Auditoría de Seguridad a Aplicaciones en iOS y Android

Autor: Héctor Pauta Martillo

Tutor: Joan Caparrós

Profesor: Víctor Font

Máster Interuniversitario de Seguridad de las Tecnologías de la Información y de las Comunicaciones (MISTIC)

TFM: Auditoría de Seguridad de aplicaciones en iOS y Android

Fecha: 29 de Mayo del 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	Auditoría de Seguridad a Aplicaciones en iOS y Android
Nombre del autor:	Héctor Pauta Martillo
Nombre del colaborador/a docente:	Joan Caparrós
Nombre del PRA:	Víctor García Font
Fecha de entrega:	06/2020
Titulación o programa:	Seguridad en Internet de las Cosas
Área del Trabajo Final:	Máster Seguridad de las Tecnologías de Información y de las Comunicaciones
Idioma del trabajo:	Español
Palabras clave	Seguridad, Android, iOS

Resumen del Trabajo

El objetivo de este TFM es desarrollar un proceso de una auditoría de Seguridad a aplicaciones para dispositivos móviles centrándonos en las implementadas para los sistemas operativos iOS y Android.

Para ello, se explicará una metodología estándar de trabajo ayudándonos del proyecto de OWASP Mobile Security Project, que es un recurso centralizado destinado a proporcionar a los desarrolladores y equipos de seguridad los recursos que necesitan para construir y mantener aplicaciones móviles seguras.

La metodología estará estructurada en tres fases: Reconocimiento, Análisis Estático y Análisis Dinámico con las que se pretende llevar a cabo un análisis completo que permita identificar las principales vulnerabilidades y debilidades de la aplicación auditada.

Inicialmente se abordará el contexto de aplicaciones para dispositivos móviles y lo que representa OWASP y su proyecto Mobile Security como referencia para el conocimiento de las principales vulnerabilidades y deficiencias para este tipo de plataformas.

Se detallarán cada de las distintas fases de la auditoría de seguridad, la preparación de un entorno de trabajo y se llevará a cabo un ejercicio práctico auditando dos aplicaciones vulnerables, siguiendo los pasos recomendados en este documento, para la ejecución de una correcta auditoría de seguridad.

Finalmente, se presentarán los resultados obtenidos de la auditoría de seguridad incluyendo como anexos los informes que facilitan algunas herramientas automáticas.

Abstract

The objective of this TFM is to develop a security audit process for applications for mobile devices focusing on those implemented for iOS and Android operating systems.

For this, a standard work methodology will be explained by helping us with the OWASP Mobile Security Project, which is a centralized resource designed to provide developers and security teams with the resources they need to build and maintain secure mobile applications.

The methodology will be structured in three phases: Reconnaissance, Static Analysis and Dynamic Analysis with which it is intended to carry out a complete analysis to identify the main vulnerabilities and weaknesses of the audited application.

Initially, the context of applications for mobile devices and what OWASP and its Mobile Security project will represent will be addressed as a reference for the knowledge of the main vulnerabilities and deficiencies for this kind of platforms.

The different phases of the safety audit, the preparation of a work environment will be explained and a practical exercise will be carried out auditing two vulnerable applications, following the steps recommended in this document, for the execution of a correct security audit.

Finally, the results of the security audit will be presented, including as annexes the reports provided by some automatic tools.

Índice

Índice de figuras

1.Introducción

1.1. Contexto y Justificación

La seguridad de los dispositivos móviles juega un papel cada vez más importante en la protección de los activos de Información de una organización, más con la aparición del Internet de las Cosas (IoT), lo que hace que también se tenga en cuenta o se aseguren adecuadamente los equipos a los cuales se conecten.

El uso de dispositivos móviles va en aumento y según Tech Crunch, la empresa de movilidad Ericsson predice que habrá más de seis mil millones de usuarios en 2020, de manera que superarán a los teléfonos fijos. ¿Por qué? Debido a que los smartphones se están haciendo más potentes a medida que las empresas adoptan la idea de las políticas tipo "trae tu propio dispositivo" y permiten a los usuarios acceder a las redes corporativas con tecnología personal. Ese aumento en el uso viene acompañado de una proliferación del malware móvil.

En 2017, la firma mundialmente reconocida, Kaspersky Labs detectó un total de 66,4 millones de ciberataques contra dispositivos móviles basados en aplicaciones maliciosas, que, tras instalarse, ganan permisos dentro del sistema operativo, toman el control y lo utilizan para transmitir otras amenazas diferentes o para extraer datos personales de las víctimas. Mientras tanto en 2018, los ciberataques contra dispositivos móviles, basados en software malicioso, han duplicado su presencia, registrando un total de 116.5 millones de ataques.

Figura 1: Comparativa de ataques por malware 2017-2018
Virología móvil 2018

Investigadores de la Universidad de Cambridge descubrieron que el 87 % de los smartphones Android están expuestos a al menos una vulnerabilidad crítica, mientras que Zimperium Labs descubrió a comienzos de 2019 que el 95 % de los dispositivos Android se podrían piratear con un simple mensaje de texto.

Apple tampoco es inmune. En septiembre, se quitaron 40 aplicaciones de la App Store porque estaban infectadas con XcodeGhost, un tipo de malware diseñado para activar los dispositivos de Apple en un botnet a gran escala. Es decir, a pesar de la afamada protección de Apple, el malware, se ubicó por encima de aplicaciones legítimas, por lo que era difícil de detectar.

Esto se agrava más con la poca información sobre riesgos latentes en este tipo de plataformas y la falta de concienciación que hay entre los usuarios sobre el Desarrollo y uso correcto de los dispositivos móviles. Cada vez que alguien desarrolla una aplicación móvil, la seguridad se deja de lado y el 52% de las veces se olvida por falta de tiempo.

Para evitar este tipo de amenazas son necesarios los análisis de seguridad periódicos. Con una auditoría de seguridad es posible descubrir vulnerabilidades en aplicaciones y servidores antes de que lo hagan los atacantes, lo que reduce el riesgo de pérdida de datos o sufrir cualquier otro tipo de ataque que suponga un impacto organizacional o reputacional.

1.2. Objetivos

El principal objetivo de este TFM será el de proporcionar una guía para ejecutar una auditoría de seguridad sobre aplicaciones en dispositivos móviles iOS y Android. Este análisis de vulnerabilidades pretende que pueda servir para prevenir amenazas y protegerse frente a posibles ataques.

Es por ello que, a lo largo del proyecto, se buscará:

- Proponer una metodología a seguir para la realización de auditorías de seguridad de aplicaciones para dispositivos iOS y Android.
- Estudiar las distintas amenazas que pueden afectar a una aplicación en dispositivos iOS y Android para ayudar a prevenirlas y protegerse frente a ellas.

- Conocer las distintas herramientas, disponibles actualmente, para la ejecución de auditorías de seguridad de aplicaciones implementadas en dispositivos iOS y Android.
- Realizar un ejercicio práctico, propuesto a modo de ejemplo, que nos permita identificar las vulnerabilidades y amenazas para la aplicación en estudio.
- Elaborar un informe de auditoría en función de los resultados obtenidos
- Presentación de conclusiones y posibles planteamientos de mejora.

1.3. Metodología y Proceso de trabajo

Para la elaboración de este TFM nos basaremos en dos partes bien diferenciadas, la parte teórica y la parte práctica.

Por un lado, en la parte teórica, se revisarán todos los conceptos necesarios para el entendimiento del entorno de análisis, su preparación, así como cada uno de los pasos a seguir para la realización de una auditoría de seguridad.

Para la parte práctica, se prepararán dos entornos utilizando sistemas operativos iOS y Android. El primero servirá para analizar el binario.ipa de la aplicación Damn Vulnerable iOS App (DVIA) y el segundo, para analizar el binario .apk de la aplicación Damn Insecure and Vulnerable App (DIVA).

Con esta información trataremos de realizar un compendio de pasos y buenas prácticas para completar una auditoría de aplicaciones móviles en las mejores condiciones.

1.4. Listado de Tareas

Para alcanzar el objetivo principal de este TFM y el resto de los objetivos enumerados se ha previsto la ejecución de las siguientes tareas:

#	Tarea
1	Revisión de esquema y estructura del TFM
2	Recopilación de Información de fuentes públicas
3	Documentación previa y listado de tareas
4	Redacción sobre Introducción, objetivos, metodología, estado del arte y riesgos
5	Corrección de errores

6	Entrega PEC1
7	Revisión y Redacción de Contenido de TFM sobre arquitectura de aplicaciones móviles, auditorías de seguridad y metodología
8	Corrección de errores
9	Revisión y Redacción sobre fases de la auditoría de seguridad, preparación del entorno y bypass de protecciones
10	Corrección de errores
11	Entrega PEC-2
12	Caso Práctico de estudio: auditoría de aplicaciones para iOS y Android
13	Conclusiones finales
14	Corrección de errores
15	Entrega PEC-3
16	Edición y Mejora de cada apartado
17	Revisión de Memoria Final
18	Corrección de errores
19	Entrega de PEC-4
20	Elaboración y edición de video
21	Corrección de errores
22	Presentación de Video
23	Preparación de Defensa de TFM
24	Defensa de TFM

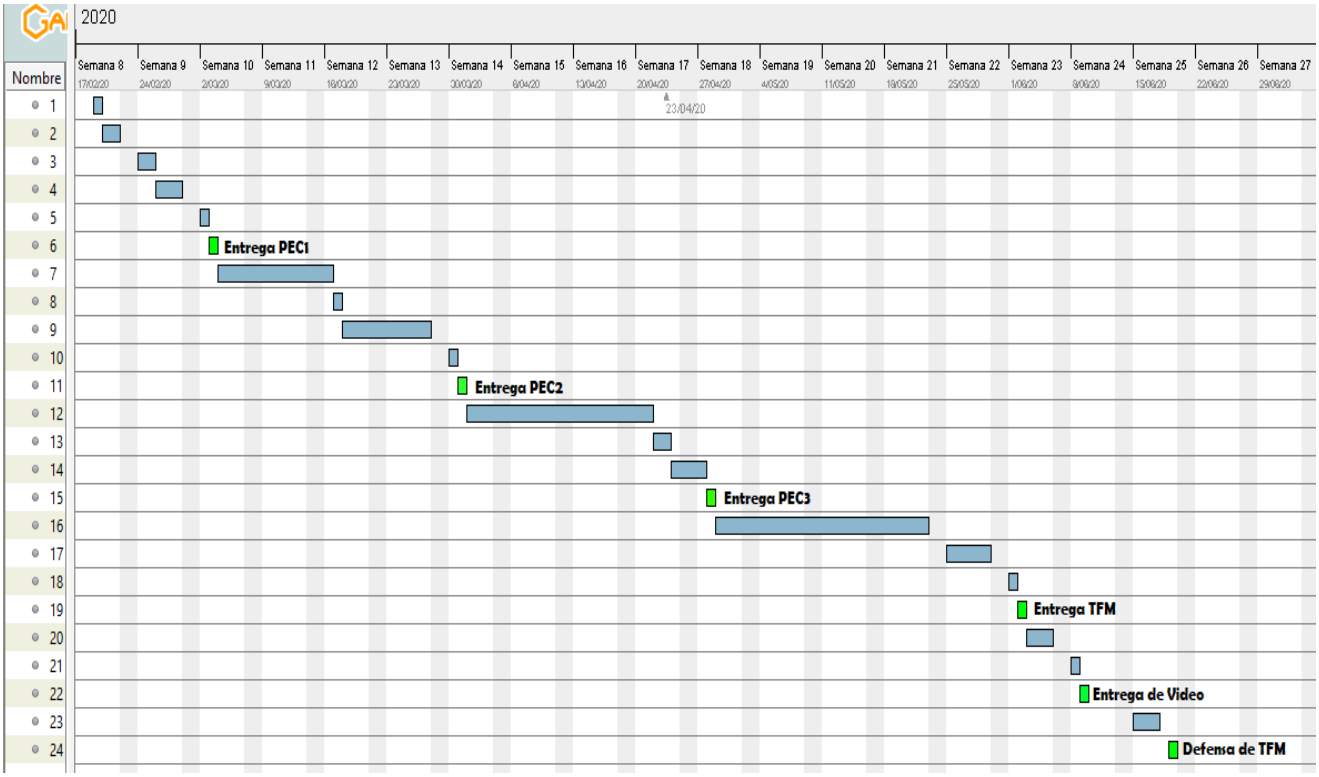
1.5. Planificación

Se seguirá la siguiente planificación en la cual se tendrá en cuenta los hitos más importantes:

#	Tarea	Duración	Inicio	Fin
0	Trabajo Fin de Máster	**84**	19/02/2020	19/06/2020
	PEC1 - Desarrollo Teórico	**10**	19/02/2020	03/03/2020
1	Revisión de esquema y estructura del TFM	1	19/02/2020	19/02/2020
2	Recopilación de Información de fuentes públicas	2	20/02/2020	21/02/2020
3	Documentación previa y listado de tareas	2	24/02/2020	25/02/2020
4	Redacción sobre Introducción, objetivos, metodología, estado del arte y riesgos	3	26/02/2020	28/02/2020
5	1era. Corrección de errores	1	02/03/2020	02/03/2020
6	Entrega PEC1	1	03/03/2020	03/03/2020

	PEC2 - Desarrollo Teórico	**20**	04/03/2020	31/03/2020
7	Revisión y Redacción de Contenido de TFM sobre arquitectura de aplicaciones móviles, auditorías de seguridad y metodología	9	04/03/2020	16/03/2020
8	2da. Corrección de errores	1	17/03/2020	17/03/2020
9	Revisión y Redacción sobre fases de la auditoría de seguridad, preparación del entorno y bypass de protecciones	8	18/03/2020	27/03/2020
10	3era. Corrección de errores	1	30/03/2020	30/03/2020
11	Entrega PEC2	1	31/03/2020	31/03/2020
	PEC3 - Desarrollo de Práctica	**20**	01/04/2020	28/04/2020
12	Caso Práctico de estudio: auditoría de aplicaciones para iOS y Android	15	01/04/2020	21/04/2020
13	Conclusiones finales	2	22/04/2020	23/04/2020
14	4ta. Corrección de errores	2	24/04/2020	27/04/2020
15	Entrega PEC3	1	28/04/2020	28/04/2020
	PEC4 - Entrega de Memoria Final	**25**	29/04/2020	02/06/2020
16	Edición y Mejora de cada apartado	18	29/04/2020	22/05/2020
17	Revisión de Memoria Final	5	25/05/2020	29/05/2020
18	5ta. Corrección de errores	1	01/06/2020	01/06/2020
19	Entrega de TFM	1	02/06/2020	02/06/2020
	Presentación de Video	**5**	03/06/2020	09/06/2020
20	Elaboración y edición de video	3	03/06/2020	05/06/2020
21	Corrección de errores	1	08/06/2020	08/06/2020
22	Presentación de Video	1	09/06/2020	09/06/2020
	Cierre de Proyecto	**4**	15/06/2020	19/06/2020
23	Preparación de Defensa de TFM	3	15/06/2020	17/06/2020
24	Defensa de TFM	1	19/06/2020	19/06/2020

Se detalla la siguiente planificación temporal mediante Diagrama de Grantt:



1.6. Estado del Arte

El Estado del Arte muestra el avance del conocimiento científico logrado en un área o temática concreta y que se encuentra plasmado en libros, artículos de revistas científicas, trabajos de investigación de instituciones educativas o laboratorios, etc.

Existen proyectos, previos a la realización de este TFM, que han ayudado a la elaboración de una metodología de trabajo. Estos proyectos están vinculados con entidades reconocidas internacionalmente como lo son OWASP y NIST.

En este TFM, se incluirá el estado del Arte sobre el análisis de seguridad en aplicaciones para dispositivos móviles con iOS y Android, revisando las vulnerabilidades más comunes que puedan afectar a este tipo de plataformas.

Se configurará un entorno de Auditoría, se detallará su preparación, el uso adecuado de diversas herramientas especializadas, así como la puesta en marcha de ciertas técnicas para la evasión de controles que puedan alterar los resultados de la auditoría de seguridad.

Se describirán, además, determinadas acciones preventivas y se plantearán algunas recomendaciones para la mitigación de riesgos de seguridad tanto a nivel de Usuario como de Desarrollador de aplicaciones para iOS y Android.

Guías de OWASP y NIST

1) Identificación:

Autores: Josh Franklin, Tom Karygiannis, Michael Ogata, Steve Quirolgico y Jeffrey Voas.
Título: Vetting the Security of Mobile Applications
Edición: Enero del 2015
Citación: Special Publication (NIST SP) - 800-163
Referencia:
https://www.nist.gov/publications/vetting-security-mobile-applications
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-163.pdf

Objetivos Generales:
La motivación principal del estándar es la verificación de seguridad de la aplicación, que el documento reconoce que puede tener lugar durante el desarrollo de la aplicación, el post-desarrollo, pero antes del despliegue y durante la difusión de una aplicación en los teléfonos móviles de una organización.

El propósito de este documento, por tanto, es ayudar a las organizaciones a:

- Comprender el proceso para examinar la seguridad de las aplicaciones móviles.
- Planificar la implementación de un proceso de investigación de aplicaciones.
- Desarrollar los requisitos de seguridad de la aplicación.
- Comprender los tipos de vulnerabilidades de la aplicación y los métodos de prueba utilizados para detectar esas vulnerabilidades.
- Determinar si una aplicación es aceptable para su implementación en los dispositivos móviles de la organización.

Muestra:
Los requisitos de desarrollo de aplicaciones detallados se han elegido como parte de una estrategia de verificación más amplia, con prácticas de desarrollo sólidas que dan confianza en la aplicación implementada, así como los métodos y herramientas de prueba empleados antes y durante la difusión de la aplicación.

Para ello contaron con importantes consultores, especialistas, y organizaciones vinculadas con el mundo de la Ciberseguridad.

Actualización:

La publicación especial NIST (SP) 800-163 Revisión 1 , Vetting the Security of Mobile Applications, (19 de abril 2019) es una actualización importante de la guía NIST sobre investigación y seguridad de aplicaciones móviles. El documento original (enero de 2015) detallaba los procesos a través de los cuales las organizaciones evalúan las aplicaciones móviles para detectar vulnerabilidades de ciberseguridad.

La revisión 1 amplía el documento original al explorar los recursos que se pueden utilizar para informar los requisitos de una organización para la seguridad de las aplicaciones móviles. Estos incluyen resúmenes de la documentación relevante de la National Information Assurance Partnership (NIAP), el Open Web Application Security Project (OWASP), The MITRE Corporation y NIST.

2) **Identificación:**

Autores: Bernhard Mueller, Sven Schleier, Jeroen Willemsen y el equipo de OWASP.
Título: OWASP Mobile AppSec Verification Standard (MASVS)
Edición: Enero del 2018
Referencia:
https://owasp.org/www-project-mobile-security/
https://github.com/OWASP/owasp-masvs/releases/tag/1.1.4

Objetivos Generales:

El objetivo general de MASVS es establecer los requisitos básicos de seguridad para garantizar la integridad y la coherencia en las pruebas de penetración de aplicaciones móviles;

El MASVS está destinado a lograr lo siguiente:

- Proporcionar requisitos para los arquitectos y desarrolladores de software que buscan desarrollar aplicaciones móviles seguras.
- Ofrecer un estándar de la industria que se pueda probar en las revisiones de seguridad de aplicaciones móviles.

- Aclarar el papel de los mecanismos de protección de software en la seguridad móvil y proporcionar requisitos para verificar su eficacia.
- Proporcionar recomendaciones específicas sobre qué nivel de seguridad se recomienda para diferentes casos de uso.

