UNIVERSIDAD TÉCNICA DE MANABÍ
Fundada en 1952

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Análisis y Detección de Malware en Dispositivos Android versión 15 mediante Técnicas de Ingeniería Inversa

Malware Analysis and Detection in Android 15 Devices through Reverse Engineering Techniques

Autores

Anthony Alexander Contreras Espinoza[1]*
acontrera3@utmachala.edu.ec

Joofre Antonio Honores Tapia[1]
jhonores@utmachala.edu.ec

Milton Rafael Valarezo Pardo[1]
mvalarezo@utmachala.edu.ec

Tania Yesminia Contreras Alonso[2]
ingtaniacontreras@gmail.com

[1] Universidad Técnica de Machala, Facultad de Ingeniería Civil, Carrera de Tecnologías de la Información, Machala, El Oro, Ecuador.
[2] Unidad Educativa Ismael Pérez Pazmiño, Machala, El Oro, Ecuador.

*Autor para correspondencia

Comó citar el artículo:
Contreras Espinoza, A.A., Honores Tapia, J.A., Valarezo Pardo, M.R. & Contreras Alonso, T.Y. 2026. Análisis y Detección de Malware en Dispositivos Android versión 15 mediante Técnicas de Ingeniería Inversa. *Informática y Sistemas, 10(1)*, 24-39. https://doi.org/10.33936/isrtic.v10i1.8098

Enviado: 15/12/2025
Aceptado: 27/01/2026
Publicado: 30/01/2026

Resumen

El presente artículo describe un estudio de caso de análisis y detección de malware en dispositivos Android, centrado en la aplicación de IPTV MagisTV ejecutada en Android 8 y Android 15. MagisTV se eligió como unidad de análisis por su amplio uso, la solicitud de permisos sensibles y la generación intensiva de tráfico de red potencialmente abusivo. Se emplea un enfoque híbrido guiado por ingeniería inversa que combina análisis estático del Manifest, componentes, bibliotecas nativas y APIs sensibles con análisis dinámico del comportamiento en escenarios controlados de uso: arranque, inicio de sesión, navegación por menús, reproducción de video en streaming, uso en segundo plano, cierre forzado y reinicio, operaciones de almacenamiento, pruebas de persistencia y evaluación de la interceptación TLS. La metodología se apoya en herramientas especializadas (MobSF, Apktool, Androguard, JADX, ADB/logcat, monkey/UIAutomator, tcpdump, Wireshark y mitmproxy) y en la captura sistemática de trazas de red y eventos del sistema, manteniendo trazabilidad entre lo observado en el código y en la ejecución. Los resultados muestran que MagisTV presenta una superficie de permisos amplia y una arquitectura compleja, pero que en los escenarios probados se comporta como un cliente de IPTV sin uso efectivo de cámara o micrófono ni autoarranque tras el reinicio, con diferencias de observabilidad entre Android 8 y Android 15 debidas al endurecimiento de la plataforma. El estudio resalta la importancia de integrar evidencia estática y dinámica y de considerar la versión de Android como contexto para la detección explicable y la formación en análisis de malware móvil.

Palabras clave: Android; malware; ingeniería inversa; análisis estático y dinámico; MagisTV

Abstract

This article presents a case study on malware analysis and detection in Android devices, focusing on the IPTV application MagisTV running on Android 8 and Android 15. MagisTV was selected as the unit of analysis due to its widespread use, its request for sensitive permissions and its intensive, potentially abusive network traffic. A hybrid, reverse-engineering-guided approach is used, combining static analysis of the Manifest, components, native libraries and sensitive APIs with dynamic analysis of runtime behaviour under controlled usage scenarios: startup, login, menu navigation, video streaming, background use, forced close and restart, storage operations, persistence test and TLS interception attempts. The methodology relies on specialised tools (MobSF, Apktool, Androguard, JADX, ADB/logcat, monkey/UIAutomator, tcpdump, Wireshark and mitmproxy) and on the systematic capture of network traces and system events, preserving end to end traceability between code level findings and observable behaviour. The results show that MagisTV exhibits a broad permission surface and complex architecture, but that in the tested scenarios it behaves as an IPTV client, without effective use of camera or microphone and without auto start after reboot, while still maintaining continuous or intermittent communications depending on the Android version. The comparison between Android 8 and Android 15 highlights differences in observability linked to platform hardening, particularly in storage access and background execution. The study underscores the importance of integrating static and dynamic evidence and of explicitly considering the Android version as contextual information for explainable detection and for the design of educational labs in mobile malware analysis.

