datos, su Reglamento y cualquier otra regulación aplicable al tratar los datos.
b) **Consentimiento:** siempre que necesario deberá obtenerse el consentimiento de los titulares para el tratamiento de sus datos personales.
c) **Información:** debe ser de conocimiento del titular la información relacionada con el tratamiento de sus datos.
d) **Calidad:** todos los datos personales recolectados deben ser exactos, completos, pertinentes, correctos y actualizados de acuerdo con sus finalidades.
e) **Finalidad:** el tratamiento de datos debe cumplir las finalidades establecidas en el aviso de privacidad fornecido al usuario.

f) **Lealtad:** proteger a los intereses del titular y la expectativa razonable de privacidad al tratar los datos personales.
g) **Proporcionalidad:** sólo los datos personales necesarios, adecuados y relevantes en relación con las finalidades podrán tratarse;
h) **Responsabilidad:** la empresa debe responder por el tratamiento de los datos personales recolectados.

2.4.3 Obligaciones de las empresas

Las obligaciones y responsabilidades por las empresas que tratan datos personales. Entre ellas, están las principales: brindar información suficiente a los titulares de los datos sobre el tratamiento de sus datos y obtener su consentimiento.

Ofrecer un aviso de privacidad en tu sitio web o aplicación

A través del aviso de privacidad, se resuelve una de las obligaciones impuestas por la Ley de Protección de Datos, que es informar a los usuarios la existencia y las características principales del tratamiento a que se someterán sus datos personales. Así que, el aviso de privacidad deberá informar:

1. La identidad del responsable
2. Qué datos se recolectan
3. Cuáles son las finalidades del tratamiento
4. Los derechos de los que goza el titular
5. Si se comparten sus datos con otras entidades

Tener el consentimiento del titular

Todo tratamiento de datos personales está sujeto al consentimiento tácito o expreso del titular de los datos, salvo las excepciones establecidas por la Ley de Protección de Datos.

Según el texto vigente en la Ley Federal de Protección de Datos Personales en Posesión de los Particulares, el consentimiento será expreso cuando la voluntad se manifieste verbalmente, por escrito, por medios electrónicos, ópticos o por cualquier otra tecnología, o por signos inequívocos. Ya el consentimiento tácito es cuando el titular consiente tácitamente el tratamiento de sus datos y habiéndose puesto a su disposición el aviso de privacidad, no manifieste su oposición.

Permitir el acceso, rectificación, cancelación y oposición del titular

Una obligación muy importante de los responsables es permitir que los titulares ejerzan sus derechos de acceso, rectificación, cancelación y oposición (denominados "Derechos **ARCO**").

2.4.4 Sanciones

La ley de protección de datos establece que las siguientes conductas llevadas a cabo por el responsable constituyen infracciones:

- No cumplir con la solicitud del titular para el acceso, rectificación, cancelación u oposición al tratamiento de sus datos personales.
- Actuar con negligencia o dolo en la transmisión y respuesta de solicitudes de acceso, rectificación, cancelación u oposición de datos personales.
- Declarar dolosamente la inexistencia de datos personales, cuando exista total o parcialmente en las bases de datos del responsable.

- Omitir en el aviso de privacidad, los datos que se recolectan, la identidad del responsable, derechos que goza el titular, si se compartes con otras organizaciones.
- Incumplir con del deber de confidencialidad: en cualquier fase del tratamiento de datos personales el responsable o terceros deberán guardar confidencialidad respecto de éstos.
- Transferir datos a terceros sin comunicar en el aviso de privacidad.
- Vulnerar la seguridad de base de datos, locales, programas o equipos cuando resulte imputable al responsable.
- Recabar o transferir datos personales sin el consentimiento expreso del titular.
- Tratar los datos personales de manera que se afecte o impida el ejercicio de los derechos de acceso, rectificación, cancelación y oposición.

Las multas por el incumplimiento de las acciones mencionadas anteriormente van desde 100 a 320,000 días de salario mínimo vigente en el distrito federal.

2.5 Recursos y configuraciones para la auditoria

Para llevar a cabo la auditoría de seguridad de acuerdo con los objetivos del proyecto se describen las herramientas empleadas, las cuales son reconocidas ampliamente por el sector de seguridad de la información. Existen actualmente multitud de ellas que permiten desde inspeccionar solicitudes y respuestas de la comunicación entre la aplicación y el servidor, descompilar las aplicaciones, examinar el comportamiento en tiempo de ejecución, incluso existen máquinas virtuales con un conjunto herramientas especialmente diseñadas para realizar auditorías de aplicaciones móviles.

Se describen las herramientas utilizadas para el análisis de seguridad de aplicaciones de este proyecto, con el objetivo de tener una completa visión de las posibilidades y funcionalidades que ofrece cada una de ellas, además, se listan los dispositivos utilizados para ejecutar la auditoría de seguridad y las configuraciones, tanto en los dispositivos Android como en iOS.

2.5.1 Hardware

Para llevar a cabo una auditoría de seguridad, el auditor deberá disponer de un dispositivo Android con acceso *root* y un dispositivo iOS con *jailbreak*, esto permite modificar el sistema operativo, es decir se obtiene el control del sistema operativo y permite evadir restricciones que nos permite ejecutar las pruebas de seguridad de una manera adecuada, para más información consultar el Anexo A - Jailbreak y Acceso root.

En el caso de Android es posible usar un emulador es decir un dispositivo virtual que se ejecuta en el ordenador que imita el hardware y software del dispositivo móvil. Las opciones más comunes para emular dispositivos Android son:

- Emulador Android (incorporado en Android Studio)
- Genymotion

Para emular dispositivos iOS existe la plataforma *corellium*, es el único emulador de iOS disponible enfocado a investigadores de seguridad e incorpora herramientas como Frida, *jailbreak*, monitor de red y más. Es una solución SaaS empresarial, cabe mencionar que no ofrece periodos de prueba gratuita.

Para ejecutar la auditoría durante este proyecto se usaron los siguientes dispositivos físicos y cables de conexión:

- Xiaomi Mi Note 10 M1910F4G con Android 9
    - Cable USB a USB-C
- iPhone 8 con iOS 14.2
    - Cable lightning a USB
- MacBook Pro-2017
    - Adaptador USB-C a USB

2.5.2 Software

Para llevar a cabo una auditoría de seguridad, el auditor deberá disponer de un conjunto de herramientas para analizar las aplicaciones, es posible descargarlas desde el repositorio de código del desarrollador o página de la herramienta, además, existen máquinas virtuales que ya incluyen un listado de herramientas preinstaladas para auditorias de aplicaciones móviles.

Adicionalmente, se pueden encontrar un listado de herramientas más extenso en: `https://mas.owasp.org/MASTG/tools/`

Ambientes para pruebas

Durante la investigación se identificaron entornos de análisis seguridad móvil (máquinas virtuales) para Android e iOS indistintamente con algunas herramientas preinstaladas, dichos entornos se listan a continuación, posteriormente se describe cada entorno de manera breve.

- Mobexler
- Androl4b
- Mobile Security ToolChain
- Santoku

Mobexler

Es una máquina virtual diseñada para pruebas de seguridad a aplicaciones móviles Android e iOS con un amplio conjunto de herramientas para estas plataformas. Mobexler está diseñado para ser una herramienta de código abierto para ayudar a los profesionales de seguridad a realizar pruebas de aplicaciones móviles.

Sitio Web: `https://mobexler.com/`

Última actualización a la fecha septiembre 2024: 12 de agosto de 2024

Androl4b

Es una máquina virtual basada en Ubuntu Mate que incluye una recopilación de herramientas de seguridad e investigación para ingeniería inversa y el análisis de *malware*. Esta máquina está enfocada exclusivamente para Android.

Repositorio: `https://github.com/sh4hin/Androl4b`
Última actualización a la fecha septiembre 2024: 27 de julio de 2020

Mobile Security ToolChain

Es un conjunto de herramientas para pruebas de seguridad en móvil Android e iOS, se basa en la sección de herramientas de la guía de pruebas de seguridad del proyecto OWASP Mobile. Este paquete de herramientas está disponible para Linux y Mac OS X 10.13.X.

Repositorio: `https://github.com/xebia/mobilehacktools`
Última actualización a la fecha septiembre 2024: 26 de mayo de 2020

Santoku

La palabra santoku se traduce como "tres virtudes" o "tres usos". Santoku Linux Es una máquina virtual basada en Ubuntu que contiene herramientas para las siguientes tareas:

- Análisis forense móvil
- Análisis de *Malware* Móvil
- Seguridad de aplicaciones móviles

Sitio Web: `https://santoku-linux.com/`
Última actualización a la fecha julio 2022: 15 de noviembre de 2014

Mobile Security Framework (MobSF)

Es un *framework* multiplataforma para el análisis de seguridad de aplicaciones móviles capaz de realizar análisis estático y dinámico. MobSF admite binarios APK de Android, IPA de iOS, entre otros.

- Análisis estático

  Desde su interfaz web permite subir el binario de la aplicación a analizar y obtener la información del archivo como tamaño, nombre del paquete, componentes como: *activities, services, content provider* entre otros detalles.

- Análisis dinámico

  Este análisis puede ejecutarse en un dispositivo físico o en un emulador según las necesidades de cada caso. Después de realizar la configuración, la herramienta permite interactuar con la aplicación en tiempo de ejecución mientras recopila toda la información de las llamadas a las *API* que hace la aplicación y las comunicaciones generadas.

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

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

Frida

Es un kit de herramientas de instrumentación dinámica para desarrolladores e investigadores de seguridad. Permite inyectar fragmentos de JavaScript en aplicaciones nativas en Windows, MacOS, GNU/ Linux, iOS y Android. Con solo unas pocas líneas del lenguaje de programación C, puede cargar fragmentos de código JavaScript en tiempo de ejecución, lo que le permite manipular funciones, enumerar bibliotecas cargadas, sus funciones importadas y exportadas, leer y escribir en memoria, escanear la memoria en busca de patrones, etc. Frida dispone de los siguientes modos de operación:

1. **Inyectado**

Esta funcionalidad es proporcionada por *frida-core*, que actúa como una capa que empaqueta *GumJS* en una biblioteca compartida que inyecta en el proceso de las aplicaciones y proporciona un canal de comunicación bidireccional para la comunicación con los scripts. Además de esta funcionalidad principal, *frida-core* también le permite enumerar aplicaciones instaladas, procesos en ejecución y dispositivos conectados. Ese componente es esencialmente un demonio que expone *frida-core* sobre TCP, escuchando en *localhost*:27042 por defecto.

2. **Embebido**

A veces no es posible usar Frida en modo inyectado, por ejemplo, en sistemas iOS y Android sin *jailbreak* y *root* respectivamente. Para tales casos, existe *frida-gadget*, una biblioteca compartida que se debe incluir dentro del paquete de la aplicación. Cargando la biblioteca permitirá interactuar con ella de forma remota, usando herramientas existentes basadas en Frida como frida-trace.

3. **Precargado**

Se configura para ejecutarse de forma autónoma cargando un script desde el sistema de archivos.

Sitio web: `https://frida.re/`

Runtime Mobile Security (RMS)

Es una herramienta basada en Frida que cuenta con una interfaz web para manipular y analizar las aplicaciones Android e iOS en tiempo de ejecución.

Permite visualizar todas las clases cargadas y métodos, manipular la aplicación, rastrear argumentos en los métodos, modificar su valor y cargar scripts, entre otras tareas.

Repositorio: `https://github.com/m0bilesecurity/RMS-Runtime-Mobile-Security`

Objection

Es un kit de herramientas de exploración en tiempo de ejecución para aplicaciones móviles Android e iOS, impulsado por Frida para evaluar la seguridad de las aplicaciones sin necesidad de contar con dispositivos con jailbreak en el caso de iOS.

Repositorio: `https://github.com/sensepost/objection`

ADB

Android Debug Bridge por sus siglas (ADB) es una herramienta de línea de comandos que permite comunicarse con dispositivos Android. El comando *adb* permite realizar una variedad de acciones en el dispositivo, como instalar, desinstalar, copiar archivos y depurar aplicaciones. Es un programa cliente-servidor que incluye tres componentes:

- El cliente, envía comandos y se ejecuta en la máquina del usuario.
- Un *daemon* (adbd), ejecuta comandos en el dispositivo. El *daemon* se ejecuta como un proceso en segundo plano en cada dispositivo móvil.
- El Servidor administra la comunicación entre el cliente y el *daemon*. El servidor se ejecuta en la máquina del usuario como un proceso en segundo plano.

Sitio Web: https://developer.android.com/studio/command-line/adb?hl=es-419

Android backup extractor

Herramienta para extraer y desempaquetar el archivo *backup.ab* generado por adb backup. Basada en BackupManagerService.java del proyecto Android Open Source Project por sus siglas en inglés (AOSP).

Repositorio: https://github.com/nelenkov/android-backup-extractor

PIDCat

Es un script escrito en lenguaje Python que ayuda filtrar información de depuración de una aplicación Android. Se basa en la herramienta *Logcat* que se incluye en el IDE de desarrollo Android Studio, muestra los mensajes del sistema, por ejemplo, cuando se produce la recolección de elementos no utilizados, y los mensajes de depuración que se agregan con la clase Log.

Repositorio: https://github.com/JakeWharton/pidcat

Otool

Es una herramienta de línea de comandos disponible en macOS, incluida en el entorno de desarrollo xcode y utilizada para analizar binarios y bibliotecas compartidas. Sirve para inspeccionar diversos aspectos de estos archivos, como sus encabezados, tablas de símbolos, dependencias de bibliotecas dinámicas, y más, soporta principalmente archivos binarios y bibliotecas compartidas en el formato Mach-O, que es el formato de archivo ejecutable utilizado en sistemas operativos macOS e iOS.

JADX-GUI

Es una herramienta escrita en el lenguaje Java bajo la licencia Apache 2.0 para descompilar aplicaciones Android a código Java legible, entre sus características

incluye búsqueda de funciones a lo largo de la aplicación, búsqueda de la declaración de variables y uso e instancias de clases.
Repositorio: `https://github.com/skylot/jadx`

Burp Suite

Burp suite es un conjunto avanzado de herramientas desarrollado por la empresa PortSwigger que permite realizar pruebas de seguridad de aplicaciones web. Cuenta con una versión gratuita (Burp Suite Community) y una versión de pago (Burp Suite Profesional).

Entre las funciones que incluye la herramienta se encuentra el servidor proxy que permite inspeccionar y modificar las solicitudes y respuestas entre la aplicación y el servidor, el escáner de vulnerabilidades que automatiza la detección de fallos de seguridad en la aplicación y muchas otras funciones más.
Sitio Web: `https://portswigger.net/burp`

Drozer

Es una aplicación de código abierto, publicado bajo una licencia BSD y mantenido por MWR InfoSecurity

Drozer permite identificar vulnerabilidades en aplicaciones Android asumiendo el rol de una aplicación maliciosa, interactuando con la máquina virtual Dalvik, los mecanismos IPC de otras aplicaciones y el sistema operativo subyacente.

También permite explotar de forma remota los dispositivos Android, mediante la creación de archivos maliciosos o páginas web para explotar vulnerabilidades conocidas. Drozer puede maximizar los permisos disponibles instalando un agente completo, inyectándolo en un proceso en ejecución o conectando una shell inversa para que actúe como una herramienta de acceso remoto.
Sitio Web: `https://labs.f-secure.com/tools/drozer/`

CyberChef

Es una herramienta basada en tecnologías web que permite analizar y decodificar datos fácilmente, sin la necesidad de utilizar múltiples herramientas ni utilizar diferentes lenguajes de programación. Permite realizar más de 100 operaciones, por ejemplo, es capaz de codificar y decodificar una cadena en Base64. También nos arroja las direcciones IP de un determinado rango, como lo haría una calculadora de direcciones IP. Además, incorpora funciones hash, soporta los algoritmos MD2, MD4, MD5, MD6, SHA1, SHA2, SHA3 entre otros.

Sitio Web: `https://gchq.github.io/CyberChef/`

2.5.3 Configuración del entorno Mobexler

En esta sección se describe el paso a paso de la configuración para la máquina virtual Mobexler, se sugiere disponer de un equipo de cómputo con al menos 60 GB de espacio libre en disco y 8GB de RAM.

1. Descargar la máquina virtual Mobexler con las herramientas para la auditoria de seguridad en: `https://mobexler.com/`
2. Instalar software de virtualización como VirtualBox o VMWare, para este proyecto se usará VMWare.
3. Importar el archivo con extensión *.ova* en el software de virtualización elegido.
4. Iniciar la máquina virtual Mobexler e iniciar sesión con la contraseña: “mobexler”.

5. Conectar dispositivo físico iOS/Android a la máquina virtual Mobexler

    - Usando VMWare Player/Workstation/Fusion

    Conectar el cable USB al equipo de cómputo → Haga clic en Máquina Virtual → Dispositivo Extraíble → Dispositivo iOS/Android, y conecte el Dispositivo a la Máquina Virtual Mobexler haciendo clic en “Conectar a la máquina virtual”.

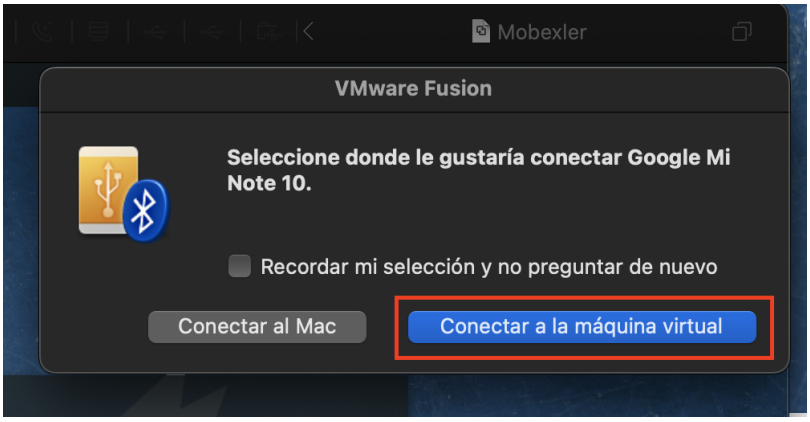


Figura 18 – Opciones de conexión VMWare Fusion

Android

Una vez se ha conectado el dispositivo Android, abra una terminal → Introduzca el comando *adb devices*, y debería mostrar su dispositivo Android conectado, como se muestra en la Figura 19.

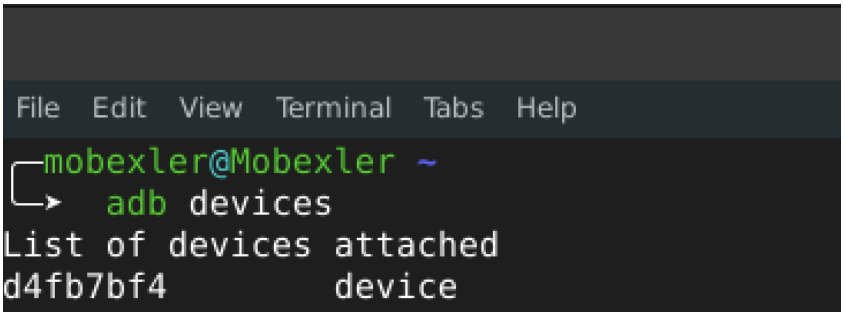


Figura 19 – Prueba de conexión dispositivo Android

iOS

Una vez se ha conectado el dispositivo iOS, abra el terminal → Introduzca el comando *ideviceinfo*, que debería mostrar la información de su dispositivo iOS conectado, como se muestra en la Figura 20.

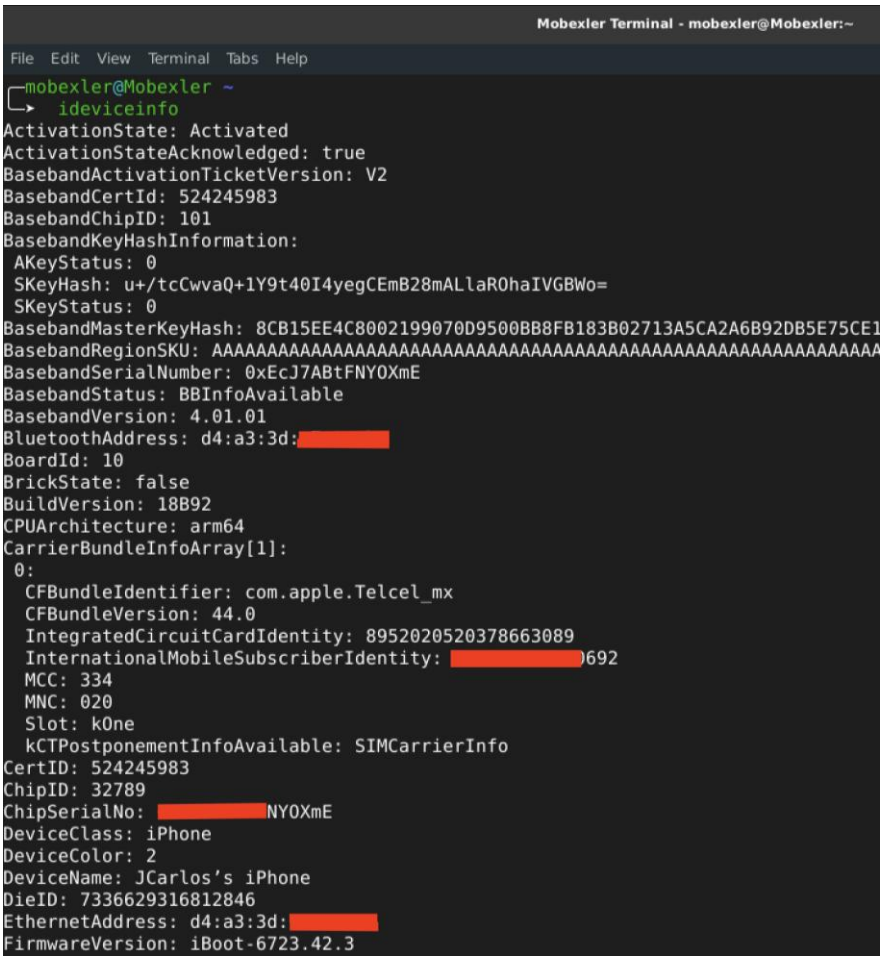


Figura 20 – Prueba de conexión dispositivo iOS

2.5.4 Configuración del dispositivo Android

En esta sección se describe la configuración e instalación de las herramientas para analizar una aplicación Android, es recomendable contar con un dispositivo Android con acceso *root*, esto le brinda control total sobre el sistema operativo y permite evadir restricciones como el aislamiento de aplicaciones.

Rootear dispositivo Android

Para *rootear* un dispositivo móvil, primero se debe desbloquear el gestor de arranque. El procedimiento de desbloqueo depende del fabricante del dispositivo. Algunos dispositivos móviles son más fáciles de *rootear* que otros, particularmente cuando se trata de pruebas de seguridad los dispositivos creados por Google y fabricados por compañías como Samsung, LG o Motorola se encuentran entre los más populares, porque son utilizados por muchos desarrolladores. En el siguiente enlace se encuentra una lista de guías para *rootear* todos los dispositivos de las principales marcas: `https://www.xda-developers.com/root/`.

Magisk es una forma de *rootear* un dispositivo Android. Su especialidad radica en la forma en que se realizan las modificaciones en el sistema. Mientras que otras herramientas alteran los datos reales en la partición del sistema, Magisk no lo hace por lo que se denomina "*systemless*". Esto permite ocultar las modificaciones de las aplicaciones sensibles, por ejemplo, para aplicaciones de banca móvil o los juegos, y permite usar las actualizaciones oficiales de Android sin necesidad de *desrootear* el dispositivo es decir eliminar las modificaciones hechas en el sistema operativo. Se pueden encontrar las instrucciones para instalar en: `https://topjohnwu.github.io/Magisk/.`

Instalación del servidor de Frida

Para interactuar con la aplicación que desea analizar, se requiere instalar el servidor de Frida en el dispositivo, para hacerlo ocuparemos la aplicación *FridaLoader* una aplicación Android para descargar y lanzar la última versión del servidor Frida, creada por el mismo autor que la aplicación InsecureBankv2, disponible en el siguiente enlace:

`https://github.com/dineshshetty/FridaLoader/releases/download/v4/FridaLoader.apk`

Descargar el archivo APK → Conectar el dispositivo a través de USB al equipo de cómputo → Instalar la aplicación con el comando `adb install FridaLoader.apk`

Abrir el menú de aplicaciones y localizar la aplicación `FridaLoader,` abrirla y dar clic en el botón Install & Run Latest Frida Server y con esto se instalará la última versión del servidor de Frida.

Instalación del explorador de Archivos Android

La aplicación root explorer es una aplicación para Android que permite a los usuarios root acceder al sistema de archivos del dispositivo.

La aplicación se puede descargar del siguiente enlace: `https://apkpure.net/es/root-explorer/com.speedsoftware.rootexplorer/download`

Descargar el archivo APK → Conectar el dispositivo a través de USB al equipo de cómputo → Instalar la aplicación con el comando `adb install Root Explorer.apk`

Instalación Drozer

1. Instalar Drozer usando el siguiente comando: *`pip install drozer`*
2. Verificar la Instalación: Después de la instalación, verifica que Drozer esté correctamente instalado ejecutando:
   `drozer console connect`

Instalar el Agente de drozer en el dispositivo Android

1. Instalar el Agente Drozer: En el dispositivo Android, descargar e instala el agente Drozer, disponible en `https://github.com/WithSecureLabs/drozer-agent/releases/download/3.0.0/drozer-agent.apk`
   `adb install /path/to/drozer-agent.apk`
2. Iniciar el Agente Drozer: Abre la aplicación Drozer Agent en su dispositivo Android. Haz clic en Start Server para iniciar el servidor del agente.
3. Reenviar el Puerto ADB: Configura el reenvío de puertos para comunicarte con el agente Drozer desde la computadora:
   adb forward tcp:31415 tcp:31415
4. Abrir una terminal en la máquina virtual y ejecutar el siguiente comando:
   drozer console connect
   Si la conexión es exitosa, se debería visualizar un mensaje de bienvenida de Drozer.

Instalación de la aplicación InsecureBankv2

1. Para instalar la aplicación que se analizará, primero descargar el repositorio del desarrollador “InsecureBankv2” de la siguiente URL: `https://github.com/dineshshetty/Android-InsecureBankv2/tree/master/AndroLabServer`
2. Después de descargar el repositorio, ingresar a la carpeta */Android-InsecureBankv2/AndroLabServer*
   Instalar las dependencias con el comando *pip install -r requirements.txt*
   Iniciar el servidor con el comando *python app.py*
3. Instalar el archivo .apk en el dispositivo Android.
   Localizar el archivo que se ubica en la carpeta: */Android-InsecureBankv2/InsecureBankv2.apk*
   Instalar la aplicación con el comando: *adb install* `InsecureBankv2.apk`

4. Po último, en el dispositivo Android, abrir la aplicación seleccionar el menú de la esquina superior derecha, seleccionar *Preferences* e ingresar la dirección IP del servidor a la escucha y el puerto 8888 ya que este puerto es por defecto en servidor, sin embargo, este se puede modificar de acuerdo con la disponibilidad de su entorno.

Configuración de Burp Suite en el dispositivo Android

Configurar un dispositivo Android para que funcione con Burp Suite es un especto fundamental durante la auditoria de aplicaciones móviles. Esto le permite inspecciones las solicitudes y respuestas entre una aplicación y el servidor.

Paso 1: Configurar la escucha del proxy de Burp

Para que la herramienta Burp Suite pueda interceptar el tráfico HTTP generado por el dispositivo Android, debe configurar un *proxy listener* y vincularlo a un puerto abierto.

1. En Burp, vaya a la pestaña Proxy → Proxy Settings.
2. En la pestaña *proxy listeners*, haga clic en Añadir.
3. En la nueva pestaña de configuración, establezca un puerto disponible comúnmente suele utilizarse el puerto 8080, pero puede elegir otro.
4. En la opción Bind to address seleccionar All interfaces y haga clic en Aceptar.

Burp ya está listo para recibir tráfico HTTP en el puerto asignado y reenviarlo al servidor web de destino.

Paso 2: Configure el dispositivo para que envíe el tráfico a través de Burp

Si está utilizando un emulador de Android, puede que tenga que añadir los detalles del proxy desde el menú de ajustes del emulador en lugar de los ajustes nativos de red o Wi-Fi en el dispositivo virtual.

1. En el dispositivo Android, ir a los ajustes de red e Internet.
2. Abra los detalles de la red Wi-Fi que desea utilizar para las pruebas y acceder al modo de edición de la red.
3. En los ajustes, elija la opción para configurar un proxy manualmente.
4. Establezca el nombre de host del proxy en la dirección IP de la máquina que está utilizando para ejecutar Burp.
5. Establezca el puerto Proxy en el puerto que asignó al nuevo oyente del proxy que configuró en Burp.
6. Guardar los cambios y conectarse a la red Wi-Fi. El tráfico web del dispositivo ahora será enviado a través de Burp Suite.

Paso 3: Añadir el certificado CA de Burp al almacén de confianza del dispositivo.

Cada instalación de Burp tiene su propia autoridad de certificación (CA) integrada. Para trabajar con tráfico HTTPS en Burp, debe agregar el certificado CA asociado al almacén de confianza de su dispositivo. Esto permite a Burp hacerse pasar por el servidor web de destino durante el proceso de negociación del protocolo TLS. Esto permitirá inspeccionar el tráfico HTTPS al igual que lo haría con el tráfico HTTP no cifrado.

1. En Burp Suite, abra el cuadro de diálogo configuración y vaya a herramientas → Proxy.
1. En el cuadro de diálogo certificado CA, seleccione exportar → Certificado en formato DER y haga clic en Siguiente e introducir un nombre de archivo y elegir la ubicación para el certificado. Tener en cuenta que debe incluir explícitamente la extensión de archivo .der.
2. Moverse al directorio donde se guardó el archivo ,der y convertir el certificado exportado de formato DER a formato PEM con los siguientes comandos:

   ```
   cd ~/Directorio
   openssl x509 -inform DER -in cacert.der -out cacert.pem
   ```

3. Obtener el *Subject hash old* del certificado en formato PEM.
   El Subject Hash Old de un certificado es un valor hash que se utiliza para identificar de manera única el "sujeto" (subject) del certificado.
   `openssl x509 -inform PEM -subject_hash_old -in cacert.pem | head -1`
4. Cambiar el nombre del certificado al valor del *Subject hash old* concatenado con ".0", quedando de la siguiente manera `[subject-hash-old].0`
   `mv cacert.pem [subject-hash-old].0`
5. Transferir el certificado en el dispositivo con el comando
   `adb push [subject-hash-old].0 /sdcard`
6. Abrir una shell interactiva en el dispositivo Android y cambia al usuario (root).
   `adb shell`
   `su`
7. Montar el sistema de archivos raíz (/) con permisos de lectura y escritura
   `mount -o rw,remount /`
8. mueve el archivo desde el almacenamiento interno (/sdcard) al directorio de certificados del sistema.
   `mv /sdcard/[subject-hash-old].0 /system/etc/security/cacerts/`
9. Cambiar los permisos del archivo de certificado para que sea legible por todos los usuarios, pero solo editable por el propietario root.
   `chmod 644 /system/etc/security/cacerts/[subject-hash-old].0`
10. Cambia el grupo propietario del archivo de certificado al grupo root.
    `chgrp root /system/etc/security/cacerts/[subject-hash-old].0`
11. El siguiente comando vuelve a montar el sistema de archivos raíz (/) con permisos de solo lectura, es decir restaura los permisos de solo lectura al sistema de archivos raíz para proteger el sistema contra modificaciones accidentales o maliciosas.
    `mount -o ro,remount /`

Paso 4: Pruebe la configuración

Para asegurarse de que ha completado la configuración con éxito y está listo para empezar a probar:

1. Verificar que el dispositivo Android y el equipo de cómputo donde se ejecuta Burp puedan comunicarse entre sí a nivel de red, se puede realizar con el comando `ping -c 4 <ip-dispositivo-Android>.`
2. En Burp, vaya a la pestaña Proxy > Interceptar.
3. Utilice el botón para activar la función Interceptar.
4. En su dispositivo Android, abra el navegador.
5. Navegue a cualquier sitio que utilice HTTPS. Si ha completado la configuración correctamente, la página debería cargarse sin advertencias de seguridad y debería ver su tráfico en Burp en la pestaña Proxy > Historial HTTP.

2.5.5 Configuración del dispositivo iOS

En esta sección se describe la configuración e instalación de las herramientas para analizar una aplicación iOS.

Jailbreak en dispositivo iOS

Es necesario contar con un dispositivo con *jailbreak*, para saber si la versión de iOS del dispositivo cuenta con una herramienta para conseguirlo consultar los siguientes recursos `https://docs.google.com/spreadsheets/d/1QjWyoDfaiF-TWhzVdvEMRqA3OQXsz6e8of3SxZB1W_M/edit` o `https://canijailbreak.com/`.
Una vez identificada la versión de iOS y la herramienta para conseguirlo, seguir los pasos que apliquen según la herramienta disponible.

Cydia es gestor de paquetes para dispositivos iOS que permite descargar e instalar aplicaciones que no están disponibles en la App Store y complementos no oficiales del sistema operativo iOS. Esta solo se encuentra disponible para dispositivos con jailbreak.

Instalación de complementos *AppSync, SSH* y *Frida*
Debido a las restricciones que implementa Apple que son, entre una de ellas, la firma de código; es decir todas las aplicaciones deben estar firmadas con un certificado emitido por Apple. Esto significa que la aplicación se puede instalar solo después de una verificación exitosa de la firma de código. Sin embargo, en un teléfono con *jailbreak*, se puede evadir esta función de seguridad con el complemento *AppSync* disponible en el repositorio `https://repo.hackyouiphone.org/`. Esta herramienta es posible instalarla a través de cydia.

A diferencia de Android en iOS no se cuenta con una interfaz de administración por defecto como lo es ADB, por ello es necesario instalar la herramienta de administración remota SSH a través de Cydia, ingresando a Cydia → Ingresar a la sección de búsqueda y escribir SSH → Seleccionar OpenSSH y presionar instalar. Después de concluir la instalación, podrá acceder al dispositivo con el usuario root con el comando `ssh root@<ip-iPhone>` con la contraseña alpine. Como medida de seguridad se recomienda cambiar la contraseña por defecto del servicio SSH.

Instalar la herramienta de instrumentación dinámica Frida, ingresando a Cydia → Ingresar a la sección de búsqueda → Sources → Seleccionar Edit y presionar Add → ingresar la URL en el campo de texto `https://build.frida.re` → regresar a la sección de búsqueda y escribir “Frida” seleccionar el paquete y dar clic en el botón instalar.

SSL Kill Switch 2 es una herramienta de código abierto diseñada para desactivar la validación de certificados SSL en aplicaciones iOS. Esto permite interceptar y analizar el tráfico de red de aplicaciones que normalmente usan SSL/TLS para proteger la comunicación. Para instalar esta herramienta seguir los siguientes pasos. Abrir Cydia y seleccionar la opción Fuentes seleccionar el botón Editar → Añadir y agrega la siguiente URL: `https://julioverne.github.io`, Ingresar a la sección de búsqueda y escribir “SSL Kill Switch 2” → Seleccionar este elemento y presionar instalar.
Por último, ir ajustes del teléfono → identificar SSL Kill Switch 2 y activarlo desde el botón tipo switch.

Instalación de la aplicación DVIA-v2
Para instalar la aplicación DVIA-v2, ingresar a Cydia y seleccionar el icono de Fuentes, presionar el botón Editar y seleccionar “Añadir” e ingresar la siguiente URL: `http://apt.thebigboss.org/repofiles/cydia/` presionar el botón añadir fuente,

seleccionar el botón “buscar” ubicado en la esquina inferior derecha e ingresar el texto: Filza, seleccionar el elemento Filza File Manager y presionar Instalar.

Posteriormente descargar el archivo .ipa de la aplicación en el dispositivo iOS desde la siguiente URL: `https://github.com/prateek147/DVIA-v2/releases/download/v2.0/DVIA-v2-swift.ipa`

Por último, dirigirse a la aplicación Filza y buscar el archivo .ipa en la ruta */var/mobile/Documents*, seleccionar el archivo de la aplicación y presionar “Instalar”, esto se puede observar en la Figura 21.

Figura 21 – Instalar aplicación DVIA-v2-swift .ipa

Configuración de Burp Suite en el dispositivo iOS

Configurar un dispositivo iOS para que funcione con Burp Suite es un especto fundamental durante la auditoria de aplicaciones móviles. Esto le permite inspecciones las solicitudes y respuestas entre una aplicación y el servidor.

Paso 1: Configurar la escucha del proxy de Burp

Para que la herramienta Burp Suite pueda interceptar el tráfico HTTP generado por el dispositivo iOS, debe configurar un *proxy listener* y vincularlo a un puerto abierto.

1. En Burp, vaya a la pestaña Proxy → Proxy Settings.
2. En la pestaña proxy listeners, haga clic en Añadir.
3. En la nueva pestaña de configuración, establezca un puerto disponible comúnmente suele utilizarse el puerto 8080, pero puede elegir otro.
4. En la opción Bind to address seleccionar All interfaces y haga clic en Aceptar.

Burp ya está listo para recibir tráfico HTTP en el puerto asignado y reenviarlo al servidor web de destino.

Paso 2: Configure su dispositivo para utilizar el proxy

Para que la herramienta Burp Suite pueda interceptar el tráfico HTTP generado por el dispositivo iOS, debe configurar un proxy listener y vincularlo a un puerto abierto.

1. En el dispositivo iOS, vaya a Ajustes > Wi-Fi.
2. Asegurarse de que el botón Wi-Fi está encendido y se encuentra conectado a su red Wi-Fi.
3. Seleccione el icono de información situado junto a su red Wi-Fi y desplazarse hasta la sección Proxy HTTP.
4. Elegir la opción Configurar proxy y dar clic en “Manual” e ingresar la dirección IP de la máquina virtual que ejecuta burp y el puerto elegido en el paso 1.

5. Por último, dar clic en guardar. El tráfico web del dispositivo ahora será enviado a través de Burp Suite.

Cada instalación de Burp tiene su propia autoridad de certificación (CA) integrada. Para trabajar con tráfico HTTPS en Burp, debe agregar el certificado CA asociado al almacén de confianza de su dispositivo. Esto permite a Burp hacerse pasar por el servidor web de destino durante el proceso de negociación del protocolo TLS. Esto permitirá inspeccionar el tráfico HTTPS en Burp igual que lo haría con el tráfico HTTP no cifrado.