Muestra:
Los requisitos de desarrollo de aplicaciones detallados se han elegido como parte de una estrategia de verificación más amplia, con prácticas de desarrollo sólidas que dan confianza en la aplicación implementada, así como los métodos y herramientas de prueba empleados antes y durante la difusión de la aplicación.

Para ello contaron con importantes consultores, especialistas, y organizaciones vinculadas con el mundo de la Ciberseguridad.

3) **Identificación:**

Autores: Bernhard Mueller, Sven Schleier, Jeroen Willemsen y el equipo de OWASP.
Título: OWASP Mobile Security Testing Guide (MSTG)
Edición: Junio del 2018
Referencia:
https://github.com/OWASP/owasp-mstg
https://owasp.org/www-project-mobile-security-testing-guide/

Objetivos Generales:
OWASP desarrolló la Guía de prueba de seguridad móvil (MSTG) para ayudar a los analistas a aprender cómo probar áreas específicas de seguridad de aplicaciones móviles. Este es un gran recurso, pero tiene el tamaño de una guía telefónica. MSTG describe los procesos técnicos para verificar los controles en el Estándar de Verificación de Aplicaciones móviles (MASVS) y, además sostiene:

- Pruebas de seguridad en el ciclo de vida de desarrollo de aplicaciones móviles
- Pruebas básicas de seguridad estática y dinámica.
- Aplicación móvil de ingeniería inversa y manipulación
- Evaluar las protecciones de software
- Casos de prueba detallados que se corresponden con los requisitos en MASVS.

Muestra:

Los requisitos de desarrollo de aplicaciones detallados se han elegido como parte de una estrategia de verificación más amplia, con prácticas de desarrollo sólidas que dan confianza en la aplicación implementada, así como los métodos y herramientas de prueba empleados antes y durante la difusión de la aplicación.

Para ello contaron con importantes consultores, especialistas, y organizaciones vinculadas con el mundo de la Ciberseguridad.

1.7. Análisis de Riesgo Preliminar

Se enumera a continuación los principales riesgos que podrían afectar, de forma temporal, el desarrollo del TFM. Para cada riesgo identificado se ha valorado la probabilidad y el impacto de la materialización del riesgo y, además, se proponen algunas acciones para su correcta mitigación.

Riesgo 1 – Contenido demasiado extenso

Riesgo: Errores a la hora de sintetizar todo el contenido propio de una Auditoría de Seguridad podrían provocar que se pierda de vista los objetivos principales y se intente profundizar en temas secundarios, que poco aporten al enriquecimiento del trabajo final.

Probabilidad	3	**Impacto**	5

Acción Mitigadora: Simplificar, siempre que sea posible, descartar temas secundarios o auxiliares, revisar de forma periódica la evolución del contenido y su enfoque. En último caso, eliminar dicho contenido.

Riesgo 2 – Información insuficiente sobre alguna de las herramientas o técnicas

Riesgo: Poca información sobre alguna de las herramientas utilizadas o técnicas implementadas, podrían afectar la documentación y desarrollo del proyecto.

Probabilidad	3	**Impacto**	2

Acción Mitigadora: Recurrir a búsqueda de información adicional en Internet o la propia Deep Web. Utilización de foros de seguridad, blogs y o publicaciones relacionadas pueden ser otro mecanismo que ayude a la documentación y enriquecimiento del proyecto. En último caso, sustitución de la herramienta o técnica.

Riesgo 3 – Actualizaciones en plataforma iOS y Android dificulte ejecución del análisis
Riesgo: Cambios recientes o actualizaciones en los controles de seguridad de iOS o Android pueden producir alguna afectación en la configuración del entorno, o en la implementación de alguna de las herramientas necesarias o pruebas de seguridad preparadas.

Probabilidad	3	**Impacto**	4

Acción Mitigadora: Para evadir la continua evolución de los mecanismos de seguridad en plataformas iOS y Android, se optaría por el diseño de nuevas estrategias de ataque, sustitución de las pruebas de seguridad y herramientas inicialmente seleccionadas y en último caso, descartar ciertas pruebas de la metodología.

1.8. Entregables

Durante el Trabajo Final del Máster se proporcionarán los siguientes entregables:
PEC1.- Se incluye la revisión del Plan y la Estructura del TFM, además del Contexto, Objetivos y Estado del Arte. Se trata de una parte introductoria que proporciona una visión general de lo que será el proyecto.

PEC2.- Incluye la investigación, la topología y el Marco legal y ético. La investigación engloba la elección de herramientas para cubrir todos los aspectos implicados en el producto que se plantea como resultado del TFM. La topología consiste en un esquema donde se mostrará los diferentes componentes elegidos y su interacción dentro de la solución propuesta. El marco legal y ético de la solución es un espacio donde valorarás a nivel ético y legal las implicaciones de la implantación de la propuesta, considerando las posibles consecuencias que un mal uso podrían implicar.

PEC3.- Incluye un ejercicio práctico para demostrar el uso de una metodología de trabajo y el diseño de un informe que detalle los resultados obtenidos tras el análisis de aplicaciones ara dispositivos iOS y Android.

PEC4.- Se genera la memoria final del proyecto, en la que se incluye todos los apartados anteriores más Conclusiones, Bibliografía, Referencias y Anexos.

Entrega del Video Resumen.- Una presentación grabada, a alto nivel, que describe todo el trabajo realizado.

1.9. Recursos para realización de TFM

Para la ejecución de este proyecto se detallan a continuación los siguientes recursos:

Recurso	Roles	Coste
Ordenador Personal	Redacción de TFM Host para laboratorio con máquinas virtuales y emuladores	400€
Conexión a Internet	Consulta de otras fuentes	20€/mes
Terminal Físico iOS	Pruebas de Seguridad	300€
Terminal Físico Android	Pruebas de Seguridad	100€

2. Aplicaciones Móviles

Según Wikipedia una aplicación móvil, “es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Este tipo de aplicaciones permiten al usuario efectuar un variado conjunto de tareas -profesional, de ocio, educativas, de acceso a servicios, etc.-, facilitando las gestiones o actividades a desarrollar”.

Las aplicaciones móviles se han convertido en los últimos años, en una herramienta necesaria de uso diario. Es la mejor forma de visualizar una web y la manera más rápida de comprar un producto o buscar y contratar algún servicio desde cualquier parte y rápidamente.

Sus principales ventajas son la descarga sencilla, implementación rápida y facilidad de uso.

2.1. Tipos de Aplicaciones Móviles

El diseño de una aplicación móvil se basará en los criterios de ejecución que se han priorizado, es decir debe tenerse en cuenta si se ejecutará únicamente sobre una plataforma en concreto (iOS o Android), sobre un navegador web o interesa que pueda utilizarse desde cualquiera de estos dos entornos.

2.1.1. Aplicación Nativa

Una aplicación nativa es la que se desarrolla de forma específica para un determinado sistema operativo, llamado Software Development Kit o SDK. Es decir, si se necesita que la

aplicación esté disponible, tanto en iOS y Android, se deberá crear una aplicación por cada lenguaje, Java para el caso de Android y Objective-c/Swift para iOS.

Este tipo de aplicaciones no necesitan de conexión internet para que funcionen, y además nos proporcionan la posibilidad de aprovechar al máximo todos los recursos de hardware del dispositivo móvil (GPS, cámara, almacenamiento, sensores etc) lo que hace que la experiencia del usuario pueda ser muy positiva.

Ventajas	Desventajas
Acceso completo al dispositivo	Requiere mayor esfuerzo de Desarrollo
Actualizaciones de la aplicación constantes	Desarrollo de una aplicación por cada entorno
Mejor aprovechamiento de los recursos del dispositivo	Código fuente no es reutilizable entre plataformas

2.1.2. Aplicación Web

Una aplicación web es la desarrollada en lenguajes muy conocidos como son Javascript, HTML y CSS. La principal diferencia con respecto a las aplicaciones nativas es la posibilidad de desarrollarlas independientemente del sistema operativo donde se vaya a ejecutar. De esta forma se pueden ejecutar en distintos dispositivos sin necesidad de crear varias aplicaciones.

Este tipo de aplicaciones no necesitan de instalación ya que se ejecutan dentro del propio navegador web del dispositivo a través de una URL.

Ventajas	Desventajas
Proceso de Desarrollo más sencillo y económico	Requiere de acceso a Internet
No requieren aprobación de la tienda de iOS o Android para su publicación	Acceso muy limitado a las características de hardware del dispositivo
Los usuarios utilizarán la versión más reciente	El rendimiento será menor, dependerá del servidor Web

2.1.3. Aplicación Híbrida

Una aplicación híbrida es una combinación de las dos anteriores, se podría decir que recoge lo mejor de cada una de ellas. Este tipo de aplicaciones se desarrollan con lenguajes propios de las aplicaciones web, es decir Javascript, HTML, CSS por lo que permite su uso en diferentes plataformas, pero también dan la posibilidad de acceder a gran parte de las características del hardware del dispositivo. A pesar de estar desarrollada en HTML, CSS, Java, es posible agrupar el código fuente y distribuirla a través de App Store.

PhoneGap es uno de los frameworks más utilizados para el desarrollo multiplataforma de aplicaciones híbridas. Otro ejemplo para desarrollar aplicaciones híbridas es Cordova.

Ventajas	Desventajas
Es posible distribuirlas en tiendas de iOS y Android El mismo código base para múltiples plataformas Acceso a parte del hardware del dispositivo	Experiencia del usuario más propia de la aplicación web que de la aplicación nativa Diseño visual no siempre relacionado con el sistema operativo en el que se muestre

En este TFM, las aplicaciones de estudio elegidas son dos aplicaciones nativas para iOS y Android por lo que la guía de la auditoría de seguridad estará especialmente enfocada en estas dos aplicaciones.

2.2. Seguridad y Bypass en iOS

Los sistemas complejos siempre tendrán vulnerabilidades, y la complejidad del software sólo aumenta con el tiempo. No importa cuán cuidadosamente se adopte prácticas de codificación seguras y se proteja contra errores, los atacantes sólo necesitan superar sus defensas una vez para tener éxito.

Existen varios mecanismos de seguridad que, aunque no evitarán los ataques contra una aplicación al 100%, al menos dificultarán su gestación o minimizarán el daño que puede causar. Éstos son App Sandbox y SSL Pinning.

2.2.1. App Sandbox

App Sandbox es una tecnología de control de acceso proporcionada en Apple, aplicada a nivel del núcleo. Está diseñado para contener daños en el sistema y los datos del usuario si una aplicación se ve comprometida. Las aplicaciones distribuidas a través de Apple Store deben adoptar App Sandbox. Las aplicaciones firmadas y distribuidas fuera de Apple Store con ID de desarrollador deberían usar App Sandbox también.

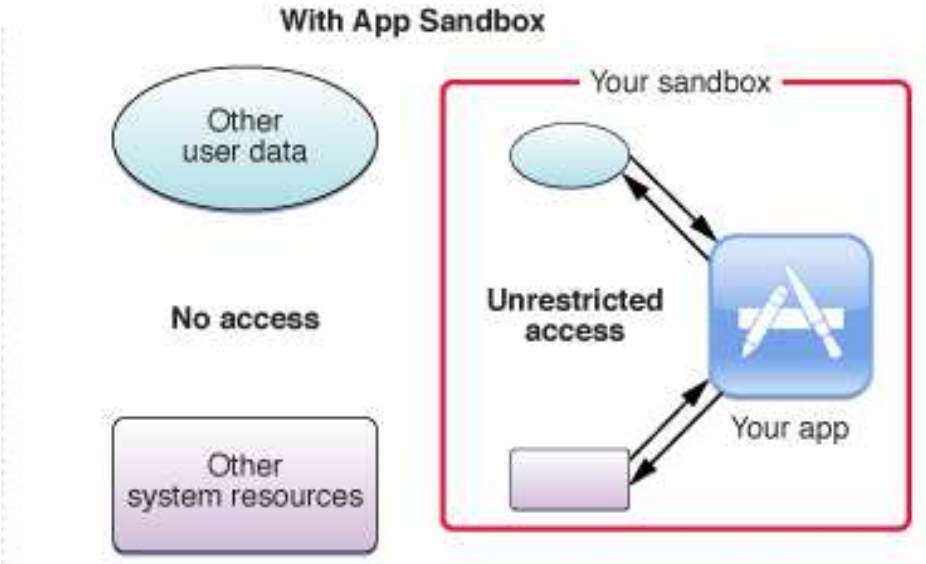


Figura 2: App Sandbox

Apple Developer

Con App Sandbox, el sistema rechaza el acceso a cualquier recurso no solicitado explícitamente en la definición del proyecto en tiempo de ejecución. Si está codificando una aplicación de Banking, por ejemplo, y sabe que su aplicación nunca necesitará acceso al micrófono, simplemente no solicita acceso, y el sistema sabe que rechazará cualquier intento que la aplicación, tal vez comprometida, haga eso.

2.2.2. SSL Pinning

El SSL Pinning se utiliza para garantizar que la aplicación se comunique sólo con el servidor designado. Uno de los requisitos previos para la SSL Pinning es guardar el certificado SSL del servidor de destino dentro del paquete de la aplicación. El certificado guardado se utiliza al definir los certificados anclados en la configuración de la sesión.

Figura 3: SSL Certificate Pinning

Guardsquare

No obstante, estos dos mecanismos de seguridad han sido vulnerados por técnicas agresivas de hacking, a nivel de Sistema Operativo, como lo son el Jailbreak y el Bypass de SSL Pinning

2.2.3. Jailbreak

Se denomina jailbreak al proceso de suprimir algunas de las limitaciones impuestas por Apple en dispositivos que utilicen el sistema operativo iOS mediante el uso de núcleos modificados. El jailbreak permite a los usuarios acceder por completo al sistema operativo, permitiendo al usuario descargar aplicaciones, extensiones y temas que no están disponibles a través de la App Store oficial.

Figura 4: Jailbreak: No App SandBox

Apple Developer

Un dispositivo con Jailbreak todavía puede usar la App Store, iTunes y todas las demás funciones, como hacer llamadas telefónicas. El Jailbreak es una forma de escalado de privilegios.

Tipos de Jailbreak

- Jailbreak Tethered es el método menos recomendado para desbloquear un terminal. El usuario deberá recurrir a un ordenador cada vez que el móvil se apague. Una vez se agote la batería o, se reinicie el terminal, el procesador de encendido se quedará a medias tintas no pasará del icono de la manzana. Para poder encenderlo correctamente se deberá enchufar a un PC y aplicar de nuevo el Jailbreak.

- Jailbreak Untethered es el más usado. Este método desbloquea el terminal, pero a diferencia del anterior método, éste sí que deja encender con normalidad el equipo de Apple y seguir funcionando sin ningún tipo de restricción. El Jailbreak estará activo hasta que el cliente decida restaurar el sistema de nuevo.

- Jailbreak Semitethered es el método más nuevo de todos. También desbloquea el terminal, pero si se termina la batería o el dispositivo se reinicia, el terminal iniciará con normalidad y únicamente dejará hacer o recibir llamadas o enviar y recibir mensajes cortos de textos (SMS). Las aplicaciones que hayan sido descargadas desde Cydia o alguna tienda alternativa no estarán disponibles.

2.2.4. Bypass del SSL Pinning

Es posible evitar el SSL Pinning mediante el uso de herramientas como SSL Kill Switch. Sin embargo, sólo está disponible para dispositivos con Jailbreak.

Figura 5: Deshabilitar SSL Pinning en iOS con SSL Kill Switch

Working with BurpSuite MobileAssistant Tool

Una vez cargado en una aplicación iOS, SSL Kill Switch parchea funciones SSL específicas de bajo nivel dentro de la API de transporte seguro para anular y deshabilitar la validación de certificado predeterminada del sistema, así como cualquier tipo de validación de certificado personalizada (SSL Pinning). Es posible también evadir el SSL Pinning con el uso de otras herramientas como Frida.

2.3. Seguridad y Bypass en Android

Los avances de la plataforma Android conseguidos con el paso de los años y la llegada de nuevas versiones del sistema han hecho que cada vez menos usuarios sientan la necesidad de realizar modificaciones avanzadas en el software de sus dispositivos. Una muy buena

opción para aumentar la seguridad en las aplicaciones es detectar bajo que entorno está siendo ejecutada la aplicación.

Existen varios mecanismos de seguridad que, aunque no evitarán los ataques contra una aplicación al 100%, al menos dificultarán su gestación o minimizarán el daño que puede causar. Éstos son App Sandbox y SSL Pinning.

2.3.1. App Sandbox

Ésto aísla las aplicaciones entre sí y protege a ellas y al sistema de aplicaciones maliciosas. Para hacer esto, Android asigna una identificación de usuario (UID) única a cada aplicación de Android y la ejecuta en su propio proceso.

Por defecto, las aplicaciones no pueden interactuar entre sí y tienen acceso limitado al sistema operativo. Si la aplicación A intenta hacer algo malicioso, como leer los datos de la aplicación B o marcar el teléfono sin permiso, se impide hacerlo porque no tiene los privilegios de usuario predeterminados apropiados.

2.3.2. SSL Pinning

El SSL Pinnng permite que la aplicación solo confíe en el certificado válido o predefinido o en la clave pública. El desarrollador de la aplicación utiliza la técnica de SSL Pinning como una capa de seguridad adicional para el tráfico de la aplicación. Como de costumbre, la aplicación confía en el certificado personalizado y permite que la aplicación intercepte el tráfico. Pero en la implementación del SSL Pinning, la aplicación no confía en los certificados personalizados y no permite que las herramientas proxy intercepten el tráfico.

No obstante, estos dos mecanismos de seguridad, App Sandbox y SSL Pinning, han sido vulnerados por técnicas agresivas de hacking a nivel de Sistema Operativo como lo son el Rooteo y el Bypass de SSL Pinning.

2.3.3. Rooteo

En Android, el rooteo se lleva a cabo con el objetivo de saltarse restricciones impuestas por el fabricante del dispositivo, o por la propia Google. De este modo, es posible llevar a cabo cambios en el sistema como la desinstalación de aplicaciones preinstaladas u otros más

sensibles como la modificación de ficheros del sistema, e incluso el reemplazo del propio sistema operativo por una versión alternativa, conocidas como ROMs de terceros.

2.3.4. Bypass del SSL Pinning

Evadir el SSL Pinning en una aplicación móvil se puede lograr con el uso de herramientas como Frida o Xposed, o descargando el APK original y modificando el archivo de configuración de seguridad de la red para confiar en los certificados proporcionados por el usuario y para desactivar el SSL Pinning. Después de la modificación, es necesario volver a empaquetar la aplicación e instalarla nuevamente en el dispositivo, después de lo cual es posible interceptar nuevamente el tráfico con un ataque Man in the Middle

Otra herramienta que es posible utilizar para evadir el SSL Pinning es Inspeckage.

2.4. Vulnerabilidades en Aplicaciones Móviles

El elevado número de descargas de aplicaciones móviles desde sitios oficiales o no oficiales, que se realizan a diario, han generado un crecimiento exponencial de las amenazas y malware, diseñados específicamente para los dispositivos móviles, y en especial para las plataformas más utilizadas como son iOS y Android.

Estudios recientes realizados, por Arxan Technology, uno de los proveedores más importantes de aplicaciones móviles, nos demuestran que el 90% de las aplicaciones analizadas son vulnerables a al menos a DOS de los DIEZ riesgos publicados en el OWASP TOP 10. Otros estudios de Arxan Tehcnology, además, nos indican que las aplicaciones híbridas, son más vulnerables a ciberataques, ya que el código base se usa en otras plataformas como la web.