Keywords: Android; malware; reverse engineering; static and dynamic analysis; MagisTV

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

1. Introducción

En términos generales, el malware se refiere a software con fines no autorizados o maliciosos, capaz de degradar el rendimiento del sistema operativo, alterar sus servicios y comprometer la confidencialidad, integridad y disponibilidad de la información que gestiona (Almomani et al., 2022; Smmarwar et al., 2024). En el ámbito móvil, la masificación de los smartphones ha hecho posible la aparición de múltiples sistemas operativos, siendo los más representativos iOS, Android (incluyendo derivados como Fire OS de Amazon), HarmonyOS y, en menor medida, plataformas como KaiOS; sin embargo, el que representa alrededor del 76% del mercado global de sistemas operativos móviles es Android, que desde la versión 5.0 (Lollipop) ha incorporado de forma continua actualizaciones en la interfaz de usuario, los servicios del sistema, la gestión de la privacidad y los mecanismos de seguridad, reforzadas en versiones posteriores hasta Android 15, lo que trae consigo múltiples preocupaciones precisamente por ser uno de los sistemas operativos más utilizados. La distribución global de sistemas operativos móviles se presenta en la Figura 1.

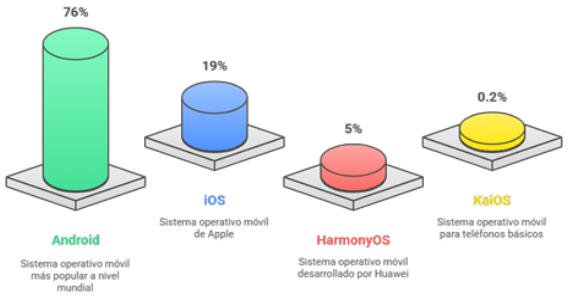

Figura 1. Sistemas operativos móviles más usados a nivel global.
Fuente: Los autores.

Esta posición dominante ha convertido a Android en un objetivo prioritario para atacantes, que despliegan familias de malware como adware, troyanos bancarios, ransomware o spyware, aprovechando permisos sensibles, llamadas a API críticas y, en variantes más avanzadas, técnicas de ofuscación, carga dinámica y código nativo (Aamir et al., 2024; Almomani et al., 2022). Frente a este escenario, la literatura reciente coincide en que los permisos declarados en el Manifest y determinadas llamadas a API proporcionan una señal explicable para modelar riesgo, que los enfoques híbridos (análisis estático + dinámico) capturan mejor el comportamiento real que el análisis estático aislado, y que las representaciones estructurales del código en forma de grafos procesadas con redes neuronales de grafos (GNN) incrementan la robustez frente a variantes y ofuscación al explotar relaciones semánticas más estables (Gao et al., 2021; Muzaffar et al., 2022; Palma et al., 2023).

A pesar de estos avances, buena parte de los trabajos se apoya en grandes conjuntos de datos públicos y en versiones anteriores de Android, priorizando la construcción de clasificadores sobre vectores de características (permisos, APIs, tráfico de red, system calls) más que el estudio detallado, mediante ingeniería inversa, de cómo se manifiesta una misma amenaza en entornos concretos (Muzaffar et al., 2022; Pathak et al., 2024). Son escasas las demostraciones comparativas y controladas que conecten, con trazabilidad de extremo a extremo, los permisos y llamadas a API extraídos del código con el tráfico de red y los perfiles de system calls observados al ejecutar la misma APK maliciosa en versiones con distinto nivel de protección, por ejemplo, Android 8 frente a Android 15, a pesar de la relevancia de estos observables para caracterizar categorías y patrones de comportamiento malicioso (Gohari et al., 2021; Manzil & Manohar Naik, 2023). En síntesis, existe un vacío de investigación relacionado con estudios comparativos controlados que vinculen de manera sistemática los hallazgos del análisis estático y dinámico de una misma aplicación maliciosa entre versiones de Android con diferente nivel de seguridad. En este trabajo se aborda esa brecha mediante el análisis, estático y dinámico, de una única aplicación de IPTV considerada maliciosa en nuestro escenario experimental, MagisTV, ejecutada en Android 8 y Android 15.