Paso 3: Añadir el certificado CA de Burp en su dispositivo iOS

Para poder interactuar con el tráfico HTTPS, necesita instalar un certificado CA de su instalación de Burp Suite en su dispositivo iOS.

Para instalar el certificado CA en su dispositivo iOS

1. Asegúrese de que Burp Suite se está ejecutando en su ordenador.
2. Abra el navegador de su dispositivo iOS y escriba http://burp en la barra de búsqueda, y seleccione la esquina superior derecha “CA Certificate”.
3. Cuando seleccione esta opción se mostrará un cuadro de dialogo que indica que se está intentando descargar un perfil de configuración, seleccione la opción permitir.
4. En la pantalla Instalar perfil, seleccione Instalar.
5. Vaya a Configuración → General → Perfil → Seleccionar el perfil descargado llamado “PortSwigger CA” → dar clic en Instalar por último el dispositivo solicitará ingresar el código de bloqueo.

Paso 4: Probar la configuración

Para asegurarse de que ha completado la configuración con éxito y está listo para empezar a probar:

1. Verificar que el dispositivo iOS y el equipo de cómputo donde se ejecuta Burp puedan comunicarse entre sí a nivel de red, se puede realizar con el comando `ping -c 4 <ip-dispositivo-iOS>.`
2. En Burp, vaya a la pestaña Proxy → Interceptar.
3. Utilice el botón para activar la función Interceptar.
4. En su dispositivo iOS, abra el navegador web.
5. Navegue a cualquier sitio que utilice HTTPS. Si ha completado la configuración correctamente, la página debería cargarse sin advertencias de seguridad y debería ver su tráfico en Burp en la pestaña Proxy → Historial HTTP.

3 TRABAJO RELACIONADO

Se han realizado numerosas investigaciones sobre auditoria de aplicaciones móviles en distintos países, este apartado se focaliza fundamentalmente en los estudios realizados en España, los cuales mantienen una estrecha relación con la presente investigación debido a los objetivos que comparten y metodologías en las que estos se basan.

Rodriguez Sánchez en [21] aborda la importancia en la seguridad de aplicaciones móviles debido a las tendencias de crecimiento de descargas en los últimos años a nivel mundial que auguran un crecimiento exponencial de seguridad y ataques con *malware* diseñados específicamente para los dispositivos móviles de las plataformas Android e iOS. La autora busca proporcionar una guía para el proceso de la auditoría de seguridad de las aplicaciones móviles aplicada al ejemplo de una aplicación real en la plataforma Android. Detalla los modelos de seguridad para aplicaciones móviles y los estándares de verificación de seguridad más reconocidos y aceptados. Presenta las distintas herramientas disponibles actualmente para la realización de auditorías de seguridad de aplicaciones móviles y proporciona las bases del marco legal al que deben ceñirse los desarrolladores en relación con la protección de datos personales, financieros, de salud, entre otros.

Rodríguez Sánchez, tomó como referencia el proyecto OWASP Mobile**,** que está formado por *Mobile Security Testing Guide* (MSTG), una guía técnica para llevar a cabo la evaluación de seguridad y por *Mobile Application Security Verification Standard* (MASVS), la cual define los niveles estrictos de verificación de seguridad con los que debe cumplir una aplicación en función del tipo de datos que esta maneje.

El marco legal bajo el que la autora realizó la investigación es el *Reglamento General de Protección de Datos* también conocido a nivel europeo con las siglas en inglés (GDPR), la cual es una normativa de la *Unión Europea* (UE) para proteger la

privacidad de sus ciudadanos, así como para armonizar la legislación de los países miembros de la UE.

La metodología de análisis usada por la autora fue: Análisis estático y análisis dinámico, que idealmente se complementan con el objetivo de cubrir la mayor área posible de ataques sobre la aplicación.

El Análisis estático también conocido como *Static Application Security Testing* por sus siglas en inglés (SAST), implica la revisión de seguridad a partir del código fuente de la aplicación, ya sea proporcionado por el equipo de desarrollo u obtenido a partir de la descompilación del código de la aplicación mediante ingeniería inversa y herramientas de *descompilación*. Análisis dinámico también conocido como *Dynamic Application Security Testing* por sus siglas en inglés (DAST) es la evaluación de la aplicación en tiempo de ejecución.

La aplicación seleccionada por la autora para llevar a cabo la auditoría de seguridad fue una aplicación de gestión de notarías que facilita una serie de servicios telemáticos para gestiones habituales en el ámbito notarial como escrituras, expedientes, pólizas, legitimaciones, facturación, tramitación, tesorería, contabilidad, informes, etc. Por lo tanto, la aplicación se evaluó con el nivel MASVS-L1 correspondiente a la criticidad de la aplicación de estudio ya que esta no maneja datos altamente sensibles.
Los resultados obtenidos a partir del análisis de vulnerabilidades arrojaron diversas deficiencias en la aplicación analizada, algunas de ellas fueron:

- Atributo *android:allowBackup* establecido con el valor *true* lo que permite al sistema operativo realizar respaldos de los archivos y bases de datos de la aplicación a través de la interfaz *adb*.

- Falta de cabeceras como: *X-Frame-Options, Cache-Control, X-XSS-Protection* y *X-Content-Type-Options* en servidor que soporta a la aplicación.
- Falta de protección en elementos de la aplicación como *Activities* y *Services*, lo que permite a otras aplicaciones en el dispositivo interactuar con estos elementos y desencadenar una potencial fuga de información.

Como conclusiones de la investigación la autora destaca el trabajo de la comunidad del proyecto de OWASP Mobile lo considera una buena referencia como guía para llevar a cabo las pruebas de seguridad en aplicaciones móviles, además se resalta la importancia de las herramientas disponibles que a pesar de que existen muchas opciones, algunas de estas se encuentran desactualizadas y debido a los cambios en las tecnologías de desarrollo y compatibilidad con las aplicaciones algunas de estas son obsoletas.

Héctor Pauta Martillo en [22] describe la problemática en el incremento del uso de los dispositivos móviles y sus aplicaciones, además a la adopción por parte de las empresas con la política *Bring Your Own Device* por sus siglas en inglés (BYOD) que consiste en que los empleados lleven sus dispositivos personales para acceder a redes corporativas, ese aumento viene acompañado de una proliferación del *malware* móvil, además se resalta el incremento de ciberataques contra dispositivos móviles basados en aplicaciones maliciosas.

El autor busca proporcionar una guía para ejecutar una auditoría de seguridad sobre aplicaciones en dispositivos iOS y Android, estudiar y conocer las distintas amenazas que puedan afectar a una aplicación para ayudar a prevenirlas, protegerse frente a ellas y conocer las herramientas disponibles para la ejecución de auditorías de seguridad en estas plataformas. El autor se basó en la guía de la publicación especial “NIST 800-163 Vetting the Security of Mobile Applications” documento donde se

detallan los procesos a través de los cuales las organizaciones evalúan las aplicaciones móviles para detectar vulnerabilidades. Además, tomo como referencia el proyecto OWASP Mobile.

Durante el desarrollo de su proyecto de auditoría de seguridad se llevaron a cabo dos casos prácticos, uno para la plataforma Android y otro para iOS. En iOS se analizó la aplicación *Damn Vulnerable iOS App* (DVIA) y en Android la aplicación *Damn Insecure and Vulnerable App* (DIVA) son aplicaciones con diferentes vulnerabilidades y su objetivo principal es proporcionar una plataforma a los entusiastas/profesionales de la seguridad móvil o estudiantes para probar sus habilidades de pruebas de penetración en un entorno destinado para ello.

Para evaluar la severidad de las vulnerabilidades el autor usó el sistema de puntuación de vulnerabilidad CVSS que consta de tres grupos métricos: base, temporal y ambiental, estas tres métricas en conjunto producen una puntuación que va de 0 a 10. Dos usos comunes de CVSS son calcular la severidad de las vulnerabilidades descubiertas en los sistemas y como un factor en la priorización de las actividades de su mitigación.

El marco legal bajo el que el autor realizó la investigación es en el Reglamento General de Protección de Datos establecido por la Unión Europea en mayo de 2018.

Las herramientas usadas por el autor para llevar a cabo la auditoría fueron: un smartphone Motorola Moto E4 con Android 7.1.1, ordenador con Windows 10, Máquina Virtual con la distribución Kali Linux 2020. Las vulnerabilidades identificadas en la aplicación Android fueron:

- Información sensible en el código fuente
- Modo de depuración habilitado
- Código fuente no ofuscado

- Métodos vulnerables a SQLI
- Registro de información sensible en logs

El nivel de seguridad Global de la aplicación DIVA determinado por el autor es bajo, ya que se detectó una vulnerabilidad de nivel alto relacionada con la exposición de claves privadas que permitirían un acceso no autorizado a la plataforma y vulnerabilidades de nivel medio descubiertas, que están relacionadas con métodos vulnerables a inyección SQL y la exposición de información sensible. Por otro lado, las vulnerabilidades detectadas en la aplicación de la plataforma iOS fueron:

- Configuración insegura de ATS
- Información sensible en base de datos local
- Información sensible en archivo .plist
- Información sensible en los logs
- *Screenshot* habilitados
- Evasión de mecanismo *jailbreak*.

El nivel de seguridad Global de la aplicación DVIA determinado por el autor es medio ya que no se detectaron vulnerabilidades de nivel alto y las vulnerabilidades de nivel medio descubiertas están relacionadas con la exposición de información sensible y la evasión de algunos mecanismos de seguridad implementados.

Los resultados obtenidos por el autor demuestran la necesidad de mejorar los controles de seguridad a los que se someten periódicamente las aplicaciones móviles. Por esta razón, la auditoría de seguridad se ha convertido en una tarea imprescindible para garantizar la confidencialidad, integridad y disponibilidad de la información gestionada por las aplicaciones internas y comerciales, su oportuna ejecución evitaría que las aplicaciones se distribuyan con defectos o vulnerabilidades que pueden llevar a la filtración de datos. Aunque el 100% de la seguridad no existe o no se puede garantizar, se debe optar por los medios necesarios para reducir la

superficie de ataque de una aplicación móvil y para ello una auditoría exhaustiva de seguridad se considera esencial durante parte del ciclo de vida del desarrollo de dicha aplicación.

David González Morte en su investigación [23] plantea la problemática en el incremento del uso de los dispositivos móviles para las tareas diarias, además recopila información sobre las estadísticas en el incremento de *malware* para estos dispositivos y hace énfasis en los riesgos y describe las posibles causas que llevan a los usuarios a ser víctimas del robo de información.

El autor afronta el estudio de las vulnerabilidades de los dispositivos, los problemas que pueden tener y como afectan estos a los usuarios, además de ver cómo reciben esta información y cómo se puede mejorar esta comunicación, para que estos sean consecuentes con las acciones que realizan.

Para el desarrollo del proyecto el autor no tomó una metodología base en concreto, realizó una búsqueda de información en Internet, biblioteca y recursos universitarios y se llevó a cabo la auditoría mediante técnicas desarrolladas por el autor de la investigación.

Las herramientas usadas por el autor fueron: un entorno virtual de la distribución Santoku basada en Linux especialmente desarrollada para auditar aplicaciones móviles, un entorno virtual *Android Virtual Device* por sus siglas en inglés (AVD) es una herramienta incluida en el entorno de desarrollo Android Studio, también se usó un dispositivo Samsung S3 mini. El marco legal bajo el que se realizó la investigación es en el Reglamento General de Protección de Datos establecido por la Unión Europea en mayo de 2018.

Durante el desarrollo del proyecto de auditoría de seguridad el autor realizó un caso práctico con una aplicación de la plataforma Android llamada theAndroidSeek,

aplicación desarrollada como trabajo de fin de carrera por el mismo autor del proyecto. TheAndroidSeek es una herramienta de geolocalización desarrollada en 2013, que tiene como objetivo dar soporte a excursionistas. La aplicación permite en momentos de urgencia enviar un mensaje a un contacto con su posición geográfica con tal que sea localizable en momentos que el excursionista se ha perdido.

La auditoría da como resultado las siguientes deficiencias identificadas en la aplicación:

- Código y funciones sin actualizar que pueden provocar incompatibilidades con algunas versiones de Android y algunos dispositivos
- Falta de bitácoras de registro del envío de SMS con coordenadas del usuario
- Solicitud de permisos innecesarios
- Código en desuso que no es relevante para el funcionamiento de la aplicación, Información embebida en el código fuente de la aplicación
- No cumple con el Reglamento General de Protección de Datos por sus siglas en inglés (RGPD) y no se informa al usuario del uso de sus datos privados

El autor concluye que, a pesar de disponer de excesiva información en Internet sobre seguridad en dispositivos móviles y de instrucciones sobre buenas prácticas en el uso de estos, este tipo de información no llega correctamente a los usuarios, ya que, su lenguaje específico dificulta tanto la comprensión como la atención. Además, el cambio constante en las nuevas tecnologías y el resto de las organizaciones deben adaptarse e incluso adelantarse a estos hechos, es decir, tanto educación, como cultura, leyes y normativas deben entender el estado actual y a la sociedad en el entorno de las nuevas tecnologías, para así, poder llegar correctamente a los usuarios.

El estado del arte actual de la auditoria o pentesting de aplicaciones móviles es complejo y en constante evolución. Las aplicaciones y sus plataformas están en constante cambio, lo que hace que sea más difícil para los auditores identificar y explotar vulnerabilidades, por otro lado, debido a la creciente complejidad de las aplicaciones, hace que sea más difícil para los desarrolladores corregir las vulnerabilidades y mantener las aplicaciones actualizadas.

Hay una serie de tendencias que están contribuyendo a mejorar el estado del arte de la auditoria de aplicaciones móviles. Una de estas tendencias es la aceptación como estándar el proyecto OWASP Mobile por diversas organizaciones del sector público, privado, aportando recursos para mejorar la calidad del contenido del proyecto.

Además, el equipo de desarrollo de Android ha adoptado como estándar de evaluación el MASVS del proyecto de OWASP Mobile a través del programa *App Defense Alliance* por sus siglas en inglés (ADA) este se enfoca en garantizar la seguridad de las aplicaciones subidas a la play store. El programa consiste en que Google registra y autoriza a un conjunto de laboratorios o empresas especializadas en autoría de aplicaciones móviles para que los desarrolladores que deseen se pongan en contacto con alguno de los laboratorios autorizados para realizar el proceso de auditoría de seguridad, si la aplicación cumple con los requerimientos de seguridad, se agrega una insignia en la play store, para conocer más detalles sobre este proyecto consultar su sitio web: `https://appdefensealliance.dev`

El aumento de la colaboración entre las comunidades de ciberseguridad: Las personas que forman parte de estas comunidades están trabajando para compartir información y conocimientos sobre las vulnerabilidades y herramientas. Esta colaboración está ayudando a mejorar la capacidad de los auditores para identificar y explotar vulnerabilidades.

Por otro lado, un problema común al que se enfrentan los auditores es la incompatibilidad de las herramientas y la falta de soporte de estas, a la fecha (septiembre 2023) existen muchas herramientas que no son compatibles con la última versión de iOS 17 y Android 14 lo que hace más difícil llevar a cabo algunas pruebas de seguridad por lo que algunas revisiones se deben realizar de forma manual.

En general, se puede decir que la seguridad móvil está mejorando, pero aún hay mucho trabajo por hacer. Las empresas, desarrolladores y los usuarios deben seguir trabajando juntos para mejorar la seguridad de las aplicaciones móviles y proteger los datos de los usuarios.

4 MÉTODO PROPUESTO

El método propuesto para realizar auditorías de aplicaciones móviles se basa en el manual de pruebas técnicas MSTG, del proyecto OWASP Mobile, consiste en una serie de pasos simplificados, en la siguiente figura se presenta el método propuesto partiendo del paso uno hasta el paso diez, de manera general, posteriormente se describen los procedimientos técnicos en cada punto para realizar el análisis de las aplicaciones Android/iOS.

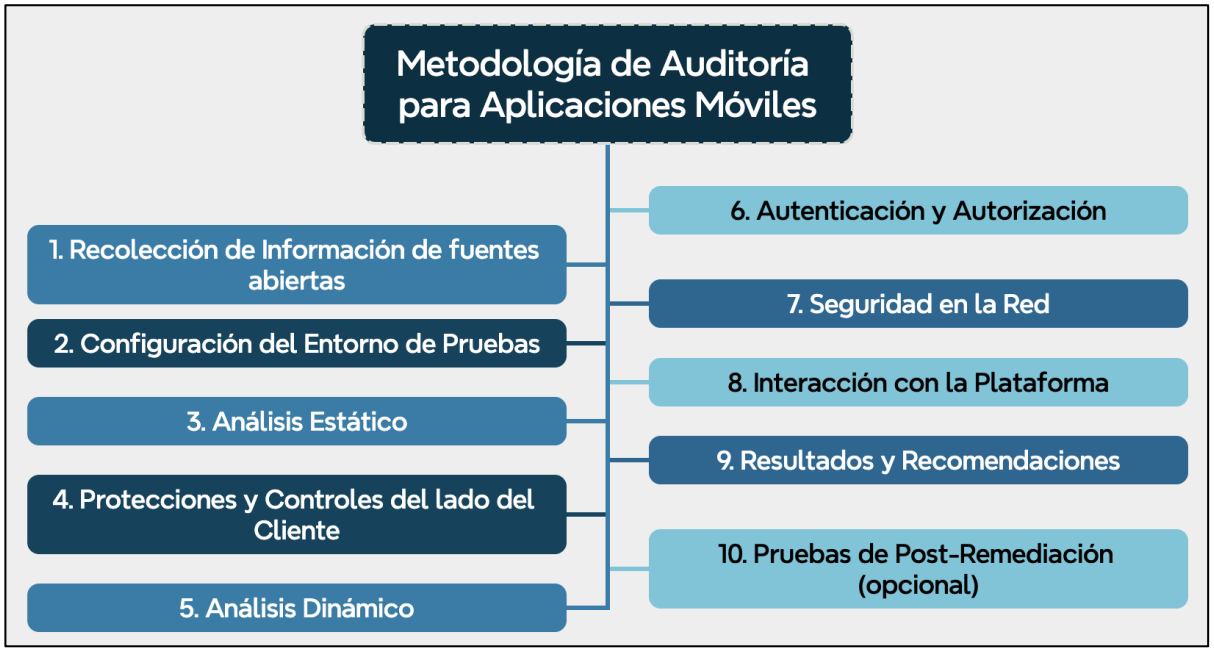


Figura 22 – Metodología propuesta para auditar aplicaciones móviles

1. Recolección de información y preparación

El objetivo de este primer paso es recopilar información preliminar y entender el funcionamiento de la aplicación, identificar los sistemas operativos compatibles: (iOS, Android) y en algunos casos identificar información expuesta como secretos o código fuente en plataformas como GitHub.

Primero, revisar la documentación de la página oficial, notas de lanzamiento, manuales de usuario sobre la aplicación, descripción en la PlayStore/App Store, esto ayudara a entender el funcionamiento de la aplicación.

Github Dorks

Es una técnica utilizada para encontrar información sensible o secretos en repositorios de GitHub mediante búsquedas específicas. Consiste en utilizar consultas avanzadas, llamadas "dorks", en el motor de búsqueda de GitHub (`https://github.com/search`) para localizar datos expuestos.

En la siguiente tabla se listan algunos dorks para identificar información en repositorios expuestos.

Dork	Descripción
`filename:.npmrc _auth`	Datos de autenticación del registro npm
`filename:.dockercfg auth`	Datos de autenticación del registro de docker
`extension:pem private`	Claves privadas
`extension:ppk private`	Claves privadas puttygen
`filename:id_rsa or filename:id_dsa`	Claves privadas ssh
`filename:.ftpconfig`	Creado por remote-ssh para Atom, contiene detalles y credenciales del servidor SFTP/SSH
`filename:credentials aws_access_key_id`	Archivos de acceso de AWS

`filename:wp-config.php`	Archivos de configuración de wordpress
`filename:.htpasswd`	Archivos htpasswd
`filename:.env DB_USERNAME NOT homestead`	Archivo de configuración de laravel .env

Google Dorks

Es una técnica de búsqueda avanzada que utiliza operadores especiales y filtros en Google para encontrar información específica. Estos operadores permiten a los usuarios filtrar resultados para descubrir datos sensibles, archivos confidenciales, vulnerabilidades de seguridad y otros contenidos expuestos accidentalmente en sitios web.

Dork	Descripción
`site:github.com "nombre-organización"`	Búsqueda básica de repositorios de una organización
`site:example.com ext:xml | ext:conf | ext:cnf | ext:reg | ext:inf | ext:rdp | ext:cfg | ext:txt | ext:ora | ext:ini`	Permite buscar archivos de configuración y otros tipos de archivos
`site:example.com ext:sql | ext:dbf | ext:mdb`	Busca archivos de base de datos expuestos
`site:example.com inurl:readme | inurl:license | inurl:install | inurl:setup | inurl:config`	Busca archivos y páginas que a menudo contienen información sobre la configuración
`site:example.com ext:bkf | ext:bkp | ext:bak | ext:old | ext:backup`	Este dork permite identificar archivos de respaldo (backups)
`inurl:apidocs | inurl:api-docs | inurl:swagger | inurl:api-explorer site:"example.com"`	Este dork permite identificar documentación de API

2. **Configuración del entorno de pruebas**

El objetivo de este paso es configurar la máquina virtual Mobexler para realizar la auditoria de seguridad en las aplicaciones móviles y configurar un entorno de prueba en emuladores o dispositivos físicos.

Para más detalles sobre la configuración consultar las secciones: Configuración del entorno Mobexler, Configuración del dispositivo Android y Configuración del dispositivo iOS.

3. Análisis estático

El objetivo de este paso es identificar posibles vulnerabilidades y malas prácticas de codificación en el código fuente de la aplicación sin ejecutarla, es decir analizando el binario de la aplicación. Es poco común disponer del código fuente de la aplicación en un entorno de producción, por lo tanto, el análisis estático se realiza normalmente a partir del binario de la aplicación, para realizar este análisis se recomienda utilizar principalmente la herramienta MobSF, que permite realizar un análisis estático automatizado a partir del binario de una aplicación iOS/Android.

El binario de la aplicación debe ser provisto por el cliente, es decir, el dueño de la aplicación o bien través de la tienda oficial de cada plataforma, Play Store/ App Store respectivamente, no se recomienda descargar la aplicación de tiendas de terceros ya que este podría ser una versión no oficial.

Android

Descargar la aplicación de la PlayStore usando ADB ejecutaremos los siguientes comandos para obtener el APK y extraer el binario del dispositivo:

Primero, listar todas las aplicaciones instaladas y filtrar por nombre de la aplicación.

adb shell pm list packages | grep <nombre_app>

El siguiente comando se utiliza para encontrar la ruta del archivo APK de una aplicación instalada en el dispositivo Android

adb shell pm path <nombre_app>

Por último, con el siguiente comando se copia el archivo APK desde el dispositivo Android al ordenador en la ruta actual de trabajo.

adb pull <path>/base.apk .

iOS

Los requisitos previos para obtener el binario de la aplicación en la plataforma iOS es contar con el dispositivo con *jailbreak*, si desea más información consultar el anexo A – Jailbreak y Acceso root que describe que es el *jailbreak* y recursos para obtenerlo, instalar el gestor de paquetes cydia u otro disponible, instalar frida en el dispositivo a través del gestor de paquetes para mayor detalle consultar la sección Configuración del dispositivo iOS.

Nota: El siguiente paso solo se deberá ejecutar si la aplicación a analizar se descarga a través de AppStore.

Descargar la aplicación a analizar desde la AppStore, después, conectar el dispositivo con *jailbreak* al ordenador, abrir una terminal en la máquina virtual y ejecutar el comando: *frida-ps -Uai | grep <nombre_app>*, para obtener el Bundle ID de la aplicación, posteriormente con ayuda de la herramienta `https://github.com/AloneMonkey/frida-ios-dump` ejecutar el comando `python3 dump.py < Bundle_ID>` con esto obtendrá el archivo de la aplicación con extensión .ipa en el directorio actual de trabajo.

Una vez se obtengan los binarios de las aplicaciones, ejecutar la herramienta MobSF, descrita en la sección Software. Una vez iniciada esta herramienta, iniciar sesión con el usuario mobsf y contraseña mobsf, elegir el binario de la aplicación y esperar a que el proceso concluya.

Cuando la herramienta haya completado el análisis automatizado del binario de la aplicación, es importante revisar manualmente los resultados, ya que pueden contener falsos positivos o falsos negativos.

Los puntos más importantes para revisar en los resultados del análisis estático son:

- Gestión de la criptografía
    - Verificación de la adecuada gestión de claves, incluyendo su almacenamiento y manejo seguro en Android (Java/Kotlin): Buscar el uso de clases criptográficas como `Cipher`, `KeyGenerator`, `KeyStore`, `SecretKey`, `PublicKey`, `PrivateKey`, `SecureRandom`. iOS (Swift/Objective-C): Busca el uso de API de criptografía como `SecKey`, `SecRandom`, `SecItemAdd`, `SecItemCopyMatching`, `SecItemUpdate`, `CCCrypt`, etc. Verifica si las claves se almacenan en el Android Keystore. El Keystore proporciona un almacenamiento seguro y evita la exportación de claves. Verifica si las claves se almacenan en el Keychain de iOS. El Keychain proporciona almacenamiento seguro y protección contra la extracción de claves.
    - Asegurarse de que los datos sensibles no se cifren con algoritmos de cifrado obsoletos o inseguros como (DES, 3DES, RSA/DSA con claves menores a 2048 bits).
    - Asegurarse de que no se usen con algoritmos de hash inseguros como (MD4, MD5 y SHA1).

- Gestión de datos sensibles
    - Identificación y protección de datos sensibles (como credenciales, información personal identificable, datos financieros) en el código y en los archivos de configuración.
    - Verificar que los datos sensibles no se almacenan en texto plano ni se incluyen en los registros de depuración.

- Control de autenticación y autorización
    - Verificación de la presencia de controles de sesión seguros.
    - Verificar que no existen puertas traseras o cuentas de usuario por defecto.

- Manejo de entradas y validación:
    - Verificar que todas las entradas del usuario son validadas y sanitizadas.
    - Asegúrate de que todas las consultas SQL usen consultas parametrizadas en lugar de concatenar directamente los valores de entrada del usuario en las consultas SQL.
    - Revisar el código en busca de métodos que ejecuten comandos del sistema o evalúen expresiones dinámicamente, como `Runtime.getRuntime().exec()`, `ProcessBuilder`, y probar introduciendo entradas maliciosas en los campos de la aplicación que podrían ejecutar comandos del sistema.
    - Buscar el uso de clases como `ObjectInputStream` y `XMLDecoder`, especialmente las llamadas a `readObject()`, que se utilizan para deserializar objetos, asegúrate de que se realicen comprobaciones de tipo y validez después de la deserialización.

- Configuraciones de seguridad y permisos:
    - Revisar los archivos de configuración para detectar configuraciones inseguras (por ejemplo, configuraciones de depuración habilitadas en producción).
    - Verificación de que la aplicación solicita solo los permisos necesarios y justificados.

- Revisión de la configuración de seguridad del entorno de ejecución, como los permisos en Android en el archivo *AndroidManifest.xm*l y las configuraciones de *Info.plist* en iOS.

- Almacenamiento y transmisión segura de datos:
  - Verificación de que los datos se transmiten de forma segura a través HTTPS/TLS para comunicaciones de red.
  - En Android analizar el archivo *AndroidManifest.xml* en Android para identificar los siguientes permisos: `READ_EXTERNAL_STORAGE`, `WRITE_EXTERNAL_STORAGE`, o `MANAGE_EXTERNAL_STORAGE`.
  - Identifica el uso de clases como `SharedPreferences`, `SQLiteDatabase` y `FileOutPutStream` especialmente las llamadas a las funciones `getExternal*`, `getWritableDatabase`, `getReadableDatabase`, `getCacheDir`, `getExternalCacheDirs`, `sharedPref.edit()`, `context.openFileOutput()`, `context.openFileInput()`, y `db.rawQuery()` el uso de estas clases y métodos son indicios de que se podría estar almacenando información sensible en texto plano en el dispositivo.
  - En iOS, no se requiere permisos explícitos. Las aplicaciones pueden leer y escribir archivos dentro de su propia *sandbox* sin requerir permisos especiales. Hay algunas clases y métodos que pueden llevar a almacenar información en texto plano como UserDefaults y FileManager, además, identificar el uso de los siguientes métodos `defaults.set()`, `fileContents.write()` y `data.value()`.

- Manejo de errores y registro:
  - Revisión del manejo de errores para asegurar que no se exponga información sensible en los mensajes de error.

- En Android (Java/Kotlin) Busca llamadas a métodos de logging como `Log.d, Log.i, Log.w, Log.e` y `System.print`.
- iOS (Swift/Objective-C) Busca el uso de funciones de logging como `NSLog, print`, etc.
- Revisión de Contenido de Logs: Asegúrate de que los logs al ejecutar la aplicación no contengan información sensible.

- Bibliotecas de terceros:
  - Revisión de las bibliotecas y dependencias de terceros para asegurar que no contienen vulnerabilidades conocidas.
  - Verificación de que las bibliotecas de terceros están actualizadas y se utilizan de manera segura.

- Protecciones contra ingeniería inversa:
  - Evaluar la presencia de técnicas para dificultar la ingeniería inversa, observa si los nombres de las clases y métodos han sido cambiados a nombres genéricos y difíciles de entender es decir si se ha aplicado ofuscación al código.

4. Protecciones y controles del lado del cliente

El propósito de este paso es identificar los controles implementados del lado del cliente, como la detección de *root/jailbreak*, detección de depuradores, detección de emuladores y la fijación de certificados SSL/TLS, estos controles suelen encontrarse en aplicaciones bancarias o que manejan datos sensibles, por los tanto dificultan las pruebas durante una auditoría de seguridad en un entorno de producción. Se debe evaluar la presencia de estos controles y en medida de lo posible, solicitar una versión de la aplicación que no los implemente. Ya que el objetivo de la auditoría es evaluar la aplicación y los servicios web asociados, no dedicar tiempo a evadir estas protecciones.

Detección de dispositivos *root/jailbreak*

Para evaluar la presencia de detección *de root/jailbreak*, consulta la sección `CODE ANALYSIS` en los resultados de la herramienta MobSF. Esta sección indica si la aplicación contiene código que detecta dispositivos con acceso *root/jailbreak*. Para confirmar esto, instala la aplicación en un dispositivo con *root/Jailbreak* e intenta ejecutarla. En algunos casos, las aplicaciones no permiten su ejecución en dispositivos alterados, pudiendo mostrar un mensaje de alerta o cerrarse sin notificación.

Detección de emuladores

El objetivo de la detección de emuladores es aumentar la dificultad para ejecutar la aplicación en un dispositivo virtual, lo que complica el uso de ciertas herramientas y técnicas durante la auditoría.

Buscar el uso de la clase `Build`, varios atributos de esta clase en Android ayudan a determinar si la aplicación está siendo ejecutada en un emulador, se presentan algunos ejemplos: `Build.ID`, `Build. MANUFACTURER`, `Build.MODEL`, `Build.PRODUCT`, `Build.RADIO`, `Build.SERIAL` y `Build.USER`. En iOS la detección se basa en características comunes de los emuladores, como el nombre del dispositivo, la versión del sistema operativo y ciertos entornos de desarrollo, a través de la clase `ProcessInfo.processInfo.environment` y la detección de la arquitectura `i386`, `x86_64`, `arm64`. Además, en la sección `APKiD ANALYSIS` del análisis realizado con la herramienta MobSF, se puede verificar si existe código destinado a detectar dispositivos virtuales. Para confirmar esto, instalar la aplicación en un dispositivo virtual y ejecutarla. En algunos casos, las aplicaciones no permiten su ejecución en dispositivos virtuales, mostrando un mensaje de alerta o cerrarse sin notificación.

Detección de depuradores

La depuración es una forma eficaz de analizar el comportamiento de una aplicación en tiempo de ejecución. Permite al auditor ejecutar el código, detener la ejecución de la aplicación en puntos arbitrarios, inspeccionar el estado de las variables, leer y modificar la memoria, y mucho más.

Las funciones *antidepuración* pueden ser preventivas o reactivas. Como su nombre indica, la *antidepuración* preventiva evita que el depurador se conecte en primer lugar; la *antidepuración* reactiva implica detectar depuradores y reaccionar ante ellos de alguna manera (por ejemplo, terminando la aplicación o activando un comportamiento oculto).

En Android existe el atributo `android:debuggable` en el archivo *AndroidManifest.xml*, si el valor de este campo se encuentra con el valor *"true"* la depuración está activada, y se permite la depuración de la aplicación, además, puede buscar el método `isDebuggerConnected` de la clase `android.os.Debug` en el código, esto sirve para determinar si un depurador está conectado.

Detectar la depuración de una aplicación iOS disponiendo solo del binario de la aplicación implica buscar patrones específicos de funciones y atributos en el código ensamblador del binario. Utilizaremos rabin2 de Radare2 para realizar este análisis.

El uso de `ptrace` para impedir que un depurador adjunte al proceso es otra técnica común. Buscamos la referencia a `ptrace` en el binario.

rabin2 -qz <BinarioApp> | grep ptrace

Para identificar si la aplicación utiliza `getppid` para determinar si fue lanzada por launchd, se analiza la referencia a `getppid` en el binario.

Comando para buscar cadenas `getppid`:

rabin2 -qz < BinarioApp > | grep getppid

Detección de fijación de certificados SSL/TLS

La fijación de certificados SSL/TLS o mejor conocido como SSLPinning es una técnica de seguridad utilizada en aplicaciones móviles para asegurar las comunicaciones entre la aplicación y el servidor. Consiste en anclar un certificado específico o una clave pública del servidor dentro de la aplicación. Esto significa que la aplicación solo aceptará conexiones HTTPS que presenten el certificado o la clave pública que coincide con el que está anclado, incluso si un certificado diferente es confiable por la cadena de certificados del sistema operativo.

Para detectar si una aplicación implementa SSLPinning, se pueden seguir diferentes enfoques a nivel de código y configuración.

Android(Java/Kotlin): Abrir el archivo apk con jadx-gui y buscar el uso de bibliotecas que soportan SSLPinning como son `OkHttp`, `Retrofit` o `HttpClient` y el uso de clases como `CertificateFactory`, `Certificate`, `TrustManagerFactory` y `SSLContext` buscar el uso de métodos como `CertificateFactory.getInstance(“X509”)`, `SSLContext.getInstance("TLS")` y `TrustManagerFactory.getDefaultAlgorithm()`.

iOS (Swift/Objective-C): Buscar el uso de bibliotecas que soportan SSLPinning, como `Alamofire`, `AFNetworking`, o configuraciones de `URLSession`, así como identificar implementaciones del método `urlSession(_:didReceive:completionHandler:)` en el delegado de `URLSession`. Para validar rápidamente si una aplicación implementa la fijación del certificado SSL/TLS, sigue estos pasos:

Configure la dirección del proxy en los ajustes de red de su dispositivo Android o iOS. Posteriormente, inicie la aplicación y observe el comportamiento. Si al intentar abrir la aplicación aparece un mensaje de error relacionado con problemas de conexión de red, entonces la aplicación está implementando la fijación del certificado.

5. Análisis dinámico

El objetivo de este punto es identificar vulnerabilidades en tiempo de ejecución. Consiste en ejecutar la aplicación en un entorno controlado y observar cómo interactúa con su entorno, qué datos maneja y cómo responde a diversas entradas.

1. Autenticación, autorización y manejo de sesiones

   El Objetivo es garantizar que solo los usuarios autorizados pueden acceder a recursos y acciones específicos. Para este punto es necesario interceptar las comunicaciones HTTP entre la aplicación y el servidor, por lo que es necesario establecer a la escucha el proxy en burp suite y configurar el proxy en los ajustes de red del dispositivo, para más detalle ingresar a la sección 2.5.4 o 2.5.5 de acuerdo con el tipo de dispositivo. Si después de configurar el proxy en el dispositivo Android/iOS y se muestra un mensaje sobre problemas en la conexión de red, quiere decir que la aplicación implementar la protección SSLPinning, tal como se describe en la sección 4. Protecciones del lado del cliente. Para evadir este control para el caso de Android existen algunos scripts de Frida, se recomienda usar frida-multiple-unpinning disponible en: `https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/` que contempla diferentes implementaciones de este control de seguridad.

1. Primero, obtener el nombre del paquete de la aplicación con el siguiente comando: `adb shell pm list packages | grep "nombre_app"`
2. Descargar el script y guardarlo en el archivo *frida_multiple_unpinning.js*, sustituir el nombre del paquete obtenido del comando anterior y ejecutar el siguiente comando: *`frida -U -f <com.example.myapp> -l frida_multiple_unpinning.js`*
3. Se iniciará la aplicación y burp podrá capturar el tráfico HTTPS, Es importarte contar con al menos un usuario y si existen diferentes roles dentro de la aplicación, con al menos un usuario con cada rol de acceso. Si la aplicación permite auto registro, registrar los usuarios para cada rol.

Para el caso de iOS, instalar la herramienta SSL Kill Switch 2 como se describe en la sección 2.5.5, después, agregar la dirección IP y puerto del proxy en las configuraciones de red y podrá interceptar e tráfico entre la aplicación y el servidor.

Control de Acceso Horizontal

Autenticarse como un usuario con privilegios limitados y tratar de acceder a recursos o realizar acciones que deberían estar restringidas a usuarios con privilegios más altos. Identificar alguna solicitud para acceder a recursos específicos: Por ejemplo, un endpoint que permite a los usuarios acceder a sus propios datos de perfil mediante una solicitud GET, como la siguiente:

```
GET /api/users/1/profile
Host: example.com
Authorization: Bearer <token_usuario1>
```

En este caso, 1 es el ID del usuario usuario1.

Ahora, intentaremos modificar la solicitud para acceder a los datos de otro usuario, por ejemplo, el usuario con ID 2. Usando Burp Suite, reenviar la solicitud anterior y modificar el endpoint de la siguiente manera:

```
GET /api/users/2/profile
Host: example.com
Authorization: Bearer <token_usuario1>
```

Si recibe una respuesta con los datos del usuario con ID 2, entonces hay un problema de Control de Acceso Horizontal en la aplicación. Si recibe un error, como `403 Forbidden` o `401 Unauthorized`, significa que se están validando correctamente los permisos.

Control de Acceso Vertical

Intentar escalar privilegios autenticándose como un usuario de bajos privilegios y accediendo a funcionalidades o datos que solo deberían estar disponibles para administradores.