Si a ello le añadimos los problemas de seguridad que padecen los propios sistemas operativos, nos lleva a tener muy presente los riesgos que estamos, a diario, afrontando. Según una encuesta realizada en Marzo del 2020, por **Which**, una de las organizaciones de Consumidores más importantes de Reino Unido, 2 de cada 5 usuarios de teléfonos móviles, con sistema operativo Android, son vulnerables a ser hackeadas o infectados por malware malicioso. Es decir, más de 1.000 millones de dispositivos Android de todo el mundo son vulnerables a ciberataques porque no reciben soporte ni actualizaciones de Google.

Figura 6: OWASP Mobile RisksTop10

Desarrollo seguro de aplicaciones para dispositivos móviles

OWASP, una organización sin ánimo de lucro, publica el Mobile Risks Top 10 que es una lista que identifica los tipos de riesgos de seguridad que enfrentan las aplicaciones móviles. Esta lista, que se actualizó en 2016, es una guía práctica para que los desarrolladores creen aplicaciones seguras e incorporen las mejores prácticas de codificación.

2.4.1. Métrica de Vulnerabilidad

El Sistema de puntuación de vulnerabilidad común (CVSS) es un marco abierto para comunicar las características y la gravedad de las vulnerabilidades de software. CVSS consta de tres grupos métricos: base, temporal y ambiental. Las métricas base producen una puntuación que va de 0 a 10. Dos usos comunes de CVSS son calcular la gravedad de las vulnerabilidades descubiertas en los sistemas y como un factor en la priorización de las actividades de su mitigación.

NVD (Base de Datos de Vulnerabilidades) proporciona clasificaciones de gravedad cualitativas de "Bajo", "Medio" y "Alto" para los rangos de puntaje base de CVSS v2.0 además de las clasificaciones de gravedad para CVSS v3.0.

Clasificaciones CVSS v2.0		Clasificaciones CVSS v3.0	
Gravedad	**Rango de puntaje base**	**Gravedad**	**Rango de puntaje base**
		Ninguna	0.0
Bajo	0.0-3.9	Bajo	0.1-3.9
Medio	4.0-6.9	Medio	4.0-6.9
Alto	7.0-10.0	Alto	7.0-8.9
		Crítico	9.0-10.0

Figura 7: Métrica de Vulnerabilidad

(NVD) National Vulnerability Database

2.5. Auditoría de Seguridad

Una auditoría de seguridad es un proceso riguroso, basado en una metodología, por el cual se identifican las vulnerabilidades en una aplicación móvil. Ésta puede ser:

- **Caja Negra:** No se proporciona información al analista a cerca de la aplicación
- **Caja Gris:** Se proporciona algo de información al analista
- **Caja Blanca:** Se proporcionan credenciales de acceso a la aplicación

Durante la auditoría, es posible el uso de herramientas automáticas como en auditorías de otro tipo, sin embargo, en el informe final de vulnerabilidades se deben incluir, además, las descubiertas de forma manual, producto de la experiencia del analista de Seguridad.

2.6. Metodología para Auditoría de Seguridad

Para la realización de Auditorías de Aplicación Móvil, se organiza la auditoría siguiendo las pautas definidas por la guía de pruebas de OWASP Mobile Security Project.

El proyecto de seguridad móvil de OWASP es un recurso centralizado destinado a proporcionar a desarrolladores y equipos de seguridad los recursos que necesitan para

construir y mantener aplicaciones móviles seguras. El objetivo es clasificar los riesgos de seguridad móvil y proporcionar controles para reducir su impacto o posibilidad de explotación.

Entre esos recursos más importantes, que se ponen a disposición, para una adecuada realización de una auditoría de seguridad, se encuentran:

MASVS	Mobile Application Security Verification Standard	Un estándar para la seguridad de aplicaciones móviles que describe los requisitos de seguridad de una aplicación móvil.
MSCH	Mobile Security Checklist	Una lista de verificación que permite mapear y puntuar fácilmente los requisitos del Estándar de verificación de seguridad de aplicaciones móviles basado en la Guía de pruebas de seguridad móvil.
MSTG	Mobile Security Testing Guide	Un manual completo para pruebas de seguridad de aplicaciones móviles e ingeniería inversa para probadores de seguridad móvil iOS y Android, así como para desarrolladores.
Top10	Mobile Top Ten Risks Mobile Top Ten Controls	Resume los 10 riesgos más importantes Resume los 10 controles más importantes

Esta guía va destinada, entre otros, a profesionales de la seguridad involucrados para el análisis de seguridad de aplicaciones móviles, proporcionando una serie de procedimientos, recursos y herramientas que garantizan un análisis exhaustivo de la plataforma. Centrándonos en este framework podemos dividir la auditoría entre fases:

- Recopilar información
- Análisis Estático
- Análisis Dinámico

2.6.1. Recopilación de Información

Para una adecuada ejecución de toda auditoría de seguridad, es imprescindible se invierta recursos y tiempo en un exhaustivo conocimiento de la aplicación a auditar. El objetivo de la aplicación, las diversas funcionalidades activas, login y logout, mecanismos de protección, almacenamiento en local, son algunos de los vectores a revisar durante esta etapa.

En el caso de Android, es ideal el ayudarse de herramientas que realicen un reconocimiento de la aplicación de forma automática distinguiendo los permisos y el resto de superficie de ataque como son los Activities, Intent, Broadcast Receivers, Content Providers y Services.

Activities (Actividades): son los componentes básicos de la aplicación que proporcionan una interfaz para el usuario, una pantalla única que puede alojar elementos de la interfaz de usuario. Una aplicación generalmente proporciona una o más actividades y permite al usuario navegar entre cada una de ellas.

Services (servicios): Los servicios son componentes de aplicaciones que se utilizan principalmente para tareas de procesamiento en segundo plano, por ejemplo, reproducir música, descargar archivos o realizar algunos cálculos que requieren mucho tiempo.

Broadcast Receivers (receptores de Difusión): Los receptores de difusión permiten intercambiar eventos entre los componentes de la aplicación o incluso entre diferentes aplicaciones. Con los receptores de difusión, los eventos se pueden entregar incluso a aplicaciones que no se están ejecutando actualmente (por ejemplo, la aplicación puede escuchar eventos del sistema). Android usa Intentos para entregar estos eventos a los receptores de transmisión.

Content Providers (Proveedores de Contenido): Los proveedores de contenido permiten que una aplicación de Android administre y comparta datos. Proporcionan una interfaz unificada para administrar los datos de la aplicación que permite utilizar un esquema de URI para asignar URI a elementos de datos.

Intent: Un Intent es un objeto de mensajería que puedes usar para solicitar una acción de otro componente de una app. Si bien las intents facilitan la comunicación entre componentes de varias formas, existen tres casos de uso principales: iniciar un activity, iniciar un service y transmitir una emisión.

Permisos: Cuando una aplicación quiere acceder a recursos o diversas capacidades del dispositivo, a menudo tiene que solicitar permisos del usuario para hacerlo. El usuario otorga algunos permisos al instalar la aplicación y algunos deben confirmarse adicionalmente mientras se ejecuta una aplicación. Los permisos solicitados se declaran en el archivo AndroidManifest.xml de la aplicación.

Android Manifest.xml: El archivo manifest.xml de Android contiene información importante sobre la aplicación que utilizan las herramientas de desarrollo, el sistema Android y las tiendas de aplicaciones. Contiene el nombre del paquete de la aplicación, la información de la versión,

las declaraciones de los componentes de la aplicación, los permisos solicitados y otras cosas importantes. Se serializa en formato binario xml y se incluye dentro del archivo APK.

Para la plataforma iOS, es ideal el ayudarse de herramientas que realicen un reconocimiento automático distinguiendo los permisos y el resto de superficie de ataque.

Payload: contiene la carpeta .app de la aplicación iOS específica. En la carpeta .app podemos ver el contenido de la aplicación, como las imágenes, los archivos nib que almacenan la interfaz de usuario, etc.

Mach-O Executable: los archivos de objetos Mach son formatos de archivo para ejecutables. Contiene sección de datos, encabezado y comandos de carga.

Info.plist: almacena la información de configuración del ejecutable. Se puede ver con un editor de texto. Si está en formato binario, se puede convertir usando:
plutil -convert xml1 Info.plist

Frameworks: carpeta con bibliotecas que usa la aplicación. Hay muchas bibliotecas de terceros. Por ejemplo, el SDK de AWS.

MobileProvision: puede encontrar información como el certificado de desarrollador, los dispositivos para los que se aprovisiona la aplicación o el identificador del equipo enembedded.mobileprovision

2.6.2. Análisis Estático

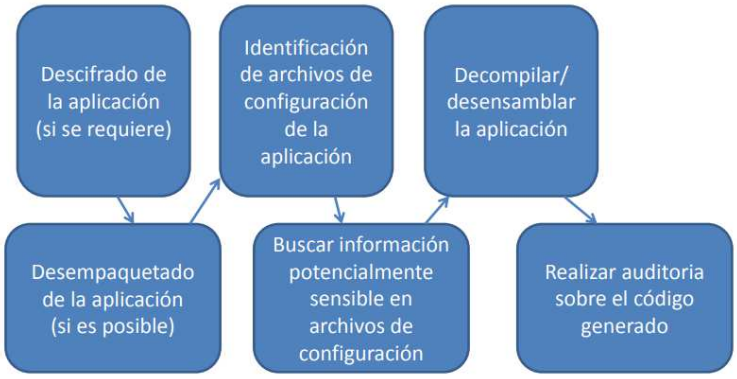


Figura 8: Tareas de un Análisis Estático

Argentesting-2019 Analizando la seguridad en Aplicaciones Móviles

El análisis estático o también denominado SAST (Static Aplication Security Testing) consiste en la revisión del código fuente de una aplicación, analizando los distintos componentes de ésta, ya sea de forma manual o automática.

En esta etapa se llevará a cabo una revisión del código fuente de la aplicación, y puede realizarse desde cualquiera de los siguientes puntos de partida:

Obteniendo el código fuente disponible y proporcionado por el equipo de desarrollo de la aplicación. En caso de ser una aplicación open-source, se accedería a su repositorio público. Utilizando ingeniería inversa, tras descarga del binario de las tiendas de iOS y Android, de forma que se obtiene el código de la aplicación utilizando decompiladores y herramientas de manipulación a bajo nivel.

Desde el punto de la seguridad los mejores resultados de una auditoria de aplicaciones móviles se suceden cuando se tiene pleno acceso al código fuente de la aplicación, y se dispone de un entorno de pruebas para la ocasión.

Estas revisiones de código, que deben ser realizadas por personal experimentado que conozca los lenguajes de programación utilizados, son esenciales porque ayudan a la identificación de potenciales vulnerabilidades en la lógica de la aplicación, malas prácticas de Desarrollo Seguro e incluso posibles fallos de Diseño, escenarios difíciles de identificar con herramientas automáticas.

A continuación, se detallan las tareas que se incluyen un Análisis Estático para aplicaciones móviles.

- Requerimientos necesarios para la Auditoria de Código
- 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

2.6.3. Análisis Dinámico

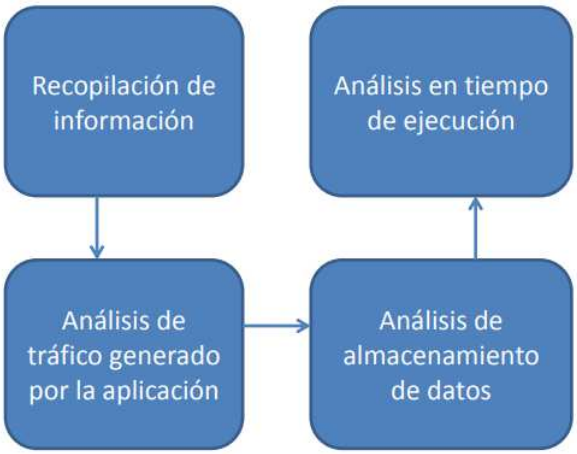


Figura 9: Tareas de un Análisis Dinámico

Argentesting-2019 Analizando la seguridad en Aplicaciones Móviles

El análisis Dinámico o también conocido como Dynamic Aplication Security Testing (DAST) consiste en el análisis de seguridad de una aplicación durante su propia ejecución, con ello es posible descubrir otro tipo de vulnerabilidades en tiempo real.

A diferencia del análisis estático, proporciona información con relación a los recursos, las características de la aplicación, los puntos de entrada desde el punto de vista del usuario e incluso nos sirve para comprobar vulnerabilidades descubiertas durante el análisis estático.

Se realiza un análisis de la aplicación de manera que se pueda decidir a qué tipos de ataques es sensible. Para hacer ésto se analizan las tecnologías empleadas, plataformas, lenguajes de programación, accesos a Bases de Datos, relaciones entre sitios web, etc. de esta forma se incluyen y definen los tests o pruebas específicas a realizar.

A través de diversas técnicas de análisis de seguridad es posible evaluar la seguridad de las aplicaciones con el objetivo de garantizar tanto la seguridad de sus usuarios, como la comunicación con otros activos de seguridad de la empresa.

En algunos casos, los métodos utilizados son exclusivos de un tipo de aplicación o a las necesidades específicas de nuestro cliente, pero en general se despliega una combinación personalizada para obtener un análisis completo y relevante de la aplicación móvil.

Este tipo de análisis llevará a cabo una revisión global de la aplicación y comprende:

- Análisis a Nivel de Aplicación Cliente
- Análisis a Nivel de Comunicaciones
- Análisis a Nivel de Servidor, Backend y API

El análisis dinámico se analizarán los patrones de las distintas solicitudes y respuestas de la aplicación móvil y se comprobará si los mecanismos de seguridad implementados protegen adecuadamente frente a los ataques más comunes: exfiltración de datos, exposición de información de la plataforma, autenticación y autorización deficiente, errores en la configuración del servidor, etc.

2.7. Entorno de Auditoría

Para llevar a cabo una auditoría de seguridad, el auditor deberá disponer de un entorno con **jailbreak o rooteado**, es de decir con acceso root. Lo podrá conseguir a través de:

- Emuladores
- Uso de frameworks o distribuciones especializadas
- Desde el propio dispositivo móvil Android o iOS

2.7.1. Emuladores

Según Wikipedia, un emulador es un software que permite ejecutar una plataforma (sea una arquitectura de hardware o un sistema operativo) diferente de aquella para la cual fueron escritos originalmente. A diferencia de un simulador, que sólo trata de reproducir el comportamiento del programa, un emulador trata de modelar de forma precisa el dispositivo de manera que este funcione como si estuviese siendo usado en el aparato original.

Para el caso de Android, el emulador será instalado en un host desde el cual se administrará el dispositivo gracias a ADB, que son las siglas de Android Debug Bridge, un sistema de comandos que permite administrar un dispositivo desde el ordenador

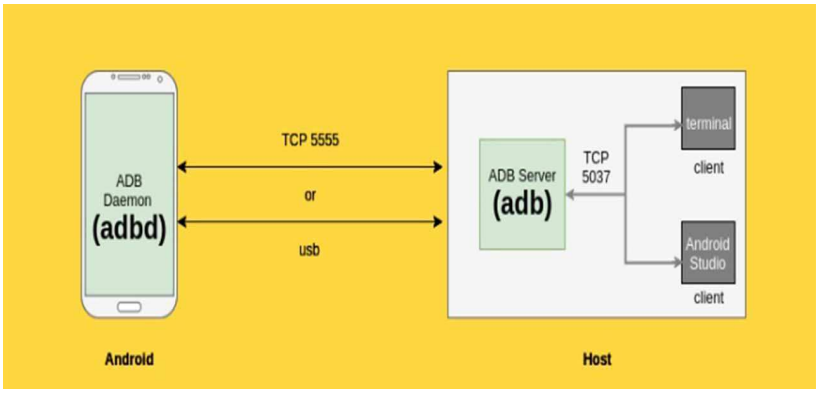


Figura 10: Terminal de comandos ADB

Terminal Commands used in Android Debug Bridge (AD)

Ventajas en el uso de Emuladores

- Posibilidad de ejecutar una aplicación en diferentes dispositivos que tengan el mismo sistema operativo.
- Hacer pruebas y análisis de nuevas aplicaciones.
- Evaluar aplicaciones antes de ser lanzadas al usuario final.
- Practicar desarrollo usando Xcode, Android Studio.

Para el caso de Android tenemos aplicaciones como Genymotion, Android Tamer, basadas en Virtualbox o el propio Android Emulator, sin embargo cuando se busca un emulador de iPhone o iPad para ejecutar aplicaciones iOS y llevar a cabo todas las funciones como de un móvil o tablet de Apple se tratase, lo cierto es que no es algo sencillo, de ahí que hayan proliferado lo que se conoce como “simuladores de iOS” que no son más que programas que simular un entorno de iOS y que sirven para que nos hagamos una idea de lo que podemos esperar de un iPhone o un iPad, pero en ningún caso emulan el hardware ni nos dejan ejecutar más aplicaciones de las que traen de serie.

Por lo contrario, un emulador es un programa que busca recrear virtualmente el hardware y software de un iPhone o un iPad. En este caso, sí que permiten instalar y ejecutar aplicaciones de iPhone en un PC con Windows. Las limitaciones vienen del lado de la compatibilidad, ya que es mucho más difícil programar un emulador de iOS que un simple simulador.

Realmente no existe ningún emulador realmente funcional que permita descargar aplicaciones directamente de App Store, y menos de forma gratuita al 100%. Lo más parecido

a ello es iPadian o el ya descontinuado MobiOne Studio, pero las aplicaciones que son posible instalar, en el primero, son únicamente webapps.

2.7.2. Frameworks o distribuciones especializadas

Androl4b es una máquina virtual segura basada en Ubuntu Mate diseñada para permitir a los auditores realizar un análisis exhaustivo de las aplicaciones para Android. Esta máquina virtual cuenta por defecto con un gran número de aplicaciones, herramientas, frameworks e incluso tutoriales pensados especialmente para auditorías de aplicaciones móviles Android. Santoku, es otra plataforma recomendable para el análisis de aplicaciones Android.

Estos frameworks o distribuciones especializadas tienen integrada ADB, para el envío de comandos desde el host.

Al no existir un emulador para iOS totalmente funcional, a día de hoy, tampoco se encuentran disponibles frameworks o distribuciones diseñadas para auditar este tipo de plataformas.

2.7.3. Móvil Android / iOS

La mejor opción, sin duda, para auditar aplicaciones es trabajar directamente desde el propio dispositivo donde se encuentra instalada, sin embargo, es la opción más costosa dado que se necesitaría un mantenimiento periódico de la plataforma o incluso sustitución del dispositivo cada cierto tiempo.

Para entornos iOS es considerada como la única opción válida para auditar dada la actualización continua de la cadena de seguridad por parte de Apple.

La comunicación de comandos con los dispositivos físicos es través de ADB, en el caso de Android, y SSH para el caso de iOS.

2.8. Herramientas para Auditoría

Para la ejecución de una auditoría de seguridad, ya sea en iOS o Android se dispone de un arsenal completo de recursos open source básicos. Entre los recursos indispensables están:

Mobile Security Framework (MobSF): es la herramienta, la que se recomienda abiertamente para el análisis dinámico y estático de aplicaciones, independiente de la plataforma (Android o iOS).