Se plantea la hipótesis de que un enfoque híbrido guiado por ingeniería inversa, integrando un análisis estático (permisos, llamadas a API y artefactos recuperados tras deofuscación) y una validación dinámica (tráfico de red y system calls) aplicado a la APK MagisTV, permitirá identificar y explicar de forma consistente los indicadores de comportamiento malicioso en un escenario comparativo en las versiones Android 8 y Android 15.

En coherencia con esta hipótesis, el objetivo general del estudio es analizar la APK de MagisTV al ejecutarse en las versiones 8 y 15 del sistema operativo Android, mediante herramientas de ingeniería inversa como JADX, Apktool, Androguard y MobSF, para detectar y explicar los indicadores maliciosos generados.

En este contexto, el aporte de este estudio son tres hallazgos. Primero, se diseña y ejecuta un experimento controlado en el que la APK de MagisTV es analizada y ejecutada en Android 8 y Android 15, permitiendo contrastar cómo las mejoras de la plataforma afectan la observabilidad del comportamiento malicioso. Segundo, se integra en un único flujo un análisis estático guiado por ingeniería inversa, mediante herramientas como JADX (decompilación de bytecode Dalvik/ART a código Java), Apktool (reconstrucción de AndroidManifest.xml y código Smali), Androguard (análisis de bytecode y, cuando procede,

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

grafos de llamadas) y MobSF (marco automatizado para análisis estático y dinámico de APKs), con una validación dinámica basada en ejecución controlada a través de ADB/logcat y generación de tráfico en emuladores o dispositivos, junto con captura y análisis de trazas de red (PCAP) mediante herramientas como tcpdump, Wireshark o man-in-the-middle proxy, manteniendo la trazabilidad entre lo inferido en el código y lo observado en tiempo de ejecución (Chen et al., 2021; Gohari et al., 2021). Tercero, se analizan las implicaciones de estos hallazgos para la detección explicable de malware móvil a la luz del estado del arte en aprendizaje automático, enfoques híbridos y representaciones estructurales de aplicaciones Android, resaltando el papel de los permisos y de las características semánticas como elementos clave para la interpretabilidad del modelo de riesgo (Almomani et al., 2022; Gu et al., 2024; Palma et al., 2023).

2. Materiales y Métodos

La investigación se desarrolló en un entorno doméstico controlado, ubicado en Pasaje (Ecuador), durante el período de septiembre a noviembre de 2025, con una duración total de 12 semanas, con el objetivo de analizar, bajo condiciones reproducibles, el comportamiento de una misma APK MagisTV en dos versiones del sistema operativo Android. Se seleccionó, mediante muestreo intencional, la APK MagisTV como caso instrumental para el análisis comparativo entre Android 8 y Android 15, y su inclusión se supeditó a que fuera instalable y ejecutable en ambos entornos, permitiera la extracción estática de artefactos clave (AndroidManifest.xml y código Smali/bytecode) y generara trazas dinámicas observables (tráfico de red y system calls) en condiciones controladas. Se registraron metadatos básicos y el hash SHA-256 para asegurar trazabilidad y reproducibilidad, conforme a recomendaciones habituales en estudios de malware Android (Muzaffar et al., 2022; Pathak et al., 2024).

2.1 Diseño del estudio y enfoque comparativo

Se empleó un diseño cuasi-experimental comparativo con dos condiciones de sistema operativo: Android 8 (Dispositivo A) y Android 15 (Dispositivo B). La unidad de análisis es una única aplicación de IPTV, MagisTV, considerada maliciosa en el escenario experimental por su capacidad de solicitar permisos sensibles y ejecutar tráfico de red potencialmente abusivo. El enfoque es aplicado, cuantitativo e híbrido: combina análisis estático guiado por ingeniería inversa (permisos del Manifest, llamadas a APIs sensibles y artefactos asociados a ofuscación/deofuscación) con observación dinámica del comportamiento en ejecución (tráfico de red y system calls), en línea con trabajos que resaltan la utilidad de integrar evidencias estáticas y dinámicas en la detección de malware en Android (Almomani et al., 2022; Gohari et al., 2021; Muzaffar et al., 2022).