Primero, iniciar sesión en la aplicación como un usuario administrador, al iniciar sesión la aplicación proporcionará un token de autenticación. Posteriormente identificar alguna solicitud que solo los administradores pueden ejecutar, por ejemplo, supongamos que hay un endpoint que es usado para acceder a la lista de todos los usuarios:

```
GET /api/admin/users
Host: example.com
Authorization: Bearer <token_adminstrador>
```

Después, iniciar sesión en la aplicación como un usuario de bajos privilegios. Por ejemplo, un usuario con el nombre de "usuario1". Posteriormente modificar la solicitud como un usuario de bajo nivel para acceder al endpoint restringido a los administradores. Usando Burp, modificar la solicitud con el token de sesión de un usuario de bajos privilegios, por ejemplo, la petición se vería de la siguiente manera:

```
GET /api/admin/users
Host: example.com
Authorization: Bearer <token_usuario>
```

Si recibe una respuesta con la lista de usuarios, entonces hay un problema de Control de Acceso Vertical en la aplicación. Si recibe un error, como `403`

`Forbidden` o `401 Unauthorized`, significa que la aplicación está validando correctamente los permisos.

Manejo de entradas y validación

Esto implica ingresar datos malformados o inesperados en la aplicación para verificar cómo maneja y valida las entradas el servidor.

Identificar un Punto de Entrada para Datos del Usuario

Supongamos que la aplicación tiene un formulario de inicio de sesión con los siguientes datos en formato JSON:

```
POST /api/login
Host: example.com
Content-Type: application/json
{
  "email": "newuser@example.com",
  "password": "Password123!"
}
```

Preparar Peticiones con Datos Malformados

Queremos probar cómo la aplicación maneja diferentes tipos de entradas malformadas o inesperadas. A continuación, se presentan algunos ejemplos:

Inyección SQL:

```
{
 "email": "newuser@example.com",
 "password": "' OR '1'='1"
}
```

Cross-Site Scripting (XSS):

```
{
  "email": "newuser@example.com",
  "password": "<script>alert('XSS')</script>"
}
```

Datos Excesivamente Largos:

```
{
  "email": "newuser@example.com",
  "password": "A".repeat(1001) //Una cadena de más de 1000 caracteres
}
```

Formato de Email Inválido:

```
{
  "email": "invalid-email",
  "password": "Password123!"
}
```

Análisis de las Respuestas

Inyección de SQL: El servidor debería rechazar la solicitud con un mensaje de error genérico, sin ejecutar la consulta SQL no deseada. Si el servidor devuelve un mensaje de error indicando un problema de sintaxis SQL, significa que la aplicación es potencialmente vulnerable a inyección SQL.

Cross-Site Scripting (XSS): El servidor debería neutralizar la secuencia <script> y rechazar la entrada o almacenarla sin efectos secundarios. Si los datos ingresados se ejecutan en el componente web, la aplicación es vulnerable a XSS.

Datos Excesivamente Largos: El servidor debería rechazar la solicitud con un mensaje de error indicando que la entrada es demasiado larga.

Formato de Email Inválido: La aplicación debería rechazar la solicitud con un mensaje de error indicando que el formato del email es inválido.
Código de Estado Esperado: El servidor debería devolver un código de estado `400 Bad Request` o un mensaje específico de validación.

Enumeración de Usuarios
El proceso consiste en usar una lista de posibles usuarios e intentar uno por uno y con base en la respuesta del sistema determinar la existencia de la cuenta debido a la diferencia entre los mensajes devueltos, cuando una cuenta existe respecto al mensaje devuelto cuando la cuenta no existe.
Identificar los puntos de entrada donde los usuarios interactúan con la autenticación, por ejemplo: Formulario de inicio de sesión, formularios de registro y funcionalidades de recuperación de contraseñas

Inicie la aplicación móvil y navegue hasta la pantalla de inicio de sesión. Introducir un nombre de usuario/email válido y capturar la solicitud con el proxy. Repetir el proceso con un nombre de usuario/email inválido y analizar lo siguiente.

- Comparar Respuestas
- Tiempos de respuesta
- Cualquier cambio en el contenido de la respuesta
- Verificar Mensajes de Error

Identificar si los mensajes de error son genéricos o revelan si el nombre de usuario/email existe:

- Mensaje correcto: "Credenciales inválidas"
- Mensaje incorrecto: "Usuario no encontrado" (revela existencia)

- Códigos de estado HTTP (como `200 OK` y `404 Not Found`)

Ejemplo en inicio de Sesión:

Capturar solicitud con usuario válido

```
POST /login
Host: example.com
Content-Type: application/json
{
    "username": "usuario_valido",
    "password": "contraseña_incorrecta"
}
```

Capturar solicitud con usuario inválido

```
POST /login
Host: example.com
Content-Type: application/json
{
    "username": "usuario_invalido ",
    "password": " contraseña_incorrecta "
}
```

Comparar las respuestas:

Respuesta usuario válido

```
{
"error": "Credenciales inválidas"
}
```

Respuesta usuario inválido:

```
{
```

```
"error": "Usuario no encontrado"
}
```

2. Almacenamiento de datos

   El objetivo de probar el almacenamiento en una aplicación móvil Android/iOS consiste en revisar cómo se maneja la información sensible y asegurar que se siguen las mejores prácticas de seguridad para evitar fugas de datos. Navegue por todas las funcionalidades disponibles, de manera la aplicación almacene datos en el dispositivo.

   Analizar el Almacenamiento del Dispositivo Android: primero, obtener el nombre del paquete de la aplicación con el siguiente comando:

   adb shell pm list packages | grep "nombre_app"

   Acceder al directorio de almacenamiento interno de la aplicación con los siguientes comandos.

   adb shell

   cd /data/data/<com.example.myapp>/

   Buscar Datos Sensibles: Inspecciona bases de datos SQLite, archivos de preferencias compartidas y archivos locales para asegurarte de que los datos sensibles están cifrados. En Android, con ayuda de la aplicación root explorer es posible navegar y analizar los archivos en los directorios *databases, shared_prefs, files*, principalmente y otros directorios relevantes del almacenamiento interno.

   Para acceder al almacenamiento externo, validar de que la aplicación hace uso de este tipo de almacenamiento. Una manera sencilla es identificar los permisos declarados en el archivo *AndroidManifest.xml* de la aplicación:

`android.permission.READ_EXTERNAL_STORAGE`
`android.permission.WRITE_EXTERNAL_STORAGE`

Conectar el dispositivo al ordenador y verifique que se encuentre en modo de depuración y navegue al directorio de almacenamiento externo con los siguientes comandos:
`adb shell`
`cd /storage/emulated/0`

Listar los directorios para identificar los datos de la aplicación, comúnmente las aplicaciones crean un directorio con su nombre de paquete en el almacenamiento externo. Por ejemplo: `com.example.myapp`

En Android, con ayuda de la aplicación root explorer analizar los archivos en el directorio del almacenamiento externo, o bien copiar los archivos al ordenador con el siguiente comando para un análisis más detallado.
`adb pull /storage/emulated/0/com.example.myapp /path/local/`

Compruebe el uso de cualquier base de datos Firebase e intente identificar si están mal configuradas ingresando a la URL: `https://_firebaseProjectName_.firebaseio.com/.json` esto, normalmente se puede identificar en la sección FIREBASE DATABASE del reporte de la herramienta MobSF.

Para la plataforma iOS, analizar los archivos almacenados en el *sandbox* de la aplicación.
Verifique que la aplicación esté instalada y en ejecución en su dispositivo con *jailbreak*.

Conectar Objection a la aplicación, para esto, es esencial obtener el identificador de proceso (PID) de la aplicación con el comando:
frida-ps -Ua

Posteriormente conectar Objection a la aplicación en ejecución con el comando *objection --gadget <NombreDelProceso> explore*
o bien usando el identificador del proceso *objection --gadget <PID> explore*

Para obtener los directorios relacionados con la aplicación es posible enumerarlos utilizando el comando *env*. Esto mostrará las ubicaciones de los directorios *Caches*, *Documents* y *Library* de la aplicación.

También es posible utilizar el comando *cd /ruta/directorio/* dentro de la herramienta objection para navegar hasta el directorio deseado. Además, se pueden listar los archivos con el comando *ls* y descargarlos utilizando el comando *download <nombre-archivo>*.

Una opción adicional es conectase al dispositivo a través de SSH con el comando:
ssh root@<ip-local>

Navegar al directorio de la aplicación:
cd /var/mobile/Containers/Data/Application/<UUID>

Crear un archivo ZIP:
zip -r app_data.zip .

Transferir el archivo ZIP al ordenador con el siguiente comando:

scp root@192.168.1.100:/var/mobile/Containers/Data/Application/<UUID>/app_data.zip /path/local/

Por último, con la herramienta unzip descomprimir el archivo y analizar el contenido:
unzip app_data.zip

Manejo de errores y registro
Revisar que los mensajes de error no revelen información sensible. Por ejemplo, nombre de usuario, contraseña, tokens de sesión o datos de pago etc.

Análisis de logs y errores en Android
Para analizar los mensajes de error en los logs de la aplicación Android seguir los siguientes pasos:

1. Instalar la aplicación en el dispositivo
2. Conectar el dispositivo a la máquina virtual y obtener el nombre del paquete de la aplicación con el siguiente comando: *adb shell pm list packages | grep "nombre_app"*
3. Ejecutar *pidcat <com.example.myapp>* Utilizar el nombre del paquete de la aplicación cuyos logs deseas visualizar.
4. Filtrar los logs por nivel de severidad (DEBUG, INFO, WARN, ERROR) utilizando las opciones de pidcat. Por ejemplo, para mostrar solo los logs de error: *pidcat -l E com.example.myapp*
5. Navegar a través de toda la aplicación y analizar la información que se muestra en los logs del dispositivo.

Análisis de logs y errores en iOS

Para analizar los mensajes de error en los logs de la aplicación seguir los siguientes pasos:

1. Iniciar Xcode
2. Conecta el dispositivo iOS al ordenador.
3. Seleccionar Ventana → Dispositivos y Simuladores.
4. Hacer clic en su dispositivo iOS conectado, este se mostrará en la sección izquierda de la ventana Dispositivos.
5. Navegue a través de la aplicación, por ejemplo, en el formulario de inicio de sesión y en secciones donde existan campos con información sensible y analizar si alguna de la información registrada se observa en la consola del dispositivo.
6. Hacer clic en el botón “Abrir consola” situado en la parte superior derecha de la ventana Dispositivos para ver los registros de la consola en una ventana independiente.

6. Autenticación y Autorización

La autenticación local se refiere cuando una aplicación autentica al usuario utilizando credenciales almacenadas en el dispositivo. Es decir, el usuario "desbloquea" la aplicación o alguna funcionalidad proporcionando un PIN, contraseña, o características biométricas como rostro o huella dactilar, que se verifican con la información guardada localmente. Sin embargo, la autenticación local siempre debe ser reforzada por un servidor remoto o basada en un método criptográfico seguro.

Probar la Autenticación Local en Android

El uso de autenticación biométrica puede detectarse analizado el archivo apk con la herramienta jadx-gui buscando la instancia de las siguientes clases `FingerprintManager`, `BiometricPrompt` y `BiometricManager` buscar el uso de

métodos como hasEnrolledFingerprints(), setAllowedAuthenticators(), canAuthenticate(), BiometricPrompt.PromptInfo.Builder(), biometricPrompt.authenticate(), onAuthenticationError(), onAuthenticationSucceeded(), onAuthenticationFailed().

Verificar de tener un dispositivo con las capacidades biométricas necesarias y configura un PIN o patrón en el dispositivo y registrar datos biométricos ya que algunas aplicaciones que hacen uso de autenticación biométrica no la permiten utilizar si no se encuentra configurada en el dispositivo.

Pruebas en tiempo de ejecución: El método onAuthenticationSucceeded se activa cuando el sistema autentica correctamente al usuario utilizando la clase BiometricPrompt. Una implementación incorrecta de la autenticación biométrica confía en que este método sea llamado sin utilizar el objeto CryptoObject. Este enfoque puede ser fácilmente explotado al manipular la aplicación y llamar directamente al método onAuthenticationSucceeded, permitiendo así que la aplicación se desbloquee sin necesidad de una autenticación biométrica válida por parte del usuario.

El script de Frida Fingerprint bypass disponible en https://github.com/WithSecureLabs/android-keystore-audit/blob/master/frida-scripts/fingerprint-bypass.js ayudará a evadir la autenticación cuando el objecto CryptoObject no se use en el método authenticate de la clase BiometricPrompt.
Es posible ejecutar el script en la aplicación con el siguiente comando:
frida -U -f com.example.myapp -l fingerprint-bypass.js

Otro caso de una incorrecta implementación de la autenticación biométrica ocurre en el manejo de Excepciones del CryptoObject.

Este método de evasión se basa en un error en la implementación de la autenticación biométrica donde se utiliza un `CryptoObject` pero no se utiliza correctamente para cifrar o descifrar los datos para que la aplicación funcione correctamente. Cuando la aplicación intenta utilizar el `CryptoObject` para operaciones criptográficas y encuentra datos que no están correctamente cifrados se provocará una excepción `javax.crypto.IllegalBlockSizeException`, por lo tanto, un script externo podría interceptar y manejar esa excepción, permitiendo así que la aplicación continúe su ejecución como si la autenticación hubiera sido exitosa, aunque no lo haya sido realmente.

El script de Frida Fingerprint bypass via exception handling disponible en `https://github.com/FSecureLABS/android-keystore-audit/blob/master/frida-scripts/fingerprint-bypass-via-exception-handling.js`
Intentará evadir la autenticación cuando se utilice `CryptoObject`, de manera incorrecta.
Se puede ejecutar el script en la aplicación utilizando el siguiente comando:
frida -U -f com.example.myapp -l fingerprint-bypass-via-exception-handling.js

Probar la Autenticación Local en iOS
El uso de autenticación biométrica puede detectarse analizando la lista de bibliotecas del binario de la aplicación. Esto se puede hacer utilizando la herramienta otool: *otool -L Payload/<AppName>.app/<AppName> | grep -i 'LocalAuthentication\.framework|Security\.framework'.*

Pruebas en tiempo de ejecución: Con ayuda de la herramienta Objection es posible probar la robustez de la implementación de autenticación en tiempo de ejecución. Objection utiliza Frida para instrumentar la función `evaluatePolicy` de modo que

retorne el valor *"true"* incluso si la autenticación no se realizó con éxito. Para probar la implementación de la autenticación biométrica seguir los siguientes pasos:

Verificar que la aplicación esté instalada y en ejecución en su dispositivo con *jailbreak*. Obtener el identificador de proceso (PID) de la aplicación con el comando:
frida-ps -Ua

Conectar Objection a la aplicación en ejecución: *objection --gadget <NombreDelProceso> explore* o bien usando el identificador del proceso:
objection --gadget <PID> explore

Interactuar con la aplicación e intentar evadir la autenticación biométrica. Navegue en la aplicación hasta llegar a la sección donde se requiere la autenticación biométrica. En ese momento, aparecerá un mensaje solicitando dicha autenticación. Para proceder, ejecutar el siguiente comando en objection:
ios ui biometrics_bypass

Si la aplicación implementa de manera insegura la autenticación permitirá acceder al área autenticada.

Fuerza bruta: Probar contra ataques de fuerza bruta. Asegurarse de que hay mecanismos para limitar los intentos de autenticación, como bloqueos temporales o permanentes tras múltiples intentos fallidos.

Borrado de datos: Verificar que la aplicación borra los datos sensibles después de un cierto número de intentos fallidos (si está configurado).

7. Seguridad en la red

El objetivo es asegurar que la comunicación de la aplicación es segura y no expone información sensible. A continuación, se describen los aspectos clave y cómo llevar a cabo estas pruebas.

Analizar los protocolos de comunicación.

Analizar los protocolos de comunicación entre la aplicación y los servidores tratando de identificar el uso de protocolos inseguros como HTTP, FTP, TELNET y SMTP. Esta tarea se puede realizar con Wireshark, esta herramienta se encuentra instalada en la máquina virtual Mobexler.

1. Verificar que el dispositivo móvil esté conectado a la red local.
2. Abrir Wireshark y configurar el adaptador de red en algunos casos, puede ser necesario configurar el adaptador de red en modo promiscuo para capturar todos los paquetes de la red. Wireshark generalmente puede hacer esto automáticamente.
3. Seleccionar la interfaz de red, en la ventana principal de Wireshark, se mostrará una lista de interfaces de red disponibles.
4. Configurar Wireshark para capturar el tráfico de la misma red en la que está el dispositivo móvil, (generalmente la interfaz se llama Ethernet o eth0)
5. Iniciar la captura de paquetes: Hacer clic en el botón azul de “Start” o “Iniciar” para comenzar la captura de paquetes en la interfaz de red seleccionada.
6. Navegue a través de las funcionalidades de la aplicación móvil, para generar tráfico.
7. En el cuadro de texto de filtros en la parte superior de la ventana, ingresar el siguiente filtro (http or ftp or telnet or smtp).
8. Analizar los paquetes capturados para identificar datos sensibles transmitidos en texto plano.

Validación de Certificados SSL/TLS

La verificación de certificados es un proceso crucial para garantizar que la aplicación se comunica únicamente con servidores legítimos y no es vulnerable a ataques de hombre en el medio (*Man-in-the-middle*).

1. Configurar el Proxy en el dispositivo móvil, ir las secciones 2.5.4 o 2.5.5 para Android e iOS respectivamente.
2. Interceptar el tráfico de la aplicación: Abra Burp Suite, inicie la aplicación móvil y realice varias operaciones que impliquen comunicación de red. A partir de ese momento, Burp Suite comenzará a interceptar el tráfico.
3. Analizar el comportamiento de la Aplicación, si valida correctamente el certificado, debería rechazar la conexión o mostrar un error en la negociación de SSL/TLS. Si la aplicación continúa la conexión y se muestra tráfico en el proxy de Burp Suite sin advertencia, esto indica que no está validando correctamente el certificado.

8. Interacción con la plataforma

Exposición de Datos a través de componentes IPC en Android

La exposición de datos a través de componentes IPC en Android puede representar un riesgo significativo para la seguridad de las aplicaciones. El archivo *AndroidManifest.xml* es clave, ya que en él se declaran todos los componentes de la aplicación, incluidos `<activity>`, `<service>`, `<provider>` y `<receiver>`. El análisis de este archivo permite identificar configuraciones inseguras o permisos excesivos que podrían ser abusados por una aplicación maliciosa.

Otras aplicaciones instaladas en un dispositivo pueden interactuar con una actividad, servicio o proveedor de contenido si el componente tiene el atributo `android:exported` configurado como `"true"` o si se ha definido un `<intent-filter>` para los componentes `<activity>`, `<service>` o `<receiver>`. Esto

significa que el elemento está expuesto a otras aplicaciones en el dispositivo. Para evitar esto, se debe verificar que el atributo `android:exported="true"` y los `<intent-filter>` no estén presentes en el archivo *AndroidManifest.xml*, a menos que sea necesario.

Si alguno de los componentes mencionados tiene el atributo `"android:permission"` otras aplicaciones que deseen interactuar con este, deben declarar un elemento `<uses-permission>` correspondiente en su propio manifiesto para iniciar, detener o enlazar con el servicio. Si se encuentra el atributo `android:protectionLevel` tiene el valor `signature`. Este valor indica que sólo las aplicaciones de la misma empresa pueden acceder a los datos (es decir, firmados con la misma clave).

Una vez identificados una lista de mecanismos IPC en el archivo *AndroidManifest.xml*, analizar el código fuente para verificar si se filtran datos sensibles cuando se utilizan estos mecanismos. Por ejemplo, los proveedores de contenido pueden ser utilizados para acceder a información de bases de datos o archivos locales, y los servicios se pueden analizar para saber si devuelven algún tipo de datos. Los receptores de difusión pueden filtrar información sensible si son analizados o interceptados.

Actividades `<activity>`

Las *activities* son componentes de la aplicación que representan una sola pantalla con una interfaz de usuario. Si en el manifiesto se define el atributo `android:exported` con el valor **“**`true`**”** la *activity* puede ser lanzada por componentes de otras aplicaciones.

Con la herramienta jadx-gui y su función de búsqueda analizar el código fuente para comprender el funcionamiento de la actividad. Buscar la declaración de las actividades y sus métodos principales: `onCreate`, `onStart`, `onResume`,

`startActivity` y `startActivityForResult` y el uso de la clase `android.app.Activity`.

Servicios `<service>`

Un *Service* en Android es un componente de aplicación que puede realizar operaciones de larga duración en segundo plano, no proporciona una interfaz de usuario. Los *Services* son útiles para tareas que necesitan ejecutarse incluso cuando la aplicación no está en primer plano, como la reproducción de música, la descarga de archivos o la sincronización de datos. El atributo `android:exported` en el manifiesto define si un *Service* puede ser accedido por otras aplicaciones. El valor `"true"` significa que el servicio es accesible por otras aplicaciones. Si no se implementan controles adecuados, esto puede permitir que aplicaciones maliciosas interactúen con el *Service* de manera no autorizada.

Analizar el código fuente para comprender el funcionamiento del servicio. Identificar la declaración de los servicios y sus métodos principales: `onStartCommand`, `onBind`, `onCreate`, `startService` y `bindService` y el uso de la clase `android.app.Service`.

Proveedores de contenido `<provider>`

Un *Content Provider* es un componente en Android que permite a las aplicaciones gestionar el acceso a un conjunto estructurado de datos, proporcionando mecanismos para definir la seguridad de los datos, proveen una interfaz para que otras aplicaciones puedan acceder y modificar la información, como contactos, archivos multimedia, de manera controlada. Si este componente tiene el atributo `android:exported` con el valor “`true`” en el manifiesto significa que el *Content Provider* está disponible para otras aplicaciones. Esto puede ser un riesgo si no se gestionan adecuadamente los permisos de acceso a los datos.

Analizar el código fuente para comprender el flujo del *Content Provider*, buscar la declaración del componente y sus métodos principales: `.query`, `.update`, `.delete`

y el uso de las clases `android.database.sqlite`, `android.content.ContentProvider` y `android.database.Cursor`.

Para acceder al contenido de un elemento *Content Provider* se requiere de una URI de contenido, esta es esencial para acceder a los datos almacenados en una aplicación de manera organizada y segura.

Receptores de emisión `<receiver>`

Un *Broadcast Receiver* es un componente de Android que permite a las aplicaciones recibir y reaccionar a mensajes globales (*broadcasts*) enviados por el sistema o por otras aplicaciones. Estos mensajes pueden anunciar eventos del sistema (como el cambio de conectividad de red o la llegada de un SMS) o eventos personalizados definidos por las aplicaciones. Si el componente tiene el atributo `android:exported` con el valor "`true`" en archivo *AndroidManifest.xml*, significa que la aplicación puede recibir *broadcasts* de otras aplicaciones. Sin controles adecuados, esto puede permitir que aplicaciones maliciosas envíen *broadcasts* falsos o malintencionados.

Analizar el código fuente para comprender el funcionamiento del servicio. Con la herramienta jadx-gui y su función de búsqueda analizar el código fuente y buscar la declaración de los broadcasts receivers y sus métodos principales: `onReceive`, `sendBroadcast`, `sendOrderedBroadcast`, `registerReceiver` y `unregisterReceivery` el uso de la `android.content.BroadcastReceiver`.

Con la herramienta Drozer es posible analizar los componentes de una aplicación Android con sus diferentes módulos. Con el módulo `app.package.attacksurface`. Drozer lista todas las actividades, servicios, proveedores de contenido y receptores de difusión exportados, es decir son aquellos elementos que pueden ser lanzados por otras aplicaciones y si no están adecuadamente protegidos, que podrían ser explotados por una aplicación maliciosa.

Iniciar la aplicación Drozer, probar la conectividad y ejecutar el siguiente comando para listar todos los componentes de una aplicación:

run app.package.attacksurface com.example.app

La salida del comando anterior podría verse como en la Figura 23 – Ejemplo de ejecución del módulo `attacksurface` de Drozer

```
dz> run app.package.attacksurface com.example.app

Package: com.example.app
Application Name: Example App
Process Name: com.example.app
UID: 10089

Activities:
  - com.example.app.LoginActivity
  - com.example.app.TransactionActivity
  - com.example.app.StatementActivity

Services:
  - com.example.app.TransactionService

Broadcast Receivers:
  - com.example.app.LoginReceiver

Content Providers:
  - com.example.app.InsecureContentProvider

Permissions:
  - android.permission.INTERNET
  - android.permission.ACCESS_NETWORK_STATE
  - android.permission.READ_EXTERNAL_STORAGE
  - android.permission.WRITE_EXTERNAL_STORAGE
```

Figura 23 – Ejemplo de ejecución del módulo attacksurface de Drozer

A manera de ejemplo, se presentan los pasos para el análisis del componente "`<provider>`", con la herramienta Drozer.

Para analizar los proveedores de contenido de una aplicación, primero se debe obtener la superficie de ataque de la aplicación, es decir obtener los componentes expuestos y potencialmente vulnerables.

dz> run app.package.attacksurface com.example.myapp

La herramienta mostrará todos los componentes expuestos (actividades, servicios, proveedores de contenido y receptores de difusión).

Para identificar URIs de proveedores de contenido dentro de la aplicación, utilizar el módulo `scanner.provider.finduris de Drozer`. Este módulo muestra las rutas y determina URIs de contenido accesibles de varias maneras:

```
dz> run scanner.provider.finduris -a com.example.myapp
```

Una vez se cuenta con una lista de proveedores de contenidos accesibles, intenta extraer datos de cada proveedor a través de las URIs con el módulo `app.provider.query`, en el siguiente ejemplo se toma la URI `content://com.example.myapp.CPmessages/msg/` el comando a ejecutar se vería se la siguiente forma.

```
dz> run app.provider.query content://com.example.myapp.Cpmessages
/msg/ --vertical
```

Este comando mostrará todos los elementos en el almacén de datos accesibles desde el proveedor de contenido. También se puede utilizar Drozer para insertar, actualizar y eliminar registros de un proveedor de contenidos vulnerable.

Insertar un registro

```
dz> run app.provider.insert
content://com.example.myapp.CPmessages/msg
    --string msg “ejemplo” \
    --string type 0 \
    --integer _id 1
```

Actualizar un registro

```
dz> run app.provider.update
content://com.example.myapp.CPmessages/msg
    --selection "_id=1" \
    --string msg "nuevo mensaje" \
    --string type 0
```

Eliminar un registro

```
dz> run app.provider.delete
content://com.example.myapp.CPmessages/msg
    --selection "_id=1"
```

Inyección SQL en Proveedores de Contenido

La plataforma Android utiliza bases de datos SQLite para almacenar los datos de los usuarios. Dado que estas bases de datos se basan en SQL, pueden ser vulnerables a la inyección SQL. El módulo `app.provider.query` de Drozer permite identificar inyecciones SQL manipulando los campos de proyección y selección que se pasan al proveedor de contenido, a continuación, se presentan algunos ejemplos.

```
dz> run app.provider.query
content://com.example.myapp.CPmessages/msg/ --projection "'"
```

```
dz> run app.provider.query
content://com.example.myapp.CPmessages/msg/ --selection "'"
```

Si una aplicación es vulnerable a inyección SQL, devolverá un mensaje de error en la sentencia SQL ejecutada. Esta vulnerabilidad puede utilizarse para modificar o consultar datos del proveedor de contenidos vulnerable. En el siguiente ejemplo, el módulo de Drozer `app.provider.query` se utiliza para listar todas las tablas de la base de datos:

```
dz> run app.provider.query content://com.example.myapp.CPmessages/msg/ --projection "* FROM SQLITE_MASTER WHERE type='table';--"
```

Puede automatizar estos pasos con el módulo `scanner.provider.injection`, que identifica automáticamente proveedores de contenido vulnerables dentro de una aplicación:

```
dz> run scanner.provider.injection -a com.example.myapp
```

Proveedores de contenidos basados en sistemas de archivos

Los proveedores de contenidos pueden proporcionar acceso al sistema de archivos. Esto permite a las aplicaciones compartir archivos (que el *sandbox* de Android normalmente impide). Se pueden utilizar los módulos de Drozer `app.provider.read` y `app.provider.download` para leer y descargar archivos, respectivamente de elementos exportados vulnerables.

A continuación, se presentan ejemplos de proveedores de contenido vulnerables a *path traversal*, lo que permite leer archivos que estarían protegidos en el *sandbox* de Android.

```
dz> run app.provider.download content://com.example.myapp.FileProvider/../../../../../data/data/com.example.myapp/database.db /sdcard/database.db
```

El módulo `scanner.provider.traversal` permite automatizar el proceso de búsqueda de proveedores de contenido vulnerables a *path traversal.* Este comando devolverá las URIs potencialmente vulnerables.

```
dz> run scanner.provider.traversal -a com.example.myapp
```

Información sensible en capturas de pantalla generadas automáticamente

Cuando una aplicación pasa a segundo plano, se toma una captura de pantalla de la aplicación actual y se muestra con fines estéticos cuando la aplicación vuelve al primer plano. Sin embargo, esto puede filtrar información sensible.

Para determinar si una aplicación Android puede exponer información sensible a través del selector de aplicaciones, analizar si se ha establecido el atributo `FLAG_SECURE`. Debería identificar algo similar al siguiente fragmento de código: `getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE, WindowManager.LayoutParams.FLAG_SECURE);`

Abrir el archivo apk con la herramienta jadx-gui y buscar el atributo `FLAG_SECURE`, o bien instalar la aplicación abrirla y desplazarse a cualquier pantalla que contenga información sensible y pulsa el botón de inicio para enviar la aplicación a segundo plano; a continuación, pulsa el botón “recientes”, comúnmente representado por un cuadrado para ver la captura autogenerada. Si está activado, la miniatura de la aplicación estará vacía o si el indicador no está activado, se mostrará la información de la actividad. Las capturas de pantalla autogeneradas se almacenan en el directorio */data/system_ce/<ID_USUARIO>/<IMAGE_FOLDER_NAME>*. El nombre de *<IMAGE_FOLDER_NAME>* depende del fabricante, pero los nombres más comunes son *snapshots* y *recent_images*.

En iOS si dispone del código fuente, busque el método `applicationDidEnterBackground` para determinar si la aplicación limpia la pantalla antes de pasar a segundo plano.

A continuación, se muestra una implementación de ejemplo en swift que utiliza una imagen de fondo predeterminada (*overlayImage.png*) siempre que la aplicación pone en segundo plano, anulando la vista actual evitando así exponer datos del usuario:

```swift
private var backgroundImage: UIImageView?

func applicationDidEnterBackground(_ application: UIApplication) {
    let myBanner = UIImageView(image: #imageLiteral(resourceName: "overlayImage"))
    myBanner.frame = UIScreen.main.bounds
    backgroundImage = myBanner
    window?.addSubview(myBanner)
}

func applicationWillEnterForeground(_ application: UIApplication) {
    backgroundImage?.removeFromSuperview()
}
```

Figura 24 – Implementación superposición en Swift

Para determinar si una aplicación iOS puede exponer información sensible a través del selector de aplicaciones puede utilizar un enfoque visual para validar este caso de prueba utilizando cualquier dispositivo iOS.

1. Navegar hasta la pantalla de la aplicación que muestre información confidencial, como un nombre de usuario, una dirección de correo electrónico o detalles de una cuenta.
2. Pasar la aplicación en segundo plano pulsando el botón Inicio de su dispositivo iOS.
3. Compruebe que se muestra una imagen predeterminada como elemento de la vista superior en lugar de la vista que contiene la información sensible

Para inspeccionar las capturas de pantalla autogeneradas conectarse al dispositivo iOS por SSH y navegue hasta el directorio *Snapshots*. La ubicación puede variar según la versión de iOS, pero suele estar dentro del directorio Library de la aplicación. */var/mobile/Containers/Data/Application/<UUID>/Library/SplashBoard/Snapshots/*

Deep Links y Universal Links

Los enlaces profundos (*deep links*) son URI (*Uniform Resource Identifier*) predefinidos que permiten acceder directamente a una actividad en una aplicación web o móvil cuando se hace clic en ellos. Estos enlaces se encuentran en páginas dentro de una aplicación web o en las WebViews de una aplicación móvil. Permiten una transición fluida desde una fuente externa, como un sitio web o una notificación, hacia una ubicación específica dentro de la aplicación móvil. Cuando el usuario hace clic en un enlace profundo y dispone de la aplicación para abrir ese tipo de enlace, una ventana emergente le sugiere abrir el enlace con la aplicación correspondiente.

Para determinar si se definen enlaces profundos (con o sin esquemas de URL personalizados) inspeccionar el archivo *AndroidManifest.xml* con la herramienta jadx-gui en busca de elementos `<intent-filter>` que a su vez llevan adentro la etiqueta `<category>` con el atributo `android:name` "`DEFAULT y BROWSABLE`"

Si el `<intent-filter>` incluye la bandera `android:autoVerify="true"`, significa que el sistema Android se comunica con el host declarado en `android:host` para intentar acceder al archivo Digital Asset Links y verificar los *App Links*. Un enlace profundo puede considerarse un *App Link* solo si la verificación es exitosa.

Cuando se enumeran los enlaces profundos, los elementos `<data>` dentro del mismo `<intent-filter>` se combinan para tener en cuenta todas las variaciones de sus atributos combinados.

```
<intent-filter>
  ...
  <data android:scheme="https" android:host="www.example.com" />
  <data android:scheme="app" android:host="open.my.app" />
</intent-filter>
```

Figura 25 – Ejemplo de <intent-filter> de un enlace profundo

Podría parecer que esto solo soporta `https://www.example.com` y `app://open.my.app`. Sin embargo, en realidad soportaría:
`https://www.example.com`
`app://open.my.app`
`app://www.example.com`
`https://open.my.app`

Verificación de la Asociación Correcta del Sitio Web
Si los enlaces profundos contienen el atributo `android:autoVerify="true"`, deben ser verificados para ser considerados App Links.

- Comprobación de la ausencia del archivo de Digital Asset Links:
  Intentar ingresar a la ruta */.well-known/* del dominio. Ejemplo: `https://www.example.com/.well-known/assetlinks.json`
- Comprobación de un archivo de Digital Asset Links válido a través de HTTP.
- Comprobación de archivos de Digital Asset Links inválidos servidos a través de HTTPS.
  Por ejemplo:
  El archivo contiene JSON inválido.
  El archivo no incluye el paquete de la aplicación de destino.

Verificación de Redirecciones
El sistema no verifica ningún *App Link* de Android si el servidor establece una redirección, como de `http://example.com` a `https://example.com` o de `example.com` a `www.example.com`.

Verificación de Subdominios

Si un filtro de intención lista varios hosts con diferentes subdominios, debe existir un archivo válido de Digital Asset Links en cada dominio. Por ejemplo, el siguiente filtro de intención incluye `www.example.com` y `mobile.example.com` como hosts de URL de intención aceptados. Para que los enlaces profundos se registren correctamente, se debe publicar un archivo válido de Digital Asset Links en ambos `https://www.example.com/.well-known/assetlinks.json` y `https://mobile.example.com/.well-known/assetlinks.json`.

Verificación de Wildcards (Comodines)

Si el nombre de host incluye un wildcard (como *.example.com), debes encontrar un archivo válido de Digital Asset Links en el nombre de host raíz: `https://example.com/.well-known/assetlinks.json`.

Verificación del Método de Manejo

Incluso si el enlace profundo está correctamente verificado, la lógica del método de manejo debe analizarse cuidadosamente. En especial a los enlaces profundos que se utilizan para transmitir datos que son controlados externamente por el usuario u otra aplicación.

Primero, obtener el nombre de la *Activity* del elemento `<activity>` en el Manifiesto de Android que define el `<intent-filter>` objetivo y buscar el uso de `getIntent().getData()`. Este enfoque general de localizar estos métodos se puede usar en la mayoría de las aplicaciones al realizar ingeniería inversa y es clave para entender cómo la aplicación utiliza enlaces profundos y maneja cualquier dato de entrada proporcionado externamente y si pudiera estar sujeto a cualquier tipo de abuso. El siguiente ejemplo es un fragmento de una aplicación en Kotlin descompilada con jadx.

```
// snippet edited for simplicity
public final class WebViewActivity extends AppCompatActivity {
    private ActivityWebViewBinding binding;

    public void onCreate(Bundle savedInstanceState) {
        Uri data = getIntent().getData();
        String html = data == null ? null : data.getQueryParameter("html");
        Uri data2 = getIntent().getData();
        String deeplink_url = data2 == null ? null : data2.getQueryParameter("url");
        View findViewById = findViewById(R.id.webView);
        if (findViewById != null) {
            WebView wv = (WebView) findViewById;
            wv.getSettings().setJavaScriptEnabled(true);
            if (deeplink_url != null) {
                wv.loadUrl(deeplink_url);
```

Figura 26 – Ejemplo de manejo de parámetros en Deep Links Android

Es posible rastrear la variable `deeplink_url` y analizar el resultado de la llamada `wv.loadUrl`. Esto significa que un potencial atacante controla la URL que se carga en el WebView (como se muestra arriba, tiene JavaScript habilitado) podría estar renderizando un parámetro controlado por el usuario. Por lo que este escenario desencadenaría Cross-Site Scripting reflejado dentro del contexto del WebView.
Sin embargo, otras posibles vulnerabilidades y métodos de prueba podrían presentarse.

Aquí utilizará la lista de enlaces profundos del identificados en el archivo *AndroidManifest.xml* para iterar y determinar cada método manejador y los datos procesados, si los hay. El siguiente ejemplo asume una aplicación de destino que acepta este enlace profundo: `deeplinkdemo://load.html`. Sin embargo, aún no conocemos el método `handler` correspondiente, ni los parámetros que acepta.
El script depp link observer `https://codeshare.frida.re/@leolashkevych/android-deep-link-observer/` para monitorear todos los enlaces profundos invocados que activan una llamada a `Intent.getData`.

Con la herramienta ADB se invoca el enlace profundo con el siguiente comando:

```
adb shell am start -W -a android.intent.action.VIEW -d "deeplinkdemo://load.html/?message=ok#part1"
```

El script de Frida mostrará la información presentada en la Figura 27.

Figura 27 – Resultado de manejo de Deep Link en Android

En este caso se ha creado el enlace profundo incluyendo parámetros arbitrarios (`?message=ok`) y un fragmento (`#part1`). En este punto no se sabe si se están utilizando. La información anterior revela información útil que puede ser utilizada para realizar ingeniería inversa en la aplicación.

Universal Links

Universal Links son una característica de iOS que permite a las aplicaciones abrir enlaces web específicos directamente en la aplicación en lugar de en el navegador web.

- Experiencia de Usuario Mejorada: Al abrir un enlace, los usuarios son llevados directamente a la aplicación correspondiente si está instalada, en lugar de abrir el navegador web.