Figura 11: Plataforma Mobsf

MobSF/Mobile-Security-Framework-MobSF

Frida, una herramienta que inyecta código JavaScript para permitirnos analizar aplicaciones de Windows, macOS, Linux, iOS, Android y QNX. Indispensable para aplicar técnicas de hooking.

Figura 12: Aplicación de Frida

Frida A world-class Dynamic Instrumentation Framework

Burp Suite es un conjunto de herramientas basada en java que nos permite comprobar la seguridad de aplicaciones web y móviles. Esta suite consiste en un servidor proxy para analizar las peticiones y respuestas http, un crawler, y funcionalidades adicionales para descubrir vulnerabilidades e incluso la ejecución de un test de Intrusión.

Figura 13: Aplicación de Burpsuite

Burpsuite - Cybersecurity Software

2.8.1. Herramientas para el Análisis Estático

Existe una variedad importante de herramientas para la ejecución del Análisis Estático, entre las principales:

MARA es un Mobile Application Reverse Engineering and Analysis Framework. Es una framework para Android que reúne varias herramientas de análisis e ingeniería inversa de aplicaciones móviles de uso común, para ayudar a probar aplicaciones móviles contra las amenazas de seguridad móvil de OWASP.
APK Easy Tool es una herramienta que te permite descompilar, compilar y firmar archivos APK de aplicaciones de Android en tu PC con Windows.
Radare2 (también conocido como r2) es un marco completo para ingeniería inversa y análisis de binarios; compuesto por un conjunto de pequeñas utilidades que se pueden usar juntas o independientemente de la línea de comando.

2.8.2. Herramientas para el Análisis Dinámico

Existe una variedad importante de herramientas para la ejecución del Análisis Dinámico, entre las principales:
Drozer, un framework para Android con multitud de herramientas y recursos de seguridad para Android.
Xposed: equivalente a hacer una inyección de código basada en Stub, pero sin modificaciones en el binario iOS o Android.
Inspeckage Android Package Inspector: análisis dinámico con ganchos de api, inicia actividades no exportadas y más. (Módulo Xposed)
Cycript permite a los desarrolladores explorar y modificar aplicaciones en ejecución en iOS o Mac OS X usando un híbrido de sintaxis Objective-C ++ y JavaScript a través de una consola interactiva que presenta resaltado de sintaxis y finalización de pestañas.
IDB es una herramienta para simplificar algunas tareas comunes para la investigación y pentesting en iOS. Originalmente había una versión de la línea de comandos de la herramienta, pero ya no está en desarrollo, por lo que debe obtener la versión GUI.

Adicionalmente, existen portales web que realizan estas dos tareas de forma automática:

[1] Quixxi Free Mobile App Vulnerability Scanner
https://vulnerabilitytest.quixxi.com/#/

[2] Ostorlab Continuos Mobile Application Security Testing
https://report.ostorlab.co/
[3] ImmuniWeb AI for Application Security
https://www.immuniweb.com/mobile/?id=tlGhtxVC

3. Marco Legal

3.1. Privacidad

La protección de la privacidad del usuario ha sido una constante en la historia de la humanidad y es esencial para garantizar el correcto y libre desarrollo del ser humano. En la medida en la que la obtención y tratamiento de datos de las personas se hace de una forma más automatizada, el número de datos disponibles se incrementa exponencialmente lo que ha permitido que las empresas puedan ofrecer a los usuarios servicios realmente útiles para ellos, personalizando en cada caso concreto gracias a esa información que explotan.

Precisamente gracias a la publicidad en los dispositivos móviles y los ingresos que ello supone para los desarrolladores de aplicaciones móviles, los usuarios pueden disfrutar gratuitamente de las funcionalidades que éstas ofrecen. No obstante, ello no justifica, en ningún ordenamiento jurídico, que puedan obviarse los derechos de privacidad de los usuarios.

Con independencia de la normativa existente, una de las principales preocupaciones del usuario es la falta de transparencia en cuanto, no sólo a los datos que se recogen a través de dispositivos móviles, sino quiénes son los que tienen acceso a esos datos y para qué los están usando. Esa falta de transparencia hace que en muchas ocasiones los usuarios se encuentren con un uso no esperado de sus datos por parte de alguna aplicación, provocando malestar en los usuarios y pérdida de credibilidad de las marcas.

3.1. RGPD

El 25 de mayo de 2018 entró en vigor el nuevo Reglamento general de protección de datos (RGPD), una reforma sobre la protección de datos a nivel europeo. El objetivo del RGPD es regular de manera uniforme el tratamiento de los datos, es decir unificar los principios de protección de datos de la Unión Europea.

La normativa afecta a todas las empresas o entidades que tratan datos personales como parte de las actividades de una de sus sucursales establecidas en la Unión Europea (UE), independientemente del lugar donde sean tratados los datos, y afecta a las empresas que están establecidas fuera de la UE, que ofrecen productos o servicios (de pago o gratuitos) y registran el comportamiento de las personas en la UE.

Las nuevas reglas tienen que estar presentes durante todo el desarrollo de una Aplicaciones Móvil, siempre se debe tener en cuenta la forma en la que se va a usar los datos de tus usuarios y además informarlos sobre el motivo de su uso. Los pasos a seguir para cumplir con la RPGD para aplicaciones móviles:

1) **Mapeo de datos**

Se debe tener claro en qué lugar de la Aplicaciones se recibirá la información de los usuarios, así como de dónde se obtiene la información y a dónde se destinará. Además, se debe informar a los usuarios del porqué se recogen sus datos personales.

2) **Seguridad**

La seguridad de los datos ya estaba establecida en la anterior ley, teniendo incluso que hacer una evaluación de impacto de la protección de esos datos (DPIA- Data Protection Impact Assessment). No es algo que vaya a afectar a todas las aplicaciones móviles, más bien solo aquellas en la que parezca que exista un riesgo importante en los derechos de los usuarios.

3) **Diseño por privacidad**

Los usuarios tendrán que aceptar los Términos y Condiciones de la Aplicaciones, los cuales tienen que ser redactados de acuerdo a la nueva ley. Lo mismo sucede con la Política de Privacidad.

4) **Derecho a borrar o derecho a ser olvidado**

Los usuarios deben tener siempre el “poder” de ver, modificar e incluso borrar cualquier información que se haya recogido.

5) **Extraterritorialidad**

Significa que el reglamento RPGD está operativo también en lugares fuera de la Unión Europea.

En general, el RGPD no recoge una nueva reorganización de la política de protección de datos, sino que, más bien, mantiene la vigencia de los principios sobre protección de datos ya conocidos. Dichos principios son el fundamento de esta nueva normativa, aunque se encuentran mejor formulados y desarrollados. Los más importantes son:

Prohibición salvo autorización: este principio significa que a priori se prohíbe cualquier procesamiento de datos personales a no ser que esté permitido. Con el Reglamento europeo de protección de datos, el principio de prohibición se aplica indiscriminadamente a cualquier tipo de datos personales.

Limitación de la finalidad: las empresas solo podrán recopilar y editar datos con unos objetivos específicos. Para ello, al empezar a recogerlos deben formularse los objetivos y documentarse el uso futuro de los datos. Este es otro de los fines que necesita una justificación por separado y solo se permiten las modificaciones posteriores de los objetivos bajo determinadas circunstancias.

Minimización de datos: Este principio exige que las empresas recopilen la menor cantidad de datos posible, aplicando el lema “lo menos posible y tanto como sea necesario”. Así, no se puede recopilar más de lo requerido para conseguir el objetivo previsto con la obtención de datos. Con ello, este principio prohíbe la recopilación de datos “desmedida”.

Transparencia: el tratamiento de los datos debe ser comprensible para los interesados. Esto, por un lado, requiere avisos de privacidad claros y, por otro, significa mayores derechos para los usuarios. Como hasta ahora, las empresas deben comunicar bajo petición cuáles son los datos existentes y cómo se van a utilizar.

Confidencialidad: las empresas tienen la obligación de proteger los datos personales de sus clientes de forma técnica y organizativa ya sea del tratamiento o modificación no autorizados, del robo o de la destrucción de dichos datos. Una novedad es, la obligación explícita a aplicar medidas técnicas de protección.

4. Caso Práctico

4.1. Auditoría de Seguridad de Aplicación en Android

4.1.1. Preparación del entorno

4.1.1.1 Aplicación Objetivo

La aplicación objetivo del análisis es **DIVA (Damm Insecure and Vulnerable App)** es una aplicación diseñada intencionalmente para ser insegura. El objetivo de la aplicación 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 esta aplicación ya que ha sido diseñada y puesta a disposición por su autor, **Payatu**, para el ejercicio de descubrimiento, explicación y mitigación de vulnerabilidades.

4.1.1.2 Recursos para Auditoría

Para la auditoría de seguridad de la aplicación DIVA se optado por auditar desde el propio terminal Android, es decir no se ha utilizado emulador. Los componentes utilizados fueron:

- 1 smartphone Motorola Moto E4 en modo Desarrollador* y Android Nougat 7.1.1
- 1 equipo con Windows 10
- 1 máquina virtual con Kali Linux 2020

* *Modo Desarrollador: Desde el propio dispositivo Ajustes → Acerca del teléfono → 7 clics en el número de compilación. Una vez habilitado, en Ajustes se debe habilitar la "Depuración por USB" y el "Desbloqueo de OEM".*

En relación a las herramientas utilizadas para la auditoría, listamos las siguientes:

- ADB (Android Debug Bridge)
- APK Analyzer
- APK Easy Tool
- JD-GUI
- Burpsuite
- Mobsf
- Frida
- SSL Trust Killer

4.1.1.3 Rooteo del Dispositivo

Rootear un terminal Android es similar a hacer un jailbreak a un iPhone, básicamente, permite al analista de seguridad sumergirse más profundamente en el subsistema del terminal.

Requerimientos:

- Platform-tools de Android (Android SDK):
- Drivers más recientes USB Motorola
- Rom Personalizada para Motorola Moto E4
- TWRP 3.3.0.0
- Super SU v2.82

Desbloqueo de Bootloader

Desbloquear el gestor de arranque o Bootloader, que es la parte de software encargada de realizar varias comprobaciones antes del arranque del sistema operativo para asegurarse de que todo está correctamente y le proporciona al sistema operativo las instrucciones necesarias para iniciarse. Para ello:

- Apagar el teléfono y encenderlo manteniendo apretado el botón de encendido y el de bajar volumen hasta que nos aparezca el Menú de Fastboot.
- Desde el ordenador, ejecutamos el siguiente comando:
  fastboot flashing unlock

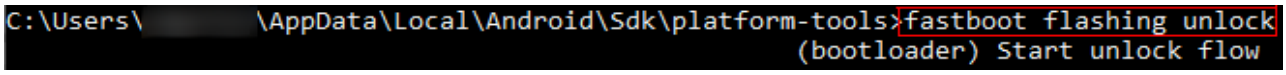


Figura 14: Fastboot

Elaboración propia

Tarda unos minutos, esperar hasta que finalice el proceso.

Configuración Rom personalizada y TWRP

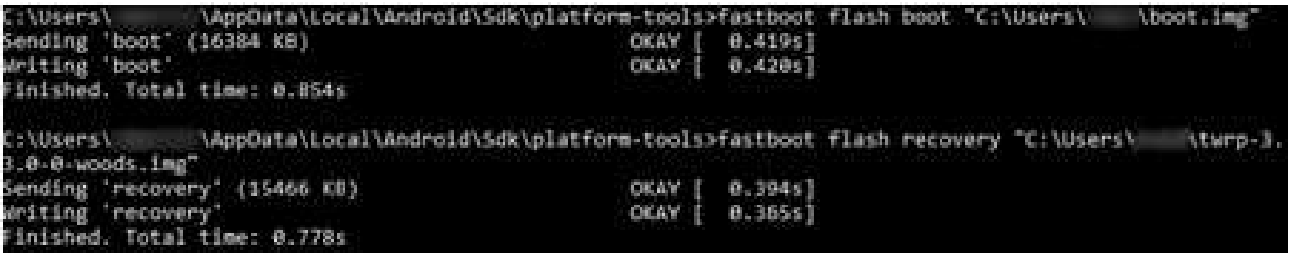


Figura 15: Flash Imagen Rom

Elaboración propia

Custom ROM es un sistema operativo modificado y personalizado para nuestro dispositivo. Lo contrario sería una Stock ROM que es la versión del sistema operativo que viene de serie. **TWRP** se trata de un modo recovery instalado cuando se decide tener un entorno rooteado.

- Descargamos el boot.img y lo flasheamos con el siguiente comando:
  fastboot flash boot boot.img
- Descargamos la imagen más actual de TWRP para flashearla en el recovery:
  fastboot flash recovery twrp.img
- Usando las teclas de volumen, seleccionar el RECOVERY MODE y presionamos el botón de encendido.
- Instalamos SuperSU. Una vez dentro de la imagen TWRP primero realizamos el Wipe de los datos para desencriptar. Nos pedirá escribir “Yes” para confirmar.
- Seguidamente nos descargamos la versión más actual de SuperSU (en este caso se usó la v2.82 SR5) y, mediante adb, pasamos el .zip al dispositivo móvil.

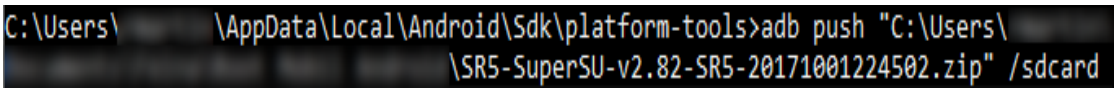


Figura 16: Copiado de SuperSU

Elaboración propia

- En la imagen de TWRP usamos la opción Install y seleccionamos el fichero .zip del SuperSU.

Figura 17: TWRP 3.2.3

Instalación Recovery personalizado

- Una vez termine, se ejecuta el "Reboot" automáticamente y cuando se inicie el dispositivo, si vemos la aplicación SuperSU instalada ya tenemos Root.
- Finalmente, dentro de la aplicación SuperSU, nos dirigimos a Ajustes y "Activar Pro".

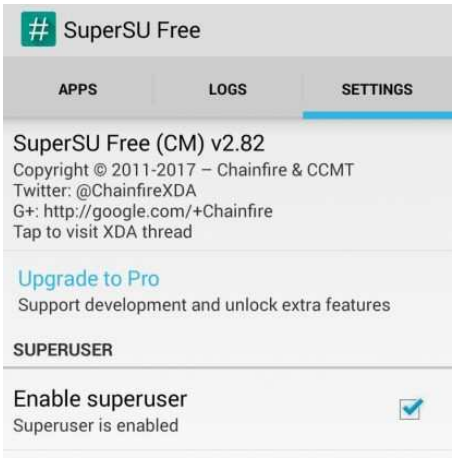


Figura 18: Aplicativo SuperSU 2.82

SuperSU Android

Después de rootear, se puede acceder a la totalidad del sistema operativo para sortear cualquier restricción que su fabricante u operador haya aplicado. Permite además el acceso a los datos almacenados en el dispositivo que de otro modo tendrían acceso restringido.

Tener en cuenta que el rooteo de un dispositivo se considera una gran responsabilidad. Cualquier aplicación que se instale a partir de ese momento dispondrá de permisos de administrador. Es decir, acceso total cualquier archivo, carpeta o dispositivo mapeado en el sistema operativo Android. Es decir, no existe unos límites que podrían de alguna forma acotar el daño en caso de la instalación de un malware.

4.1.1.4 Instalación y Firmado de Aplicación Objetivo

Una vez se compruebe la conexión con el terminal Android rooteado desde el equipo de Kali Linux, puede que al intentar instalar la aplicación objetivo de análisis con ADB se nos muestre un error relacionado con el firmado de la app.

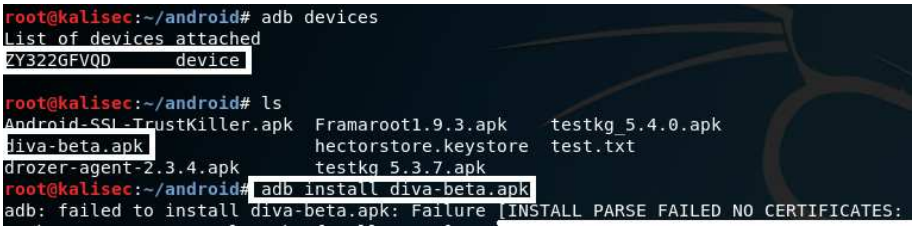


Figura 19: Instalación Aplicación objetivo con ADB

Elaboración propia

Android requiere que todos los archivos .apk estén firmados de manera digital con un certificado para poder instalarse en un dispositivo. Es posible volver a firmar una aplicación para subirla a Google Play y para ello se puede utilizar APK Easy Tool y la opción “Sign APK”.

Figura 20: Firmar APK con APK Easy Tool

APK Easy Tool 2.3.4

Se genera un nuevo archivo, el cual se instala sin problemas en el terminal

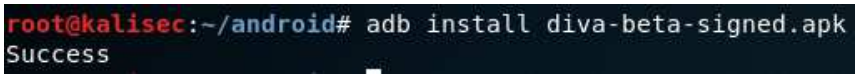


Figura 21: Instalación de Aplicación objetivo firmada

Elaboración propia

Ejecutamos la aplicación desde el propio dispositivo.

Figura 22: Aplicación Objetivo Diva

Elaboración Propia

4.1.1.5 Bypass de SSL Pinning

Hoy en día, la mayoría de aplicaciones utiliza SSL Pinning para evitar que las comunicaciones entre app y servidor puedan ser interceptadas.

Evadir el SSL Pinning en una aplicación móvil se puede lograr con el uso de herramientas como Frida o Xposed, o descargando el APK original y modificando el archivo de configuración de seguridad de la red para confiar en los certificados proporcionados por el usuario y para desactivar el SSL Pinning.

En este caso se utiliza el aplicativo SSL Trust Killer, que es una herramienta que aprovecha Cydia Substrate para conectar varios métodos a fin de evitar la fijación de certificados al aceptar cualquier certificado SSL. SSL Trust Killer se instala en el dispositivo con ADB:

```
adb install Android-SSL-TrustKiller.apk
```

A continuación se agrega el certificado de CA de la herramienta de proxy, en este caso Burpsuite, al almacén de confianza del dispositivo.

4.1.1.6 Configuración de Proxy e Instalación Certificado del Proxy

Para poder interceptar las comunicaciones entre la App y el servidor se utiliza el proxy Burpsuite de Portswigger. Conectamos en la misma red Wireless 10.11.11.0/24:

- El equipo con Kali Linux y Burpsuite: 10.11.11.100
- El terminal Android: 10.11.11.107

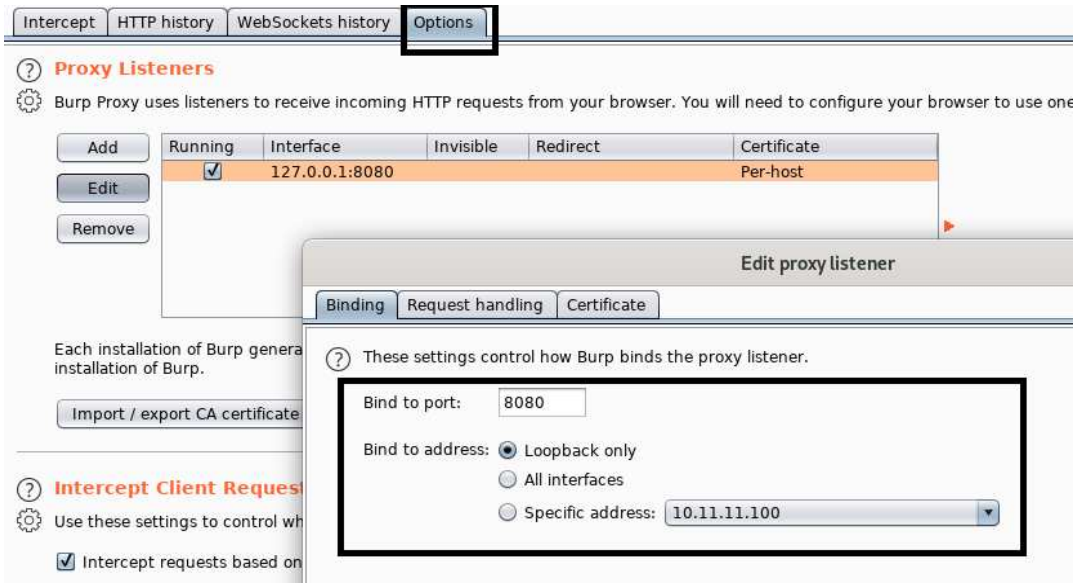


Figura 23: Configuración de Proxy Burpsuite

Elaboración Propia

Configuración de Proxy en terminal Android

Para ello en el propio terminal modificamos la conexión Wireless y, de forma manual, colocamos los parámetros del proxy anteriormente configurado.