En este marco, la versión de Android (8 vs 15) se define como variable independiente, mientras que las variables dependientes agrupan indicadores estáticos (permisos, APIs, señales de ofuscación) y dinámicos (flujo de red y patrones de system calls). El objetivo del diseño comparativo es evaluar cómo las diferencias de seguridad entre Android 8 y Android 15 afectan la observabilidad y la expresión de comportamientos maliciosos de una misma APK bajo condiciones controladas (Palma et al., 2023).

2.2 Entorno de evaluación y herramientas

El entorno de evaluación se estructuró alrededor de un host de orquestación con Kali Linux, encargado de gestionar la instalación, ejecución y monitorización de la APK MagisTV en dos dispositivos físicos reales: Samsung Galaxy J7 Prime (SM-G610M) con Android 8.1.0 (Dispositivo A; revisión de seguridad: 1 de junio de 2020) e Infinix Note 40 Pro (X6850) con Android 15 (Dispositivo B; actualización de seguridad: 1 de octubre de 2025). En ambos casos, los equipos se mantuvieron sin acceso root (no rooteados) para preservar condiciones de operación cercanas a un entorno real. Con el fin de centralizar la observación y mejorar la reproducibilidad, se habilitaron las opciones de desarrollador, incluyendo depuración USB/ADB, y se mantuvo desactivado el desbloqueo de OEM en ambos dispositivos, evitando modificaciones de arranque durante el periodo experimental. Adicionalmente, se configuró en Kali Linux un punto de acceso (hotspot) al cual se conectaron ambos dispositivos durante las sesiones experimentales, permitiendo la captura de tráfico sin exponer sistemas externos. Sobre este host se instalaron las herramientas de análisis estático MobSF, Apktool, Androguard y JADX, utilizadas de forma complementaria para escanear la APK, descompilar el paquete, recuperar el AndroidManifest.xml y el código Smali/bytecode, extraer características estructuradas (permisos, APIs, componentes) y revisar manualmente fragmentos de código decompilado cuando fue necesario (Palma et al., 2023; Pathak et al., 2024). El flujo del análisis estático inicial con MobSF se resume en la Figura 2, y el conjunto de herramientas de ingeniería inversa para el análisis detallado se muestra en la Figura 3.

Figura 2. Flujo de análisis estático inicial con MobSF.
Fuente: Los autores.

Figura 3. Conjunto de herramientas de ingeniería inversa para el análisis estático detallado.
Fuente: Los autores.

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Para el análisis dinámico se emplearon ADB y logcat en la gestión de instalación, control y registro de eventos, mientras que monkey/UIAutomator permitieron generar secuencias de interacción reproducibles sobre la aplicación. La captura de tráfico se realizó con tcpdump y Wireshark, complementadas con mitmproxy cuando fue necesario inspeccionar tráfico HTTP(S) y derivar flujos a formatos compatibles con herramientas de extracción de características como CICFlowMeter (Gohari et al., 2021). La automatización de tareas repetitivas se apoyó en scripts bash y python, y, cuando se requirió evaluar resistencia a técnicas como certificate pinning, se utilizó Frida para instrumentar la aplicación en tiempo de ejecución, siguiendo prácticas habituales en la caracterización dinámica de malware Android (Manzil & Manohar Naik, 2023). Las herramientas empleadas, junto con su versión y propósito, se resumen en la Tabla 1. El conjunto de herramientas de captura y análisis dinámico se presenta en la Figura 4.

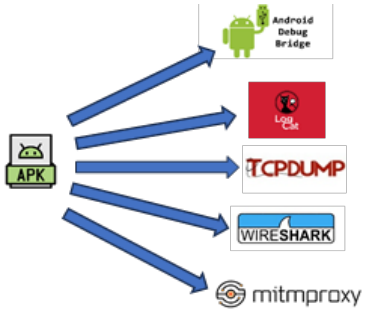

Figura 4. Herramientas de captura y análisis dinámico de tráfico y eventos.