- Seguridad y Control: Universal Links proporcionan una forma segura de manejar enlaces, ya que solo el propietario del dominio puede habilitar su uso dentro de la aplicación.
- Retención de Usuarios: Mejora la retención de usuarios al redirigir automáticamente a la aplicación, donde la experiencia es más optimizada que en la web móvil.
- Contexto Persistente: Mantiene el contexto del usuario y proporciona una navegación más coherente y contextualizada.

Revisar la capacidad de Associated Domains

Los Universal Links requieren que el desarrollador agregue la capacidad de Associated Domains e incluya en ella una lista de los dominios que la aplicación soporta.

En Xcode ir a la pestaña "Capabilities" y buscar "Associated Domains". También es posible inspeccionar el archivo *.entitlements* buscando `com.apple.developer.associated-domains`. Cada uno de los dominios debe estar con el prefijo "`applinks:`", por ejemplo, `applinks:www.example.com`.

Si no se dispone del código fuente original, se puede obtener el archivo *entitlements.plist* de la siguiente manera:

Extraer el contenido del archivo .ipa

```
unzip App.ipa
```

En la carpeta Payload, se encuentra una carpeta con el nombre de la aplicación (por ejemplo, MyApp.app). Dentro de esta carpeta, habrá un archivo ejecutable con el nombre de la aplicación (por ejemplo, MyApp).

```
cd /Payload/MyApp.app/
```

Utilizar la herramienta codesign para extraer el archivo .entitlements del ejecutable de la aplicación:

codesign -d --entitlements - App > entitlements.plist

Recuperar el archivo Apple App Site Association

Intentar recuperar el archivo *apple-app-site-association* desde el servidor usando los dominios asociados obtenidos en el paso anterior. Este archivo debe ser accesible vía HTTPS, sin redirecciones, en `https://<dominio>/apple-app-site-association` o `https://<dominio>/.well-known/apple-app-site-association`.

Alternativamente, se puede usar el validador de Apple App Site Association (`https://branch.io/resources/aasa-validator/`). Después de ingresar el dominio, mostrará el archivo, lo verificará y mostrará los resultados.

iOS no abre ningún enlace no verificado.

La verificación de enlaces universales se produce en el momento de la instalación. iOS recupera el archivo para los dominios declarados en el archivo *entitlements*. iOS se negará a abrir esos enlaces si la verificación no ha tenido éxito.

Verificación del Método Receptor de Enlaces

Para recibir y manejar enlaces de manera adecuada, el delegado de la aplicación debe implementar `application:continueUserActivity:restorationHandler:`. Si se cuenta con el proyecto original, intentar buscar este método.

Si no se dispone del código fuente original, con el comando strings buscar las cadenas del método receptor de enlaces:

strings <ruta_al_binario> | grep 'application:continueUserActivity:restorationHandler:'

Esto permitirá identificar si el método está presente en el binario de la aplicación y analizar cómo se manejan los enlaces universales en ausencia del código fuente original.

Verificación del Método de Manejo de Datos

Se debe verificar cómo se valida la información recibida. Apple advierte explícitamente sobre esto: "Los enlaces universales ofrecen un vector de ataque potencial a tu aplicación, por lo que asegúrate de validar todos los parámetros de URL y descartar cualquier URL malformada. Además, limitar las acciones disponibles a aquellas que no pongan en riesgo los datos del usuario." Si la URL incluye parámetros, no deben ser confiables antes de ser validados (incluso si provienen de un dominio confiable). Por ejemplo, podrían haber sido manipulados o podrían incluir datos mal formados. Si ese es el caso, se debe descartar toda la URL y, por lo tanto, la solicitud del enlace universal.

Verificación si la Aplicación Está Llamando a Enlaces Universales de Otras Aplicaciones

Una aplicación puede estar llamando a otras aplicaciones mediante enlaces universales para simplemente activar algunas acciones o transferir información. En ese caso, se debe verificar que no esté filtrando información sensible.

Si se dispone del código fuente original, es posible buscar el método `openURL:options:completionHandler:` y revisar los datos que se están manejando. Este método no solo se utiliza para abrir enlaces universales, sino también para llamar a esquemas de URL personalizados.

Si no se dispone del código fuente original, buscar los símbolos y en las cadenas del binario de la aplicación. Por ejemplo, para buscar métodos en Objective-C que contengan "openURL": *`rabin2 -zq App | grep "openURL`*

WebViews en Android/iOS

Un WebView en Android un componente que permite a las aplicaciones mostrar contenido web directamente dentro de una actividad. Básicamente, es un navegador web integrado en la aplicación. Las aplicaciones pueden cargar y mostrar páginas web, ejecutar JavaScript y manejar interacciones web sin salir de la aplicación.

Ejecución de JavaScript en WebViews Android

JavaScript está desactivado por defecto para WebViews y debe ser activado explícitamente. Buscar el método `setJavaScriptEnabled` con la herramienta jadx-gui para comprobar la activación de JavaScript.

Para probar si existe una forma de inyectar JavaScript malicioso en el componente Web View de la aplicación analizar los campos de entrada de la página web que se renderiza dentro de este componente, con esto se enviará al WebView de la aplicación móvil cuando el usuario navegue a la función vulnerable.

Con ayuda del proxy de Burp Suite manipular los campos de entrada y respuesta del servidor que aloja la página web para inyectar código JavaScript. Revisar si se cargan archivos almacenados fuera del almacenamiento interno de la aplicación en el WebView donde sea posible inyectar código JavaScript.

Manejadores de protocolo WebView

Analizar el archivo .apk con la herramienta jadx-gui, para comprobar el uso de WebView buscar la clase `android.webkit.WebView` en el código fuente. Los siguientes métodos de WebView controlan el acceso a recursos: `setAllowContentAccess`, `setAllowFileAccess`, `setAllowFileAccessFromFileURLs` y `setAllowUniversalAccessFromFileURLs` se utilizan en la clase `WebSettings` de Android para controlar el acceso del contenido web a los archivos y recursos del dispositivo. Estos métodos son importantes para

garantizar la seguridad y evitar que el código JavaScript malicioso acceda a datos sensibles.

`setAllowContentAccess`: Este método permite o deniega el acceso a URL de contenido que permite a los WebViews cargar contenido desde un proveedor de contenido (habilitado por defecto).

`setAllowFileAccess`: Habilita/deshabilita el acceso al sistema de archivos dentro de un WebView. El acceso a activos y recursos no se ve afectado y es accesible a través de `file:///android_asset` y `file:///android_res`.

`setAllowFileAccessFromFileURLs`: Permite o no que JavaScript que se ejecute en el contexto de una URL de esquema de archivos y acceda al contenido de otras URL de esquema de archivos.

`setAllowUniversalAccessFromFileURLs`: Permite o no que JavaScript que se ejecute en el contexto de una URL de esquema de archivos acceda al contenido desde cualquier origen.

Si uno o varios de los métodos anteriores están activados, determinar si son realmente necesarios para que la aplicación funcione correctamente.

Abrir el archivo .apk con la herramienta jadx-gui y buscar instancias de la clase `android.webkit.WebView`. Analizado los siguientes fragmentos de código donde se cargue un archivo local en el webview

```
WebView = new WebView(this);
webView.loadUrl("file:///android_asset/filename.html");
```

Verificar la ubicación desde la que se carga el archivo HTML. Si el archivo se carga desde un almacenamiento externo, este puede ser leído y modificado por otra aplicación. El siguiente ejemplo presenta la carga de un archivo html desde el almacenamiento externo del dispositivo.

```
webview.loadUrl("file:///"+Environment.getExternalStorageDirectory()
.getPath() + "index.html");
```

La URL en el método `loadURL` también debe comprobarse en busca de parámetros dinámicos que puedan ser manipulados; su manipulación puede llevar a la inclusión de archivos locales.

Por último, crear una lista que defina las páginas web locales, remotas y los protocolos que se permiten cargar.

Limpieza de datos en WebViews

Las API de WebView proporcionan diferentes métodos para almacenar datos, como `cacheData()`, `saveData()`, y `setWebViewClient()`. Es importante utilizar la API correcta para cada tipo de dato y asegurarse de que los datos se eliminan cuando ya no son necesarios. Por lo tanto, es importante analizar la aplicación para verificar que los datos almacenados en las vistas web se eliminen correctamente cuando ya no son necesarios.

Analizar el código fuente de la aplicación para identificar las API de WebView que se utilizan para almacenar datos. Después, verificar que estas API se utilizan correctamente y que los datos se eliminan cuando ya no son necesarios.

Inicialización: Una aplicación puede inicializar el componente WebView de forma que evite almacenar cierta información utilizando los métodos `setDomStorageEnabled`, `setAppCacheEnabled` o `setDatabaseEnabled` de `android.webkit.WebSettings`.

`setDomStorageEnabled(false)`: Desactiva / activa el almacenamiento DOM. Cuando está desactivado, las páginas web no podrán almacenar datos en formato HTML5.

`setAppCacheEnabled(false)`: Desactiva / activa el caché de aplicaciones. Cuando está desactivado, el WebView no guardará recursos web para su uso posterior.

`setDatabaseEnabled(false)`: Desactiva / activa el almacenamiento de bases de datos. Cuando está desactivado, las páginas web no podrán almacenar datos en bases de datos locales.

Cache: La clase WebView de Android ofrece el método `clearCache()` para borrar la cache de todas los WebViews utilizadas por la aplicación. Además, comprobar si la aplicación está sobrescribiendo el método `onRenderProcessUnresponsive()` para el caso en que la WebView no responda, ya que el método `clearCache()` también podría ser llamado desde ahí.

WebStorage: El método `WebStorage.deleteAllData()` se puede utilizar para borrar todo el almacenamiento utilizado actualmente por las API de almacenamiento de JavaScript, incluyendo la base de datos Web SQL y las API de almacenamiento Web HTML5

Cookies: todas las cookies existentes pueden ser eliminadas utilizando el método `CookieManager.removeAllCookies()`.

Archivos: para borrar adecuadamente los datos en algunos directorios como directorios ocultos y almacenamiento compartido puede no ser tan sencillo, algunas aplicaciones utilizan una solución pragmática que consiste en eliminar manualmente directorios seleccionados que se sabe que contienen datos del usuario. Esto se

puede hacer utilizando la API `java.io.File` como `java.io.File.deleteRecursively`.

Ejecutar la aplicación en el dispositivo y monitorear el almacenamiento para identificar los archivos y datos que son creados por las vistas web. Luego, verificar que estos archivos y datos se eliminan cuando la aplicación se cierra o cuando el usuario navega a una nueva página, el análisis del almacenamiento se puede realizar con la aplicación root explorer y los archivos se pueden encontrar en las siguientes ubicaciones:

Almacenamiento interno de la aplicación

- Caché: Esta carpeta contiene recursos web como imágenes, CSS y JavaScript que el WebView ha descargado para un acceso más rápido en futuras visitas. Ruta: */data/data/<com.example.myapp>/cache/webview/Cache*
- Bases de datos: Si el WebView ha habilitado el almacenamiento de bases de datos, las bases de datos locales se almacenan en: */data/data/<com.example.myapp>/databases*
- Archivos de caché de DOM: El almacenamiento DOM se utiliza para almacenar datos estructurados en formato HTML5. La ruta común es: */data/data/<com.example.myapp>/files/webappcache*

Almacenamiento externo

- Caché: Si la aplicación tiene habilitado el almacenamiento en tarjeta SD, los recursos web se almacenan en: */storage/sdcard/Android/data/<com.example.myapp >/cache/webview/Cache*
- Bases de datos: Si el WebView ha habilitado el almacenamiento en tarjeta SD, las bases de datos locales se almacenan en: */storage/sdcard/Android/data/<com.example.myapp>/databases*

WebViews en iOS

`UIWebView` y `WKWebView` son tecnologías utilizadas para añadir contenido web en aplicaciones iOS. `UIWebView` es la tecnología más antigua y está obsoleta desde iOS 12. `WKWebView` es la tecnología más reciente, y es la opción recomendada para incrustar contenido web en nuevas aplicaciones iOS. JavaScript no se puede desactivar para `UIWebView` que es otra razón para evitar usarlo.

Identificar el uso de WebView

Buscar usos de las clases `UIWebView` y `WKWebView` con la herramienta de búsqueda en Xcode, si se dispone del código fuente. O bien analizando el binario compilado buscar en sus símbolos y cadenas:

La herramienta strings permite extraer cadenas de texto imprimibles de un archivo binario, esta se puede usar en conjunto con grep para buscar las clases `UIWebView`, `WKWebView` o `SFSafariViewController`:

```
strings ./binaryApp | grep -i "UIWebView"
strings ./binaryApp | grep -i "WKWebView"
```

`WKWebView` presenta varias ventajas de seguridad con respecto a `UIWebView`:

- `javaScriptEnabled` este método permite deshabilitar JavaScript para evitar vulnerabilidades de inyección de código JavaScript.
- `JavaScriptCanOpenWindowsAutomatically` evita que JavaScript abra nuevas ventanas, como ventanas emergentes.
- La propiedad `hasOnlySecureContent` verifica que los recursos cargados por el WebView se recuperan a través de conexiones cifradas (HTTPS).
- Aislamiento de errores: `WKWebView` implementa el renderizado fuera de proceso, lo que significa que los errores de corrupción de memoria en el `WebView` no afectarán al proceso principal de la aplicación.

Si una o varias de las propiedades anteriores están activadas, se debe determinar si son realmente necesarias para que la aplicación funcione correctamente.
Analice las entradas en los campos de la página web en el WebView, intentar inyectar código JavaScript malicioso y analizar si es interpretado por el navegador.

WebView Protocol Handlers

Cuando se carga contenido en un WebView desde el directorio de datos de la aplicación, los usuarios no deberían poder cambiar el nombre del archivo o la ruta desde la que se carga el archivo, y no deberían poder editar el archivo cargado.
Esto se vuelve especialmente crítico cuando se utilizan los métodos obsoletos `loadHTMLString:baseURL:` o `loadData:MIMEType:textEncodingName:baseURL:` con un `baseURL` establecido en `nil`, `file:` o `applewebdata:`. Estos métodos podrían otorgar acceso no autorizado a archivos locales en el dispositivo.

Para evitar esto, establecer el `baseURL` en `about:blank` (una página vacía). El enfoque más seguro es evitar `UIWebView` por completo y utilizar `WKWebView` en su lugar. `WKWebView` ofrece mejores funciones de seguridad y mecanismos de *sandboxing*.

Al implementar `WKWebViews`, Apple recomienda utilizar `loadHTMLString:baseURL:` o `loadData:MIMEType:textEncodingName:baseURL:` para cargar archivos HTML locales y `loadRequest:` para el contenido web. Normalmente, los archivos locales se cargan en combinación con métodos que incluyen, entre otros: `pathForResource:ofType:`, `URLForResource:withExtension:` o `init(contentsOf:encoding:)`.

Si sólo dispone del binario compilado, buscar estos métodos con el siguiente comando:

```
strings ./ binaryApp | grep -i "loadHTMLString"
```

Verificar si la aplicación está utilizando el método `loadFileURL: allowingReadAccessToURL:`. El primer parámetro es URL y contiene la URL a cargar en el `WebView`, el segundo parámetro `allowingReadAccessToURL` puede contener un único archivo o un directorio. Si contiene un único archivo, ese archivo estará disponible para el `WebView`. Sin embargo, si contiene un directorio, todos los archivos de ese directorio estarán disponibles para el `WebView`. En caso de que sea un directorio, verificar que no se encuentren datos sensibles en este directorio.

Si sólo dispone del binario compilado, buscar el parámetro con el comando:
`strings ./ binaryApp | grep -i " allowingReadAccessToURL "`

Si se ha identificado `UIWebView` en uso, entonces se aplica lo siguiente:

- El esquema `file://` siempre está habilitado.
- El acceso a ficheros desde URLs `file://` está siempre activado.
- El acceso universal desde URLs `file://` está siempre habilitado.

Respecto a WKWebViews aplica lo siguiente:

- El esquema `file://` también está siempre activado y no se puede desactivar.
- El acceso a archivos desde URLs `file://` está deshabilitado por defecto, pero puede habilitarse

Detección de números de teléfono
En Safari iOS, la detección de números telefónicos está activada de forma predeterminada. Si una página HTML contiene números que pueden interpretarse como números de teléfono, pero en realidad no lo son, es posible que se quiera evitar este comportamiento para evitar que el DOM se modifique cuando el navegador lo analiza.

Para desactivar la detección de números telefónicos en Safari para iOS, utilizar la etiqueta meta de detección de formato (`<meta name="format-detection" content="telephone=no">`). Posteriormente, se deberá utilizar enlaces telefónicos (por ejemplo, `<a href="tel:1-408-555-5555">1-408-555-5555</a>`) para crear un enlace de forma explícita.

Si es posible cargar archivos locales a través de una `WebView`, la aplicación podría ser vulnerable a *path traversal*. Esto permitiría acceder a todos los archivos dentro del *sandbox* o incluso escapar de este con acceso completo al sistema de archivos (si el dispositivo tiene *jailbreak*). Por lo tanto, debe verificarse si un usuario puede cambiar el nombre del archivo o la ruta desde la que se carga el archivo, y no debe permitir editar el archivo cargado. En una aplicación en producción, JavaScript puede ser inyectado a través de una vulnerabilidad de *Stored Cross-Site Scripting* o un ataque MITM.

Comprobando Cómo se Cargan las WebViews

Si se carga un archivo a través del método en el componente `WKWebViews`, la aplicación lo hará llamando a `URLForResource:withExtension:` y `loadHTMLString:baseURL`.

Para inspeccionar estos métodos, usar frida-trace y rastrear todos los métodos "`loadHTMLString`" y "`URLForResource:withExtension:`".

```
frida-trace -U "Example App?"    -m "*[WKWebView *loadHTMLString*]"
-m "*[* URLForResource:withExtension:]"
```

Esto mostrará en la consola el contenido la ruta del archivo en el parámetro `URLForResource` cargado al `WKWebView` y el parámetro `baseURL` con la extensión.

Determinar el acceso a archivos WebView

Aunque no se disponga del código fuente, es posible determinar si un componente `WebView` de la aplicación permiten el acceso a ficheros y de qué tipo. Para esto, navegue al componente `WebView` en la aplicación e inspeccionar todas sus instancias, para cada una de ellas obtener los valores mencionados de los atributos `allowFileAccessFromFileURLs` y `allowUniversalAccessFromFileURLs`. Esto sólo se aplica a las `WKWebViews` (las `UIWebVIews` siempre permiten el acceso a ficheros).

El siguiente script de Frida, obtiene los atributos URL, `javaScriptEnabled` y `allowFileAccessFromFileURLS` de los objetos `WKWebViews.` Para ejecutar este script en la aplicación guardar el código con el nombre *webviews_inspector.js*

```
ObjC.choose(ObjC.classes['WKWebView'], {
  onMatch: function (wk) {
    console.log('onMatch: ', wk);
    console.log('URL: ', wk.URL().toString());
    console.log('javaScriptEnabled: ', wk.configuration().preferences().javaScriptEnabled());
    console.log('allowFileAccessFromFileURLs: ',

 wk.configuration().preferences().valueForKey_('allowFileAccessFromFileURLs').toString());
    console.log('hasOnlySecureContent: ', wk.hasOnlySecureContent().toString());
    console.log('allowUniversalAccessFromFileURLs: ',
            wk.configuration().valueForKey_('allowUniversalAccessFromFileURLs').toString());
  },
  onComplete: function () {
    console.log('done for WKWebView!');
  }
});
```

Figura 28 – Script de Frida para obtener parámetros de WKWebView

Posteriormente, obtener el Bundle ID de la aplicación a analizar con el comando:

`frida-ps -U`

En la lista obtenida, identificar el nombre del paquete de la aplicación que se requiere analizar. El nombre del paquete suele estar en la columna "Name" o "Identifier".

Después ejecutar con el comando para analizar el componente WebView:

`frida -U -f <com.example.myapp> -l webviews_inspector.js`

9. Informe de resultados y recomendaciones

El informe de resultados debe incluir la documentación detallada de las vulnerabilidades identificadas durante la auditoría, descripción, impacto potencial, puntuación y vector CVSS utilizando las tres métricas base, temporal y ambiental. Es importante que el informe incluya recomendaciones claras para mitigar las vulnerabilidades encontradas. A continuación, se describen las secciones clave que deben formar parte de un informe de resultados de auditoría de aplicaciones.

- **Portada**: Incluye el título del informe, el nombre de la empresa que realizó la auditoria, fecha y otros datos identificativos.
- **Resumen ejecutivo**: Proporciona una visión general de los hallazgos y el impacto general en la seguridad de la organización, dirigido a audiencias no técnicas.
- **Alcance del proyecto**: Detalla los objetivos, el alcance, las limitaciones y el contexto en que se realizó la auditoria.
- **Metodología/Enfoque**: Describe el enfoque y las metodologías empleadas, como OWASP, PTES, o estándares propios de la organización.
- **Hallazgos**: Lista cada vulnerabilidad encontrada en detalle, incluyendo:
    - Nombre de la vulnerabilidad.
    - Descripción: Explicación de la vulnerabilidad y su impacto.
    - Puntuación y vector CVSS.
    - Evidencia: Pruebas, capturas de pantalla, y detalles técnicos que muestran cómo fue identificada.
    - Detalles de la explotación: Descripción de cómo puede ser explotada.
    - Recomendaciones: Propuesta de solución para mitigar la vulnerabilidad.

10. Pruebas de Post-Remediación

Después implementar las recomendaciones de mitigación de vulnerabilidades, es importante realizar auditorías de seguimiento. Esto ayudará a garantizar que las vulnerabilidades se han mitigado eficazmente y que no han surgido otras nuevas debido al proceso de corrección, además, sirve para confirmar que las soluciones son efectivas y cumplen con el nivel de seguridad requerido

5 CASO PRÁCTICO

En este capítulo se desarrolla la auditoria a dos aplicaciones, utilizando los recursos descritos en la sección Recursos y configuraciones para la auditoria.

Las aplicaciones elegidas para el análisis son: InsecureBankv2 para la plataforma Android y la aplicación DVIA-v2 por sus siglas en inglés (*Damn Vulnerable iOS Application*) para la plataforma iOS, son aplicaciones diseñadas intencionalmente para ser inseguras. El objetivo es estas es enseñar a los desarrolladores y profesionales de seguridad fallas que generalmente están presentes en las aplicaciones debido a prácticas de codificación deficientes o inseguras.
No se requiere una autorización para la ejecución de la auditoría sobre estas aplicaciones, ya que han sido diseñadas y publicadas en la plataforma GitHub por sus autores, dineshshetty y Prateekg147 respectivamente.

En la siguiente tabla se presentan los detalles de la auditoría sobres las aplicaciones seleccionadas como caso de estudio.

Aplicación	Plataforma	Modalidad de las pruebas
InsecureBankv2	Android	Caja Negra
DVIA-v2	iOS	Caja Negra

La auditoría se realizó en modalidad de caja negra, debido a que es el enfoque que tendría un atacante que solo tiene acceso al binario de la aplicación y se llevó a cabo siguiendo la metodología propuesta descrita en la sección Método propuesto
De acuerdo con el capítulo Recursos y configuraciones para la auditoria se elegio la máquina virtual Mobexler, ya que tiene las herramientas necesarias preinstaladas, además, cuenta con soporte constante actualización por parte de sus desarrolladores y la comunidad.

5.1 Android

En esta sección se describen los procesos técnicos ejecutados de acuerdo con la metodología propuesta para auditar la aplicación Android en busca de fallas de seguridad.

Deficiencias en componentes Android

Para iniciar se realizará un análisis estático con MobSF, solo es necesario cargar el archivo .apk en la herramienta, esta cuenta con un escáner automatizado. Al finalizar, la herramienta presenta los resultados como se pude apreciar en la Figura 29.

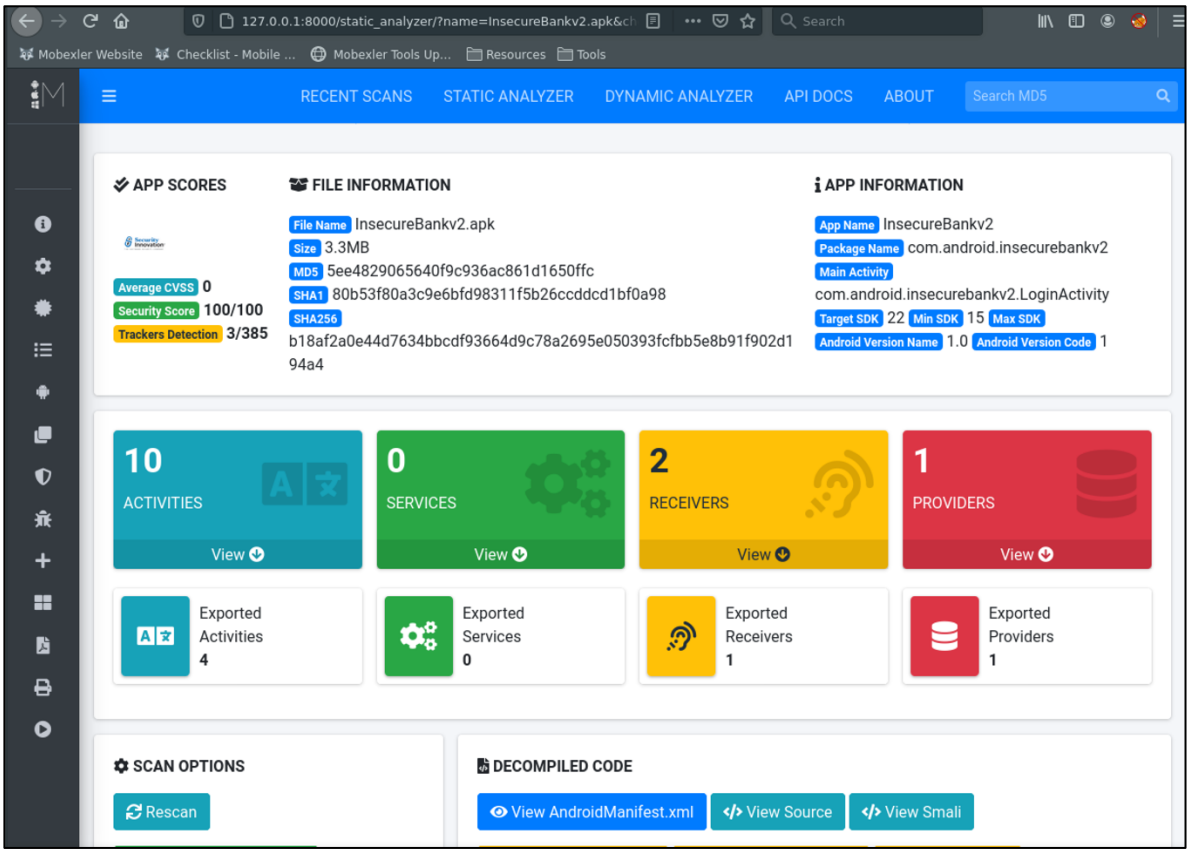


Figura 29 – Resultados análisis estático con MobSF

Se puede observar que en los resultados se incluyen detalles de la aplicación analizada, como su nombre, el nombre del paquete y sus componentes (*activities, services, content providers* y *brodcasst receiver*), además, una puntación de seguridad con un valor 100, resulta interésate este valor ya que es una aplicación con múltiples vulnerabilidades. Aunque esta herramienta es de gran ayuda durante una auditoria, es importante validar los resultados de forma manual ya que pueden presentarse falsos positivos o falsos negativos.

Activities expuestas

Se utilizó la herramienta jadx-gui para abrir el archivo *InsecureBankv2.apk* y analizando el *AndroidManifest.xml* que se presenta en la Figura 30, se observan algunas *activities* que forman parte de la aplicación.

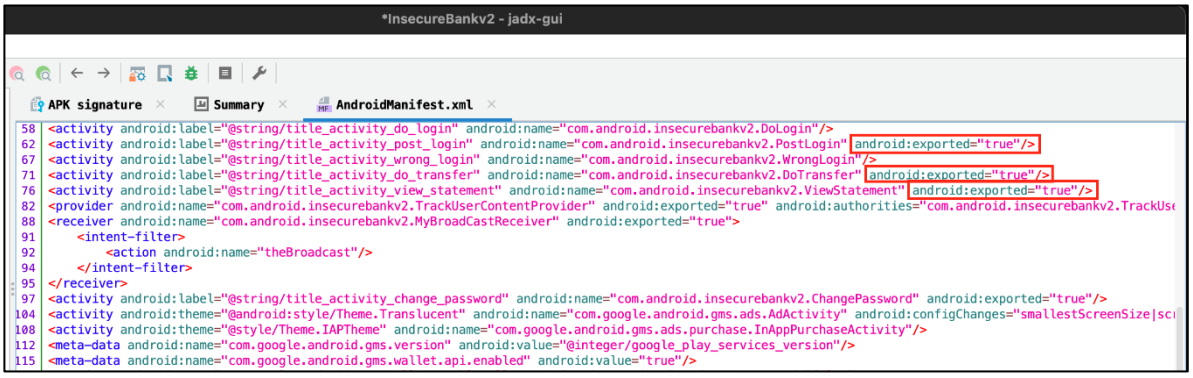


Figura 30 – Activites expuestas de la aplicación InsecureBankv2

Tal como se describe en el punto 8. Interacción con la plataforma de la metodología propuesta, un elemento con el atributo `android:exported` establecido como `true` permite que otras aplicaciones en el dispositivo puedan interactuar con este ya sea una *activity, service* o cualquier componente de la aplicación que tenga este valor. En la Figura 30 se puede observar el nombre de algunas *activities* y resaltado en color rojo el valor del atributo `android:exported` establecido como `true`.

Analizando los nombres de estas *activities*, resulta interesante probar si es posible interactuar con algunas de estas que se encuentran expuestas.
Usando la herramienta ADB se llama a la *activity* `PostLogin` con el comando:

```
adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.PostLogin
```

Después de ejecutar el comando, se logró acceder a la *activity* `PostLogin`, evadiendo la autenticación e ingresando a la sección que permite realizar transferencias y cambiar la contraseña del usuario, como se puede observar en la Figura 31.

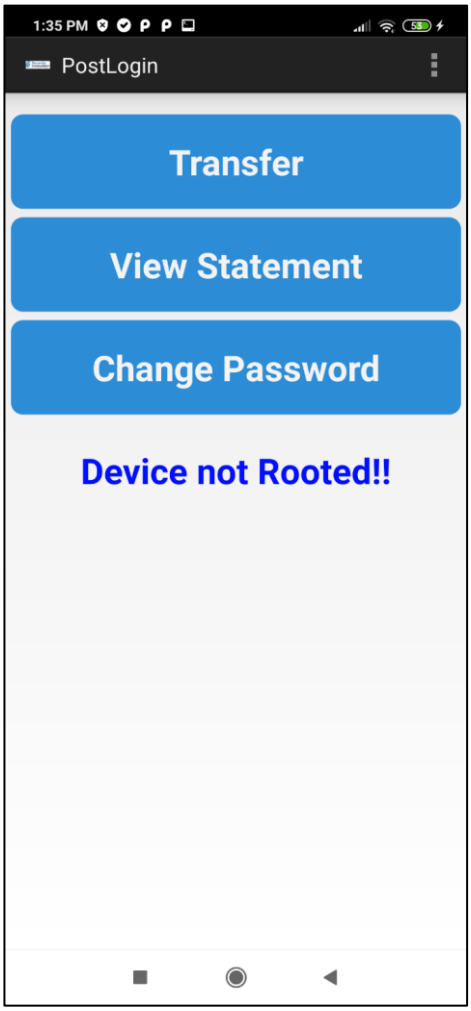


Figura 31 – Acceso a la activity PostLogin

De igual manera ocurre con las *activities:* `DoTransfer` *y* `ViewStatement` lo que permite a un atacante acceder a funciones críticas evadiendo la autenticación.

Broadcast Receiver expuestos

Con la herramienta jadx-gui se analizó el archivo *AndroidManifest.xml* y se identificó el componente *Broadcast Receiver* expuesto como se puede observar en la Figura 32, además, no requiere de algún permiso para que una aplicación externa pueda enviar emisiones a este componente.

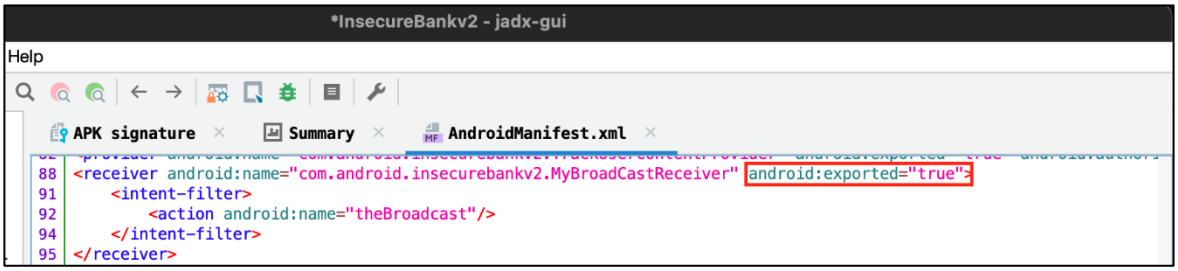


Figura 32 – Análisis BroadcastReceiver

Para analizar este elemento se utilizó la herramienta Drozer, con ayuda de esta herramienta se logró obtener más información de este componente con el siguiente comando:

```
dz> run app.broadcast.info -a com.android.insecurebankv2 -i
```

Con el que se obtuvo la siguiente información presentada en la Figura 33.

```
drozer Console (v2.4.4)
dz> run app.broadcast.info -a com.android.insecurebankv2 -i
Package: com.android.insecurebankv2
  com.android.insecurebankv2.MyBroadCastReceiver
    Intent Filter:
      Actions:
        - theBroadcast
    Permission: null
```

Figura 33 – Análisis BroadcastReceiver

El nombre del componente es *MyBroadCastReceiver* y se ha exportado con permisos nulos, lo que significa que no se requieren permisos para enviar emisiones al receptor. Se ha establecido un *intent filter* para el componente con la acción de transmisión, el receptor se suscribe a "*theBroadcast*".

Para comprender el funcionamiento del componente es importante inspeccionar la clase donde se implementa la lógica de este, buscando la clase BroadcastReceiver, tal como se menciona en el punto 8. Interacción con la plataforma. La implementación de dicho componente se detectó el archivo *MyBroadcastReceiver.java*, tal como se puede observar en la Figura 34.

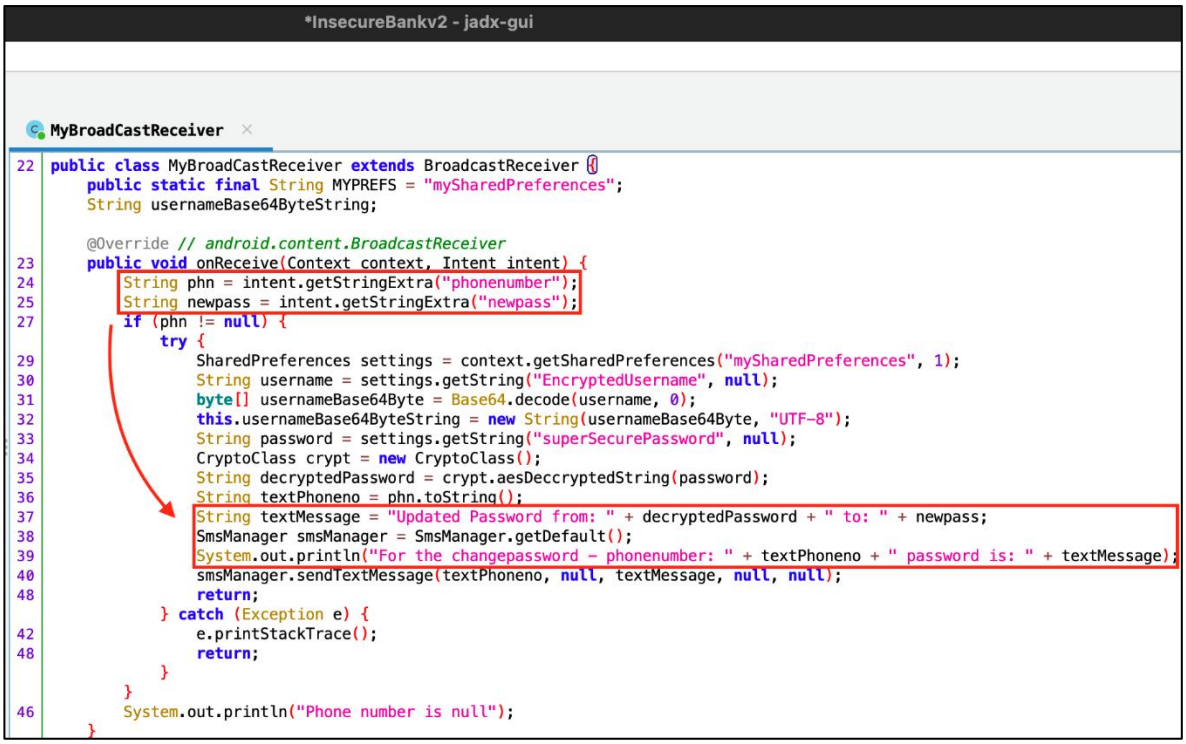


Figura 34 – Análisis código fuente BroadcastReceiver

En el código se puede observar que el *intent filter* se utiliza para obtener dos parámetros: *phonenumber* y *newpass*. Las credenciales en el archivo *mySharedPreferences* se recuperan y descifran para enviarse en una cadena de texto

a través de SMS junto con la nueva contraseña. A partir de la información obtenida del código fuente es posible invocar el componente con el siguiente comando de la herramienta Drozer con los parámetros *phonenumber* y *newpass*:

```
dz> run app.broadcast.send -action theBroadcast -extra string phonenumber 2216370887 -extra string newpass supersecurepass
```

Como resultado en el teléfono se obtiene un mensaje con la contraseña anterior y la nueva contraseña del usuario, como se observa en la Figura 35.