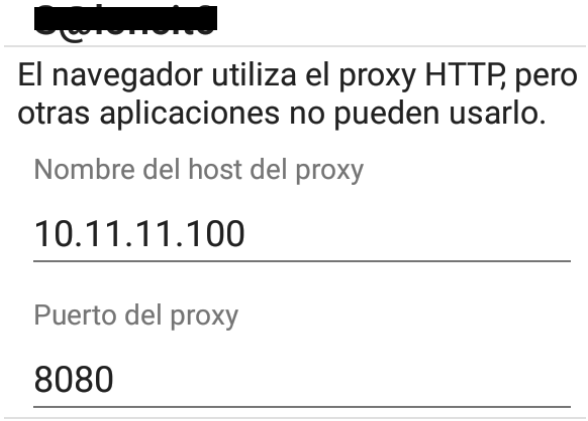


Figura 24: Configuración de proxy en Terminal Android

Elaboración propia

Instalación de certificado de Burpsuite

En el navegador web, se coloca la URL del proxy y el puerto determinado (8080) y se instala el certificado.

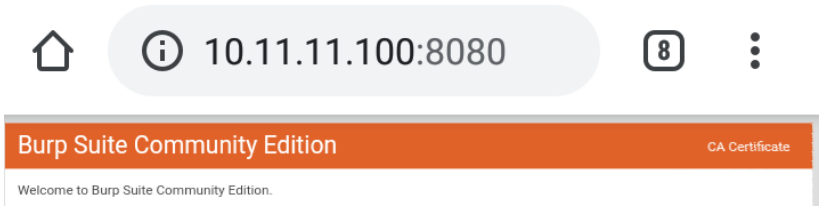


Figura 25: Descarga de certificado de Burpsuite

Elaboración propia

Ha quedado configurado el entorno para empezar cada una de las fases de la auditoría de seguridad de la aplicación DIVA

4.1.2. Recopilación de Información

En primer lugar, se realiza un reconocimiento de la aplicación a nivel de usuario y posteriormente un análisis de la aplicación utilizando APK Analyzer para la obtención de información relacionada con permisos, características y recursos, etc.

A nivel de Usuario:

Diva se trata de aplicación nativa en lenguaje Java y C que incluye ejercicios relacionados con la codificación o el desarrollo de la aplicación, el registro inseguro de datos y el almacenamiento inseguro de datos.

Al ejecutar la aplicación se nos muestra varios menús y cada uno de ellos representa un ejercicio para practicar una determinada vulnerabilidad.

APK Analyzer:

Detalles de la aplicación	
Application Name	Diva
Filename	diva-beta signed.apk
Package Name	Jakhar.aseem.diva
Process Name	Jakhar.aseem.diva
Version Name	1.0
Plataforma	Android
Tamaño	1,4 MB
MD5 Checksum	ff02f9091cdf4615498a1a0e9aec75a0
SHA1 Checksum	8f51ba350d5bfee6bd6bc3aa2354661adb6ee4e5
Minimal Supported Android	Ice Cream Sandwich (4.0.3 – 4.0.4)

PERMISOS	
Write_External_Storage	Modificar/Eliminar almacenamiento USB.
Read_External_Storage	Leer almacenamiento USB
Internet	Acceso a Internet
Access_Coarse_Location	Permite que una aplicación acceda a una ubicación aproximada.
Access_Fine_Location	Permite que una aplicación acceda a una ubicación precisa.

ACTIVITIES	
jakhar.aseem.diva.MainActivity	jakhar.aseem.diva.LogActivity
jakhar.aseem.diva.HardcodeActivity	jakhar.aseem.diva.InsecureDataStorage1Activity
jakhar.aseem.diva.InsecureDataStorage2Activity	jakhar.aseem.diva.InsecureDataStorage3Activity
jakhar.aseem.diva.InsecureDataStorage4Activity	jakhar.aseem.diva.SQLInjectionActivity
jakhar.aseem.diva.AccessControl1Activity	jakhar.aseem.diva.InputValidation2URISchemeActi
jakhar.aseem.diva.APICredsActivity	jakhar.aseem.diva.AccessControl2Activity

jakhar.aseem.diva.APICreds2Activity	jakhar.aseem.diva.AccessControl3Activity
jakhar.aseem.diva.Hardcode2Activity	jakhar.aseem.diva.AccessControl3NotesActivity
jakhar.aseem.diva.InputValidation3Activity	
CONTENT PROVIDERS	
jakhar.aseem.diva.NotesProvider	
INTENTS	
android.intent.action.MAIN	android.intent.category.DEFAULT
android.intent.category.LAUNCHER	jakhar.aseem.diva.action.VIEW_CREDS
jakhar.aseem.diva.action.VIEW_CREDS2	

Android manifest.xml

Figura 26: Ejemplo de archivo manifest.xml

Ostorlab

4.1.3. Análisis Estático

Uno de los primeros pasos para encontrar vulnerabilidades es el análisis estático tras realizar un reversing a la aplicación. Para la ingeniería inversa podemos hacer uso del aplicativo dex2jar, seleccionamos el archivo .apk a auditar y lo transformamos a .jar.

Figura 27: Transformar .apk a .jar con Dex2jar

Elaboración propia

Luego utilizaremos JD-GUI para analizar la información decompilada.

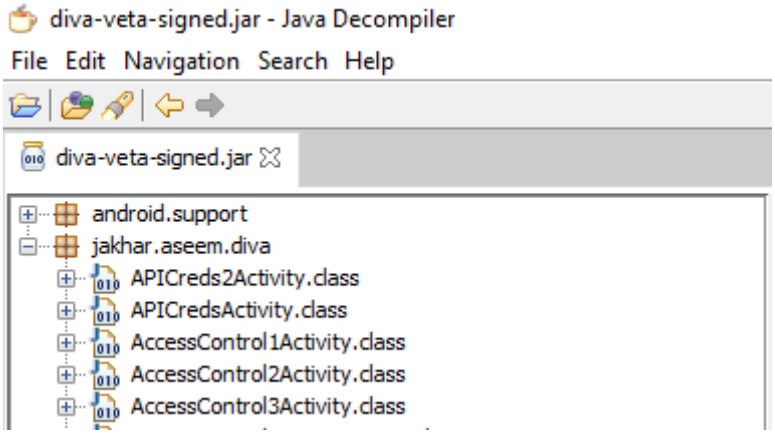


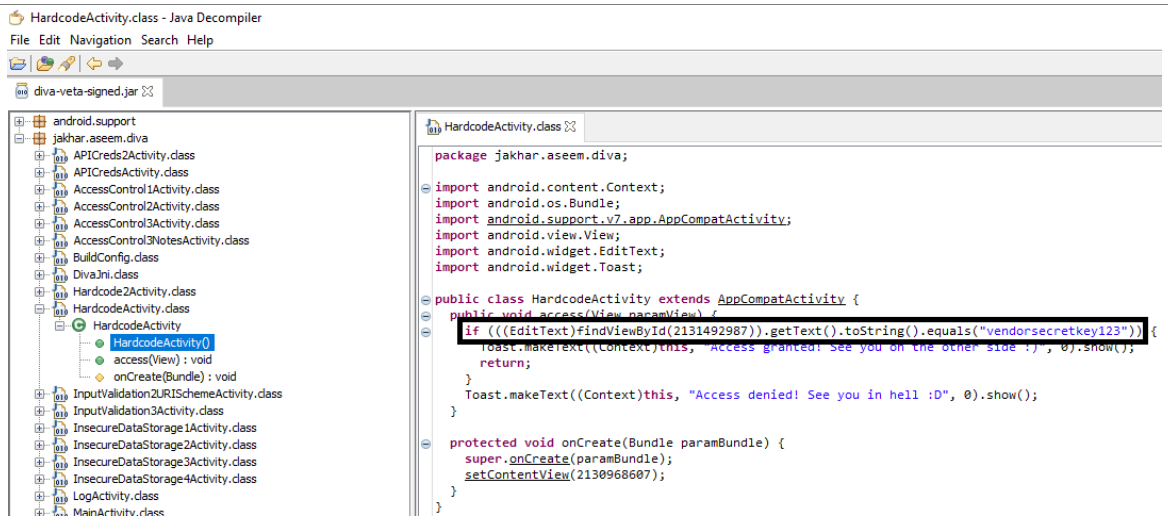
Figura 28: Análisis de .jar desde J-GUI

Elaboración propia

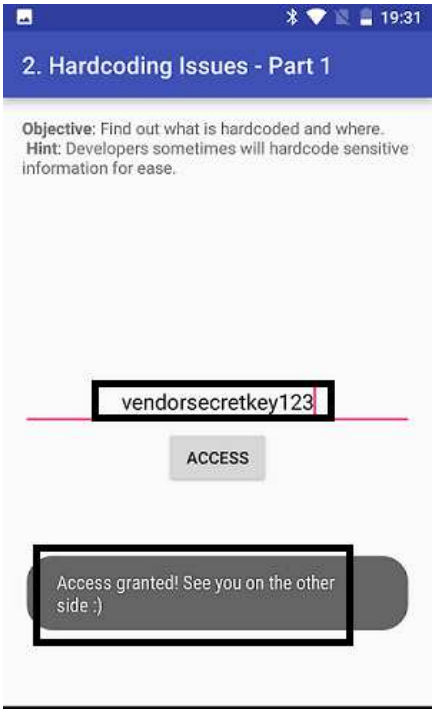
Informe de Vulnerabilidades

A continuación documentaremos algunas de las vulnerabilidades descubiertas en la plataforma y para ello usaremos una métrica de CVSS v2.

001	**Información sensible en el código fuente**		CVSS 7,2		
diva-beta signed.apk					
AV	**AC**	**Au**	**C**	**I**	**A**
Local	Baja	NA	Completo	Completa	Completa
Descripción					
El desarrollador ha codificado información sensible en el código fuente de la aplicación. Se trata de una clave del proveedor que permitiría un acceso no controlado de la aplicación. Abrimos el archivo "HardcodeActivity.class" usando JD-GUI y observamos el siguiente fragmento de código. La clave secreta se ha codificado para que coincida con la entrada del usuario como se muestra en la línea a continuación.					



Comprobamos que es posible acceder con la clave descubierta “vendorsecretkey123”



Recomendaciones

No colocar información sensible en el código fuente de la aplicación o al menos intentar ofuscarla.

002	Modo de Depuración habilitado		CVSS 6,1

diva-beta signed.apk

AV	AC	Au	C	I	A
Local	Medio	NA	Completa	Parcial	Parcial

Descripción

La aplicación se compila con el modo de depuración habilitado. Esto permite a los atacantes acceder al sistema de archivos de la aplicación y adjuntar un depurador para acceder a datos confidenciales o realizar acciones maliciosas.

El atacante puede depurar la aplicación sin acceso al código fuente y aprovecharla para realizar acciones maliciosas en nombre del usuario, modificar el comportamiento de la aplicación o acceder a datos confidenciales como credenciales y cookies de sesión.

Revisamos el archivo manifest.xml y comprobamos que el valor es “true” or lo tanto está activado.

Recomendaciones

El valor debe ser modificado a “False” para deshabilitar el modo “depuración”.

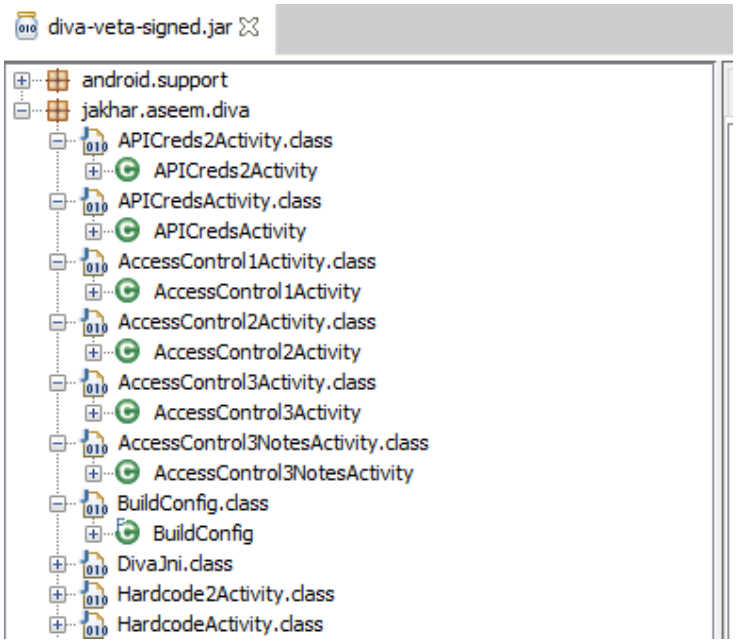
003	Código fuente no ofuscado		CVSS 4,4

diva-beta signed.apk

AV	AC	Au	C	I	A
Local	Medio	NA	Parcial	Parcial	Parcial

Descripción

Es posible decompilar la aplicación y revisar el código fuente en el que se encuentran métodos y clases de forma bastante clara.



Recomendaciones

Es necesario diseñar la aplicación para agregar protecciones a la ingeniería inversa de la aplicación. Por ejemplo, ofusque el código fuente de Java con herramientas como Proguard o Dexguard.

004	Métodos vulnerables a SQLi		CVSS 6,9

diva-beta signed.apk

AV	AC	Au	C	I	A
Local	Medio	NA	Completa	Completa	Completa

Descripción

La construcción incorrecta de consultas SQL podría conducir a la inyección SQL. Un ataque de este tipo consiste en inyectar una consulta SQL a través de los datos de entrada del cliente a la aplicación.

Los métodos vulnerables son:

- jakhar.aseem.diva.SQLInjectionActivity.search()
- jakhar.aseem.diva.InsecureDataStorage2Activity.saveCredentials()

Y según el análisis, no hay ningún módulo cargado por otro lado para sanitizar el input en ambos métodos.

Método jakhar.aseem.diva.SQLInjectionActivity.search() :

```
public void search(android.view.View p8)
{
    android.widget.EditText v2_1 = ((android.widget.EditText) this.findViewById(2131493017));
    try {
        android.database.Cursor v0 = this.mDB.rawQuery(new StringBuilder().append("SELECT * FROM sqliuser WHERE user = \'"
        StringBuilder v3_1 = new StringBuilder("");
    } catch (Exception v1) {
        android.util.Log.d("Diva-sqli", new StringBuilder().append("Error occurred while searching in database: ").append(
        return;
    }
    if ((v0 == null) || (v0.getCount() <= 0)) {
        v3_1.append(new StringBuilder().append("User: (").append(v2_1.getText().toString()).append(") not found").toString
    } else {
        v0.moveToFirst();
        do {
            v3_1.append(new StringBuilder().append("User: (").append(v0.getString(0)).append(") pass: (").append(v0.getStr
        } while(v0.moveToNext());
    }
    android.widget.Toast.makeText(this, v3_1.toString(), 0).show();
    return;
}
```

Método jakhar.aseem.diva.InsecureDataStorage2Activity.saveCredentials() :

```
public void saveCredentials(android.view.View p7)
{
    try {
        this.mDB.execSQL(new StringBuilder().append("INSERT INTO myuser VALUES (\'") append(((android.widget.EditText) thi
        this.mDB.close();
    } catch (Exception v0) {
        android.util.Log.d("Diva", new StringBuilder().append("Error occurred while inserting into database: ").append(v0.
    }
    android.widget.Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
    return;
}
```

Recomendaciones

Sin una eliminación o cita suficiente de la sintaxis SQL en las entradas controlables por el usuario, la consulta SQL generada puede hacer que esas entradas se interpreten como SQL en lugar de datos de usuario normales. Esto se puede utilizar para alterar la lógica de consulta para evitar las comprobaciones de seguridad o acceder a contenido no autorizado.

4.1.4. Análisis Dinámico

El análisis dinámico incluye el interceptar las comunicaciones de la aplicación con el servidor, el comportamiento de la App antes y después de un login, revisión de logs, etc.

Informe de Vulnerabilidades

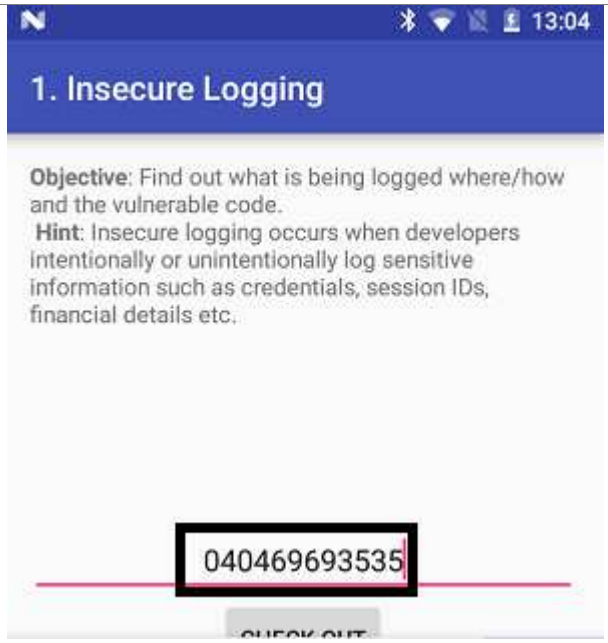
A continuación documentaremos algunas de las vulnerabilidades descubiertas en la plataforma y para ello usaremos una métrica de CVSS v2.

005	Registro de información sensible en Logs		CVSS 4,4
diva-beta signed.apk			

AV	AC	Au	C	I	A
Local	Medio	NA	Parcial	Parcial	Parcial

Descripción

El aplicativo registra informacion sensible y confidencial en Logcat debido a una mala práctica de Desarrollo.




```
03-25 13:07:06.602   300   300 I wmt_launcher: fw log ctrl flag has been set
03-25 13:07:06.602   300   300 I wmt_launcher: fw dynamic ctrl flag has been set
03-25 13:07:06.695   971  1151 I PerfService: PerfServiceNative_boostEnableTimeoutMsAsync:6, 100
03-25 13:07:06.698 26193 26193 E diva-log: Error while processing transaction with credit card: 040469693535
03-25 13:07:06.701   971  1491 I NotificationService: enqueueToast pkg=jakhar.aseem.diva callback=android.app.I
be7c3e duration=0
03-25 13:07:06.725   307   307 I BufferQueue: [unnamed-307-970](this:0xac709400,id:970,api:0,p:-1,c:-1) BufferQ
flinger)
03-25 13:07:06.726   307   307 I BufferQueueConsumer: [unnamed-307-970](this:0xac709400,id:970,api:0,p:-1,c:307
m/bin/surfaceflinger) controlledByApp=false
```

Según comprobamos en esta línea de código se indica el almacenamiento de los datos ingresados por el usuario en Logcat.