2.3 Procedimiento general por fases

El estudio se organizó en fases secuenciales para garantizar la trazabilidad de todo el proceso, desde la preparación del entorno hasta la comparación de los resultados entre Android 8 y Android 15 utilizando la misma APK (MagisTV). La estructura metodológica se resume en la Figura 5.

Figura 5. Estructura metodológica de 8 fases para el análisis cuasi-experimental de la aplicación.
Fuente: Los autores.

Tabla 1. Herramientas, versiones y propósito en el estudio.
Fuente: Los autores.

Herramienta	Versión	Propósito en el estudio
MobSF	V4.4.3	Análisis estático automatizado (puntuación de seguridad, permisos, componentes, SDKs).
Apktool	2.7.0	Desempaquetado y extracción de recursos/AndroidManifest; soporte a ingeniería inversa.
JADX	1.5.3	Decompilación y revisión del código; búsqueda dirigida de llamadas a API/patrones sensibles.
Androguard	3.4.0a1	Extracción programática de metadatos y apoyo en análisis estático.
ADB	1.0.41	Instalación/ejecución controlada, gestión de pruebas y recolección de evidencias.
Logcat	1.0.41	Registro de eventos del sistema y trazas durante la ejecución.
UIAutomator	Android 8.1.0: v1.0 + (API 27). Android 15: v2.0 + (API 35).	Automatización de interacción (escenarios reproducibles de uso).
Monkey	Android 8.1.0: Integrada (API 27). Android 15: Integrada (API 35).	Generación de estímulos/eventos de UI para explorar rutas de ejecución.
Tcpdump	4.99.5	Captura de tráfico de red en el dispositivo (PCAP).
Wireshark	4.4.9	Inspección y análisis del tráfico capturado (PCAP).
CICFlowMeter	0.5.0	Transformación de PCAP a flujos y métricas cuantitativas para comparación.
Mitmproxy	12.2.0	Intercepción/observación de tráfico HTTP(S) en laboratorio cuando la aplicación lo permite.
Frida	17.5.2	Instrumentación dinámica puntual para observación de funciones y comportamientos (cuando aplica).

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

Fase 1. Planificación y diseño comparativo.

Definición de la hipótesis y de las variables del estudio: versión de Android (8 vs 15) como variable independiente; indicadores estáticos (permisos, APIs, artefactos de ofuscación/deofuscación) e indicadores dinámicos (tráfico de red y system calls) como variables dependientes. Se establecieron los criterios de inclusión de la APK y los lineamientos para asegurar que las condiciones de prueba fueran comparables entre ambos dispositivos.

Fase 2. Preparación del entorno.

Se configuró el host de orquestación (Kali Linux), se habilitó un punto de acceso (hotspot) y se conectaron los dispositivos físicos (Android 8 y Android 15) a dicha red de laboratorio. Las pruebas se realizaron sin privilegios de root en ambos dispositivos, habilitando únicamente opciones de desarrollador y depuración USB para la instrumentación. Se instalaron y verificaron las herramientas de análisis estático (MobSF, Apktool, Androguard, JADX) y dinámico (ADB, logcat, monkey/UIAutomator, tcpdump y Wireshark, mitmproxy), así como los scripts auxiliares de automatización necesarios para repetir de forma consistente las sesiones de prueba en ambos entornos.

Fase 3. Adquisición y verificación de la APK.

La APK bajo estudio MagisTV se obtuvo desde un sitio web público del distribuidor donde actualmente la aplicación se presenta bajo la denominación Xuper TV (marca empleada en la distribución pública vigente; no se identificó un repositorio oficial verificable de "MagisTV" en tiendas oficiales para la muestra analizada). Con fines de reproducibilidad, se documentó el URL de la página de referencia https://es.coimobile.io/download/xuper-tv-33978/, consulta:01-10-25, y se aseguró la trazabilidad la posibilidad de actualización obligatoria de la aplicación; en el caso de MagisTV, al intentar su ejecución se produjo una migración desde la versión 6.4.2 hacia 6.5.2 el 12-11-2025 (15:00), por lo que la evidencia dinámica reportada en este estudio corresponde a la versión efectiva en ejecución tras dicha actualización. En adelante, se denomina MagisTV v6.4.2 (estático) al artefacto analizado estáticamente y MagisTV v6.5.2 (dinámico) a la versión observada en ejecución.