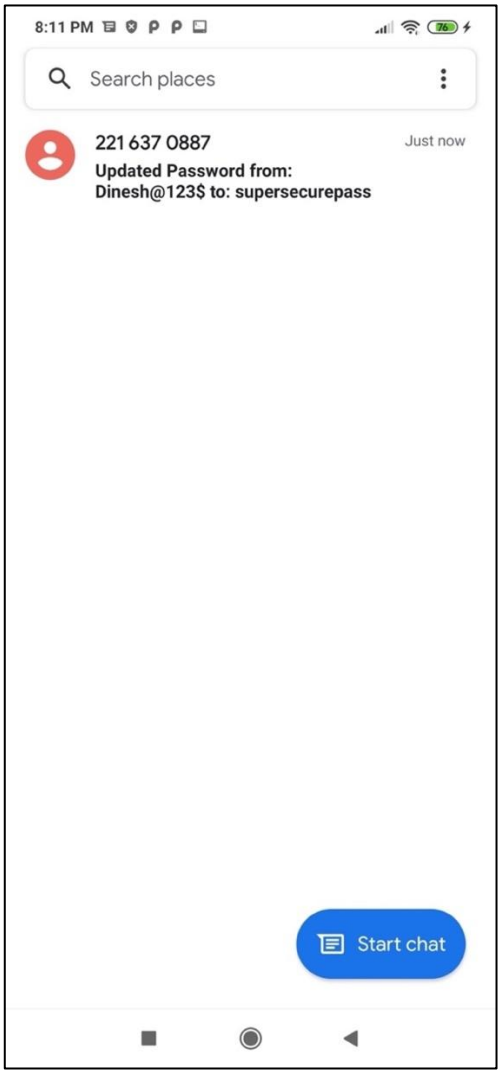


Figura 35 – SMS con la nueva contraseña del usuario

Esto demuestra como una aplicación maliciosa puede interactuar con los componentes no protegidos sin ninguna restricción y cambiar la contraseña del usuario.

Content provider expuestos

Por lo general, son dos las situaciones en las que se trabaja con proveedores de contenido: cuando se necesita acceder a un proveedor de contenido de otra aplicación o crear un proveedor de contenido para compartir datos a otra aplicación.

Si la información de entrada no se valida correctamente o no se configuran los permisos de lectura y escritura, el componente puede ser propenso a fuga de información o inyección SQL mientras otras aplicaciones interactúan con él.

Al igual que los elementos anteriores se identificó el componente *Content Provider*, con ayuda de la herramienta jadx-gui, en la Figura 36 se puede apreciar este elemento.

Figura 36 – Content Provider en el archivo AndroidManifest.xml

Con ayuda de Drozer se analizó el componente para obtener más información con el comando:

```
dz> run app.provider.info -a com.android.insecurebankv2
```

El resultado obtenido se puede observar en la Figura 37.

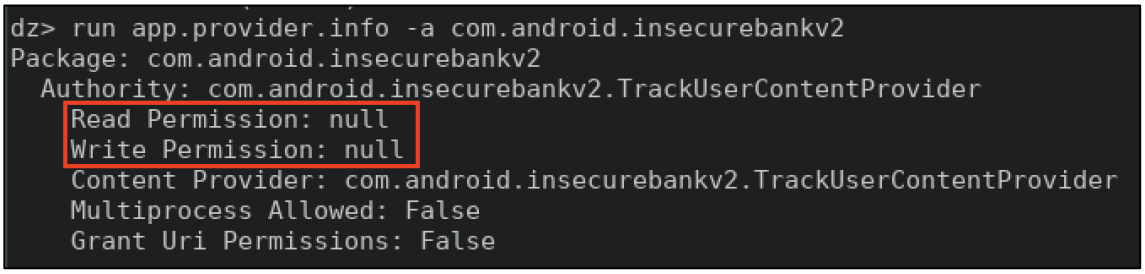


Figura 37 – Análisis Content Provider con la herramienta Drozer

Los permisos de lectura y escritura se establecen en `null`, significa que no hay nada que impida consultar los datos a través del componente. Para obtener el contenido, nuevamente se usará la herramienta Drozer para identificar cualquier URI accesible disponible a través del componente con el siguiente comando:

dz> run scanner.provider.finduris -a com.android.insecurebankv2

Se descubrió una URI accesible como se puede observar en la Figura 38.

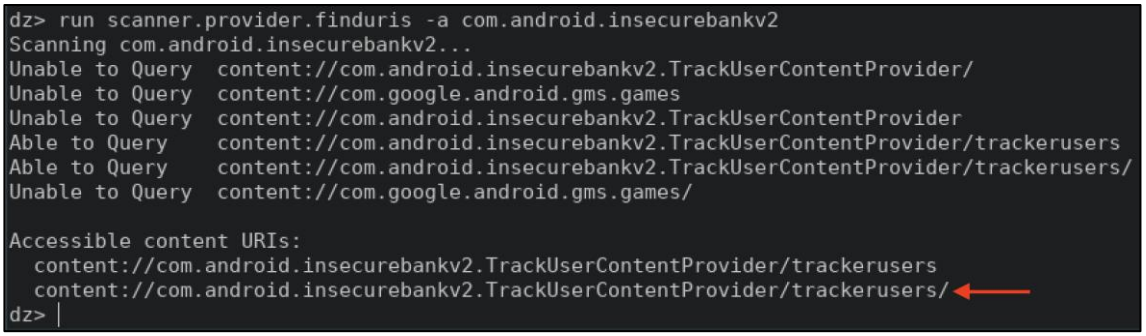


Figura 38 – Análisis de URIs accesibles

Es posible consultar el contenido del *Content Provider* usando la URI identificada. La consulta se realiza con el siguiente comando:

dz> run app.provider.query content://com.android.insecurebankv2.TrackUserContentProvider/trackusers/

Esto devuelve la tabla presentada en la Figura 39 con los ID y nombres de usuario de la base de datos que registra el inicio de sesión de los usuarios en la aplicación.

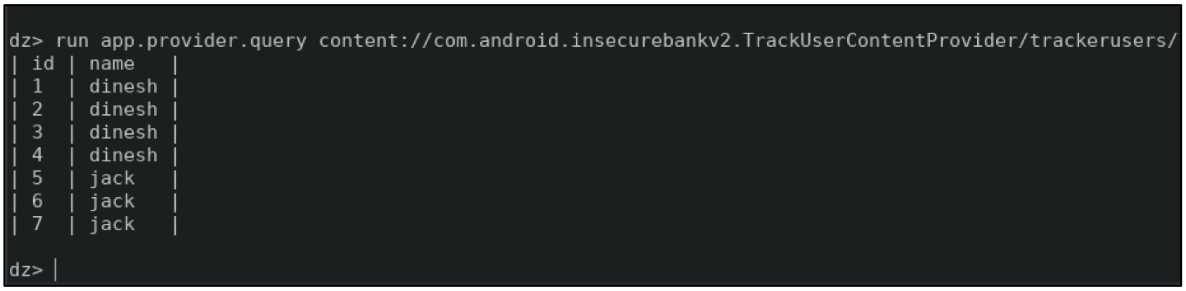


Figura 39 – Usuarios obtenidos a través del Content Provider

Dado que no se protege el componente, cualquier aplicación instalada en el dispositivo puede consultar la información. Lo que provoca una fuga de datos.

Deficiencias en Web View

La aplicación InsecureBankv2 permite visualizar los estados de cuenta en un *WebView*. Analizando el código de la *activity ViewStatements* se puede determinar que se carga un archivo *.html* donde se almacena la información del estado de cuenta del usuario, además, se observa que el elemento *WebView* tiene habilitado JavaScript con el método `setJavaScriptEnable(true),` esto se puede observar en la Figura 40.

com.android.insecurebankv2.ViewStatement

```
package com.android.insecurebankv2;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.view.Menu;
import android.view.MenuItem;
import android.webkit.WebChromeClient;
import android.webkit.WebView;
import android.widget.Toast;
import java.io.File;

21 public class ViewStatement extends Activity {
    String uname;

    /* access modifiers changed from: protected */
22  public void onCreate(Bundle savedInstanceState) {
23      super.onCreate(savedInstanceState);
24      setContentView(R.layout.activity_view_statement);
26      this.uname = getIntent().getStringExtra("uname");
29      File fileToCheck = new File(Environment.getExternalStorageDirectory(), "Statements_" + this.uname + ".html");
30      System.out.println(fileToCheck.toString());
31      if (fileToCheck.exists()) {
34          WebView mWebView = (WebView) findViewById(R.id.webView1);
36          mWebView.loadUrl("file://" + Environment.getExternalStorageDirectory() + "/Statements_" + this.uname + ".html");
37          mWebView.getSettings().setJavaScriptEnabled(true);
38          mWebView.getSettings().setSaveFormData(true);
39          mWebView.getSettings().setBuiltInZoomControls(true);
40          mWebView.setWebViewClient(new MyWebViewClient());
42          mWebView.setWebChromeClient(new WebChromeClient());
52          return;
        }
46      startActivity(new Intent(this, PostLogin.class));
47      Toast.makeText(this, "Statement does not Exist!!", 1).show();
    }
```

Figura 40 – Código fuente de la clase ViewStatement

Analizando el almacenamiento externo del dispositivo se observó el archivo *Statements_jack.html* donde se guarda la información del usuario, en la Figura 41 se puede observar el contenido del archivo.

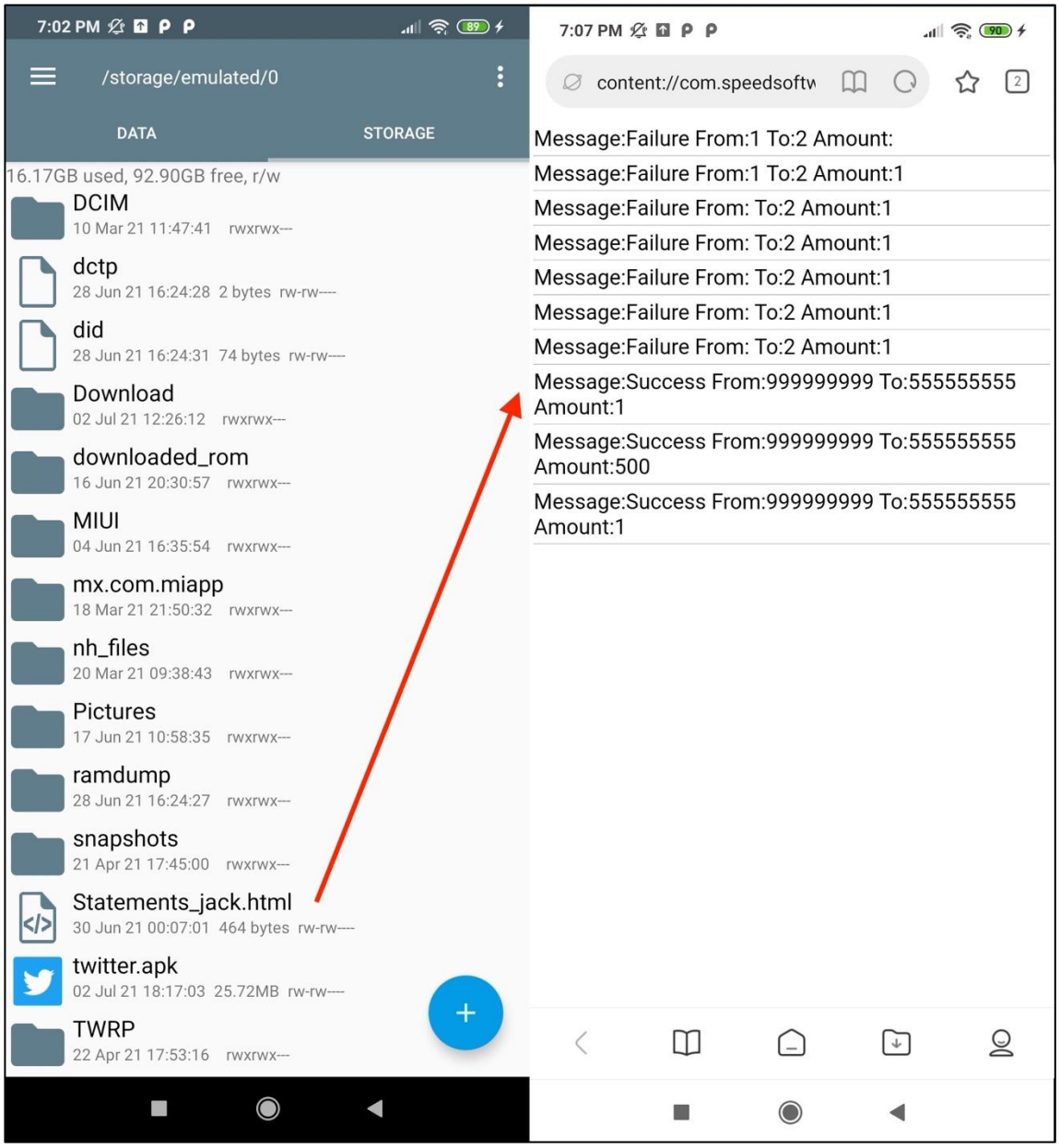


Figura 41 – Archivo en almacenamiento externo

Dado que el archivo se carga desde el almacenamiento externo, cualquier aplicación puede leer y escribir sobre él. Para demostrar la vulnerabilidad se agregó código JavaScript que se ejecutará cuando se cargue el archivo. A modo de demostración

se presenta en la Figura 42 donde se ejecuta el código JavaScript en el *WebView* lo que abre la puerta a ataques del *tipo Cross-Site scripting* (XSS).

Figura 42 – XSS en WebView

Almacenamiento inseguro

Se considera almacenamiento inseguro a cualquier almacén de datos que guarde información personal o información considerada sensible en el contexto de la aplicación en texto claro en el dispositivo, un actor malicioso puede utilizar esta información para ataques como ingeniería social, secuestro de la cuenta, por mencionar algunos.

Para identificar si la aplicación hace uso del almacenamiento externo se analizó el archivo *AndroidManifest.xml* en busca de permisos de escritura o lectura `android.permission.WRITE_EXTERNAL_STORAGE` y `android.permission.READ_EXTERNAL_STORAGE`
Además, se identificó en el código fuente las palabras clave y las llamadas de clases y funciones como que se utilizan para almacenar datos como: `SharedPreferences`, `FileOutPutStream`, `getExternal*`, `getWritableDatabase`, `etc.` Esto se describe de manera detallada en la metodología propuesta 3. Análisis estático.

Analizando el código fuente descompilado con la herramienta jadx-gui, se identificó el uso `SharedPreferences` en algunas clases de la aplicación, esto se puede observar en la Figura 43.

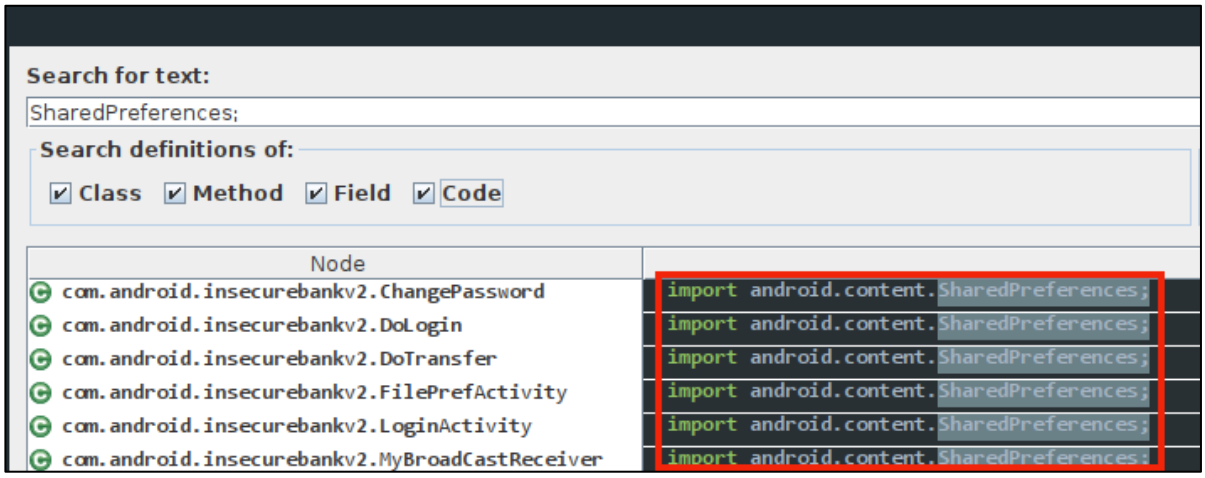


Figura 43 – Uso de SharedPrefences

Buscando el método donde se usa esta clase se puede observar que se almacena en el archivo *mySharedPreferences* el usuario y contraseña. En la Figura 44 se puede observar fragmento del código donde se almacenan las credenciales.

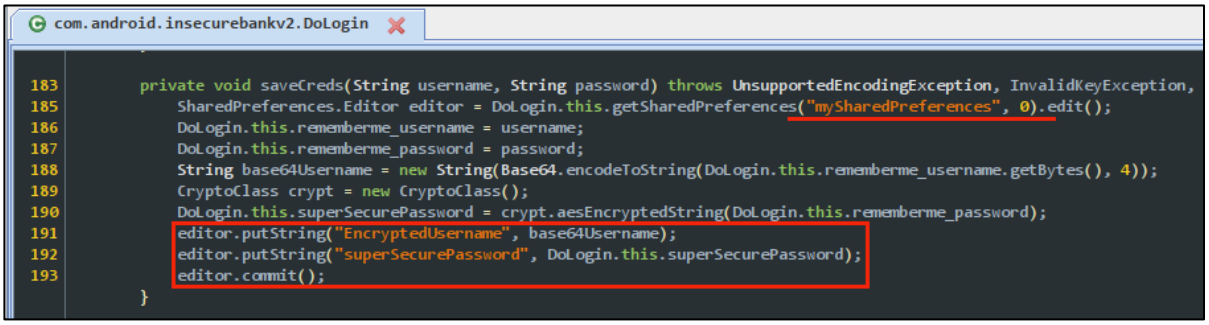


Figura 44 – Método saveCreds() de la clase DoLogin

Con un explorador de archivos, en este caso se usó la aplicación root explorer para navegar en el almacenamiento de interno de la aplicación (se requiere tener acceso root en el dispositivo móvil). El directorio de archivos de la aplicación se encuentra en el almacenamiento interno */data/data/com.android.insecurebankv2/shared_prefs* donde se localizaron las credenciales del usuario codificadas en base64 en el archivo *mySharedPreferences.xml* el contenido del archivo se puede apreciar en la Figura 45.

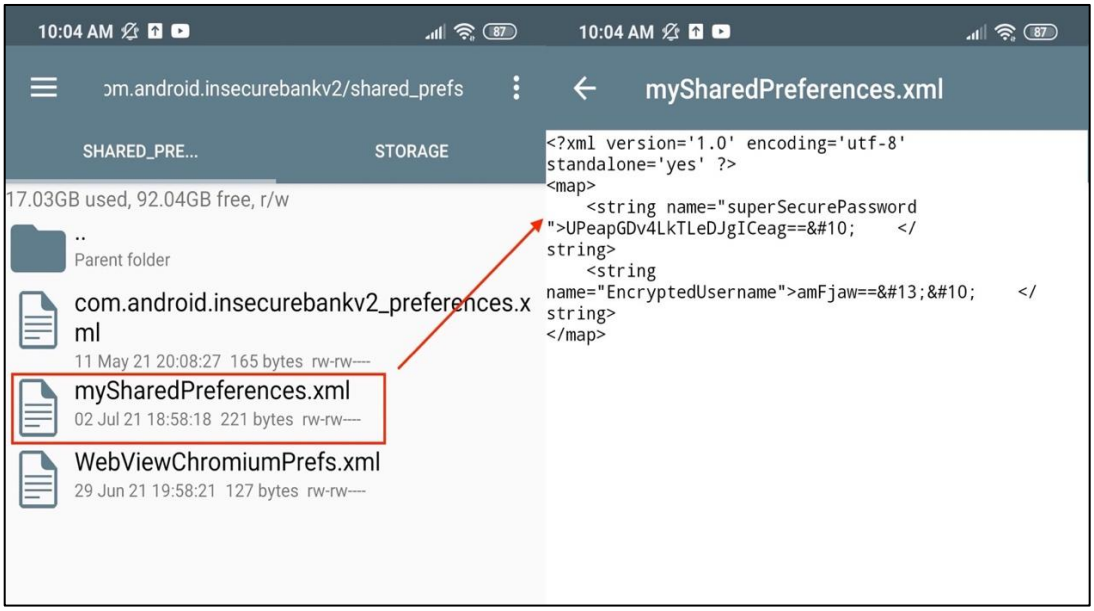


Figura 45 – Credenciales del usuario almacenadas en el archivo mySharedPreferences.xml

Al analizar el código fuente se observa que el nombre del usuario se codifica en base64 y se guarda. Con la herramienta CyberChef se puede decodificar fácilmente el nombre del usuario, esto se puede observar en la Figura 46.

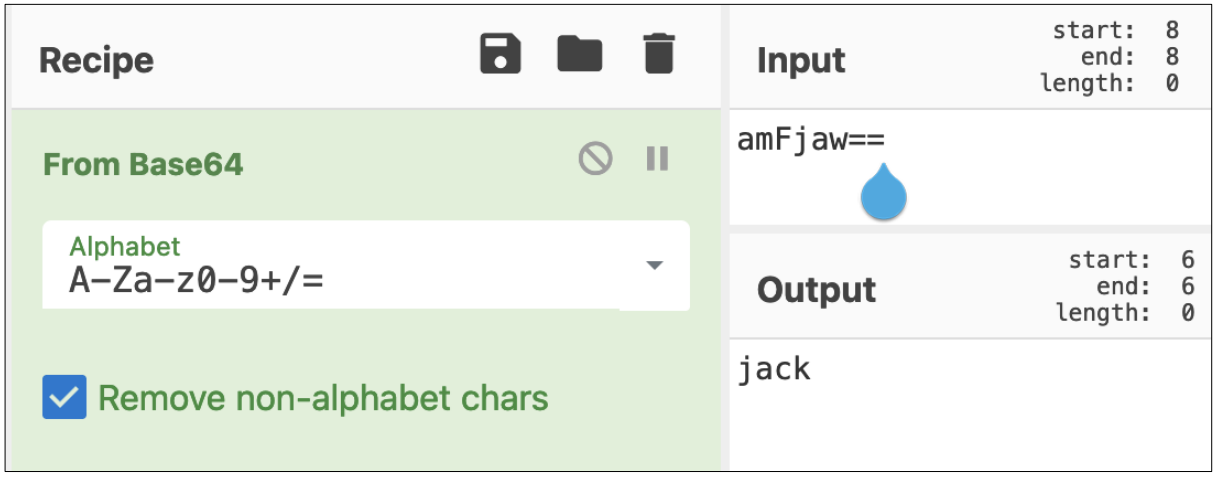


Figura 46 – Nombre de usuario decodificado

Observando el código mostrado anteriormente en la Figura 44 se puede ver que la contraseña se cifra con el método `aesEncryptedString()` de la clase `cyptoClass`. Analizando esta clase se puede saber que se utilizó el algoritmo AES en modo *Cipher Block Chaining* (CBC) que utiliza un vector de inicialización (IV) y una clave, ambas se encuentran embebidas en el código fuente, mostradas en la Figura 47.

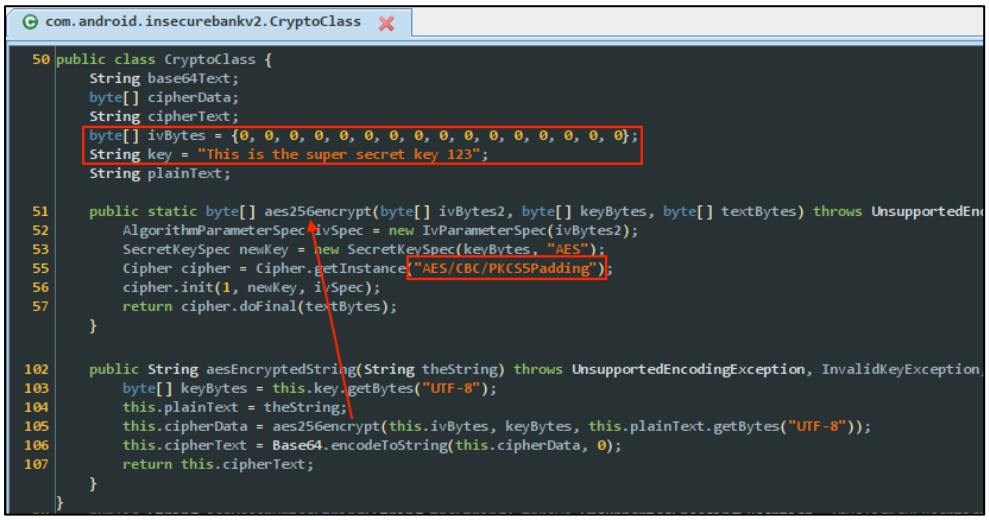


Figura 47 – Clave de cifrado y vector de inicialización

Conociendo la clave y el vector de inicialización se puede obtener la contraseña en texto plano. Para este caso se usó un script en Python que se muestra en la Figura 48, pero se puede utilizar la herramienta CyberChef o bien programar una función en otro lenguaje de programación.

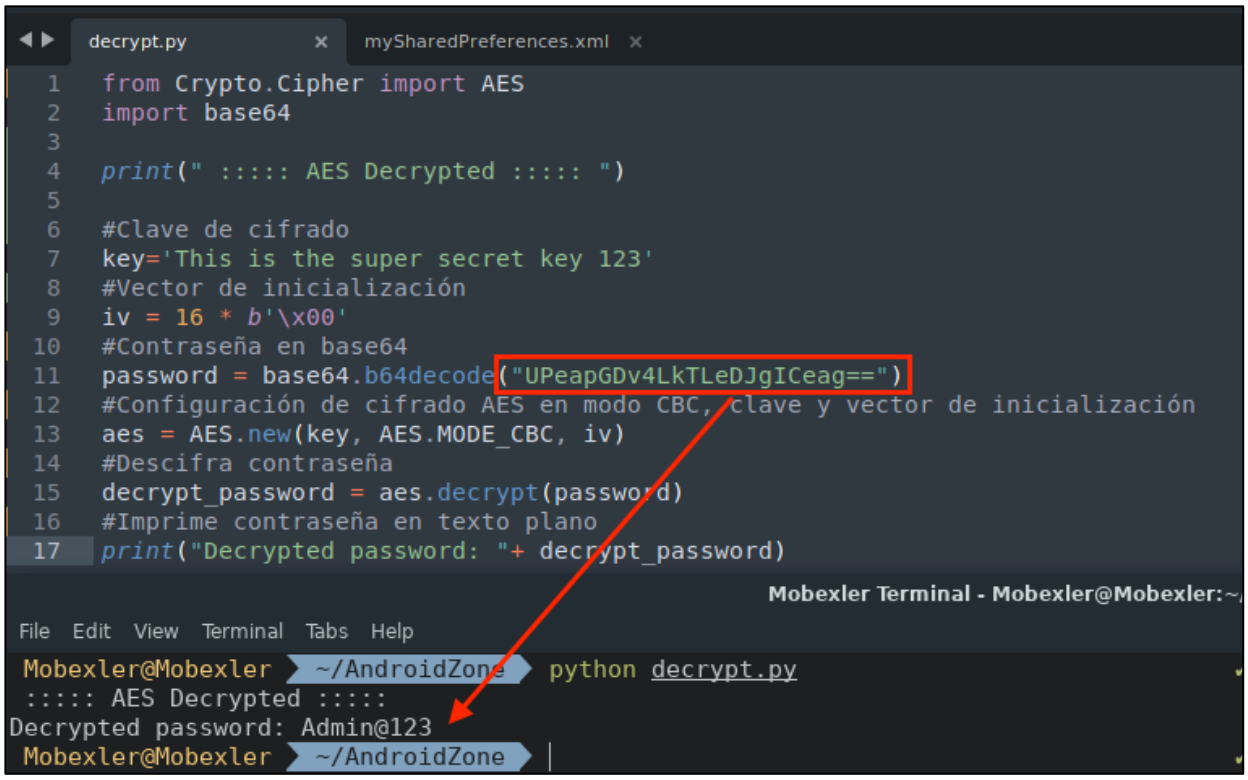


Figura 48 – Script para descifrar la contraseña

Funciones de depuración activas

Las funciones de depuración son muy útiles durante la fase de desarrollo, ayudan al desarrollador a identificar errores al momento de ejecución, sin embargo, en algunos casos la información desplegada por estas funciones podría contener datos sensibles como credenciales de usuario, cookies, tokens de sesión etc.

La herramienta MobSF reporta este tipo de deficiencias en las aplicaciones en la sección FILE ANALYSIS del reporte de resultados, sin embargo, en el reporte de la aplicación objeto de estudio no arroja información en esta sección por lo que se trata de un falso negativo.

Para analizar la aplicación en busca de funciones de depuración se realizó con la herramienta jadx-gui con la opción “Búsqueda de texto” presentada en la Figura 49.

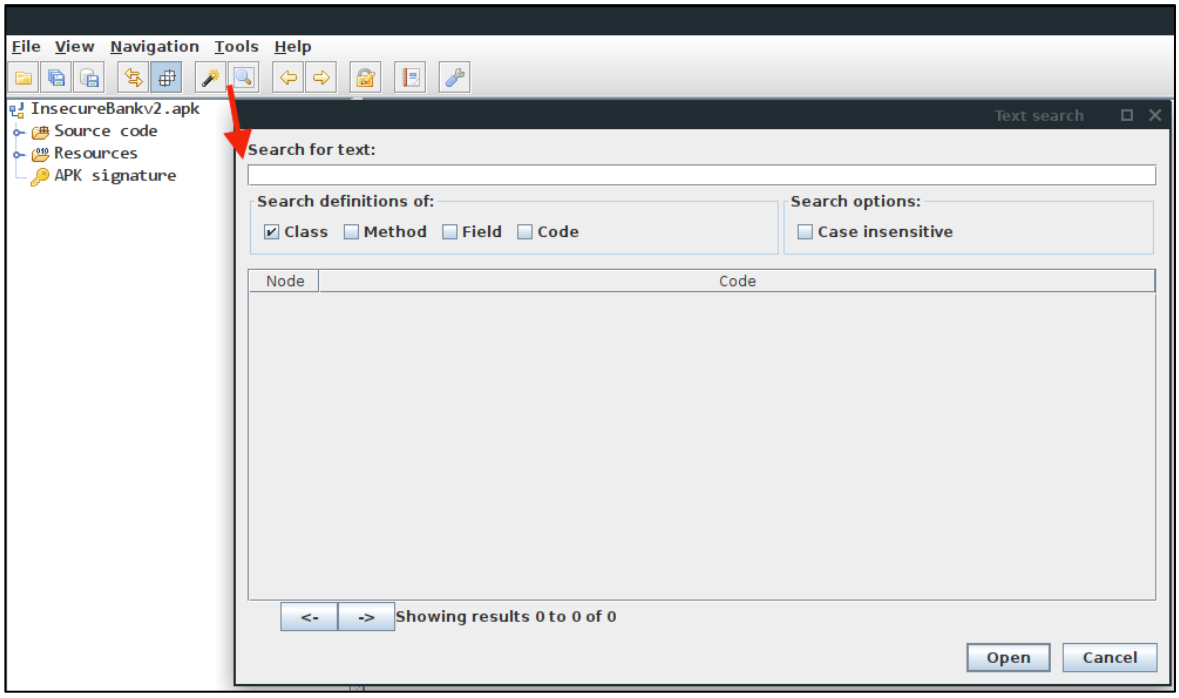


Figura 49 – Búsqueda de texto de la herramienta jadx-gui

Las funciones depuración en la plataforma Android son: `Log.v()`, `Log.d()`, `Log.i()`, `Log.w()`, `Log.e()` con ayuda de la herramienta jadx-gui se localizó una instancia de una de estas funciones en la clase *DoLogin*, esto se describe en el punto 3. Análisis estático del método propuesto. En la Figura 50 se puede observar el uso de las funciones de depuración previamente mencionadas.

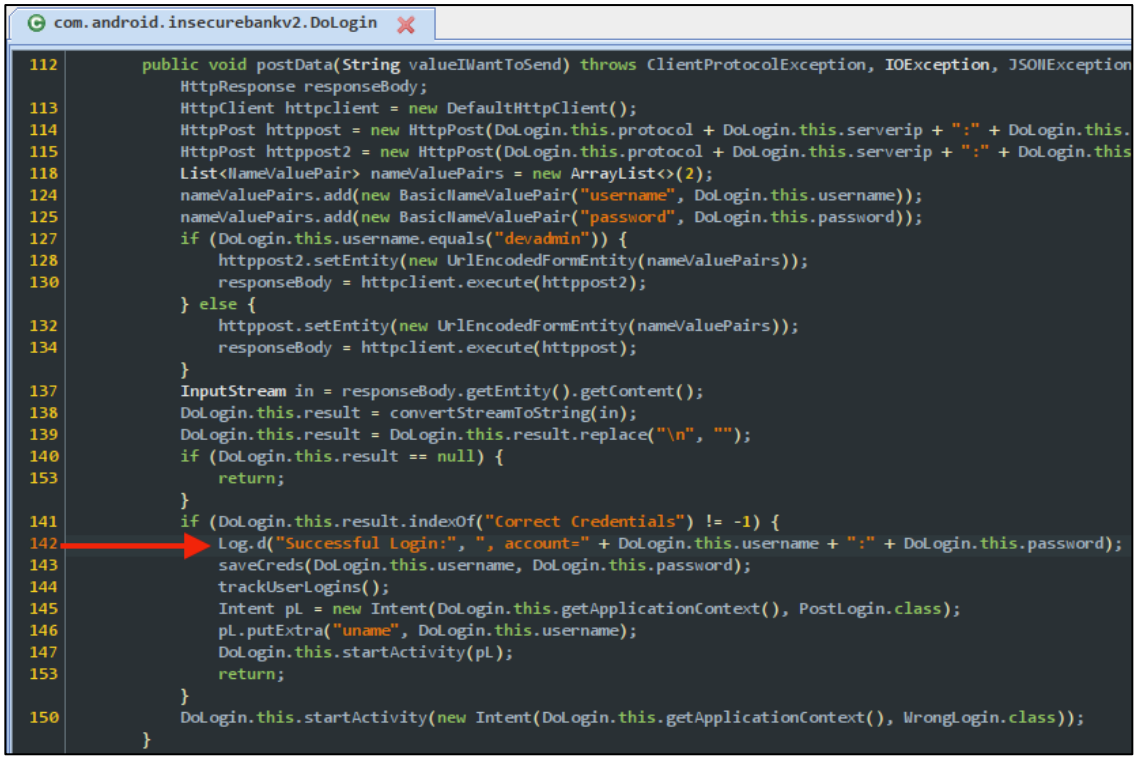


Figura 50 – Análisis de la clase DoLogin

Dada la nomenclatura de las variables se puede observar que esta función incluye las credenciales cuando el usuario las ha ingresado correctamente. Con ayuda de la herramienta Pidcat se analiza la información en la consola de dispositivo Android presentada en la Figura 51 permitiendo observar la información que se despliega.

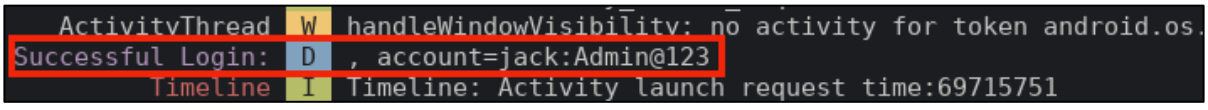


Figura 51 – Credenciales de usuario en logs de la consola del dispositivo

Uso de protocolos sin cifrado

Para probar si las comunicaciones entre la aplicación y el servidor se realizan por un canal cifrado, es necesario interceptar esta comunicación, esto se describe en el punto 7. Seguridad en la red del método propuesto.

La herramienta Burp Suite permite interceptar, inspeccionar y modificar las solicitudes en ambas direcciones entre un cliente y el servidor. Para esto es necesario una serie de configuraciones que se puede encontrar a detalle en las secciones Configuración del dispositivo Android y Configuración del dispositivo iOS de acuerdo con el tipo de dispositivo.

En la Figura 52 se puede observar la dirección IP y puerto a la escucha del proxy de la herramienta.

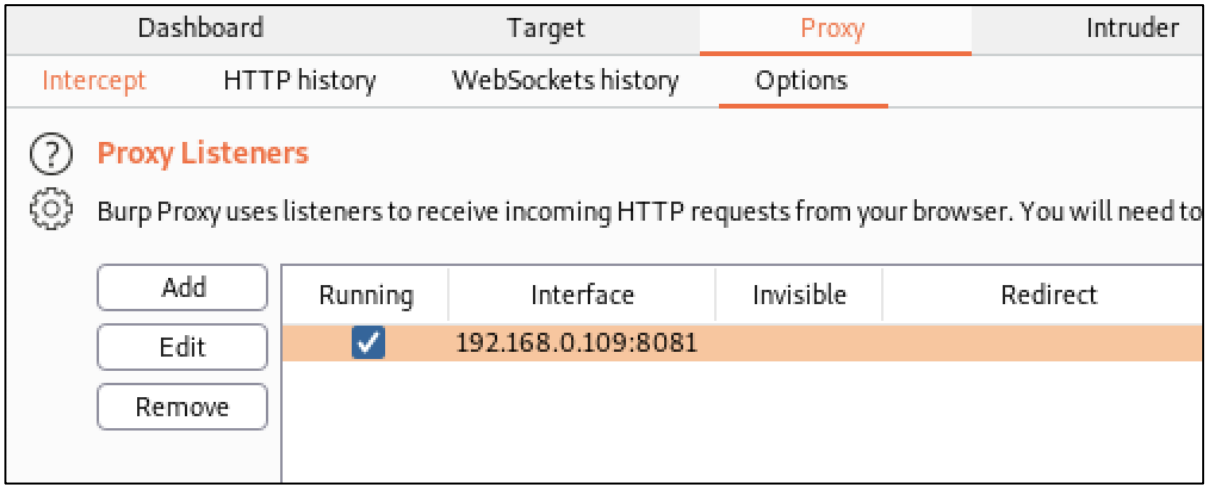


Figura 52 – Dirección IP y puerto del proxy Burp Suite

En el dispositivo móvil se configura la dirección IP y puerto del proxy en la sección ajustes de red tal como se presenta en la Figura 53.