LogActivity.class

```
package jakhar.aseem.diva;

import android.content.Context;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class LogActivity extends AppCompatActivity {
  private void processCC(String paramString) {
    throw new RuntimeException();
  }

  public void checkout(View paramView) {
    EditText editText = (EditText)findViewById(2131493014);
    try {
      processCC(editText.getText().toString());
    } catch (RuntimeException runtimeException) {
      Log.e("diva-log", "Error while processing transaction with credit card: " + editText.getText().toString());
      Toast.makeText((Context)this, "An error occured. Please try again later", 0).show();
    }
  }

  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    setContentView(2130968615);
  }
}
```

Recomendaciones
Es importante estar al tanto de lo que está registrando y sólo registrar información no confidencial. La salida de registro detallado es muy útil para los desarrolladores, pero puede ser una mina de oro de información confidencial para los atacantes. Se debe tener especial cuidado al registrar las claves de sesión y las URL que pueden contener valores importantes.

4.1.5. Informe automático con MOBSF

MobSF es una herramienta recomendada por OWASP en su Guía de pruebas de seguridad móvil para plataformas Android y Java. Esta herramienta se basa en patrones del Estándar de verificación de seguridad de aplicaciones móviles OWASP y de la Guía de pruebas de seguridad móvil OWASP.

Para entornos Android proporcinará un análisis estático y dinámico de la Aplicación.

Diva (1.0)

File Name:	diva-beta signed.apk
Package Name:	jakhar.aseem.diva
Average CVSS Score:	6.1
App Security Score:	55/100 (MEDIUM RISK)

FILE INFORMATION

File Name: diva-beta signed.apk
Size: 1.44MB
MD5: ff02f9091cdf4615498a1a0e9aec75a0
SHA1: 8f51ba350d5bfee6bd6bc3aa2354661adb6ee4e5
SHA256: 77519a9186408ff3617c83231e0e1ba2d2cf4d38e7310ca4ce3ba73526ba3a93

APP INFORMATION

App Name: Diva
Package Name: jakhar.aseem.diva
Main Activity: jakhar.aseem.diva.MainActivity
Target SDK: 23
Min SDK: 15
Max SDK:
Android Version Name: 1.0
Android Version Code: 1

APP COMPONENTS

Activities: 17
Services: 0
Receivers: 0
Providers: 1
Exported Activities: 2
Exported Services: 0
Exported Receivers: 0
Exported Providers: 1

CERTIFICATE INFORMATION

APK is signed
v1 signature: True
v2 signature: True
v3 signature: True
Found 1 unique certificates
Subject: C=US, ST=California, L=Mountain View, O=Android, OU=Android, CN=Android, E=android@android.com
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2008-02-29 01:33:46+00:00
Valid To: 2035-07-17 01:33:46+00:00
Issuer: C=US, ST=California, L=Mountain View, O=Android, OU=Android, CN=Android, E=android@android.com
Serial Number: 0x936eacbe07f201df
Hash Algorithm: sha1
md5: e89b158e4bcf988ebd09eb83f5378e87
sha1: 61ed377e85d386a8dfee6b864bd85b0bfaa5af81
sha256: a40da80a59d170caa950cf15c18c454d47a39b26989d8b640ecd745ba71bf5dc
sha512:
5216ccb62004c4534f35c780ad7c582f4ee528371e27d4151f0553325de9ccbe6b34ec4233f5f640703581053abfea303977272d17958704d89b7711292a4569

PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: f9f32662753449dc550fd88f1ed90e94b81adef9389ba16b89a6f3579c112e75

Certificate Status: Bad
Description: The app is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues.

APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete SD card contents	Allows an application to write to the SD card.
android.permission.READ_EXTERNAL_STORAGE	dangerous	read SD card contents	Allows an application to read from SD Card.
android.permission.INTERNET	dangerous	full Internet access	Allows an application to create network sockets.

SHARED LIBRARY BINARY ANALYSIS

ISSUE	SEVERITY	DESCRIPTION	FILES
Found elf built without Position Independent Executable (PIE) flag	high	In order to prevent an attacker from reliably jumping to, for example, a particular exploited function in memory, Address space layout randomization (ASLR) randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries. Built with option -pie.	lib/mips64/libdivajni.so

APKID ANALYSIS

FILE	DETAILS
classes.dex	FINDINGS: Compiler — DETAILS: dexlib 2.x

MANIFEST ANALYSIS

ISSUE	SEVERITY	DESCRIPTION
Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.

ISSUE	SEVERITY	DESCRIPTION
Application Data can be Backed up [android:allowBackup=true]	medium	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
Activity (jakhar.aseem.diva.APICredsActivity) is not Protected. An intent-filter exists.	high	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Activity (jakhar.aseem.diva.APICreds2Activity) is not Protected. An intent-filter exists.	high	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
Content Provider (jakhar.aseem.diva.NotesProvider) is not Protected. [android:exported=true]	high	A Content Provider is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.

</> CODE ANALYSIS

ISSUE	SEVERITY	STANDARDS	FILES
The App logs information. Sensitive information should never be logged.	info	CVSS V2: 7.5 (high) CWE: CWE-532 - Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	jakhar/aseem/diva/AccessControl1Activity.java jakhar/aseem/diva/SQLInjectionActivity.java jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/InsecureDataStorage3Activity.java jakhar/aseem/diva/InsecureDataStorage4Activity.java jakhar/aseem/diva/AccessControl2Activity.java jakhar/aseem/diva/LogActivity.java
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	high	CVSS V2: 5.9 (medium) CWE: CWE-89 - Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') OWASP Top 10: M7: Client Code Quality	jakhar/aseem/diva/SQLInjectionActivity.java jakhar/aseem/diva/NotesProvider.java jakhar/aseem/diva/InsecureDataStorage2Activity.java
App creates temp file. Sensitive information should never be written into a temp file.	high	CVSS V2: 5.5 (medium) CWE: CWE-276 - Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	jakhar/aseem/diva/InsecureDataStorage3Activity.java

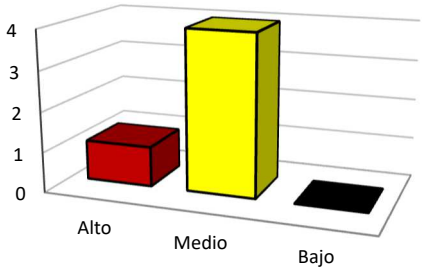
ISSUE	SEVERITY	STANDARDS	FILES
App can read/write to External Storage. Any App can read data written to External Storage.	high	CVSS V2: 5.5 (medium) CWE: CWE-276 - Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	jakhar/aseem/diva/InsecureDataStorage4Activity.java

4.1.6. Resumen Ejecutivo

Nivel de seguridad Global

NIVEL BAJO
4 / 10

Total Vulnerabilidades



RESUMEN	
Nivel	**Total**
Alto	1
Medio	4
Bajo	0
TOTAL	**5**

El objeto del análisis de aplicación Android es el de detectar las deficiencias de seguridad mediante una exhaustiva Auditoría Externa de Seguridad, aportando, en caso de existir diferentes alternativas, la solución o soluciones más adecuada a cada una de estas deficiencias.

El nivel de seguridad Global es Bajo ya que se ha detectado una vulnerailidad de Nivel Alto relacionada con la iexposición de claves privadas que permitirían un acceso no autorizado a la plataforma y vulnerabilidades de Nivel Medio descubiertas están relacionadas con métodos vulnerales a SQL injection y la exposición de información sensible.

Se recomienda que, tras implantar las medidas correctoras definidas en este informe, se realice una Auditoría de Verificación con el fin de validar que todas las vulnerabilidades han sido solventadas y su solución no ha provocado nuevas deficiencias.

4.2. Auditoría de Seguridad de Aplicación en iOS

4.2.1. Preparación del entorno

4.2.1.1 Aplicación Objetivo

La aplicación objetivo del análisis es **DVIA (Damm Vulnerable iOS App)** es una aplicación diseñada intencionalmente para ser insegura. El objetivo de la aplicación es enseñar a los desarrolladores / QA / 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 esta aplicación ya que ha sido diseñada y puesta a disposición por su autor para el ejercicio de descubrimiento, explicación y mitigación de vulnerabilidades.

4.2.1.2 Recursos para Auditoría

Para la auditoría de seguridad de la aplicación DVIA se optado por auditar desde el propio terminal iOS, es decir no se ha utilizado emulador. Los componentes utilizados fueron:

- 1 Iphone 6 con Sistema Operativo iOS 13
- 1 Equipo con Windows 10
- 1 Máquina virtual con Kali Linux 2020

En relación a las herramientas utilizadas para la auditoría listamos las siguientes:

- Objection
- IDB
- Burpsuite
- Mobsf
- Frida
- SSL Kill Switch2

4.2.1.3 Jailbreak del Dispositivo

Tras recientes cambios que hizo Apple sobre la autenticación de sus servicios para las cuentas gratuitas varios de los métodos más conocidos de Jailbreak de Pangu o Phoenix, así como la instalación de aplicaciones fuera de Apple Store con Cydia Impactor o iTunes dejaron de funcionar.

Con la bienvenida de iOS13 apareció el nuevo Jailbreak de Checkra1n. Se trata de un Jailbreak semi-tethered y basado en checkm8 bootrom. Se encuentra disponible para iPhone 5S y hasta el modelo X y por el momento sólo se puede lanzar desde macOS. Para más información podéis visitar su sitio web https://checkra.in/.

Instalación de Checkrain en dispositivo IPhone

Conectamos el dispositivo IPhone con el MacOS a través de USB y ejecutamos Checkrain.

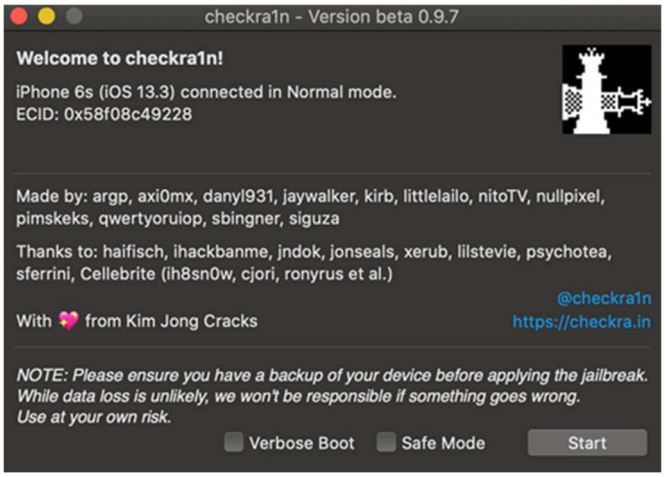


Figura 29: Checkrain desde MacOS

Elaboración propia

Tras conectar el dispositivo por USB va a solicitar que activemos el modo DFU (Device Firmware Upgrade), basta con pulsar primero el botón de encendido e inicio (4s) y luego dejar de pulsar el de encendido, y mantener el de inicio, otros 10 segundos más.

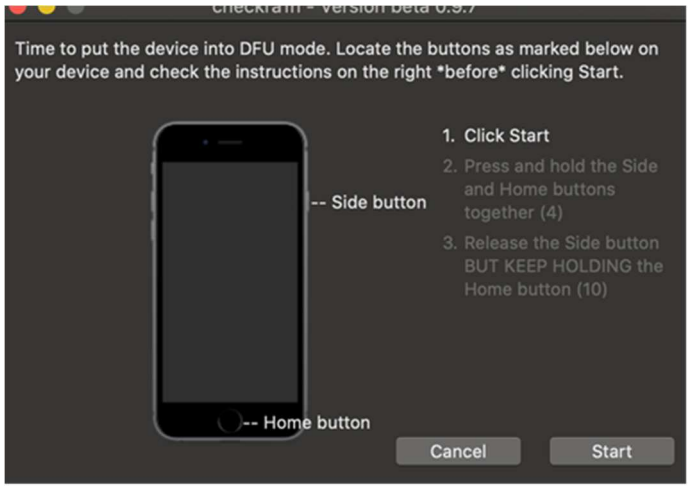


Figura 30: Activación de DFU

Elaboración propia

Tras finalizar este proceso aparecerá en el inicio la app checkra1n, y accediendo a ella podremos finalizar el Jailbreak instalando Cydia. **Cydia** es la tienda, no oficial, de aplicaciones iOS más popular que ofrece una gran variedad de aplicaciones y herramientas para iOS.

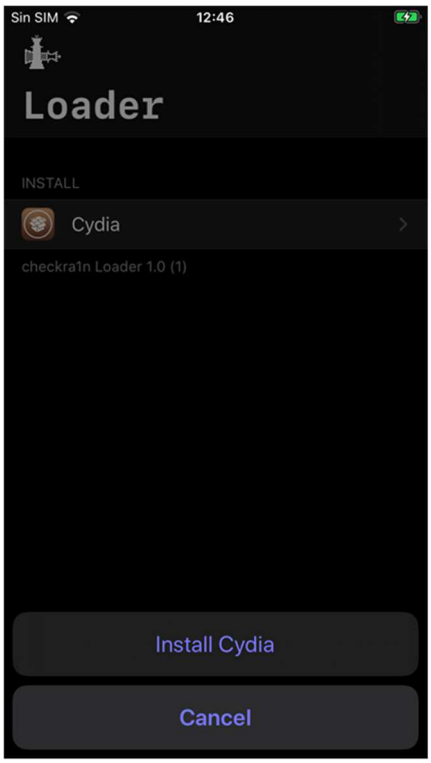


Figura 31: Cydia Loader

Checkra1n Jailbreak Tool for iOS 12 up to iOS 13

4.2.1.4 Firmado de Aplicación Objetivo e Instalación

En algunos casos para subir una aplicación a Apple Store o instalarla en un dispositivo iOS es necesario firmarla previamente. Para ello es posible generando una nueva identificación para la aplicación en el portal de Desarrolladores de Apple contando con una licencia de Desarrollador (99$) o utilizando métodos o herramientas que ayudarán con el proceso de firma. Por ejemplo, con EasyResigny una herramienta que genera un IPA que realmente puede implementarse en un dispositivo iOS. Esta herramienta hace que el proceso de re-firma sea realmente fácil, para ello:

- Seleccione la IPA que desea volver a firmar
- Seleccione el certificado para volver a firmar el IPA con
- Seleccione el perfil de aprovisionamiento para usar
- Haga clic en Iniciar EasyResigny!

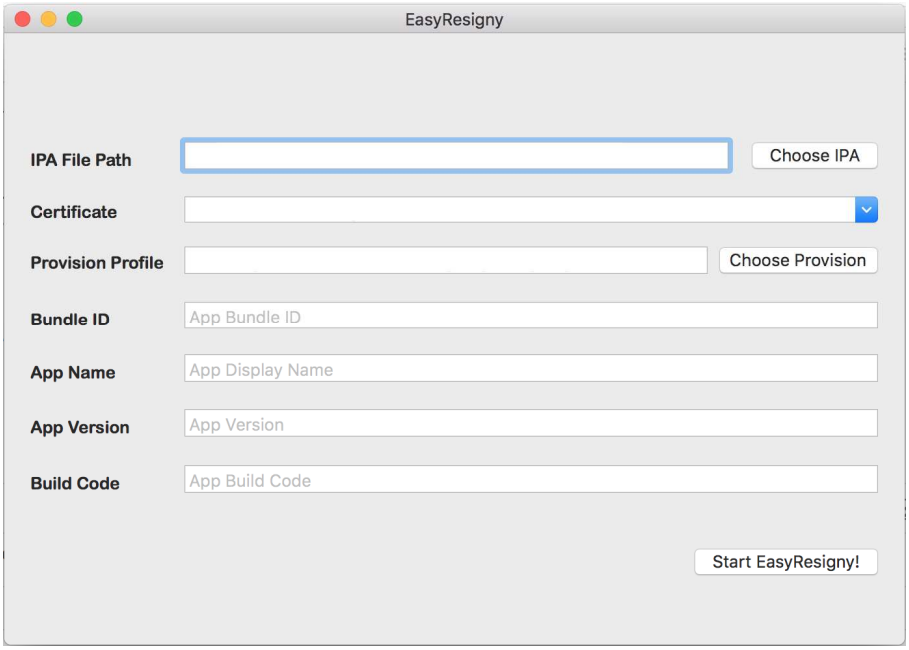


Figura 32: Herramienta EasyResigny

Re-signing iOS apps

Instalación de Aplicación Objetivo

Para la instalación de aplicaciones no firmadas o fuera de Apple Store, se recomienda el uso de Filza, compatible también con el Jailbreak de Checkra1n anteriormente aplicado.

Filza se trata de un explorador de archivos avanzado que sirve para para administrar todos los archivos en el dispositivo con Jailbreak. Permite listar, buscar archivos y carpetas, ejecutar scripts de shell e instalar paquetes ipa.

- Filza App - File Manager para iOS 13. Añadir el siguiente repo de Cydia http://apt.thebigboss.org/repofiles/cydia/

Instalación DVIA:

1) Abra el archivo IPA en el navegador Safari.

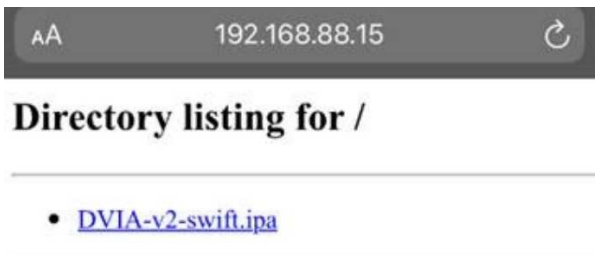


Figura 33: Instalable de Aplicación objetivo

Elaboración propia

2) Después de descargarlo, seleccione copiar archivo a Filza.
3) El archivo aparecerá en la carpeta Documentos.

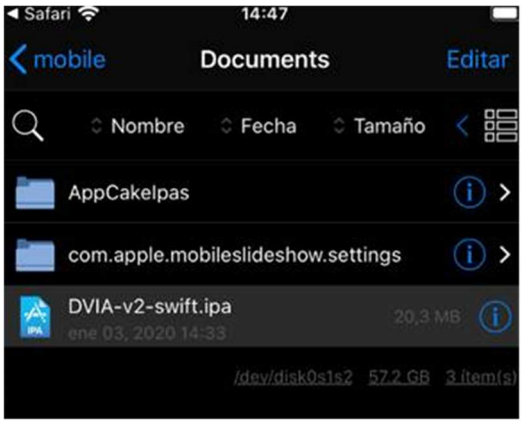


Figura 34: Archivo .ipa en Documents

Elaboración propia

4) Clic en el archivo IPA para mostrar todos los detalles sobre la aplicación. Y pulsar en "Instalar" para extraer el archivo IPA.

Figura 35: Extraer archivo .ipa

Elaboración propia

5) Ejecutar la aplicación desde la pantalla de inicio.

4.2.1.5 Bypass de SSL Pinning

La aplicación usa SSL/TLS para cifrar las comunicaciones entre la aplicación y el servidor. No obstante, los usuarios pueden instalar certificados de CA adicionales en su dispositivo para poder interceptar dichas comunicaciones.

Las aplicaciones móviles disponen de un control llamado “SSL Certificate Pinning” con el que se evita que la aplicación acepte cualquier certificado instalado en el dispositivo y sólo use el que tiene la aplicación.