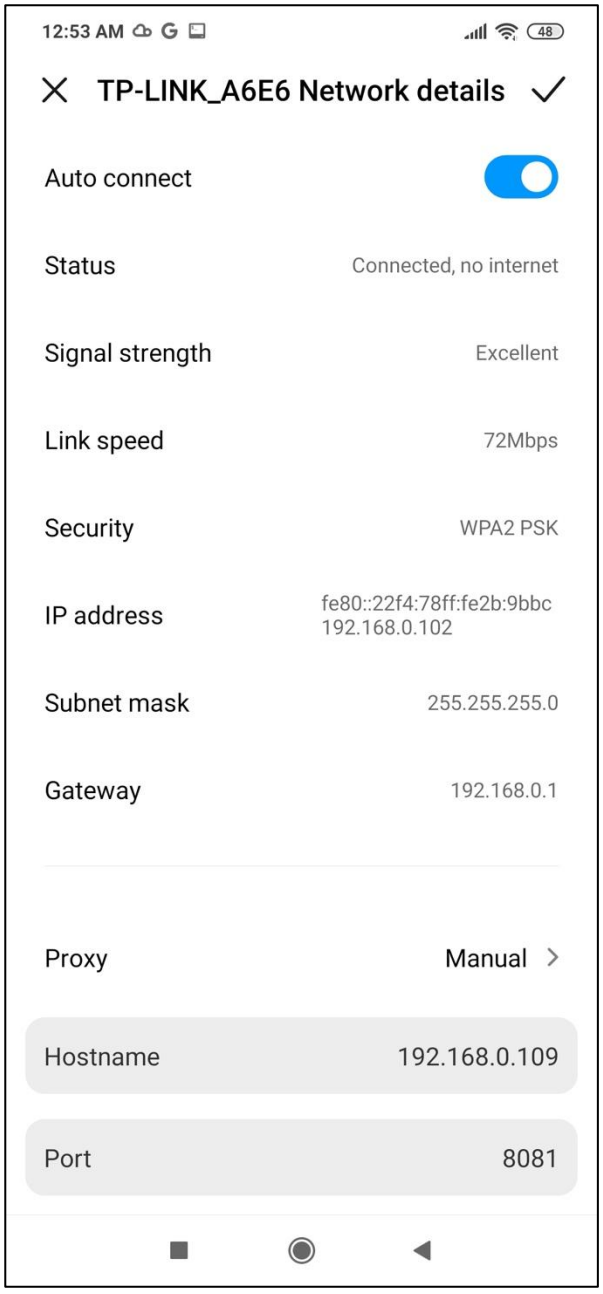


Figura 53 – Configuración proxy en el dispositivo Android

Después de guardar la configuración en el dispositivo es posible observar las solicitudes entre la aplicación y el servidor, en caso de presentarse un error en la conexión de red quiere decir que se implementa el mecanismo de seguridad *SSLPinning* para más detalle consultar los puntos: 4. Protecciones y controles del lado del cliente y 5. Análisis dinámico del método de análisis propuesto.

En la Figura 54 se puede observar que la comunicación se realiza mediante el protocolo HTTP sin una capa de cifrado.

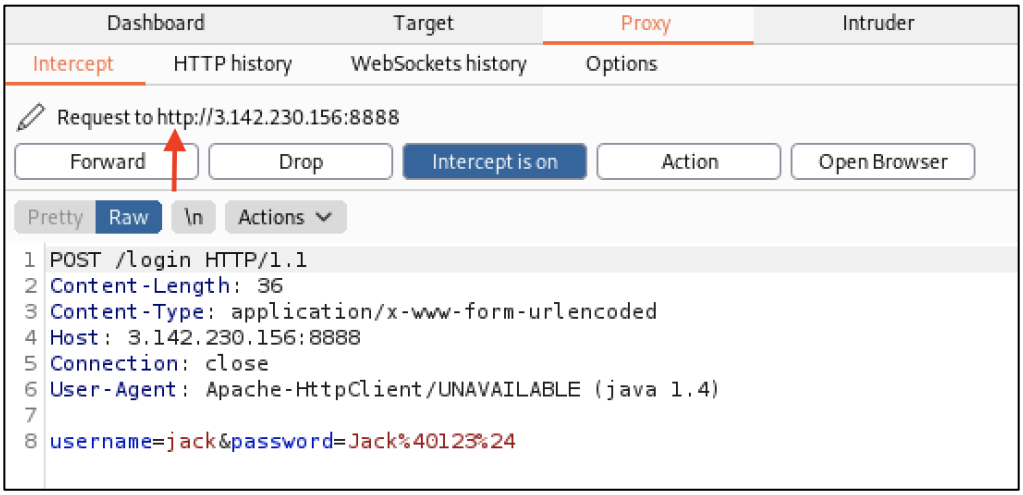


Figura 54 – Comunicación entre la aplicación y el servidor

Un elemento que tenga acceso al canal de comunicaciones ya sea porque está en el mismo segmento de red, o por ser un dispositivo intermedio puede interceptar la información intercambiada.

Deficiencias en AndroidManifest

Algunos de los atributos definidos en el *AndroidManifest.xml* están configurados con los valores por defecto por ejemplo algunos de ellos son: el ícono de la aplicación, el tema usado, versión de la aplicación, etc. Es importante que el desarrollador determine qué atributos son estrictamente necesarios para el buen funcionamiento de la aplicación de acuerdo con la lógica de negocio de esta.

El atributo *android:allowBackup* permite al sistema operativo realizar una copia de seguridad o restauración de la aplicación, incluso mediante una copia de seguridad del sistema completo. El valor predeterminado de este atributo es *true*.

A través del análisis estático a la aplicación se identificó que esta tiene el valor *true* en el atributo *android:allowBackup* en al archivo *AndroidManifest.xml*, esta situación se puede observar en la Figura 55.

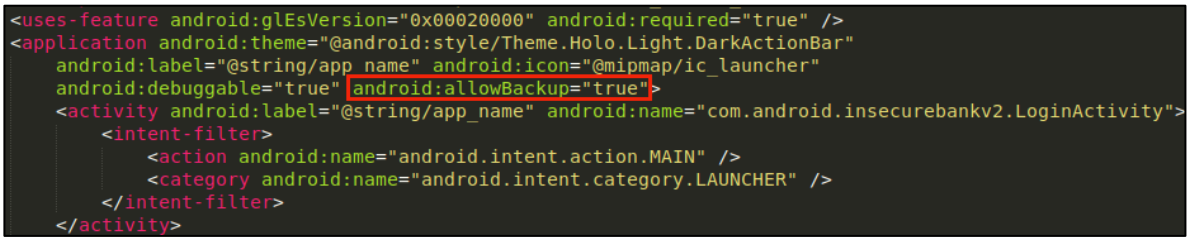


Figura 55 – Atributo allowBackup

Esto permite a un atacante realizar un respaldo de los datos de la aplicación a través de la interfaz ADB si el dispositivo tiene habilitadas las opciones de desarrollador y aunque el dispositivo no se encuentre *rooteado*.

Con ayuda de la herramienta ADB fue posible realizar un respaldo de los datos de la aplicación con el comando:

```
adb backup com.android.insecurebankv2
```

Se obtuvo el archivo *backup.ab* como se puede observar en la Figura 56

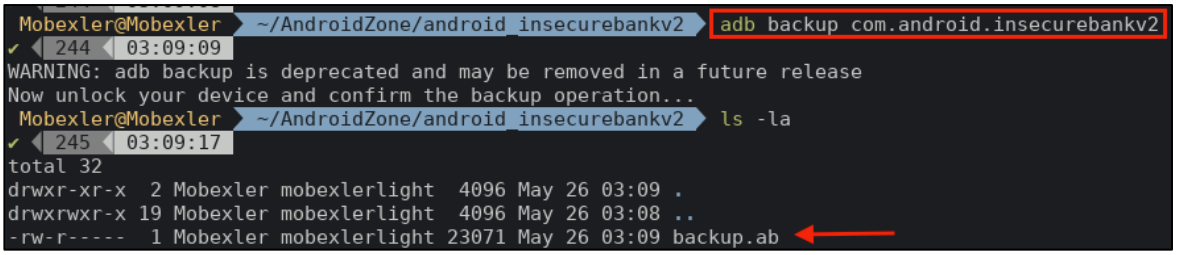


Figura 56 – Respaldo de la aplicación Inscurebankv2

Para obtener la información del archivo es necesario realizar el desempaquetado. Esto se realizó con ayuda de la herramienta Android backup extractor para convertir el archivo de respaldo en un archivo *.tar*, con el comando: *java -jar abe.jar unpack backup.ab application-data.tar <password>*

En la Figura 57 se puede observar el desempaquetado de archivo de respaldo.

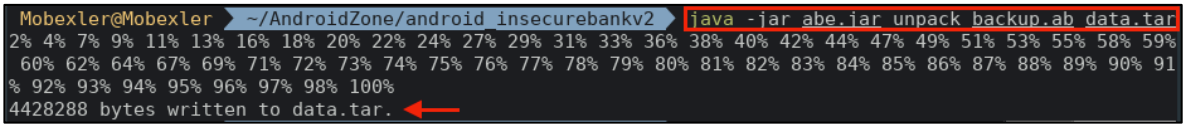


Figura 57 – Desempaquetado del archivo de respaldo backup.ab

Como resultado se obtuvo el archivo *data.tar,* para descomprimir este archivo se realizó con el comando:

tar xvf application-data.tar

En la Figura 58 se puede observar el listado de archivos obtenido gracias al atributo *android:allowBackup* en el archivo *AndroidManifest.xml.*

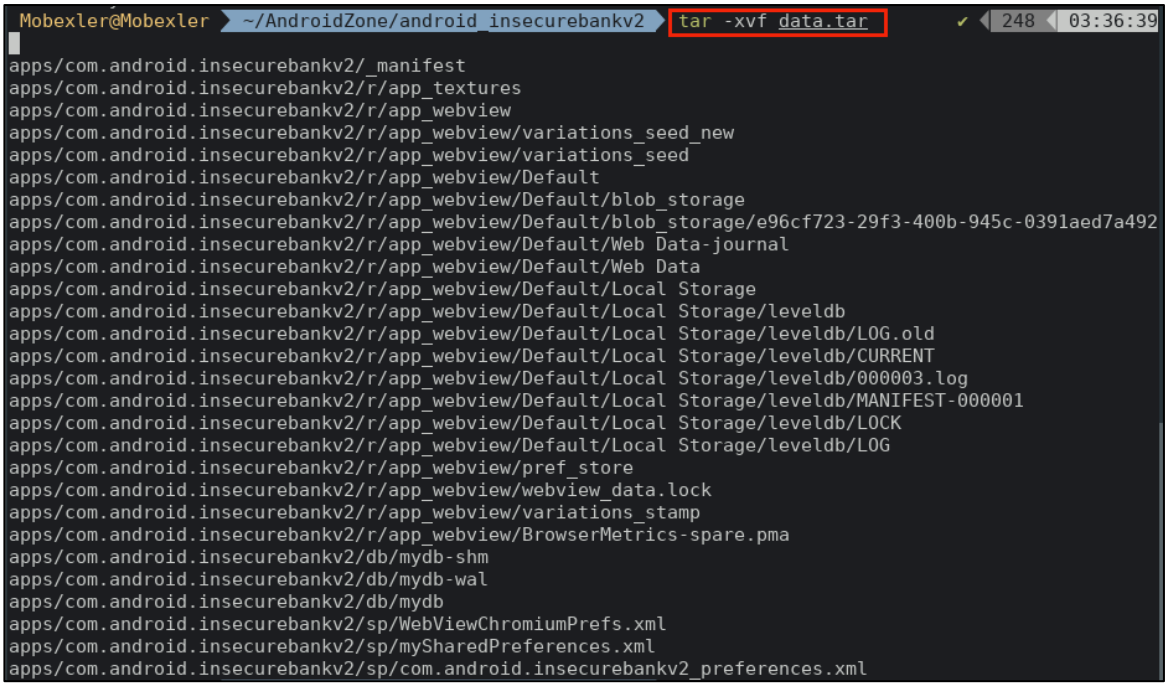


Figura 58 – Contenido del archivo de respaldo data.tar

Enumeración de usuarios

La enumeración de usuarios permite identificar nombres de usuario válidos en una aplicación a través de mensajes de error diferenciados o respuestas específicas del sistema durante intentos de autenticación, registro o recuperación de contraseñas. La descripción detallada de esta prueba se describe en el punto 5. Análisis dinámico del método propuesto.

Con ayuda de la herramienta Burp Suite se evaluaron los *web services* que sirven a la aplicación. En especificó el proceso de inicio de sesión. Para esto es necesario configurar el proxy en el dispositivo en los ajustes de red del dispositivo móvil y en el apartado proxy se ingresa la dirección IP y puerto a la escucha, como se muestra en la Figura 53.

Se interceptó la petición que genera la aplicación al iniciar sesión, misma que se presenta en la Figura 59, en esta petición se puede observar el parámetro *username* con un nombre de usuario inexistente y en la respuesta se puede ver el mensaje indicando que el nombre de usuario no existe.

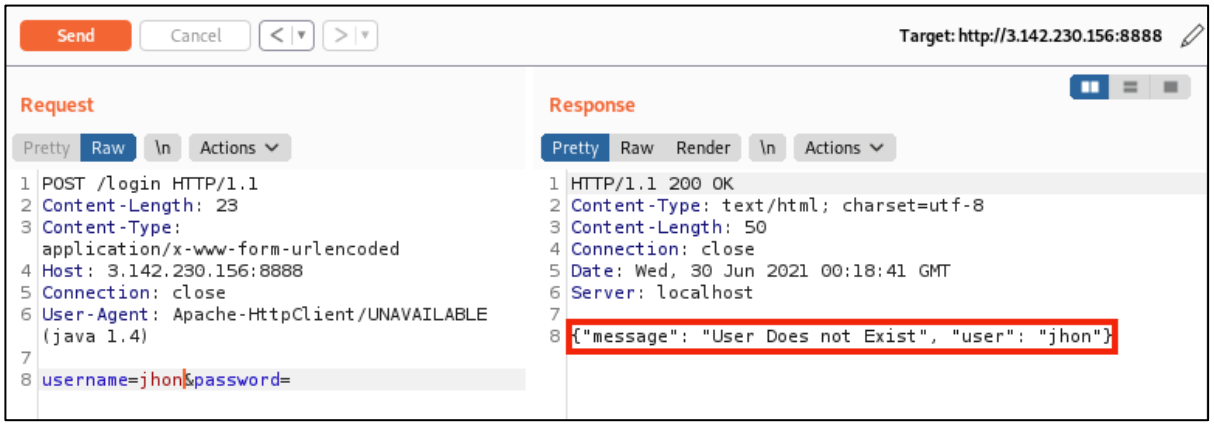


Figura 59 – Mensaje del servidor cuando un usuario no existe

Por otro lado, si se envía un usuario válido el servidor responde con un mensaje diferente, indicando que la contraseña es incorrecta, esto se observa en la Figura 60.

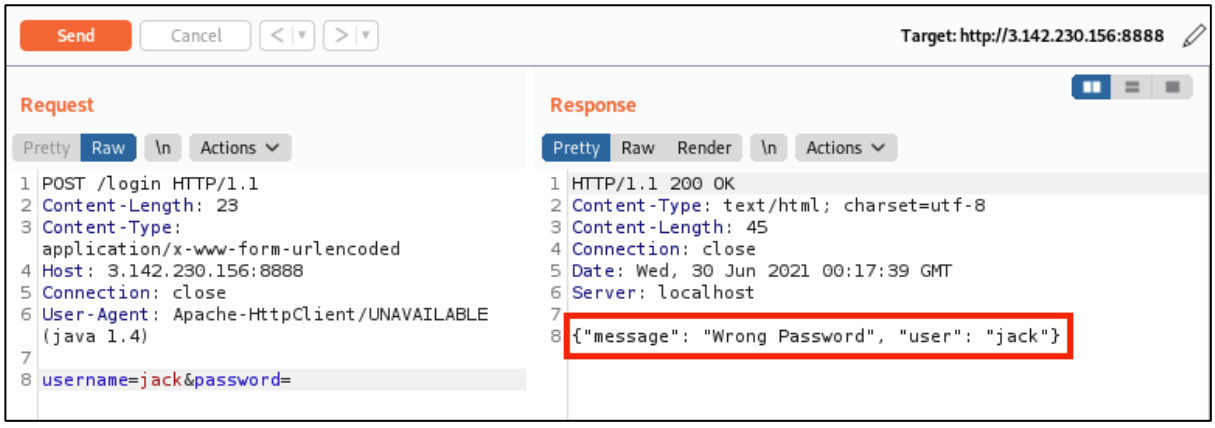


Figura 60 – Mensaje del servidor cuando un usuario existe

Para automatizar este proceso se envió una lista de posibles nombres de usuario a los *web services* que sirven a la aplicación, con la herramienta *intruder*, incluida en Burp Suite. Para hacer uso de ella basta con hacer clic derecho sobre la petición y presionar la opción “Send to *Intruder*” como se observa en Figura 61.

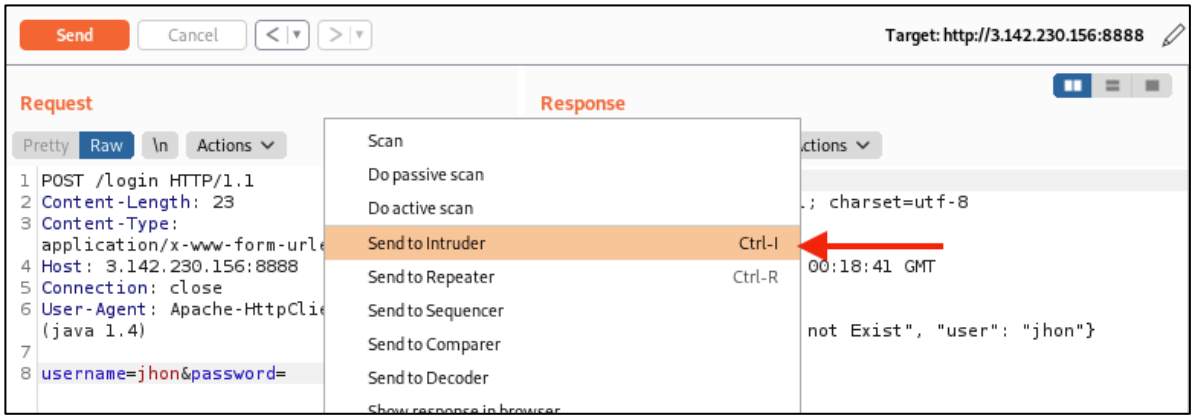


Figura 61 – Uso de la herramienta intruder de Burp Suite

Después, es necesario indicar la posición donde se sustituirán la lista de posibles nombres de usuarios, esta se marca con dos caracteres §§. El tipo de ataque se establece en *Sniper* como se ve en la Figura 62, este tipo de ataque usa un solo conjunto de *payloads* y apunta a cada posición por turno y coloca cada *payload* en esa posición a su vez.

Figura 62 – Configuración intruder de Burp Suite

Por último, se añade una lista de posibles nombres de usuario que se enviarán al servidor, como se muestra en la Figura 63.

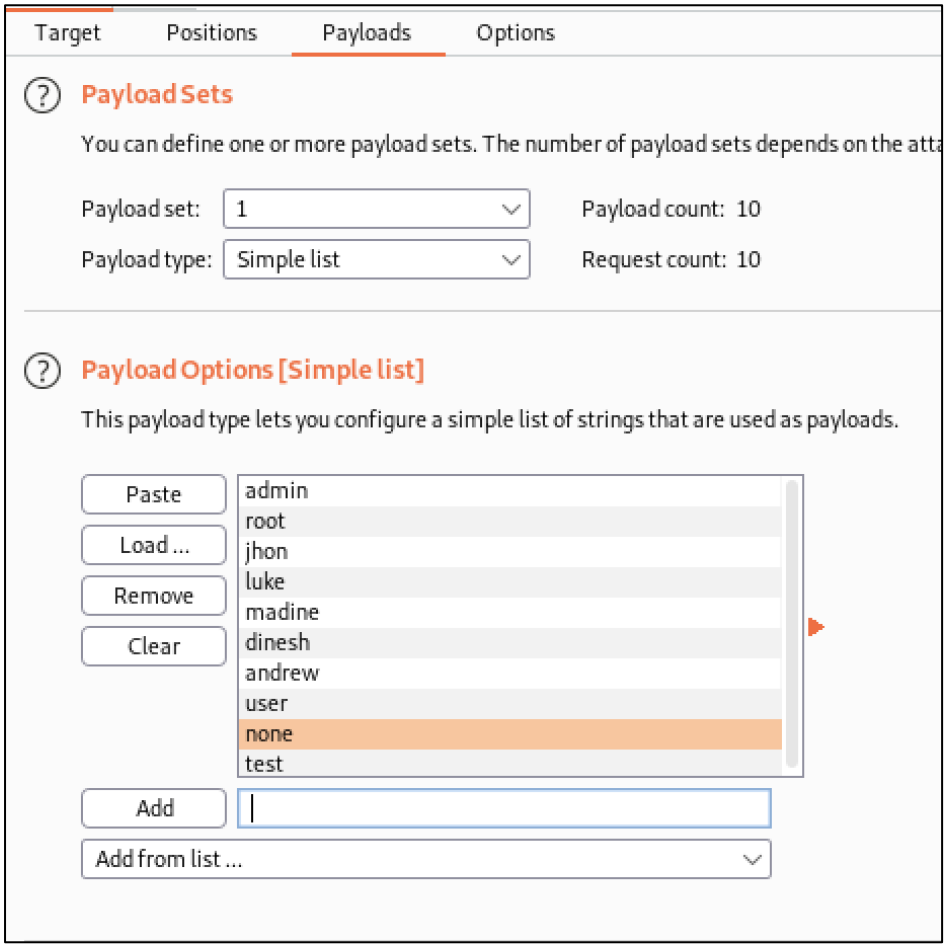


Figura 63 – Lista de posibles nombres de usuario

Finalmente se lanza la herramienta con el botón “*start attack*”

La herramienta envía una petición con cada nombre de usuario definido en la lista. En la Figura 64 se puede observar en la columna uno la lista de nombres probados, en la columna dos la respuesta que corresponde a cada elemento de la lista.

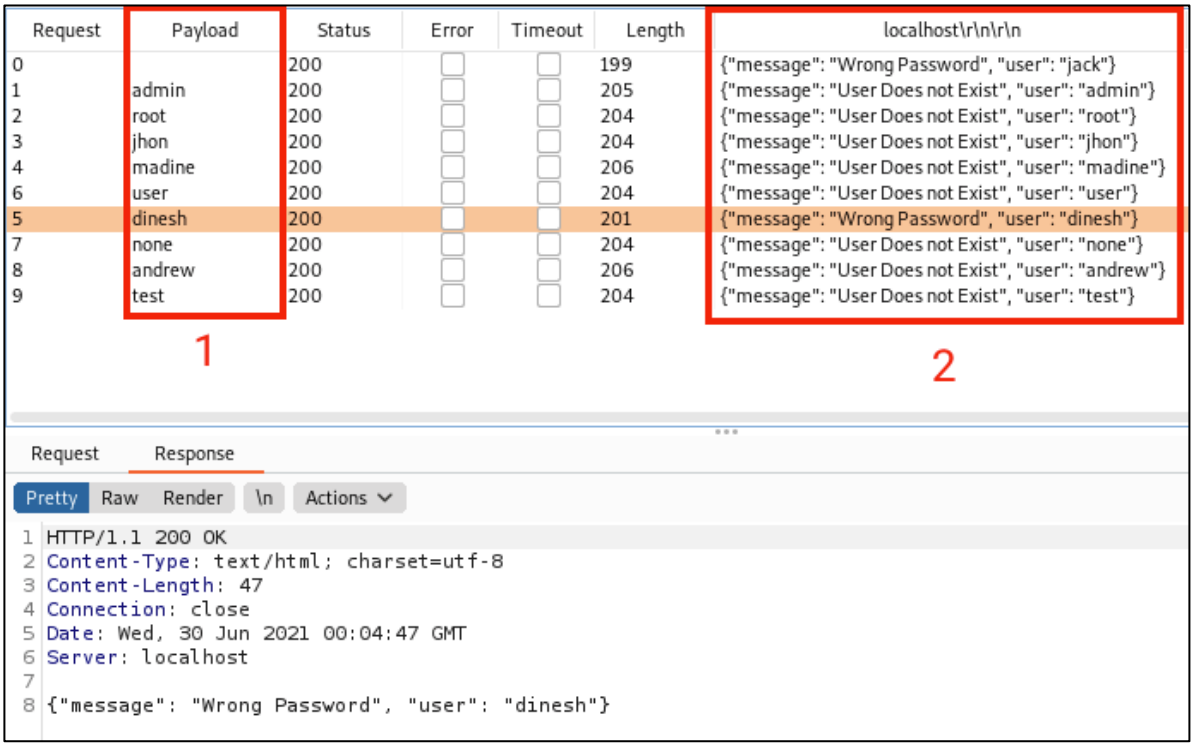


Figura 64 – Enumeración de usuarios con la herramienta intruder

Deficiencias en gestión de sesión

Las cookies o tokens son usados para realizar un seguimiento del usuario, esto asegura la capacidad de identificar al usuario en cualquier solicitud posterior, así como poder aplicar controles de acceso de seguridad, acceso autorizado a los datos privados del usuario y aumentar la usabilidad de la aplicación. Esta prueba se describe en el punto 5. Análisis dinámico del método propuesto.

Con ayuda de la herramienta Burp Suite al interceptar la comunicación entre la aplicación y el servidor, se identificó que no se implementa un mecanismo de identificador de sesión. En la Figura 65 se puede observar una petición de inicio de sesión, en la respuesta un mensaje indicando que se han ingresado las credenciales correctas, sin embargo, no se observa un token o cookie de sesión.

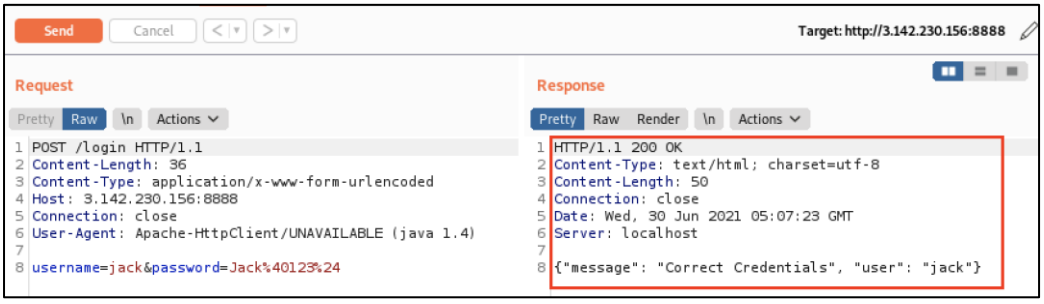


Figura 65 – Falta de identificador de sesión

Esta situación permite cambiar la contraseña de cualquier usuario con solo enviar el nombre de usuario válido y la nueva contraseña. En la Figura 66 se puede observar el cambio de contraseña para el usuario Jack.

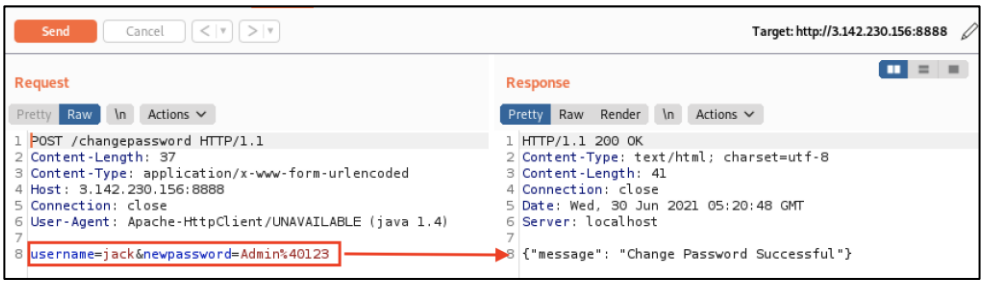


Figura 66 – Cambio de contraseña del usuario jack

5.2 iOS

En esta sección se describen los procesos técnicos ejecutados de acuerdo con la metodología propuesta para auditar la aplicación iOS en busca de fallas de seguridad.

Almacenamiento inseguro

La protección de datos sensibles, como tokens de autenticación e información personal, es un factor clave para la seguridad en aplicaciones móviles. De ser posible se debe guardar la menor cantidad posible de datos confidenciales. Aunque en la mayoría de los escenarios prácticos, se deben almacenar algunos datos del usuario.

Para realizar esta prueba, con ayuda de la herramienta Objection se obtuvo la ruta completa de los archivos almacenados por la aplicación.

Primero, con el iPhone conectado a través de USB a la máquina virtual Mobexler se obtuvo el PID de la aplicación con Frida y el comando:

```
frida-ps -Ua
```

Después, para instrumentar la aplicación y obtener el directorio de archivos, se utiliza el comando:

```
objection -g <pid> explore y env
```

En la Figura 67 se presenta el resultado de este proceso, y después se muestra el contenido del archivo *userInfo.plist* obteniendo el nombre del usuario y contraseña almacenados sin cifrado.

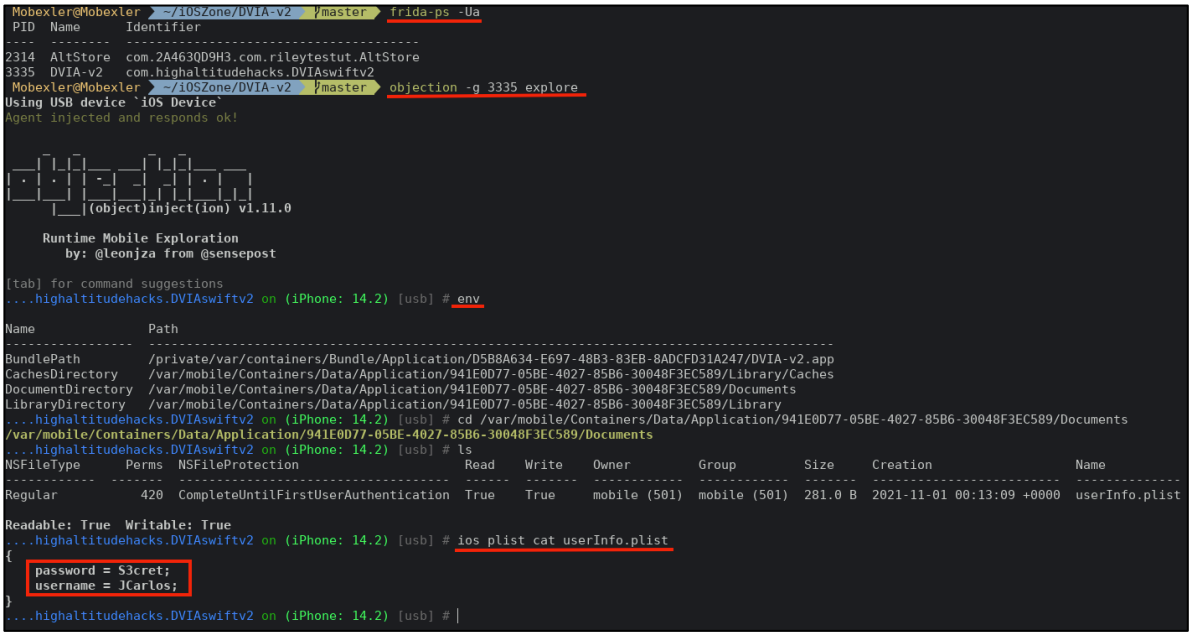


Figura 67 – Contenido del archivo userInfo.plist

NSUserDefaults

Es una interfaz para la base de datos predeterminada del usuario, donde se almacena la información en formato clave-valor. Los parámetros se conocen como predeterminados porque se usan para determinar el estado predeterminado de una aplicación al iniciar. Este sistema de valores predeterminados permite que una aplicación personalice su comportamiento, es decir las preferencias del usuario.

En la aplicación DVIA-v2 en el apartado *local data storage* se guardó información con *UserDefaults*, en la Figura 68 se observa la aplicación implementando este tipo de almacenamiento.

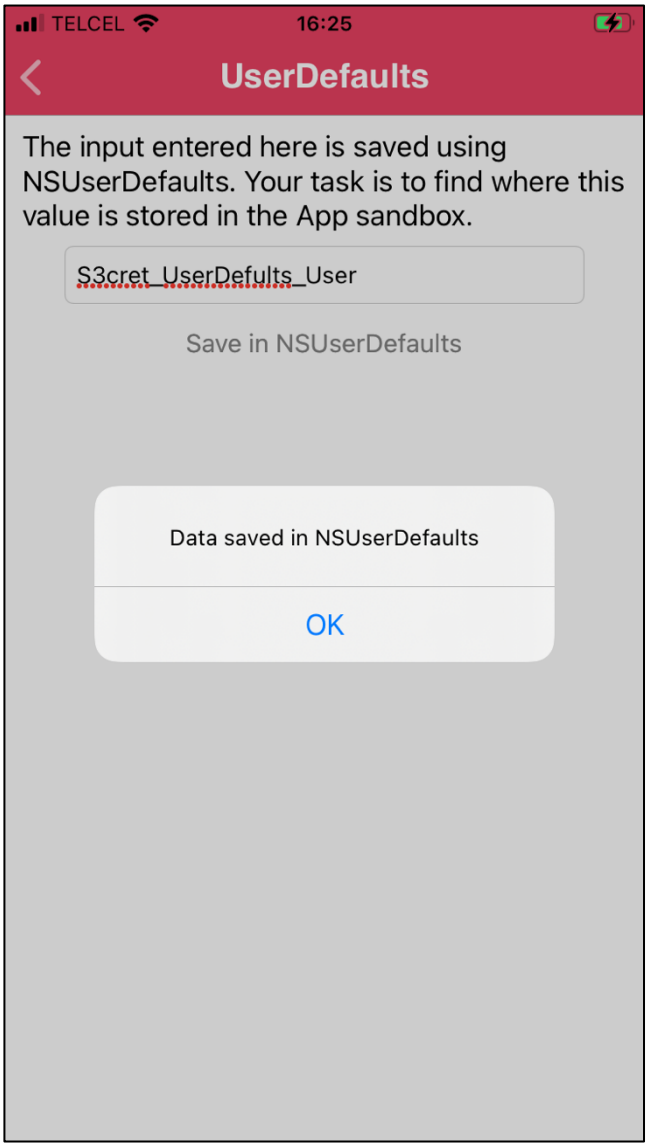


Figura 68 – Uso de NSUserDefaults

Para probar el uso de NSUserDefauls se utilizó la herramienta Objection nuevamente, con los siguientes comandos para obtener la información almacenada en la base de datos:

```
Objection -g <PID> explore

ios nsuserdefaults get
```

Como se puede observar en Figura 69, se almacena la información en texto plano, por lo que no es viable usar este tipo de almacenamiento para guardar información sensible.

```
Mobexler@Mobexler ~ objection -g 3439 explore
Checking for a newer version of objection...
Using USB device `iOS Device`
Agent injected and responds ok!


     _   _         _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
....highaltitudehacks.DVIAswiftv2 on (iPhone: 14.2) [usb] # ios nsuserdefaults get
{
    AKLastEmailListRequestDateKey = "2021-11-02 09:38:40 +0000";
    AKLastIDMSEnvironment = 0;
    AddingEmojiKeybordHandled = 1;
    AppleITunesStoreItemKinds =     (
        "itunes-u",
        movie,
        ringtone,
        album,
        "software-update",
        booklet,
        tone,
        "music-video",
        song,
        podcast,
        software,
        "podcast-episode",
        wemix,
        eBook,
        artist,
        mix,
        document
    );
    AppleLanguages =     (
        "es-MX"
    );
    AppleLanguagesDidMigrate = 18B92;
    AppleLanguagesSchemaVersion = 2000;
    AppleLocale = "es_MX";
    ApplePasscodeKeyboards =     (
        "es_MX@sw=QWERTY-Spanish;hw=Automatic",
        "emoji@sw=Emoji"
    );
    DemoValue = "S3cret_UserDefults_User";
    INNextFreshmintRefreshDateKey = "657495523.421012";
```

Figura 69 – Información almacenada con NSUserDefaults

Evasión de autenticación local

Una aplicación puede hacer uso de los métodos de autenticación local por ejemplo un PIN, una contraseña o características biométricas como el rostro o huella digital para iniciar la aplicación o bien hacer uso de una función crítica por ejemplo transferencia de fondos o acceder a información sensible. La descripción de este tipo de pruebas se encuentra en 6. Autenticación y Autorización del método propuesto.

El *framework* de autenticación local proporciona una *API* para que los desarrolladores desplieguen un cuadro de diálogo de autenticación al usuario. En iOS la autenticación de huellas dactilares se conoce como *TouchID*, el sensor de huellas es operado a través del coprocesador de seguridad *SecureEnclave* y no expone los datos de huellas a ninguna parte del sistema.

Existen dos opciones para incorporar la autenticación con el *TouchID*:

- LocalAuthentication.framework es una API que se puede utilizar para la autenticación mediante *TouchID*. La aplicación no puede acceder a ningún dato asociado con la huella digital registrada y solo se notifica si la autenticación se realizó correctamente.

- Security.framework es una API para acceder a los servicios del *keychain*. Esta es una opción segura si la aplicación necesita proteger datos confidenciales con autenticación biométrica, ya que el control de acceso se administra a nivel de sistema y no se puede evadir fácilmente.

El *framework* de autenticación local proporciona funciones para solicitar una contraseña o autenticación a través de *TouchID*, esto mediante la función `evalPolicy` de la clase `LAContext`, esta función devuelve un valor booleano que indica si el usuario se ha autenticado correctamente.

Aunque la autenticación a través del *TouchID* es eficaz a nivel interfaz de usuario, se puede evadir fácilmente mediante herramientas de instrumentación como frida y Objection. Por lo tanto, se debe reforzar con servicios del *keychain* verificando que los indicadores de control de acceso estén configurados para el elemento del *keychain*, lo que garantiza que los datos solo se puedan desbloquear mediante la autenticación del usuario.

La aplicación DVIA-v2 cuenta con un módulo donde se implementa la autenticación local con *TouchID*.

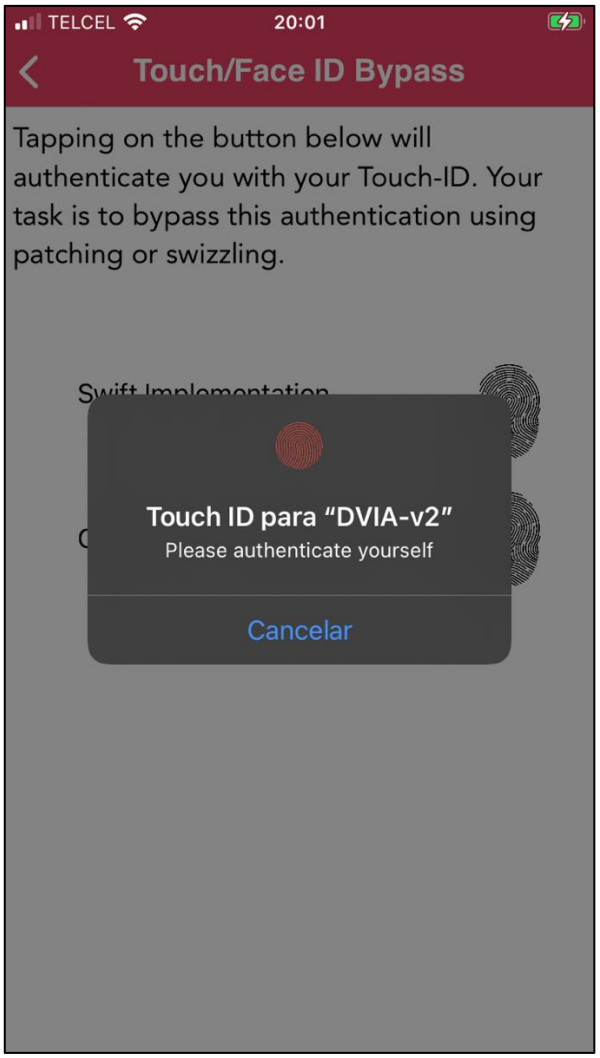


Figura 70 – Autenticación local con touchID

Para probar el método de autenticación local se realizó con el dispositivo conectado a través de USB a la máquina virtual y la herramienta Objection, que permite cambiar el valor que regresa la clase `LAContext` con los siguientes comandos:

frida-ps -Ui

Objection -g <PID> explore

ios ui biometric_bypass

Cuando se intenta autenticar con la huella, la aplicación despliega una ventana de autenticación, si se coloca una huella no registrada en el dispositivo el método `evalPolicy` devuelva el valor *`False`*, como se presenta en la Figura 71.

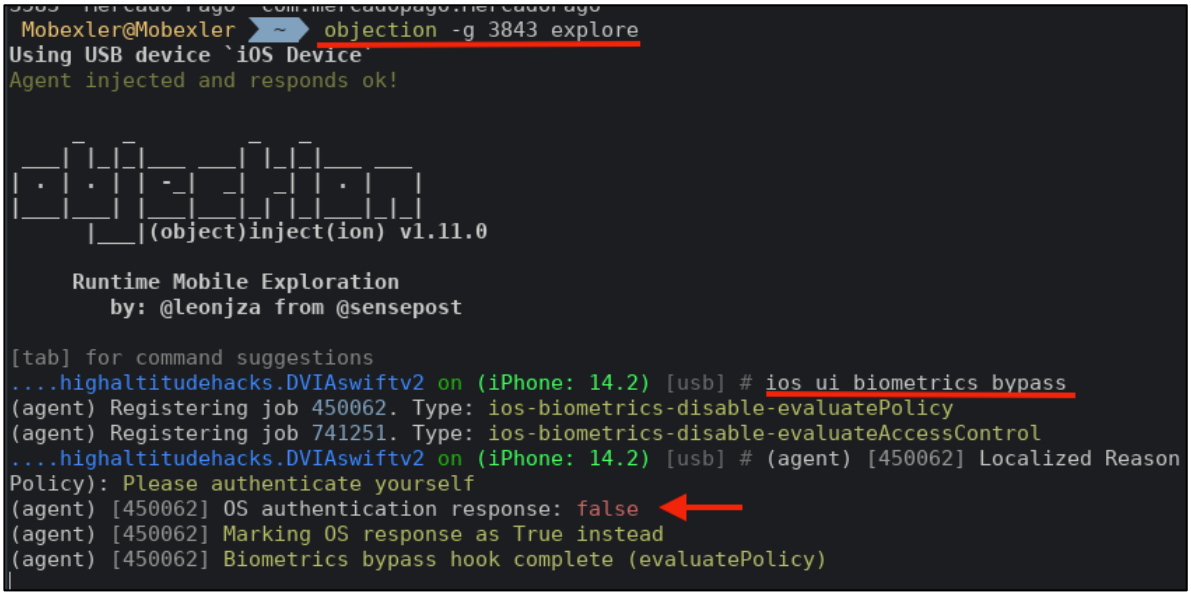


Figura 71 – Evasión de la autenticación con touchID

La herramienta Objection se cambia este valor a *`True`*, lo que permite autenticarse correctamente en la aplicación como si se hubiera colocado la huella correcta. En la Figura 72 se muestra el mensaje presentado por la aplicación.

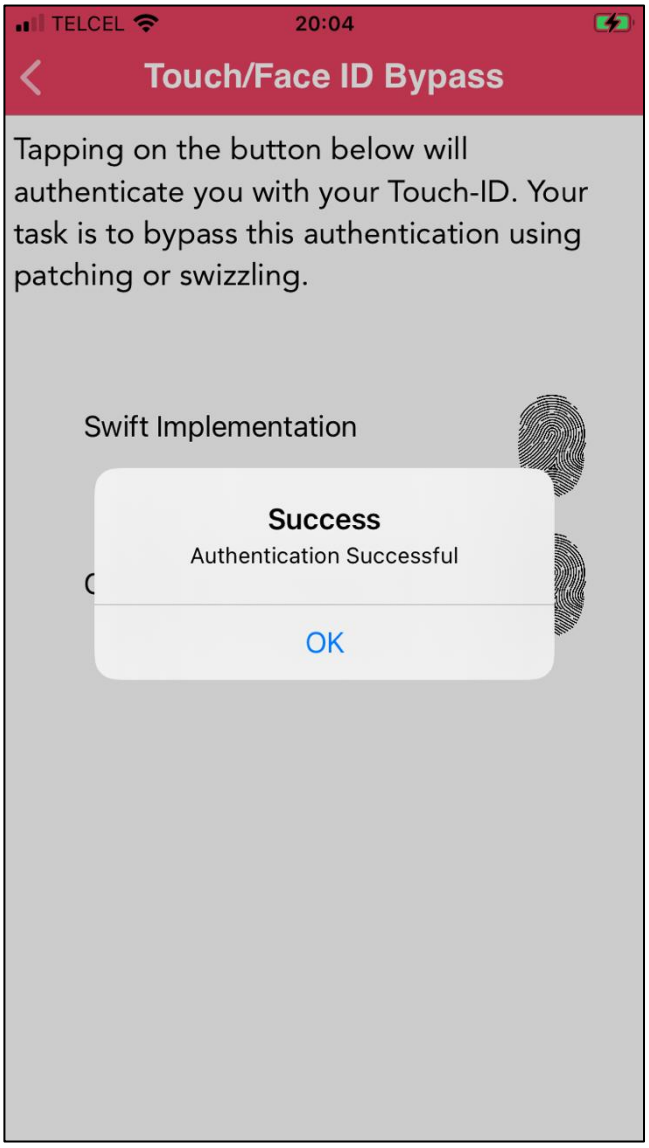


Figura 72 – Mensaje de autenticación correcta

Uso de protocolos sin cifrado

Por sí solo, el protocolo HTTP no incorpora medidas suficientes para garantizar la seguridad de las comunicaciones y es susceptible de ataques de diferente índole, como ataques de hombre en el medio (*Man-In-the-Middle*) o la interceptación de las comunicaciones. La descripción de este tipo de prueba se describe en el punto 7. Seguridad en la red del método propuesto.

Para saber si las comunicaciones entre la aplicación y el servidor se realizan por un canal cifrado, es necesario interceptar las comunicaciones. La herramienta Burp Suite permite interceptar, inspeccionar y modificar las solicitudes en ambas direcciones entre la aplicación y el servidor. Para esto es necesario una serie de configuraciones que se puede encontrar a detalle en las secciones Configuración del dispositivo Android y Configuración del dispositivo iOS de acuerdo con el tipo de dispositivo.

En la Figura 73 se observa la dirección IP y puerto a la escucha del proxy de la herramienta.

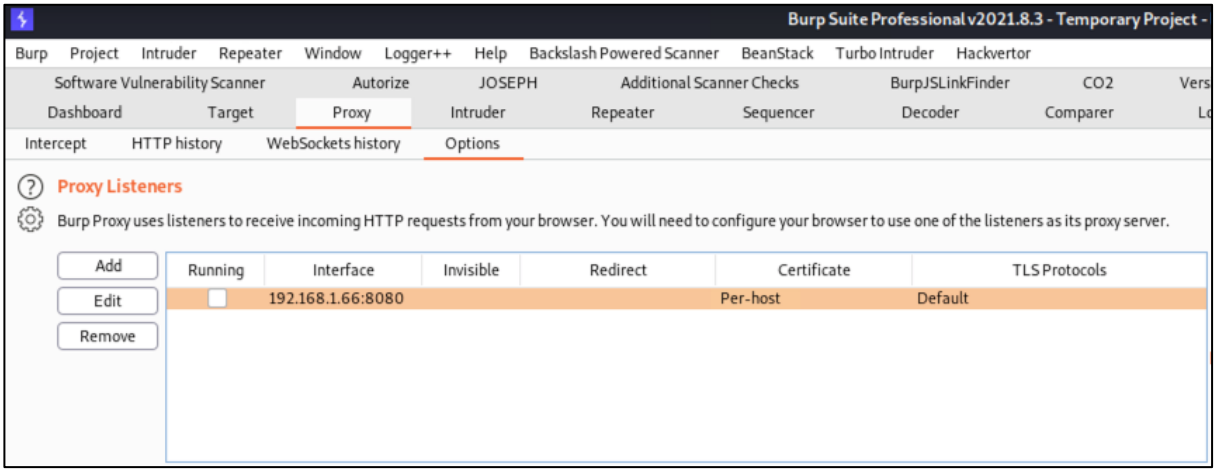


Figura 73 – Dirección IP y puerto del proxy Burp Suite

En el dispositivo iOS se configura la dirección IP y puerto del proxy en Configuración → Wi-Fi → Ajustes de red → Configurar proxy.

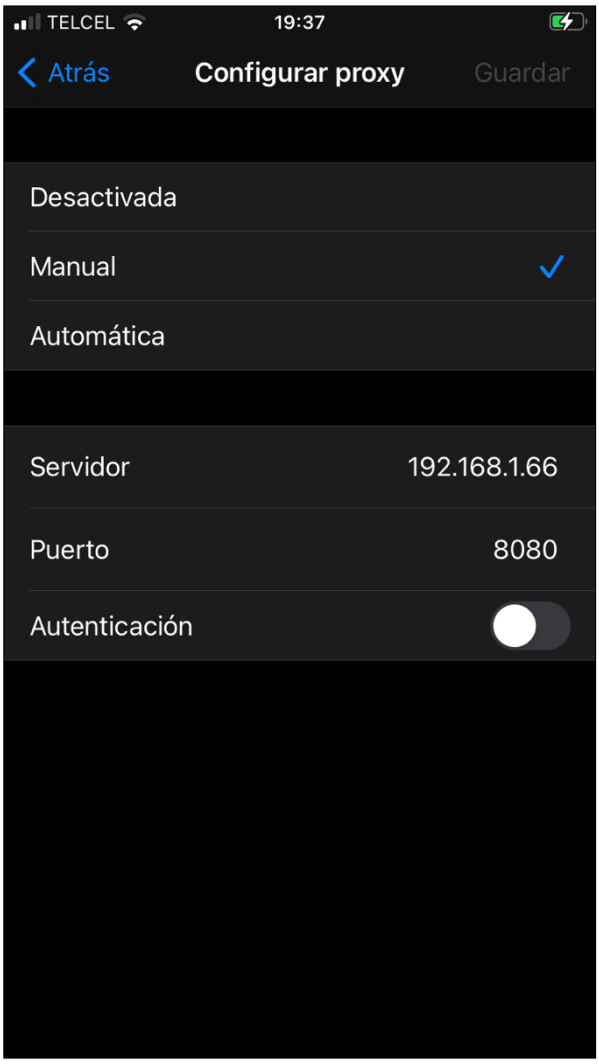


Figura 74 – Configuración proxy en el dispositivo iOS

Después de guardar la configuración en el dispositivo es posible observar las solicitudes que contienen información sensible entre la aplicación y el servidor, mediante el protocolo HTTP sin una capa de cifrado.

En caso de presentarse un error en la conexión de red quiere decir que se implementa el mecanismo de seguridad *SSLPinning* para más detalle consultar los puntos: 4. Protecciones y controles del lado del cliente y 5. Análisis dinámico del método de análisis propuesto.

A manera de ejemplo en la Figura 75 se presenta una petición en texto plano que incluye datos sensibles.

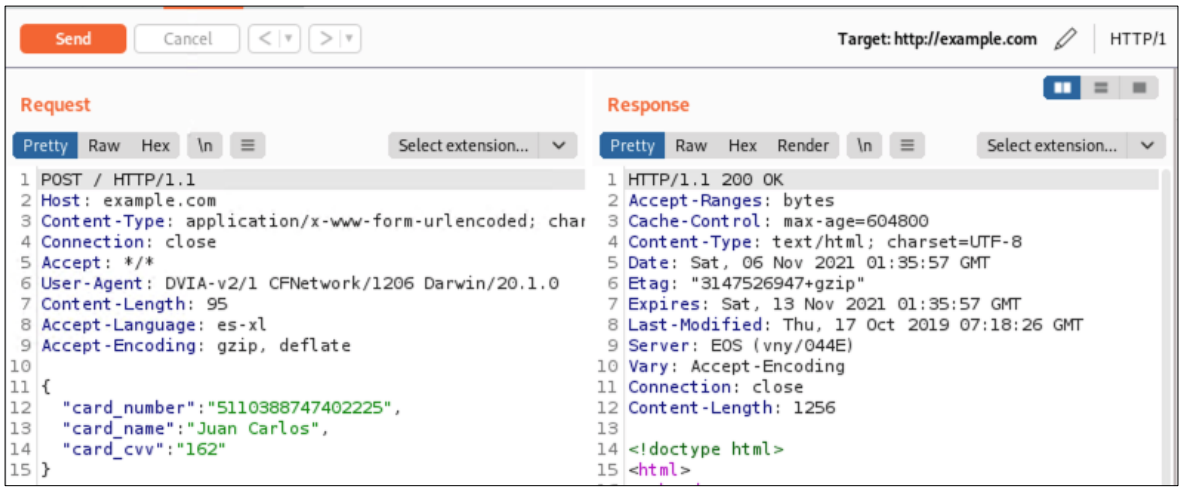


Figura 75 – Comunicación entre la aplicación DVIA y el servidor

Fuga de datos a través de servicios de terceros

La información confidencial se puede filtrar a terceros por diversas formas. En iOS, normalmente a través de servicios de terceros integrados en una aplicación. Las características que brindan estos servicios pueden incluir servicios de seguimiento para monitorear el comportamiento del usuario mientras usa la aplicación, anuncios, etc.
La desventaja es que los desarrolladores no suelen conocer los detalles del código ejecutado a través de las bibliotecas de terceros. En consecuencia, no se debe enviar a un servicio más información de la necesaria y no se debe divulgar información sensible.

La mayoría de los servicios de terceros se implementan de dos formas:

- Una librería independiente
- Un SDK completo

Todos los datos que se envían a servicios de terceros deben ser anonimizados para evitar la exposición de información personal. Para revisar la información que se envía a servicios de terceros se debe interceptar el tráfico entre la aplicación y el servidor, usando una herramienta tipo proxy.

La herramienta Burp Suite permite interceptar e inspeccionar las solicitudes entre la aplicación y el servidor, en la Figura 76 se observa parte de la configuración de la herramienta Burp Suite.

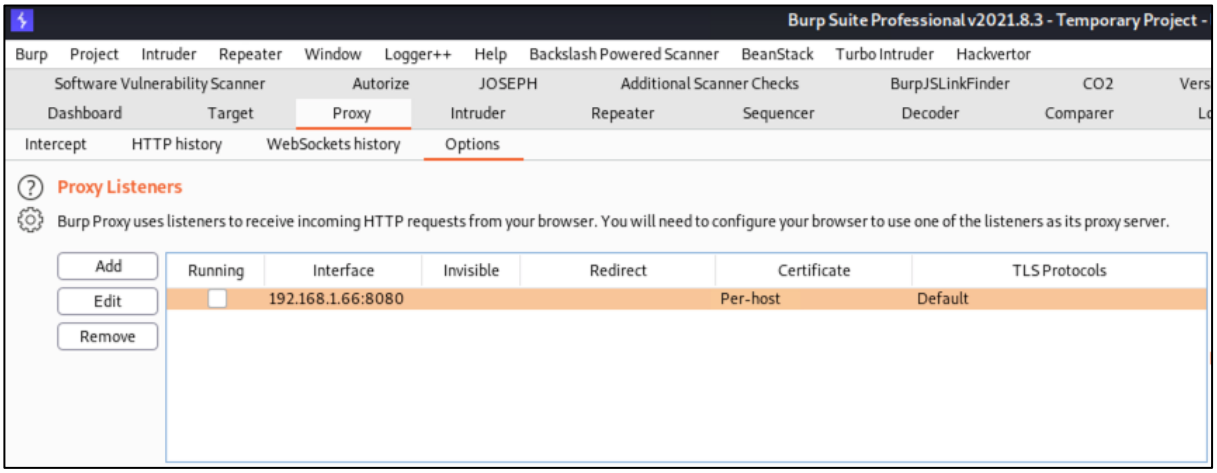


Figura 76 – Dirección IP y puerto del proxy Burp Suite

En el dispositivo se configura la dirección IP y puerto del proxy en: Configuración → Wi-Fi → Ajustes de red → Configurar proxy.

Figura 77 – Configuración de proxy en dispositivo iOS

Por último, navegar a través de la aplicación e inspeccionar las solicitudes y la información que se envía.

Se identificó el uso de Flurry, un servicio de analítica y publicidad para aplicaciones móviles se descubrió que se envía el número del teléfono del usuario a este servicio, esto se puede observar en la petición en la Figura 78.

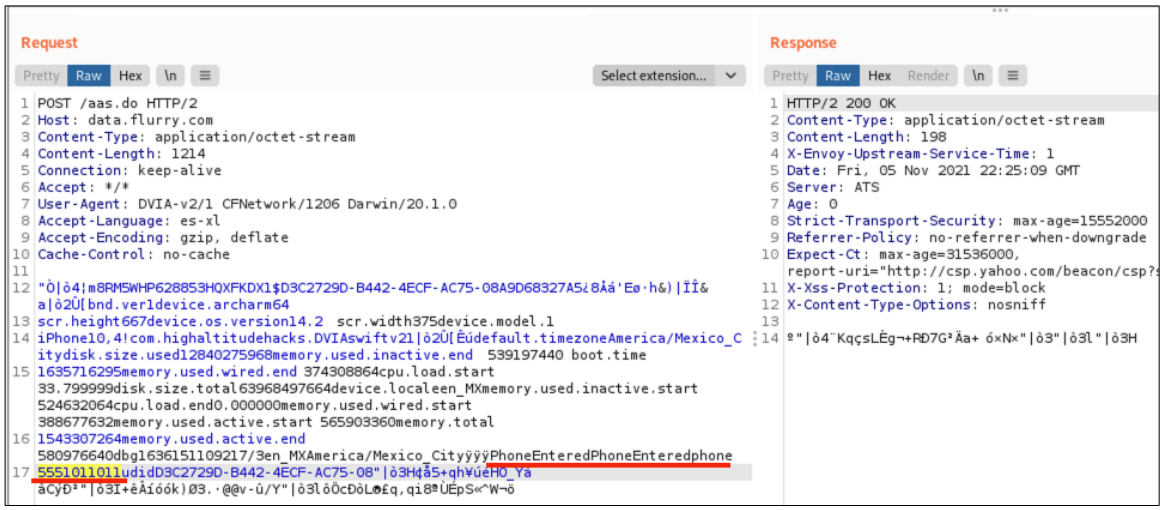


Figura 78 – Información del usuario enviada a servicios de terceros

Deficiencias en WebView

WebViews son componentes del navegador integrados en la aplicación para mostrar contenido web. Se puede utilizar para agregar contenido web directamente en la interfaz de usuario de una aplicación.

UIWebView es obsoleto a partir de iOS 12 y se debería evitar usar. Además, JavaScript no se puede deshabilitar.

WKWebView se introdujo a partir de iOS 8, permite controlar el contenido mostrado es decir evitar que el usuario ingrese una URL arbitraria, también aumenta el rendimiento de las aplicaciones a través del motor de JavaScript Nitro.

Además, cuenta con varias características de seguridad:

- JavaScript se encuentra habilitado de forma predeterminada, pero con la propiedad `javaScriptEnabled` de *WKWebView*, se puede deshabilitar por completo, evitando todos los problemas relacionados a inyección de scripts.

- JavaScriptCanOpenWindowsAutomatically se puede utilizar para evitar que JavaScript abra nuevas ventanas, como ventanas emergentes.

- La propiedad hasOnlySecureContent se puede utilizar para verificar que los recursos cargados por *WebView* se recuperen a través de conexiones cifradas.

- *WKWebView* implementa *out-of-process rendering*, por lo que los errores de corrupción de memoria no afectan el proceso principal de la aplicación.

La descripción de este tipo de pruebas se encuentra en 8. Interacción con la plataforma del método propuesto.

El primer paso para probar estos componentes es identificar el tipo de *WebView* usado por la aplicación. Con ayuda del comando:

```
strings DVIA-v2 | grep "UIWebView"
```

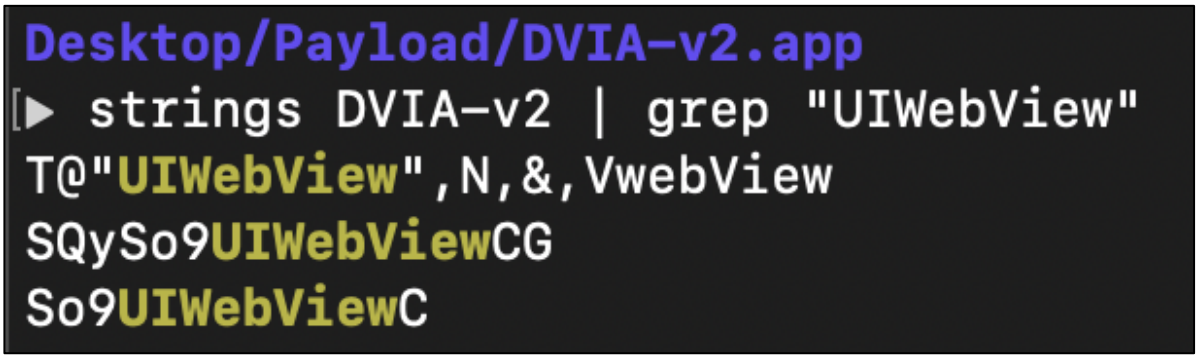


Figura 79 – Uso de UIWebView en la aplicación DVIA

Como se mencionó anteriormente, el componente *UIWebView* tiene habilitado JavaScript por defecto lo que permite ejecutar código JavaScript.

En la aplicación DVIA se identificó que no se validan los datos de entrada del usuario, lo que permite inyectar código JavaScript, en la Figura 79 se puede observar una sentencia en el campo de entrada “nombre”, está sentencia es ejecutada por el componente UIWebView.

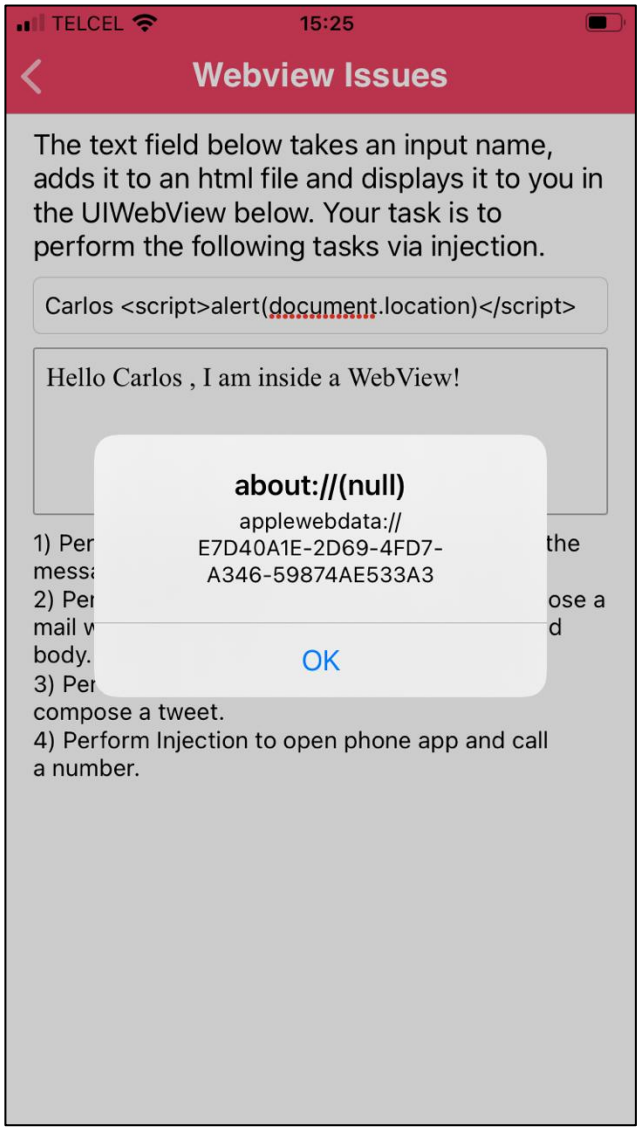


Figura 80 – XSS en la aplicación DVIA

Es posible inyectar JavaScript con URL de protocolos personalizados para realizar acciones, por ejemplo: llamar a un número de teléfono, en la Figura 81 se muestra un ejemplo para realizar una llamada. Esta acción necesita ser confirmada por el usuario, pero en versiones anteriores de iOS no se requería confirmación.

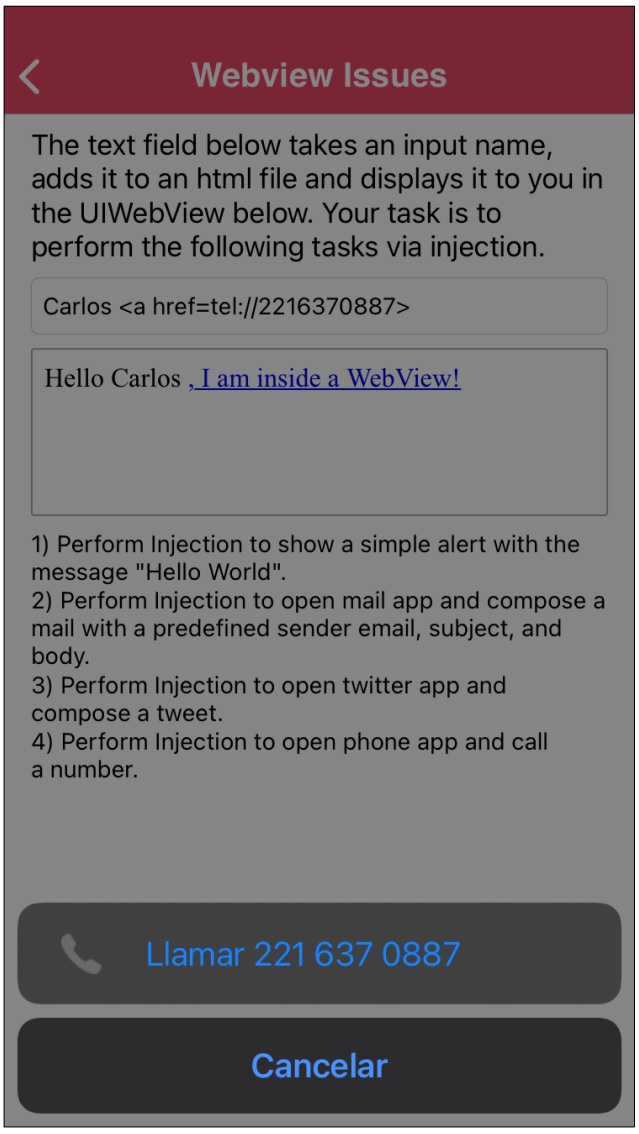


Figura 81 – Inyección de JavaScript con URL de protocolos personalizados

Información sensible en capturas de pantalla autogeneradas

Los fabricantes proporcionan un efecto agradable cuando se inicia o termina una aplicación, por lo que se introdujo el concepto de guardar una captura de pantalla cuando la aplicación pasa a segundo plano. Esta función puede representar un riesgo porque las capturas de pantalla podrían contener información sensible como: números de tarjetas de crédito, información personal, etc. Estas capturas se guardan en el almacenamiento local y podrían ser recuperadas por una aplicación maliciosa o con acceso físico al dispositivo. Para más detalles respecto a esta prueba consultar el punto 8. Interacción con la plataforma del método propuesto.

Para realizar esta prueba, primero se debe navegar por la aplicación en pantallas que muestren información confidencial, después se pasa la aplicación en segundo plano con el botón Home y se verifica que en la imagen en miniatura no se pueda observar información sensible, por último, se revisa la ruta donde se almacenan estas capturas generadas por el sistema operativo.

En la Figura 82 se puede observar que aplicación DVIA-v2 no oculta la información sensible cuando pasa a segundo plano.

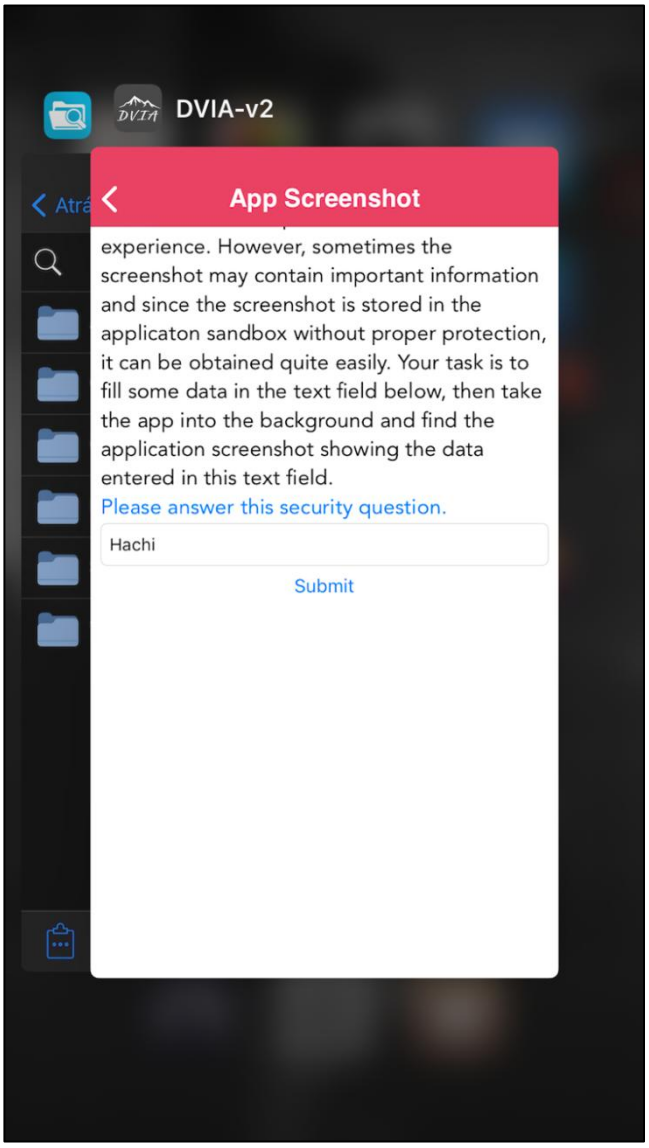


Figura 82 – Aplicación muestra información sensible

Las capturas autogeneradas se almacenan en el directorio */var/mobile/Containers/Data/Application/<UUID>/Library/SplashBoard/Snapshots/sceneID:highaltitudehacks.DVIAswiftv2-defaults/.* La ubicación puede diferir en cada versión de iOS, pero generalmente se encuentra dentro del directorio *Library* de la aplicación.

Al recuperar las capturas, se puede observar que divulgan información sensible del usuario en la Figura 83.

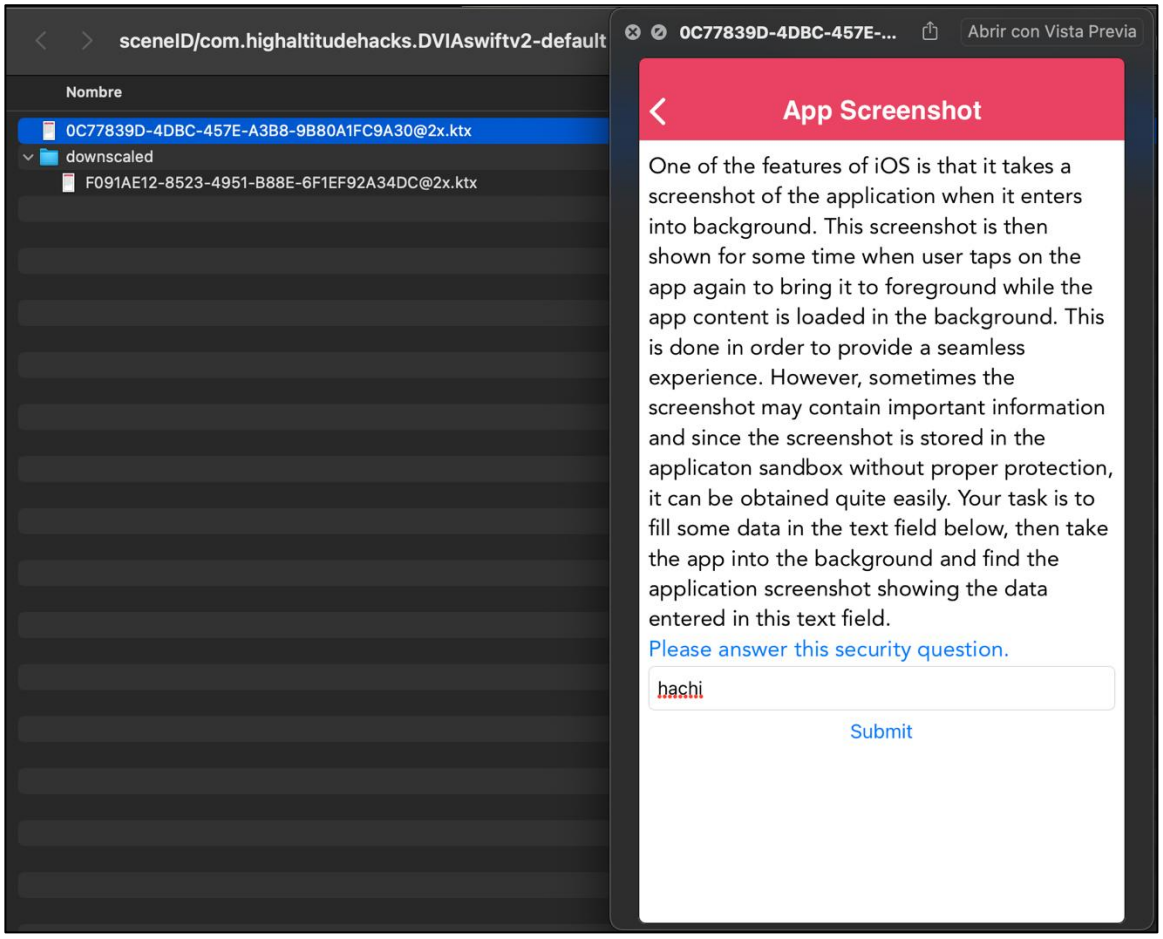


Figura 83 – Capturas autogeneradas que muestran información sensible

5.3 Resultados

Como resultado de la auditoría se presentan las vulnerabilidades identificadas en las aplicaciones evaluadas, puntuadas con el estándar CVSS 3.1 usando las métricas Base, Temporal y Ambiental, con una breve descripción del impacto de la vulnerabilidad y la recomendación para mitigarla.

Aplicación InsecureBankv2 – Android

En esta sección se presentan las vulnerabilidades identificadas en la aplicación InsecureBankv2, organizadas según su puntuación CVSS.

[01] – Deficiencias en componentes Android		
CVSS	Puntuación: 9.1	Crítica
	Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:N
Impacto	Un atacante podría abusar de estos elementos alterando el funcionamiento de la aplicación para evadir controles de seguridad y obtener información del usuario.	
Recomendación	Mediante el atributo *android:exported* restringir el acceso a los componentes de la aplicación InsecureBankv2 a otras aplicaciones estableciendo el valor *false* para evitar exponer los elementos a otras aplicaciones.	

[02] – Uso de protocolos sin cifrado		
CVSS	Puntuación: 8.4	Alta
	Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N/E:F/RL:O/RC:C/CR:M/IR:M/AR:X/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:X
Impacto	Un elemento que tenga acceso al canal de comunicaciones ya sea porque está en el mismo segmento de red que el cliente o el servidor, puede interceptar la información intercambiada.	
Recomendación	Deshabilitar el uso de HTTP y redirigir a los clientes que hagan uso de este a HTTPS, haciendo uso de los protocolos TLS 1.2 o 1.3 y haciendo uso de conjuntos de cifrado robustos.	

[03] – Deficiencias en gestión de sesión		
CVSS	Puntuación: 7.5	Alta
	Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:H/MA:N
Impacto	Un atacante podría obtener acceso la cuenta de un usuario determinado, con solo conocer su nombre de usuario dentro de la aplicación.	
Recomendación	Evitar cualquier tipo de procesamiento de la información hasta validar correctamente que el usuario está autorizado a efectuar la solicitud enviada validando el identificador de sesión. Implementar un mecanismo de autenticación seguro, como cookies de sesión o tokens de acceso (por ejemplo, JWT). Asegurase de: • Configurar las cookies con las banderas HttpOnly, Secure y SameSite. • Usar HTTPS para proteger las comunicaciones. • Establecer un tiempo de expiración adecuado y gestionar la revocación de sesiones.	

[04] – Deficiencias en AndroidManifest		
CVSS	Puntuación: 5.9	Media
	Vector:	AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:P/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:N/MA:N
Impacto	La configuración en el archivo *AndroidManifest.xml* permite realizar respaldos de la información almacenada por la aplicación, como consecuencia podría derivar fugas de información.	
Recomendación	Restringir los respaldos en el archivo *AndroidManifest.xml*, cambiando el valor del atributo `android:allowBackup` a `false` para deshabilitarlos.	

[05] – Almacenamiento inseguro		
CVSS	Puntuación: 5.7	Media
	Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:N
Impacto	Un atacante con acceso remoto o acceso local al dispositivo puede obtener la información almacenada en texto plano.	
Recomendación	Implementar el uso de Android *KeyStore* para almacenar información sensible.	

[06] – Enumeración de Usuarios		
CVSS	Puntuación: 5.7	Media
	Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:L/MI:N/MA:N
Impacto	Un atacante puede obtener un listado de usuarios validos dentro de la aplicación para para después realizar ataques de fuerza bruta o phishing.	
Recomendación	Mostrar la misma respuesta HTTP en el proceso de autenticación, independientemente si el usuario existe o no en el sistema mediante mensajes de error genéricos, de manera que no se pueda identificar por estos la existencia o no de una cuenta de usuario.	