Si el control detecta que se está usando un certificado que no es el esperado, existen dos comportamientos: se realizan las comunicaciones con el certificado correcto o la aplicación no se ejecuta. En ambos casos, el atacante no podría interceptar las comunicaciones.

No obstante, fue posible evadir el SSL certificate pinning mediante la aplicación SSLKillSwitch2:

- URL: https://github.com/nabla-c0d3/ssl-kill-switch2

Se instala en el dispositivo mediante dpkg:

```
Prateeks-IPad:~ root# dpkg -i com.isecpartners.nabla.sslkillswitch_v0.6-iOS_7.0.deb
(Reading database ... 3626 files and directories currently installed.)
Preparing to replace com.isecpartners.nabla.sslkillswitch 0.6-1 (using com.isecpartners.nabla.sslkillswitch_v0.6-iOS_
7.0.deb) ...
Unpacking replacement com.isecpartners.nabla.sslkillswitch ...
Setting up com.isecpartners.nabla.sslkillswitch (0.6-1) ...
Prateeks-IPad:~ root#
```

Figura 36: Instalación de SSL Kill Switch 2 en iOS

Elaboración propia

Y se activa desde los ajustes del propio dispositivo:

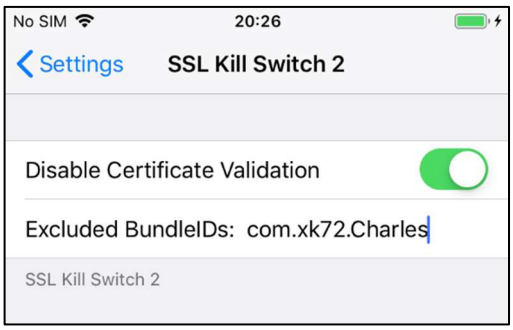


Figura 37: Habilitar SSL Kill Swich 2 en iOS

Elaboración propia

A partir de la activación de esta funcionalidad, el analista podría interceptar las comunicaciones con el objetivo de analizarlas y entender la lógica de la aplicación que le permita realizar ataques más sofisticados.

4.2.1.6 Configuración de Proxy e Instalación de Certificado Proxy

Para poder interceptar las comunicaciones entre la App y el servidor se utiliza la herramienta proxy Burpsuite de Portswigger. En la misma red Wireless 10.11.11.0/24 conectamos:

- El equipo con Kali Linux y Burpsuite: 10.11.11.100
- El terminal IPHONE: 10.11.11.103

Configuración de Proxy en Kali Linux

En Burpsuite, se configura la interfaz 10.11.11.106 y el puerto 8080.

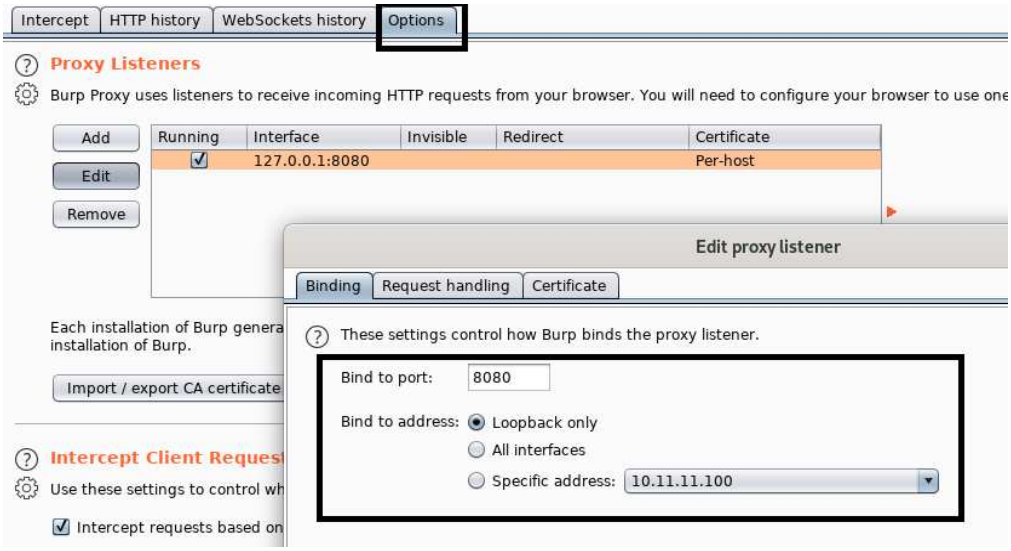


Figura 38: Configuración de proxy en Kali Linux

Elaboración propia

Configuración de Proxy en terminal IPhone

Para ello en el propio terminal modificamos la conexión Wireless y, de forma manual, colocamos los parámetros del proxy anteriormente configurado.

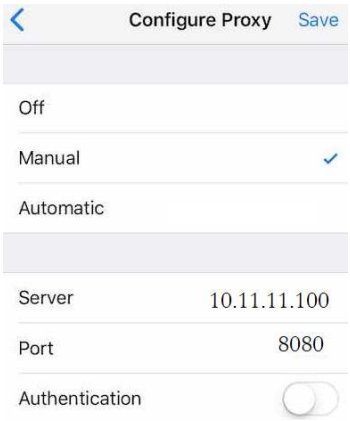


Figura 39: Configuración de proxy en terminal IPhone

Elaboración propia

Instalación de certificado de Burpsuite

En el navegador web, se coloca la URL del proxy y el puerto determinado (8080) y se instala el certificado.

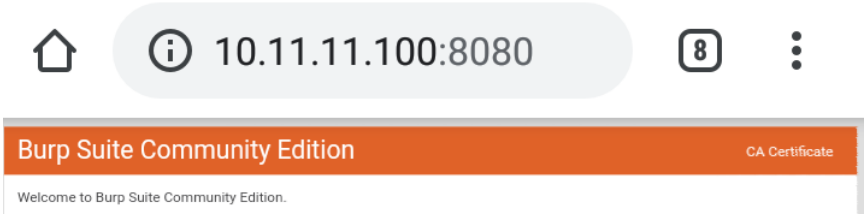


Figura 40: Descarga de certificado de Burpsuite

Elaboración propia

Ha quedado configurado el entorno para empezar cada una de las fases de la auditoría de seguridad de la aplicación DVIA.

4.2.2. Recopilación de Información

Similar a un APK para aplicaciones de Android, un IPA es un archivo que contiene la aplicación de iOS. Los archivos con extensión .ipa se pueden extraer con la utilidad de descompresión (winzip, 7zip) para proporcionarnos el contenido en el directorio Payload / AppName.app.

La aplicación DVIA cubre todas las vulnerabilidades comunes que se encuentran en las aplicaciones iOS (siguiendo los 10 riesgos móviles principales de OWASP) y contiene varios desafíos que el usuario puede probar. Esta aplicación también contiene una sección donde un usuario puede leer varios artículos sobre seguridad de aplicaciones iOS. Este proyecto es desarrollado y mantenido por **@prateekg147**.

Detalles de la aplicación	
Application Name	DVIA
Filename	DVIA-v2-swift.ipa
Package Name	com.highaltitudehacks.DVIAswiftv2
Process Name	com.highaltitudehacks.DVIAswiftv2
Version Name	1.0
Plataforma	iOS
Tamaño	19 MB
MD5 Checksum	35469622303ba10a2195557a3ad1810a
SHA1 Checksum	85174824d6cd7c83df98c518247acf8a14b28882
Minimal Supported Android	iOS 8.0

iOS Info.plist

Los .plist son archivos que almacenan objetos serializados. A menudo almacenan la configuración de los usuarios. También se utilizan para almacenar información sobre paquetes y aplicaciones.

Archivo de lista Payload/DVIA-v2.app/Frameworks/Bolts.framework/Info.plist :


```
{
    "BuildMachineOSBuild": "17D47",
    "CFBundleDevelopmentRegion": "en",
    "CFBundleExecutable": "Bolts",
    "CFBundleIdentifier": "org.cocoapods.Bolts",
    "CFBundleInfoDictionaryVersion": "6.0",
    "CFBundleName": "Bolts",
    "CFBundlePackageType": "FMWK",
    "CFBundleShortVersionString": "1.9.0",
    "CFBundleSignature": "????",
    "CFBundleSupportedPlatforms": [
        "iPhoneOS"
    ],
    "CFBundleVersion": "1",
    "DTCompiler": "com.apple.compilers.llvm.clang.1_0",
    "DTPlatformBuild": "15C107",
    "DTPlatformName": "iphoneos",
    "DTPlatformVersion": "11.2",
    "DTSDKBuild": "15C107",
    "DTSDKName": "iphoneos11.2",
    "DTXcode": "0920",
    "MinimumOSVersion": "8.0",
    "UIDeviceFamily": [
```


Figura 41: Info.plist de Aplicación objetivo

Elaboración propia

4.2.3. Análisis Estático

Uno de los primeros pasos para encontrar vulnerabilidades es el análisis estático tras realizar un reversing a la aplicación.

Informe de Vulnerabilidades

A continuación documentaremos algunas de las vulnerabilidades descubiertas en la plataforma y para ello usaremos una métrica de CVSS v2.

1	**Configuración insegra de ATS**		**CVSS 4.9**

DVIA-v2-swift.ipa

AV	AC	Au	C	I	A
Local	Baja	Ninguna	Completa	Ninguna	Ninguna

Descripción

App Transport Security (ATS) aplica las mejores prácticas en las conexiones seguras entre una aplicación y su back-end. ATS evita la divulgación accidental, proporciona un comportamiento predeterminado seguro y es fácil de adoptar

Uno de los parámetros, **NSAllowsArbitraryLoads**, si se establece en True, deshabilita todas las restricciones ATS para todas las conexiones de red, aparte de las conexiones a dominios que configura individualmente en el diccionario opcional NSExceptionDomains. El valor predeterminado es No.

Recomendaciones

Si se está desarrollando una nueva aplicación, debe usar HTTPS exclusivamente. Si se tiene una aplicación existente, debe usar HTTPS y crear un plan para migrar el resto de su aplicación lo antes posible. Además, su comunicación a través de API de nivel superior debe cifrarse con TLS versión 1.2 con confidencialidad directa.

4.2.4. Análisis Dinámico

Para el análisis dinámico se incluye el interceptar las comunicaciones de la aplicación con el servidor, el comportamiento de la App antes y después de un login, revisión de logs, etc.

Informe de Vulnerabilidades

A continuación documentaremos algunas de las vulnerabilidades descubiertas en la plataforma y para ello usaremos una métrica de CVSS v2.

2	Información sensible en base de datos		CVSS 4.9

DVIA-v2-swift.ipa

AV	AC	Au	C	I	A
Local	Baja	Ninguna	Completa	Ninguna	Ninguna

Descripción

Algunas aplicaciones almacenan peticiones en la base de datos de cache para poder acceder a ellas más tarde.

Las caches están en el siguiente directorio:

- /var/mobile/Containers/Data/Application/DVIA}/Library/Caches/

Se ha detectado que la aplicación almacena información sensible en la cache (Cache.db y Cache.db-wal):

- Información del usuario
- Tokens
- Contraseña en claro

```
iPhone-de-isec:/var/mobile/Containers/Data/Application/5D9A5147-D485-4464-94F1-B250EED79790 root
# grep -rnw . -e isecdb
Binary file ./Library/Application Support/CouchbaseLite/dvcouchbasedb.cblite matches
iPhone-de-isec:/var/mobile/Containers/Data/Application/5D9A5147-D485-4464-94F1-B250EED79790 root
# sqlite3 Library/Application\ Support/CouchbaseLite/dvcouchbasedb.cblite
SQLite version 3.24.0 2018-06-04 19:24:41
Enter ".help" for usage hints.
sqlite> .tables
attachments        docs               fulltext_segments  replicators
bboxes             fulltext           fulltext_stat      revs
bboxes_node        fulltext_content   info               views
bboxes_parent      fulltext_docsize   localdocs
bboxes_rowid       fulltext_segdir    maps
sqlite> SELECT * FROM revs;
1|1|1-068abd51dd8943620f317e51f187cf6f||1|0|{"password":"isecdbpass","username":"isecdb"}|1
```

Credenciales del usuario

Un atacante con acceso al directorio de datos de la aplicación podría obtener información sensible del usuario, incluyendo su contraseña en claro.

Recomendaciones

Se recomienda no almacenar información sensible no cifrada en los ficheros de cache de la aplicación.

Se debe especificar explícitamente en el código que no se quiere usar caché, ya que automáticamente se almacenan las peticiones cuando se usa *NSURLCache* y se crea una *NSURLRequest*.

Alternativamente, se puede usar SQLcipher de Sqlite para almacenar datos cifrados en base de datos.

3	Información sensible en plist		CVSS 4.9

DVIA-v2-swift.ipa

AV	AC	Au	C	I	A
Local	Baja	Ninguna	Completa	Ninguna	Ninguna

Descripción

Plist (lista de propiedades) es un formato flexible y conveniente para almacenar los datos de aplicaciones. Fue originalmente definido por Apple para su uso en dispositivos iPhone y posteriormente se extendió a otras aplicaciones.

Dado que las plists son realmente archivos XML, se puede utilizar un editor de texto simple para traducirlas.

Se ha detectado que la aplicación almacena información sensible, como usuarios y contraseñas, en archivos PLIST sin cifrar.

Mediante la siguiente comanda se ha podido listar todos los archivos con extensión plist:

```
iPhone-de-isec:/var/mobile/Containers/Data/Application/5D9A5147-D485-4464-94F1-B250EED79790 r
# grep -rnw . -e isecplist
./Documents/userInfo.plist:8:    <string>isecplist</string>
```

Se ha detectado que la aplicación almacena información sensible en userInfo.plist:

- Información del usuario
- Contraseña en claro

```
iPhone-de-isec:/var/mobile/Containers/Data/Application/5D9A5147-D485-4464-94F1-B250EED79790 roo
# cat Documents/userInfo.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.
.dtd">
<plist version="1.0">
<dict>
        <key>password</key>
        <string>isecplistpass</string>
        <key>username</key>
        <string>isecplist</string>
</dict>
</plist>
```

Credenciales del usuario

Un atacante con acceso al directorio de datos de la aplicación podría obtener información sensible del usuario, incluyendo su contraseña en claro.

Recomendaciones

Se recomienda no almacenar información sensible no cifrada en los ficheros plist ya que no es un formato diseñado con este fin.

A medida de lo posible, toda la información sensible debería estar almacenada en el servidor. De no ser posible, debería almacenarse en bases de datos cifradas.

4	Información sensible en los logs		CVSS 4.9

DVIA-v2-swift.ipa

AV	AC	Au	C	I	A
Local	Baja	Ninguna	Completa	Ninguna	Ninguna

Descripción

Durante el desarrollo de la aplicación es común que los desarrolladores habiliten el uso de logs de DEBUG para comprobar el correcto funcionamiento de la aplicación.

Mediante el uso de socat o oslog, ha sido posible leer información sensible al registrar un usuario.

```
iPhone-de-isec:/var/mobile/Containers/Data/Application/5D9A5147-D485-4464-94F1-B250EED79790 root
# socat - UNIX-CONNECT:/var/run/lockdown/syslog.sock

=======================
ASL is here to serve you
> watch
OK

Jul 30 14:53:44 iPhone-de-isec kernel[0] <Debug>:
 Jul 30 14:53:44 iPhone-de-isec com.apple.xpc.launchd[1] (com.apple.ReportCrash[1684]) <Notice>:
 Service exited due to SIGSEGV | sent by exc handler[1684]
```

Activación de socat

```
Jul 30 15:05:06 iPhone-de-isec kernel[0] <Debug>:
 Jul 30 15:05:07 iPhone-de-isec kernel[0] <Debug>:
 Jul 30 15:05:07 iPhone-de-isec com.apple.xpc.launchd[1] (com.apple.ReportCrash[1776]) <Notice
 Service exited due to SIGSEGV | sent by exc handler[1776]
 Jul 30 15:05:07 iPhone-de-isec com.apple.xpc.launchd[1] (com.apple.ReportCrash) <Notice>: Ser
ce only ran for 1 seconds. Pushing respawn out by 9 seconds.
 Jul 30 15:05:08 iPhone-de-isec DamnVulnerableIOSApp[1532] <Warning>: user saved: <Person: 0x2
5d8000, objectId: new, localId: (null)> {
            email = "debug mode";
            name = "isec logging";
            password = "isec adminpassword";
            phone = 1234567890;
        }
 Jul 30 15:05:09 iPhone-de-isec DamnVulnerableIOSApp[1532] <Warning>: [Error]: Error Domain=PF
FNetworkingErrorDomain Code=-1011 "Expected status code in (200-299), got 410" UserInfo={PF_AF
tworkingOperationFailingURLRequestErrorKey=<NSMutableURLRequest: 0x2816d2150> { URL: https://a
.parse.com/2/create }, NSLocalizedRecoverySuggestion=, NSErrorFailingURLKey=https://api.parse.
m/2/create, PF_AFNetworkingOperationFailingURLResponseErrorKey=<NSHTTPURLResponse: 0x2814bc9a0
{ URL: https://api.parse.com/2/create } { Status Code: 410, Headers {
            Connection =     (
```

Información sensible

Un atacante con acceso al dispositivo podría obtener información sensible del usuario, incluyendo su contraseña en claro.

Recomendaciones

Antes de subir la aplicación a producción, se recomienda desactivar el modo DEBUG y eliminar todas las lineas de código que muestren o almacenen información sensible el los logs.

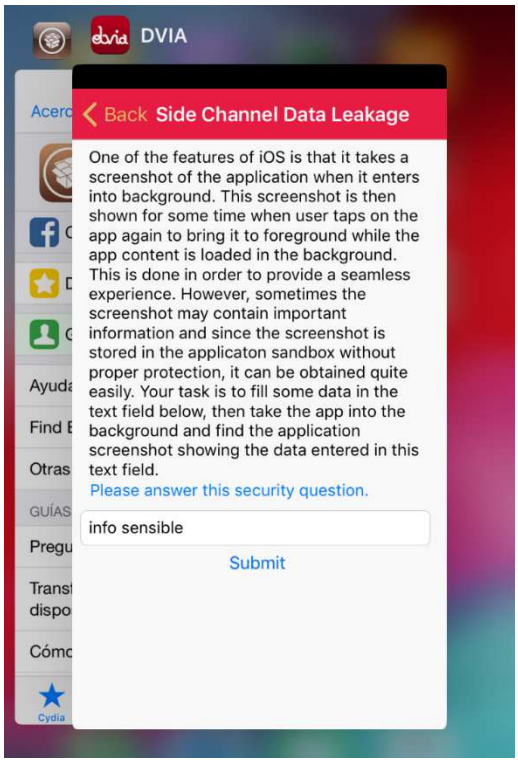
5	Screen Shot habilitados		CVSS 2.1

DVIA-v2-swift.ipa

AV	AC	Au	C	I	A
Local	Baja	Ninguna	Parcial	Ninguna	Ninguna

Descripción

Cuando el usuario pulsa el botón de “home” del dispositivo y la aplicación pasa a segundo plano, el sistema operativo automáticamente hace una captura de pantalla de la aplicación para mostrar al usuario la vista en miniatura.



Vista de la aplicación en background

Cada vez que la aplicación va en segundo plano, iOS hace una captura de pantalla y la almacena en:
/private/var/mobile/Containers/Data/Application/{DIRECTORIO DATOS APPLICACIÓN}/Library/Caches/Snapshots

Un atacante con acceso al dispositivo podría descargar las fotos y ver la información que hubiera en la aplicación en el momento de passar a segundo plano, incluyendo información sensible.

Recomendaciones

Se recomienda que se configure la aplicación para que en el momento de pasar a segundo plano, cambie la vista y muestre una imagen genérica. Por ejemplo, una imagen toda en negro o blanco o el logo de la empresa.

De esta manera, la captura de pantalla que haga el sistema operativo no contendrá información sensible.

6	Evasión de Mecanismo Jailbreak		CVSS 6.3

DVIA-v2-swift.ipa

AV	AC	Au	C	I	A
Local	Media	Ninguna	Completa	Completa	Ninguna

Descripción

Jailbreak es una técnica para eliminar los sistemas de seguridad de los dispositivos iOS, como por ejemplo el "sandbox" o la protección de la ejecución de comandos como root.

Se ha detectado que la aplicación dispone de un control de anti-jailbreak ya que sale un pop-up al intentar ejecutar la funcionalidad:

Detección de Jailbreak

Existen dos formas de evadir el control de Jailbreak:

- Dinámicamente: Objection o Frida
- Estáticamene: IDA Pro o Hopper

Objection

Objection permite realizar un análisis de la aplicación en tiempo de ejecución.

- URL: https://github.com/sensepost/objection

Primero se debe detectar cual es la función que realiza el control de anti-jailbreak.
En este caso, buscando clases con el nombre "jail" en el, se ha obtenido el siguiente resultado:

```
com.highaltitudehacks.dvia on (iPhone: 12.1.1) [usb] # ios hooking search classes ja
JailbreakDetectionVC
```

Clase obtenida con Objection

A continuación, se listan los métodos de la clase "JailbreakDetectionVC":

```
com.highaltitudehacks.dvia on (iPhone: 12.1.1) [usb] # ios hooking list class_methods JailbreakDetect
- isJailbroken
- readArticleTapped:
- jailbreakTest1Tapped:
- jailbreakTest2Tapped:
- initWithNibName:bundle:
- viewDidLoad
- didReceiveMemoryWarning

Found 7 methods
```

Metodos de “JailbreakDetectionVC”

Por el nombre, se decide que el método “isJailbroken” es probablemente el que comprueba si el dispositivo tiene Jailbreak.

Para analizar el comportamiento del método, se realiza un hookeo:

```
com.highaltitudehacks.dvia on (iPhone: 12.1.1) [usb] # ios hooking watch method "-[JailbreakDetectionVC isJailbroke
--dump-args --dump-return
(agent) Found selector at 0x100afb808 as -[JailbreakDetectionVC isJailbroken]
(agent) Registering job aogw2k3iwxq. Type: watch-method for: -[JailbreakDetectionVC isJailbroken]
```

Hook al método “isJailbroken”

Cuando pulsamos la funcionalidad y aparece el pop-up que bloquea la ejecución, se puede observar que se llama al método “isJailbroken” y este devuelve un 1 en hexadecimal:

```
com.highaltitudehacks.dvia on (iPhone: 12.1.1) [usb] # (agent) [owtk2mxfnkp] Called: -[JailbreakDetectionVC jailbre
t1Tapped:] 1 arguments(Kind: instance) (Super: UIViewController)
(agent) [owtk2mxfnkp] Argument dump: [JailbreakDetectionVC jailbreakTest1Tapped: <UIButton: 0x109e4ba90; frame = (6
 320 44); opaque = NO; autoresize = RM+BM; layer = <CALayer: 0x282161700>>]
(agent) [aogw2k3iwxq] Called: -[JailbreakDetectionVC isJailbroken] 0 arguments(Kind: instance) (Super: UIViewContro

(agent) [aogw2k3iwxq] Return Value: 0x1
(agent) [owtk2mxfnkp] Return Value: 0x109e4ba90
```

Respuesta del método “isJailbroken”

Por lo tanto, podemos suponer que el método devuelve un “1” si el dispositivo esta con Jailbrek y un “0” si no lo está.

Mediante el siguiente comando de Objection, se realiza un hook en el método de tal manera que al devolver el “1”, automáticamente sea cambiado por un “0”:

```
com.highaltitudehacks.dvia on (iPhone: 12.1.1) [usb] # ios hooking set return_value "-[JailbreakDetectionVC isJail
n]" False
(agent) Found selector at 0x100afb808 as -[JailbreakDetectionVC isJailbroken]
(agent) Registering job 6flb6ica3gj. Type: set-method-return for: -[JailbreakDetectionVC isJailbroken]
com.highaltitudehacks.dvia on (iPhone: 12.1.1) [usb] # (agent) [owtk2mxfnkp] Called: -[JailbreakDetectionVC jailbr
t1Tapped:] 1 arguments(Kind: instance) (Super: UIViewController)
(agent) [owtk2mxfnkp] Argument dump: [JailbreakDetectionVC jailbreakTest1Tapped: <UIButton: 0x109e4ba90; frame = (
 320 44); opaque = NO; autoresize = RM+BM; layer = <CALayer: 0x282161700>>]
(agent) [aogw2k3iwxq] Called: -[JailbreakDetectionVC isJailbroken] 0 arguments(Kind: instance) (Super: UIViewContr

(agent) [aogw2k3iwxq] Return Value: 0x1
(agent) [6flb6ica3gj] -[JailbreakDetectionVC isJailbroken] Return value was: 0x1, overriding to 0x0
(agent) [owtk2mxfnkp] Return Value: 0x109e4ba90
```

Modificación del resultado

IDA Pro

Permite hacer un análisis del código fuente de la aplicación para entender como funciona e incluso modificar su comportamiento.
A diferencia del análisis dinámico, es necesario signar y reinstalar la aplicación usando el binario modificado para poder ver los cambios.

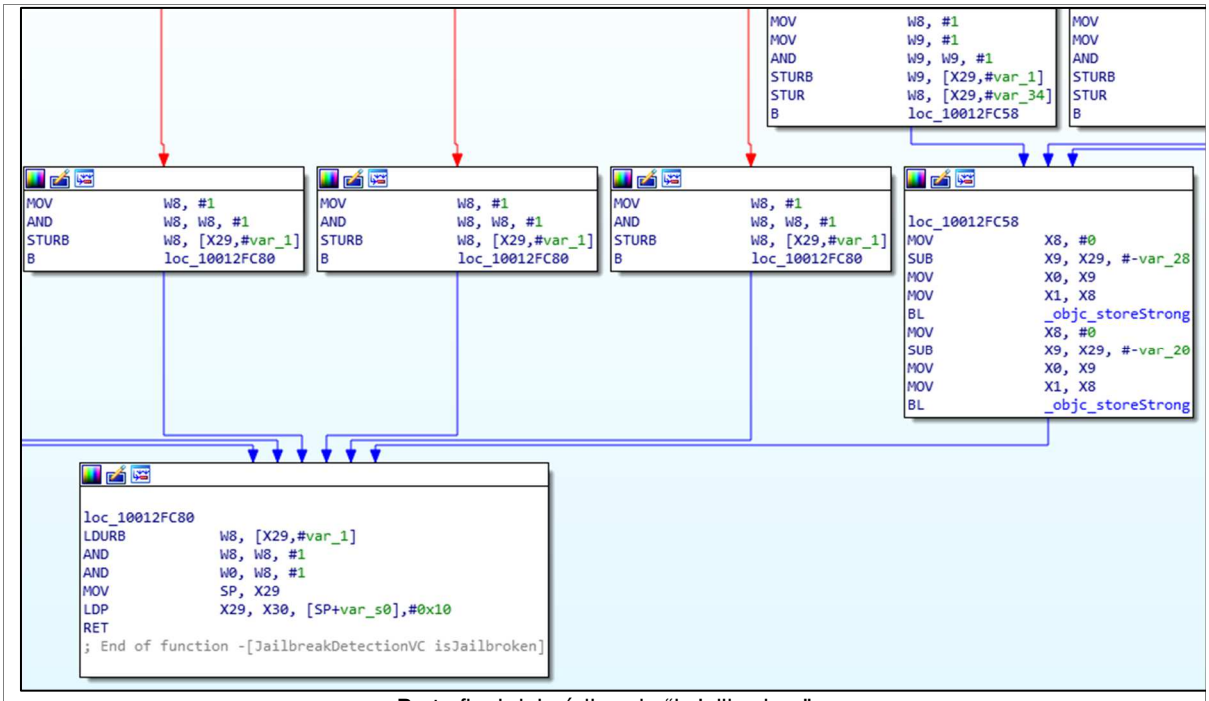
Analizando las funciones que contienen “jail” en el nombre, “jailbreakTest1Tapped” tiene el siguiente código:

```
; void __cdecl -[JailbreakDetectionVC jailbreakTest1Tapped:](JailbreakDetectionVC *self, SEL, id)
__JailbreakDetectionVC_jailbreakTest1Tapped__

var_24= -0x24
var_20= -0x20
var_18= -0x18
var_10= -0x10
var_8= -8
var_s0=  0

STP             X29, X30, [SP,#-0x10+var_s0]!
MOV             X29, SP
SUB             SP, SP, #0x30
ADD             X8, SP, #0x30+var_18
MOV             X9, #0
STUR            X0, [X29,#var_8]
STUR            X1, [X29,#var_10]
STR             X9, [SP,#0x30+var_18]
MOV             X0, X8
MOV             X1, X2
BL              _objc_storeStrong
ADRP            X8, #selRef_isJailbroken@PAGE
ADD             X8, X8, #selRef_isJailbroken@PAGEOFF
ADRP            X9, #classRef_DamnVulnerableAppUtilities@PAGE
ADD             X9, X9, #classRef_DamnVulnerableAppUtilities@PAGEOFF
LDR             X9, [X9] ; _OBJC_CLASS_$_DamnVulnerableAppUtilities
LDUR            X0, [X29,#var_8]
LDR             X1, [X8] ; "isJailbroken"
STR             X9, [SP,#0x30+var_20]
BL              _objc_msgSend
ADRP            X8, #selRef_showAlertForJailbreakTestIsJailbroken_@PAGE
ADD             X8, X8, #selRef_showAlertForJailbreakTestIsJailbroken_@PAGEOFF
LDR             X1, [X8] ; "showAlertForJailbreakTestIsJailbroken:"
LDR             X8, [SP,#0x30+var_20]
STR             W0, [SP,#0x30+var_24]
MOV             X0, X8  ; void *
LDR             W10, [SP,#0x30+var_24]
AND             W2, W10, #1
BL              _objc_msgSend
ADD             X0, SP, #0x30+var_18
MOV             X8, #0
MOV             X1, X8
BL              _objc_storeStrong
MOV             SP, X29
LDP             X29, X30, [SP+var_s0],#0x10
RET
; End of function -[JailbreakDetectionVC jailbreakTest1Tapped:]
```

Código de “jailbrakTest1Tapped”

Se puede observar como se hace una llamada a “isJailbroken” y que dependiendo del resultado se lanzará el pop-up.

Cada caja hace una comprobación distinta con el objetivo de detectar si el dispositivo tiene Jailbreak. Si la comprobación es correcta (no Jailbreak), la ejecución del programa pasa a realizar la siguiente comprobación. De lo contrario, salta directamente al final y devuelve un “1”.



Parte final del código de "isJailbroken"

Por lo tanto, si se modifica el registro W8 para que siempre devuelva un "0", es posible evadir el control de anti-Jailbreak.

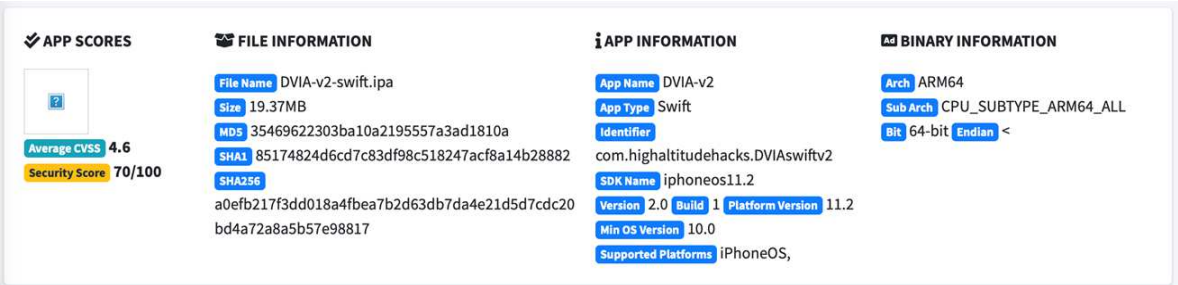
Recomendaciones

Se recomienda implementar un control de Jailbreak más complejo.

Además, devido a que el atacante dispone de un tiempo indefinido para evadir el control, se recomienda añadir complejidad mediante la ofuscación del código.

4.2.5. Informe automático con Mobsf

MobSF es una herramienta recomendada por OWASP en su Guía de pruebas de seguridad móvil para plataformas iOS tanto en Objective-c como Swift. Esta herramienta se basa en patrones del Estándar de verificación de seguridad de aplicaciones móviles OWASP y de la Guía de pruebas de seguridad móvil OWASP. Para entornos iOS, Mobsf únicamente proporcionará un análisis estático de la Aplicación objetivo.



DVIA-v2 (2.0)

File Name:	**DVIA-v2-swift.ipa**
Identifier:	**com.highaltitudehacks.DVIAswiftv2**
Average CVSS Score:	**4.6**
App Security Score:	**70/100 (MEDIUM RISK)**

FILE INFORMATION

File Name: DVIA-v2-swift.ipa
Size: 19.37MB
MD5: 35469622303ba10a2195557a3ad1810a
SHA1: 85174824d6cd7c83df98c518247acf8a14b28882
SHA256: a0efb217f3dd018a4fbea7b2d63db7da4e21d5d7cdc20bd4a72a8a5b57e98817

APP INFORMATION

App Name: DVIA-v2
App Type: Swift
Identifier: com.highaltitudehacks.DVIAswiftv2
SDK Name: iphoneos11.2
Version: 2.0
Build: 1
Platform Version: 11.2
Min OS Version: 10.0
Supported Platforms: iPhoneOS,

BINARY INFORMATION

Arch: ARM64
Sub Arch: CPU_SUBTYPE_ARM64_ALL
Bit: 64-bit
Endian: <

#CUSTOM URL SCHEMES

URL NAME	SCHEMES

URL NAME	SCHEMES
com.highaltitudehacks.DVIAswiftv2	dvia dviaswift

APPLICATION PERMISSIONS

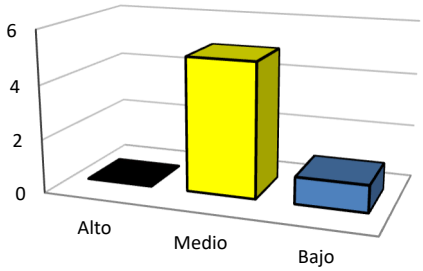
PERMISSIONS	STATUS	DESCRIPTION	REASON IN MANIFEST
NSCameraUsageDescription	dangerous	Access the Camera.	To demonstrate the misuse of Camera, please grant it permission once.

4.2.6. Resumen Ejecutivo

Nivel de seguridad Global

NIVEL MEDIO
6 / 10

Total Vulnerabilidades



RESUMEN	
Nivel	**Total**
Alto	0
Medio	5
Bajo	1
TOTAL	**6**

El objeto del análisis de aplicación iOS es el de detectar las deficiencias de seguridad mediante una exhaustiva Auditoría Externa de Seguridad, aportando, en caso de existir diferentes alternativas, la solución o soluciones más adecuada a cada una de estas deficiencias.

El nivel de seguridad Global es Medio ya que no se ha detectado vulnerabilidades de Nivel Alto y las vulnerabilidades de Nivel Medio descubiertas están relacionadas con la exposición de información sensible y la evasion de algunos mecanismos de seguridad implementados.

Se recomienda que, tras implantar las medidas correctoras definidas en este informe, se realice una Auditoría de Verificación con el fin de validar que todas las vulnerabilidades han sido solventadas y su solución no ha provocado nuevas deficiencias.

5.Conclusiones

Las millones de Aplicaciones Móviles disponibles en repositorios o tiendas especializadas como Aple Store o Google Play, al ofrecer todo tipo de servicios, se han vuelto una parte esencial de nuestro día a día y manejan más datos sensibles que cualquier otro medio con lo que se deben tomar las precauciones necesarias para evitar cualquier incidente de seguridad.

McAfee estima que para 2030, los usuarios tendrían hasta 15 dispositivos conectados. Ésto hace pensar que los ataques estarán cada vez más dirigidos contra los canales en los que las personas pasen más tiempo, utilizando técnicas como inicios de sesión no autorizados, anuncios no deseados y malware, entre otras.

Además se deberían tener en cuenta los ataques como consecuencia del uso de bibliotecas externas (casi el 80% de aplicaciones móviles usan bibliotecas de terceros) o como resultado de un error durante el proceso de pruebas y la subida a producción de una aplicación.

Estos datos 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 y su oportuna ejecución evitaría que las aplicaciones móviles se distribuyan con defectos o vulnerabilidades que pueden llevar a la filtración de datos y a actividades maliciosas.

A la hora de planificar la revisión de seguridad de una application móvil se debe identificar qué recursos gestiona dicha aplicación, qué información se almacena en el teléfono móvil, tablet o IPAD y qué información se transmite. Dependiendo de los requisitos de seguridad de la aplicación y de su criticidad entran en juego diferentes técnicas de análisis de seguridad agrupadas en las siguientes fases: Recopilacion de información, un análisis estático y dinámico de la aplicación objetivo.

En resumen, aunque el 100% de seguridad no existe o no se puede garantizar, debemos 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.

Bibliografía

[1] Amenazas a la seguridad móvil para Android
https://www.kaspersky.es/resource-center/threats/mobile
[2] OWASP Mobile Security Project
https://owasp.org/www-project-mobile-security/
[3] OWASP Mobile Application Security Verification Standard
https://github.com/OWASP/owasp-masvs
[4] OWASP Mobile Security Testing Guide
https://github.com/OWASP/owasp-mstg
[5] Desarrollo seguro de aplicaciones para dispositivos móviles
https://www.incibe-cert.es/blog/desarrollo-seguro-de-aplicaciones-para-dispositivos-moviles
[6] Guía de Privacidad en el móvil
https://mmaspain.com/wp-content/uploads/Guia-Privacidad-Movil.pdf
[7] El RGPD: El Reglamento de Protección de Datos
https://www.ionos.es/digitalguide/paginas-web/derecho-digital/el-rgpd-normativa-europea-de-proteccion-de-datos/
[8] More than one billion Android devices at risk of malware threats
https://www.which.co.uk/news/2020/03/more-than-one-billion-android-devices-at-risk-of-malware-threats/
[9] Los 3 Tipos De Aplicaciones Móviles: Ventajas E Inconvenientes
https://www.lancetalent.com/blog/tipos-de-aplicaciones-moviles-ventajas-inconvenientes/
[10] Auditando Aplicaciones en Android
https://hacking-etico.com/2016/02/25/auditando-aplicaciones-en-android/
[11] Cómo ejecutar Apps en Android Emulator
https://developer.android.com/studio/run/emulator
[12] Herramientas para analizar APK
https://blog.segu-info.com.ar/2017/03/herramientas-para-analizar-apk-app.html?m=0
[13] 9 Mobile App Scanner to Find Security Vulnerabilities
https://geekflare.com/mobile-app-security-scanner/#Mobile-App-Scanner
[14] Application Sandbox
https://source.android.com/security/app-sandbox
[15] Tampering and Reverse Engineering on iOS
https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06c-reverse-engineering-and-tampering