[07] – Funciones de depuración activas		
CVSS	Puntuación: 5.5	Media
	Vector:	AV:P/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:P/MAC:L/MPR:L/MUI:R/MS:U/MC:H/MI:N/MA:N
Impacto	Una aplicación maliciosa instalada en el dispositivo puede leer los logs y obtener la información del usuario que se presenta en la consola de depuración.	
Recomendación	Eliminar funciones de depuración, comentarios en el código, metadatos, archivos de pruebas (archivos dummy), datos de prueba y cualquier otra información descriptiva usada durante la fase de desarrollo y depuración de la aplicación.	

[08] – Deficiencias en WebView		
CVSS	Puntuación: 5.5	Media
	Vector:	AV:P/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:P/MAC:L/MPR:L/MUI:R/MS:U/MC:H/MI:N/MA:N
Impacto	Un atacante puede insertar código malicioso (JavaScript) en el WebView vulnerable para robar información confidencial, secuestrar sesiones de usuario o alterar la funcionalidad de la aplicación	
Recomendación	Deshabilitar el uso de JavaScript en el componente WebView cuando no sea estrictamente necesario con el método `setJavaScriptEnabled(false)`	

En la Figura 84 se presenta la clasificación de vulnerabilidades identificadas por severidad de la aplicación Android.

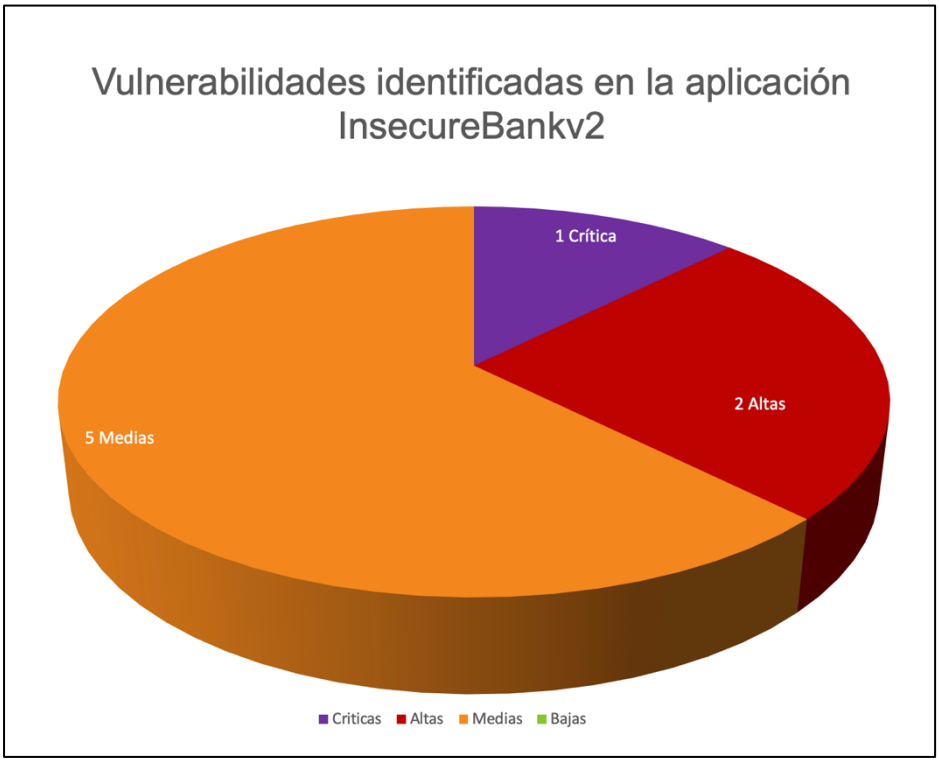


Figura 84 – Gráfica de clasificación de vulnerabilidades de la aplicación InsecureBankv2

Aplicación DVIA-v2 – iOS

En esta sección se presentan las vulnerabilidades identificadas en la aplicación DVIA-v2, organizadas según su puntuación CVSS.

[01] – Uso de protocolos sin cifrado		
CVSS	Puntuación: 8.6	Alta
	Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:N/MA:N
Impacto	Un elemento que tenga acceso al canal de comunicaciones ya sea porque está en el mismo segmento de red que el cliente o el servidor, puede interceptar la información intercambiada.	
Recomendación	Deshabilitar el uso de HTTP y redirigir a los clientes que hagan uso de este a HTTPS, haciendo uso de los protocolos TLS 1.2 o 1.3 y haciendo uso de conjuntos de cifrado robustos.	

[02] – Almacenamiento inseguro		
CVSS	Puntuación: 5.3	Media
	Vector:	AV:P/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:P/MAC:L/MPR:H/MUI:X/MS:X/MC:H/MI:N/MA:N
Impacto	Un atacante con acceso remoto o acceso local al dispositivo podría obtener la información almacenada en texto plano.	
Recomendación	Implementar el uso de *Keychain* en iOS para almacenar información sensible.	

[03] – Fuga de datos a través de servicios de terceros		
CVSS	Puntuación: 4.9	Media
	Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:F/RL:O/RC:C/CR:M/IR:M/AR:M/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:L/MI:N/MA:N
Impacto	Debido a que parte de la información del usuario es enviada a servicios de terceros es posible que se produzca una fuga de información causando violaciones en el tratamiento de datos personales.	
Recomendación	Restringir el acceso a la información del usuario a librerías de terceros, en caso de que no sea estrictamente necesario.	

[04] – Información sensible en capturas de pantalla autogeneradas		
CVSS	Puntuación: 4.3	Media
	Vector:	AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:F/RL:O/RC:C/CR:M/IR:M/AR:M/MAV:P/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:N/MA:N
Impacto	Un atacante a través de una aplicación maliciosa podría recuperar las capturas de pantalla autogeneradas, obteniendo así información sensible que permite obtener acceso a la cuenta de un usuario.	
Recomendación	Establecer en las configuraciones del desarrollo de la aplicación iOS *`false`* la configuración *`allowScreenShot`*. Por defecto está configurado como *`true`*.	

[05] – Evasión de autenticación con biométricos		
CVSS	Puntuación: 3.7	Baja
	Vector:	AV:P/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N/E:F/RL:W/RC:C/CR:M/IR:M/AR:M/MAV:P/MAC:L/MPR:H/MUI:N/MS:U/MC:H/MI:N/MA:N
Impacto	Un atacante con acceso privilegiado al dispositivo puede evadir los mecanismos de autenticación del *TouchID*, lo que le permitiría acceder a información sensible o funcionalidades críticas.	
Recomendación	Validar que las banderas de control de acceso kSecAccessControlBiometricCurrentSet y *kSecAccessControlBiometricAny* estén configuradas para el keychain lo que garantiza que los datos solo se pueden desbloquear mediante la autenticación del usuario.	

[06] – Deficiencias en WebView		
CVSS	Puntuación: 3.7	Baja
	Vector:	AV:P/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N/E:F/RL:W/RC:C/CR:M/IR:M/AR:M/MAV:P/MAC:L/MPR:H/MUI:N/MS:U/MC:H/MI:N/MA:N
Impacto	Permite a un atacante insertar código JavaScript malicioso en el *WebView* vulnerable. De esta manera puede ser utilizado para obtener información sensible, secuestrar sesiones de usuario, alterar la funcionalidad de la aplicación, etc.	
Recomendación	Reemplazar el componente UIWebView debido a que se encuentra obsoleto a partir de iOS 12 y no permite deshabilitar JavaScript. Hacer uso de WKWebView y deshabilitar JavaScript con el atributo `javaScriptEnabled = false.`	

En la Figura 85 se presenta la clasificación de vulnerabilidades identificadas por severidad de la aplicación iOS.

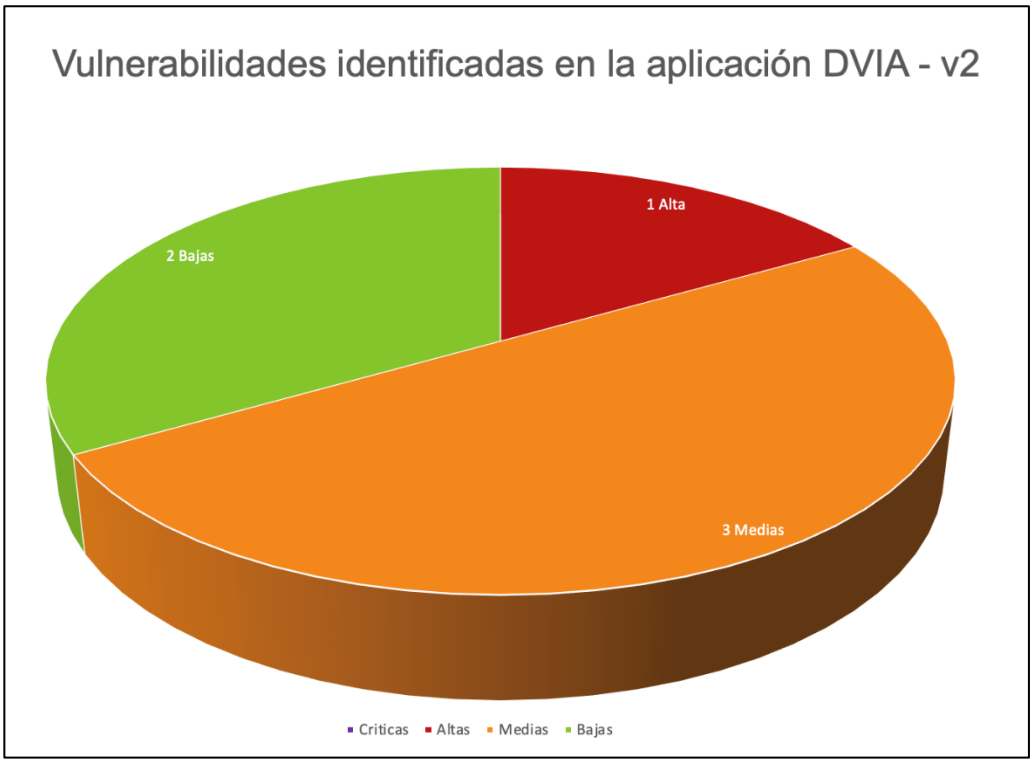


Figura 85 – Gráfica de clasificación de vulnerabilidades de la aplicación DVIA-v2

6 CONCLUSIONES Y TRABAJO A FUTURO

Las auditorias de seguridad para aplicaciones móviles, son una práctica necesaria para garantizar que se cumplen los requerimientos básicos de seguridad ante el incremento de las amenazas a estos dispositivos y sus aplicaciones.

Uno de los mayores retos al realizar este proyecto fue seleccionar las herramientas para realizar el análisis estático y dinámico sobre las aplicaciones de estudio. Por otro lado, se ha podido comprobar que existen muchas herramientas que permiten llevar a cabo análisis automatizados, pero en muchos casos estas herramientas han quedado obsoletas, funcionando solo para ciertas versiones de Android o iOS y prácticamente en desuso, siendo incompatibles con las nuevas versiones de los sistemas operativos.

La auditoría se ha realizado basándose en la metodología propuesta, tomando como referencia los recursos del proyecto OWASP Mobile Security Project como guía de referencia. De acuerdo con los resultados obtenidos, se identificaron un total de 16 fallos en la aplicación InsecureBankv2 considerando un total de 19 vulnerabilidades. Por lo tanto, la efectividad de la metodología propuesta es (16/19) x100 = 84.2% en la aplicación Android. En el caso de la aplicación en la plataforma iOS la efectividad es (7/10) x100 = 70%.

Con base en los resultados obtenidos se puede concluir que, a pesar de las medidas de seguridad en los sistemas operativos móviles y hardware, existen vulnerabilidades que pueden poner en riesgo la información del usuario o el propio dispositivo, por lo que se deben considerar distintos controles para reforzar la seguridad en las aplicaciones móviles y los dispositivos, algunas de estas medidas son: prácticas de codificación segura al momento de desarrollar una aplicación. Integrando los requerimientos de seguridad durante la fase de desarrollo, lo que se conoce como *Secure Software Develoment Life Cycle* por sus siglas en inglés (SSDLC).

Análisis de amenazas: es un proceso estructurado con estos objetivos identificar los requisitos de seguridad, señalar las amenazas a la seguridad y las posibles vulnerabilidades, cuantificar la criticidad de las amenazas y vulnerabilidades y priorizar los métodos de corrección.

Configuraciones de seguridad: dado que las buenas prácticas de seguridad durante el desarrollo no son suficientes para mitigar la explotación de las vulnerabilidades, en el Anexo B se describe una serie de configuraciones de seguridad para los sistemas operativos Android e iOS recomendada por el CIS, que permiten reducir la superficie de ataque y mitigar la explotación de algunas vulnerabilidades.

Trabajo a futuro

Debido a que las aplicaciones y los sistemas operativos se encuentran en constante cambio es una tarea ardua la actualización y soporte de los recursos de MSTG y las herramientas, por lo que se lista un conjunto de tareas a futuro:

- Optimización de la Tasa de detección: Mejorar la efectividad de la metodología propuesta, añadiendo casos de prueba.
- Automatización de tareas: crear y recopilar scripts para automatizar pruebas.
- Colaboración e Investigación: Crear un blog con recursos para la auditoria de aplicaciones entre los que se destacan:
    - Artículos de interés sobre temas seguridad en aplicaciones móviles.
    - Scripts y herramientas para evadir los controles *SSLPinning* y *Root detection* que a menudo son una de las tareas que consumen gran parte del tiempo de un proyecto de auditoría de seguridad en aplicaciones móviles.
    - Resolución de retos y laboratorios de aplicaciones vulnerables.
- Obtener las certificaciones enfocadas a seguridad en aplicaciones móviles en:
    - eMAPT de eLearn Security

      https://certs.ine.com/c76fa793-05f7-4da9-9077-615bb7292826

Figura 86 – Certificado eMAPT

BIBLIOGRAFÍA

[1] T. Reid, “OF TASTE,” in Essays on the Intellectual Powers of Man, Cambridge: Cambridge University Press, 2011, pp. 713–766

[2] Statcounter, "Mobile Operating System Market Share Worldwide," *gs.statcounter.com*, [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202306-202405-bar. [Accessed: Jun. 24, 2024].

[3] R. Fernández, "Internet: dispositivos de acceso preferidos en 2023," *gs.statcounter.com*, [Online]. Available: https://es.statista.com/estadisticas/1331273/internet-cuota-de-acceso-por-dispotivo-en-el-mundo/. [Accessed: Jun. 25, 2023].

[4] L. Ceci, "Internet: dispositivos de acceso preferidos en 2023," *www.statista.com* , [Online]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. [Accessed: Sept. 18, 2023].

[5] NIST, "Statistics Mobile Threat Catalogue," pages.nist.gov, [Online]. [Available: Jul. 18, 2022]

[6] L. Ceci, "Annual number of mobile app downloads worldwide 2023," *www.statista.com*, [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. [Accessed: Sep. 18, 2023].

[7] Z. Whittaker, "Amazon’s Ring Neighbors app exposed users’ precise locations and home addresses," *techcrunch.com*, [Online]. Available: https://techcrunch.com/2021/01/14/ring-neighbors-exposed-locations-addresses/. [Accessed: Apr. 24, 2022].

[8] R. HAGER, "That Slack email you just got asking to reset your password is legit, not a scam," *www.androidpolice.com*, [Online]. Available: https://www.androidpolice.com/2021/02/05/that-slack-email-you-just-got-asking-to-reset-your-password-is-legit-not-a-scam/ [Accessed: Apr. 24, 2022].

[9] C. Cimpanu, "Security bugs left unpatched in Android app with one billion downloads," *www.zdnet.com*, [Online]. Available: https://www.zdnet.com/article/security-bugs-left-unpatched-in-android-app-with-one-billion-downloads/. [Accessed: May 7, 2022].

[10] S. Gatlan, "Facebook Messenger bug allowed Android users to spy on each other," *www.bleepingcomputer.com*, [Online]. Available: https://www.bleepingcomputer.com/news/security/facebook-messenger-bug-allowed-android-users-to-spy-on-each-other/. [Accessed: Apr. 24, 2022].

[11] INCIBE, "Amenaza vs Vulnerabilidad, ¿sabes en qué se diferencian," *www.incibe.es*, [Online]. Available: https://www.incibe.es/protege-tu-empresa/blog/amenaza-vs-vulnerabilidad-sabes-se-diferencian. [Accessed: Feb. 25, 2021].

[12] S. Schleier, J. Willemsen, S. Schleier y C. Holguera, "Mobile Application Security Testing," *mobile-security.gitbook.io*, [Online]. Available: https://mobile-security.gitbook.io/mobile-security-testing-guide/overview/0x04b-mobile-app-security-testing. [Accessed: Feb. 10, 2021].

[13] FIRST, "Common Vulnerability Scoring System v3.1: Specification Document," *www.first.org*, [Online]. Available: https://www.first.org/cvss/specification-document. [Accessed: May 05, 2022].

[14] S. Schleier y C. Holguera, "OWASP Mobile Application Security," *mas.owasp.org*, [Online]. Available: https://mas.owasp.org/. [Accessed: Oct. 17, 2022].

[15] S. Schleier, C. Holguera, B. Mueller y J. Willemsen,"OWASP MASVS - OWASP Mobile Application Security," *mas.owasp.org*, [Online]. Available: https://mobile-security.gitbook.io/masvs/. [Accessed: Jan. 14, 2022].

[16] OWASP, "OWASP MAS Checklist," mas.owasp.org, [Online]. Available: https://mas.owasp.org/checklists/. [Último acceso: Feb. 02, 2022].

[17] M. Singh Thakur, A. MESBAHI, K. Atuly, M. Benchikh, S. Lortz y M. Junaid Tariq, "OWASP Mobile Top 10," *owasp.org*, [Online]. Available: https://owasp.org/www-project-mobile-top-10/. [Accessed: Jan 04, 2024].

[18] S. Schleier, C. Holguera, J. Beckers, B. Mueller y J Willemsen, "The Mobile Application Security Verification Standard," *mas.owasp.org*, [Online]. Available: https://mas.owasp.org/MASVS/Intro/0x03-Using_the_MASVS/. [Accessed: Nov. 19, 2022].

[19] OWASP Foundation, "OWASP Mobile Checklist," *mas.owasp.org*, [Online]. Available: https://github.com/OWASP/owasp-mastg/releases/latest/download/OWASP_MAS_Checklist.xlsx. [Accessed: Nov. 20, 2023].

[20] Cámara de Diputados del H. Congreso de la Unión, "Ley Federal de Protección de Datos Personales en Posesión de los Particulares," www.diputados.gob.mx, [Online]. Available: https://www.diputados.gob.mx/LeyesBiblio/pdf/LFPDPPP.pdf. [Accessed: Nov. 19, 2021].

[21] E. Rodriguez Sanchez, "Trabajo Fin de Máster: Auditoría de seguridad de aplicaciones móviles," openaccess.uoc.edu [Online]. Available: https://openaccess.uoc.edu/webapps/o2/bitstream/10609/95927/7/evarodriguezsTFM0619memoria.pdf. [Accessed: May 7, 2022].

[22] H. Pauta Martillo, "Auditoría de Seguridad a Aplicaciones en iOS y Android," *openaccess.uoc.edu*, [Online]. Available: https://openaccess.uoc.edu/webapps/o2/bitstream/10609/118328/7/hpautaTFM0620memoria.pdf. [Accessed: Feb. 20, 2021].

[23] D. Gonzalez Morte, "Estudio de dispositivos móviles, vulnerabilidades y auditoría de seguridad de aplicaciones móviles," openaccess.uoc.edu, [Online]. Available:

https://openaccess.uoc.edu/webapps/o2/bitstream/10609/95126/6/dagonzalezmoTFM0619memoria.pdf. [Accessed: Feb. 23, 2021].

ANEXO A - JAILBREAK Y ACCESO ROOT

Se denomina *jailbreak* al proceso de suprimir algunas de las limitaciones impuestas por Apple, como lo son la ejecución de código sin la previa verificación por parte de Apple en dispositivos que utilicen el sistema operativo iOS mediante el uso de núcleos modificados.

Una de las principales razones para hacer *jailbreak* es para expandir las características impuestas por Apple y su App Store. La mayoría de las herramientas para obtener *jailbreak* en un dispositivo instalan automáticamente Cydia, un cliente nativo *Advanced Package Tool* por sus siglas en inglés (APT) para iOS que incluye una interfaz gráfica de usuario para ayudar a la instalación de aplicaciones.

Tipos de Jailbreak

- **Tethered**

  No es persistente durante el reinicio, por lo que para volver hacer el *jailbreak* se requiere que el dispositivo esté conectado a un ordenador durante cada reinicio. Es posible que el dispositivo no se reinicie si no está conectado al ordenador.

- **Semi-tethered**

  No es persistente durante el reinicio, para volver hacer el *jailbreak* se requiere que el dispositivo esté conectado a un ordenador. Es posible reiniciar el dispositivo, pero sin las modificaciones del *jailbreak*.

- **Semi-untethered**

  Es posible que el dispositivo se reinicie por sí solo, pero el usuario debe volver hacer *jailbreak* iniciando una aplicación o visitando un sitio web, no se requiere que el dispositivo esté conectado a la computadora.

- **Untethered**

  Es persistente a través de los reinicios y solo requiere hacer el *jailbreak* una sola vez.

¿Qué herramienta de Jailbreaking usar?

Las diferentes versiones de iOS requieren diferentes técnicas de jailbreaking. La escena del *jailbreak* iOS evoluciona tan rápidamente que es difícil proporcionar instrucciones actualizadas. Sin embargo, existen algunas fuentes que actualmente proporcionan información sobre la disponibilidad de *jailbreak* de un dispositivo en específico con determinada versión del sistema operativos iOS.

- `https://docs.google.com/spreadsheets/d/1QjWyoDfaiF-TWhzVdvEMRqA3OQXsz6e8of3SxZB1W_M/edit`
- `https://www.reddit.com/r/jailbreak/`
- `https://canijailbreak.com/`

El acceso *root* es el proceso que permite a los usuarios de teléfonos inteligentes, tabletas y otros dispositivos que ejecutan el sistema operativo móvil Android obtener un control privilegiado (conocido como *root access*), además es posible acceder a los comandos administrativos y a las funciones del sistema operativo. Da la capacidad, o permiso de alterar o reemplazar aplicaciones, archivos y configuraciones del sistema, eliminando las aplicaciones preinstaladas y ejecutando aplicaciones especializadas que requieren permisos de nivel de administrador. Por lo general, el

acceso *root* se consigue mediante un *exploit* o desbloqueando el gestor de arranque del teléfono. Los *exploits* no siempre están disponibles y el gestor de arranque puede estar bloqueado de forma permanente o solo puede desbloquearse una vez que se haya rescindido el contrato con el operador.

La seguridad y protección del sistema no pueden ser garantizadas después del acceso *root* en el teléfono. Los datos de los dispositivos están en riesgo, incluyendo el acceso a información personal como listas de contactos, correos electrónicos y otros datos como credenciales y contraseñas. En un dispositivo con acceso *root*, un usuario o aplicación maliciosa puede elevar sus permisos para acceder a los datos privados de otras aplicaciones.

El *jailbreak* en dispositivos iOS o el acceso *root* en dispositivos Android no representa por sí solo una amenaza, sin embargo, facilita el escenario de un posible ataque por una aplicación maliciosa.

Las organizaciones financieras pueden requerir saber si su aplicación se está ejecutando en un dispositivo con acceso *root* en el caso de Android o en un dispositivo con *jailbreak* en el caso de iOS por una serie de razones. Por ejemplo, pueden optar por no procesar una transacción debido a la incertidumbre de su entorno de seguridad.

ANEXO B – CONFIGURACIONES DE SEGURIDAD POR EL CIS

En esta sección se listan las configuraciones de seguridad en Android 10 y iOS 14 propuestas por la comunidad que conforman el CIS por sus siglas en inglés (*Center for Internet Security*), una organización sin fines de lucro encargada de recopilar las mejores prácticas reconocidas a nivel mundial para proteger los sistemas y datos de TI.

CONFGURACIONES DE SEGURIDAD DEL SISTEMA OPERATIVO ANDROID	
Asegurarse que el firmware se encuentra actualizado.	No *rootear* el dispositivo móvil.
Asegurarse que la pantalla de bloqueo se encuentra habilitada.	Asegurarse que el bloqueo inteligente se encuentra deshabilitado.
Asegurarse que la visibilidad del patrón de bloqueo se encuentra inhabilitada.	Asegurarse que el bloqueo de tarjeta SIM se encuentra habilitado.
Asegurarse que el bloqueo automático se encuentra defino como “Inmediatamente”	Asegurarse que la opción “Encontrar mi dispositivo” se encuentra habilitada.
Asegurarse que el bloqueo instantáneo con el botón de encendido se encuentra habilitado.	Asegurarse que la función “Localizar este dispositivo de manera remota” se encuentra habilitado.
Asegurarse que el mensaje de bloqueo se encuentra configurado.	Asegurarse que la función “Permitir borrar y bloquear el dispositivo de manera remota” se encuentra habilitada.
No conectar el dispositivo a redes Wi-Fi no confiables, por ejemplo, redes en sitios públicos como aeropuertos y centros comerciales.	Asegurarse que se encuentra habilitada la opción “Analizar el dispositivo en busca de amenazas de seguridad” en las configuraciones de google.
Asegurarse que la visibilidad de las contraseñas en los campos de entrada se encuentra deshabilitadas.	Asegurarse que se encuentra habilitada la opción “Mejorar la detección de aplicaciones dañinas” en las configuraciones de google.
Asegurarse que las opciones de desarrollador se encuentran deshabilitadas.	Asegurarse que se solicite el PIN/Contraseña antes de desanclar la pantalla, si se tiene configurada.

CONFGURACIONES DE SEGURIDAD DEL SISTEMA OPERATIVO ANDROID	
Asegurarse que se encuentra deshabilitada la opción “Instalar aplicaciones de fuentes desconocidas”.	Asegurarse que el tiempo de espera de la pantalla se encuentra establecido en 1 minuto o menos.
Asegurarse que el asistente de Wi-Fi se encuentra deshabilitado.	Asegurarse que el perfil de invitado no exista.
Mantener las aplicaciones actualizadas.	Revisar los permisos de las aplicaciones constantemente.
Asegurarse que la función agregar perfiles, se encuentra deshabilitada en la pantalla de bloqueo.	

Tabla 1 – Lista de configuraciones de seguridad en Android 10

CONFGURACIONES DE SEGURIDAD DEL SISTEMA OPERATIVO IOS	
Asegurarse que la marcación por voz mientras el dispositivo está bloqueado se encuentra deshabilitada.	Asegurarse que se encuentra habilitada la opción “Forzar advertencia de fraude” en el navegador safari.
Asegurarse la opción “Permitir Siri mientras el dispositivo está bloqueado” se en encuentra deshabilitada.	Asegurarse que la opción “Aceptar Cookies” se encuentra configurada con la opción “De los sitios web que visito” o del sitio web actual.
Asegurarse que la función “Permitir aplicaciones administradas para almacenar datos en el iCloud” se encuentre deshabilitada.	Asegurarse que se encuentre configurada la opción “Dominios web de safari administrados”.
Asegurarse que la opción “Forzar el cifrado de los respaldos” se encuentre habilitada.	Asegurarse que la opción “Permitir valores simples” se encuentre deshabilitado en las configuraciones de códigos de acceso.
Asegurarse que la opción “Permitir que los usuarios acepten confiar en certificados TLS no confiables” se encuentre deshabilitada.	Asegurarse que la longitud mínima del código de acceso se encuentra establecida en 6 o mayor en las configuraciones de códigos de acceso.
Asegurarse que la opción “Permitir documentos de fuentes administradas en destinos no administrados” se encuentre deshabilitado.	Asegurarse que el bloqueo automático máximo se encuentra configurado en 2 minutos o menos.
Asegurarse que la opción “Permitir documentos de fuentes no administradas en destinos administrados” se encuentre deshabilitado.	Asegurarse que el periodo de gracia máximo para el bloqueo del dispositivo se encuentra establecido en “Inmediatamente”.
Asegurarse que la opción “Tratar AirDrop como destino no administrado” esté habilitado.	Asegurarse que el número máximo de intentos fallidos esté establecido en 6
Asegurarse que la opción “Permitir Handoff” este deshabilitada.	Asegurarse que “Mostrar centro de notificaciones en la pantalla de bloqueo” se encuentre deshabilitado.
Asegurarse que la opción “Forzar la detección de muñeca del Apple Watch” se encuentre activado.	Asegurarse que la opción “Mostrar el centro de control en la pantalla de bloqueo” se encuentre deshabilitado.

Tabla 2 – Lista de configuraciones de seguridad para iOS 14

Para consultar la(s) listas de configuraciones de otras versiones de sistemas operativos y los detalles para aplicar estas configuraciones consultar: https://downloads.cisecurity.org/#/

ANEXO C – RECURSOS Y MATERIAL DE ESTUDIO

En esta sección se presenta una lista de recursos y materiales de estudio recopilados durante el desarrollo de este proyecto y experiencias personales. Estos recursos sirven como referencia y apoyo fundamental para llevar a cabo auditorías de seguridad en aplicaciones móviles, proporcionando guías, herramientas y acceso a comunidades de investigadores.

Proyecto OWASP Mobile

- https://mas.owasp.org/

Blogs:

- https://medium.com/@justmobilesec
- https://medium.com/androgoat
- https://cmrodriguez.me/
- https://www.cobalt.io/blog/ios-pentesting-101
- https://www.yeswehack.com/learn-bug-bounty/getting-started-ios-penetration-testing-part-1
- https://www.yeswehack.com/learn-bug-bounty/getting-started-ios-penetration-testing-part-2
- https://book.hacktricks.xyz/mobile-pentesting/android-checklist
- https://www.corellium.com/blog/tag/mobile-technical-articles
- https://www.hackthebox.com/blog/intro-to-mobile-pentesting
- https://x.com/androidmalware2
- https://manifestsecurity.com/android-application-security/
- https://csbygb.gitbook.io/pentips/mobile-app-pentest/android

Comunidades:

- https://x.com/MobileHackingES
- https://www.linkedin.com/company/mobilehackingspace

Certificaciones

- https://www.giac.org/certifications/mobile-device-security-analyst-gmob/
- https://security.ine.com/certifications/emapt-certification/
- https://www.mobilehackinglab.com/course/android-appsec-exam

- https://www.crest-approved.org/membership/crest-ovs-programme/ovs-programme-accreditation-process/
- https://secops.group/product/certified-mobile-pentester-cmpen-android/
- https://secops.group/product/certified-mobile-pentester-cmpen-ios/
- https://certifications.tcm-sec.com/pjmt/

Cursos:

- https://www.mobilehackinglab.com/courses
- https://www.youtube.com/watch?v=niRooMwDUPU&list=PLmqenIp2RQcjBWzwMZQbIkbuVDmkYi_KF
- https://www.hacker101.com/playlists/mobile_hacking.html
- https://academy.tcm-sec.com/p/mobile-application-penetration-testing
- https://www.hacker101.com/playlists/mobile_hacking.html
- https://app.hextree.io/map/android

Laboratorios:

- https://mas.owasp.org/MASTG/apps/
- https://mas.owasp.org/crackmes/
- https://github.com/DERE-ad2001/Frida-Labs
- https://payatu.com/blog/damn-insecure-and-vulnerable-app-diva/
- https://app.hackthebox.com/tracks/Intro-to-Android-Exploitation
- https://www.mobilehackinglab.com/free-mobile-hacking-labs
- https://pentester.land/blog/list-of-intentionally-vulnerable-android-apps/
- https://github.com/lucideus-repo/UnSAFE_Bank

ANEXO D – GLOSARIO

Framework: En programación, un *framework* es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software. Utilizar *frameworks* puede simplificar significativamente una tarea o proceso.

Payload: En seguridad computacional, el payload se refiere a la parte del *malware* que realiza la acción maliciosa. En el análisis de software malicioso como gusanos, virus o Troyanos, se refiere a los resultados del ataque del software. Ejemplos de payloads podrían ser destrucción de datos, mensajes ofensivos o correo electrónico basura enviado a una gran cantidad de personas (spam).

Descompilar: esta tarea consiste en traducir el código ejecutable (compilado) a código fuente legible, donde el programa ejecutable está en código máquina, un lenguaje de bajo nivel, el código fuente está en un lenguaje de alto nivel, más inteligible. Después de este proceso, el código fuente puede volver a ser compilado para producir nuevamente un ejecutable que se comportará como el original.

Malware: El *malware* es un programa informático que tiene efectos no deseados o maliciosos. Incluye virus, gusanos, troyanos y puertas traseras. El *malware* a menudo utiliza herramientas de comunicación populares, como el correo electrónico y la mensajería instantánea, y medios magnéticos extraíbles, como dispositivos USB, para difundirse. También se propaga a través de descargas inadvertidas y ataques a las vulnerabilidades de seguridad en el software. La mayoría del *malware* peligroso actualmente busca robar información personal que pueda ser utilizada por los atacantes para cometer fechorías.

Ingeniería inversa: La ingeniería inversa de código permite invertir los procesos de compilación de una aplicación, y de esta forma, obtener el código fuente. La deconstrucción y el desarrollo invertido de una aplicación permite estudiar el código fuente, entender la arquitectura del programa, el funcionamiento y las estructuras internas. Además, este conocimiento sobre los procesos, adquirido gracias a la

ingeniería inversa de software, también hace posible la subsanación de fallos en aplicaciones.

GitHub: es un servicio de alojamiento de código para el desarrollo de software y el control de versiones utilizando Git. Proporciona el control de versiones distribuidas de Git más control de acceso, seguimiento de errores, integración continua, entre otras.

Intent: Un Intent es una clase de la API de Android que permite solicitar una acción de otro componente de una aplicación. Si bien las intents facilitan la comunicación entre componentes de varias formas, existen tres casos de uso principales: Iniciar una actividad, Iniciar un servicio y transmitir una emisión.

SQL: es un acrónimo en inglés de *Structured Query Language*. Un Lenguaje de Consulta Estructurado que permite insertar, borrar y recuperar datos de una base de datos relacional.

SQLite: SQLite es un motor de base de datos escrito en el lenguaje de programación C. No es una aplicación independiente, es una biblioteca que los desarrolladores de software integran en sus aplicaciones. Es el motor de base de datos más implementado, ya que lo emplean varios de los principales navegadores web, sistemas operativos, teléfonos móviles y otros sistemas integrados.

Ingeniería social: En el contexto de seguridad de la información, la ingeniería social consiste en la manipulación psicológica de las personas para que realicen acciones o divulguen información confidencial como contraseñas y tokens de acceso.

HTML: es su acrónimo en inglés de *HyperText Markup Language* es un lenguaje de marcado que se utiliza para el desarrollo de páginas web. Este lenguaje permite estructurar el contenido de una página web por medio de etiquetas de hipertexto. Otras tecnologías que se emplean en conjunto con HTML para describir la apariencia y la funcionalidad son CSS y JavaScript respectivamente.

***Cross-Site scripting* (XSS):** Es un tipo de vulnerabilidad que permite a un usuario malintencionado inyectar sentencias de JavaScript malicioso en una aplicación web para que este sea procesado y ejecutado por el navegador web, este proceso que se basa en la confianza que tiene la aplicación sobre la entrada de los datos del usuario.

Cipher Block Chaining (CBC) es un modo de funcionamiento de un cifrado por bloques, en el que una secuencia de bits se cifra como una sola unidad, o bloque, con una clave de cifrado aplicada a todo el bloque. utiliza lo que se conoce como un vector de inicialización (IV) de cierta longitud. Al utilizarlo junto con una única clave de cifrado, permite descifrar con seguridad grandes cantidades de texto

Vector de inicialización por sus siglas en inglés (IV): es un número arbitrario que se puede usar con una clave secreta para el cifrado de datos. Este número, también llamado nonce (número que se usa una vez), se emplea solo una vez en cualquier sesión para evitar el descifrado no autorizado del mensaje por parte de un actor malicioso. El uso de este evita la repetición de una secuencia de texto en el cifrado de datos. Específicamente, durante el cifrado, evita que una secuencia de texto sin formato que sea idéntica a una secuencia de texto sin formato anterior produzca el mismo texto cifrado.

Base64: es un grupo de esquemas de codificación similares a *binary-to-text* que representan datos binarios en un formato de cadena ASCII traduciéndolos a una representación radix-64. Los esquemas de codificación Base64 se usan comúnmente cuando existe la necesidad de codificar datos binarios que deben almacenarse y transferirse a través de medios diseñados para manejar ASCII. Esto es para garantizar que los datos permanezcan intactos sin modificaciones durante el transporte.

TouchID: Touch ID es una función de reconocimiento electrónico de huellas dactilares diseñada por Apple que permite a los usuarios desbloquear dispositivos, realizar compras en tiendas de medios digitales de Apple (iTunes Store, App Store y Apple Books Store) y como método de autenticación en Apple Pay o en aplicaciones. También se puede usar para bloquear y desbloquear notas protegidas con contraseña en iPhone y iPad.

Keychain: Keychain es el sistema de administración de contraseñas en macOS y iOS, desarrollado por Apple, se pueden almacenar varios tipos de datos: contraseñas (para sitios web, servidores FTP, cuentas SSH, redes compartidas, redes

inalámbricas, aplicaciones de trabajo en grupo, imágenes de disco encriptadas), claves privadas, certificados y notas seguras.

Intruder: Burp Intruder es una herramienta que permite enviar peticiones de forma automatizada para realizar ataques contra aplicaciones web. Esta herramienta posee diferentes características y se puede utilizar para realizar una gran variedad de tareas, desde descubrimiento de directorios por fuerza bruta hasta la explotación activa de vulnerabilidades complejas de inyección ciega de SQL.

Cookie: es una pequeña pieza de información enviada por el sitio web y almacenada en el navegador del usuario, conocidas como HTTP cookies se utilizan para identificar a usuarios específicos y mejorar su experiencia de navegación, además permite que los sitios web recuerden al usuario, los inicios de sesión, los productos en los carritos de compras y más.

Token de sesión: los datos del usuario se cifran en un *JSON Web Token* un por sus siglas en inglés (JWT) con un secreto y luego se envían de vuelta al usuario.

El JWT se almacena en el lado del cliente, principalmente *localStorage*, y se envía como un encabezado para cada solicitud posterior. El servidor recibe y valida el JWT antes de proceder a enviar una respuesta al cliente.

UUID: es un acrónimo de *Universally Unique Identifier* es una etiqueta alfanumérica de 36 caracteres que se utiliza para proporcionar una identidad única a cualquier recopilación de información dentro de un sistema informático. Debido a su baja probabilidad de duplicación, los UUID son una herramienta ampliamente adoptada para otorgar identidades persistentes y únicas.