Fase 4. Análisis estático guiado por ingeniería inversa.

Se realizo la ejecución de MobSF sobre la APK para obtener un primer informe de permisos, componentes y posibles indicadores de riesgo, a su vez se realizó descompilación con Apktool para recuperar el archivo AndroidManifest.xml y el código Smali, complementada con Androguard para la extracción de características estructurales (permisos, APIs sensibles, relaciones entre componentes) y con JADX para la revisión manual del código decompilado cuando se requirió refinar la interpretación. En esta fase se consolidó un conjunto de indicadores estáticos (vector de permisos, listas de APIs críticas, evidencia de ofuscación/deofuscación) que servirán como base para la comparación entre Android 8 y Android 15 (Palma et al., 2023; Pathak et al., 2024).

Fase 5. Instrumentación dinámica y generación de estímulos.

Instalación de la APK de MagisTV en los dispositivos A y B, configuración de ADB y logcat, y definición de guiones de interacción reproducibles mediante monkey/UIAutomator (por ejemplo, secuencias de navegación típicas dentro de la aplicación). Cada sesión se ejecutó bajo condiciones controladas, asegurando que las acciones realizadas fueran equivalentes en Android 8 y Android 15 de modo que las diferencias observadas pudieran atribuirse principalmente a la versión del sistema y no a variaciones en el uso (Gohari et al., 2021).

Se seleccionaron los escenarios dinámicos con criterios de representatividad funcional y cobertura de superficie de riesgo, buscando reproducir el recorrido típico de una aplicación IPTV y, a la vez, activar condiciones en las que suele manifestarse comportamiento malicioso (comunicación de red, ejecución en segundo plano, persistencia y acceso a almacenamiento).

Con este criterio, los guiones automatizados abarcaron once situaciones: actualización inicial, arranque, inicio de sesión, navegación por menús, reproducción de canal (streaming), uso en segundo plano, cierre forzado y reinicio, operaciones de lectura/escritura en almacenamiento, verificación de uso efectivo de cámara y micrófono, persistencia tras reinicio (autoarranque) e interceptación/evasión de TLS. Se priorizó un conjunto acotado y repetible para mantener la comparabilidad entre Android 8.1.0 y Android 15, y se excluyeron interacciones no soportadas por la aplicación o que no aportan señales adicionales a las variables medidas.

Fase 6. Captura de tráfico y eventos de sistema.

Durante las sesiones dinámicas se capturó tráfico de red en formato PCAP mediante tcpdump y se analizaron las trazas con Wireshark y, cuando fue necesario, con mitmproxy para inspeccionar tráfico HTTP(S). De forma paralela, se registraron eventos relevantes mediante logcat y, cuando el dispositivo lo permitió, se recogió información sobre system calls asociadas a la ejecución de la aplicación, siguiendo enfoques que combinan tráfico y llamadas al sistema como observables complementarios del comportamiento malicioso (Gohari et al., 2021; Manzil & Manohar Naik, 2023).

Fase 7. Transformación y organización de datos.

Los archivos PCAP se transformaron en vectores de flujo mediante herramientas de extracción de características de red (por ejemplo, CICFlowMeter), generando descriptores como número de paquetes, bytes transferidos, duración de las conexiones y puertos/protocolos utilizados. Las system calls y eventos registrados se agregaron en forma de frecuencias o secuencias resumidas por sesión para cada versión de Android. En paralelo, los resultados del análisis estático (permisos, APIs, evidencias de ofuscación) se estructuraron en tablas comparativas, manteniendo siempre la asociación con la única muestra analizada (MagisTV).

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Fase 8. Integración y comparación Android 8 vs 15.

Finalmente, se integraron los indicadores estáticos y dinámicos en un esquema de análisis comparativo, contrastando cómo la misma APK expresa sus permisos, APIs sensibles, patrones de tráfico y eventos del sistema en Android 8 frente a Android 15. Esta fase incluyó la elaboración de tablas y gráficos de diferencias y la interpretación de los hallazgos a la luz del estado del arte sobre análisis estático, dinámico e híbrido en malware Android, con el fin de evaluar la hipótesis y valorar el impacto de la versión del sistema en la observabilidad del comportamiento malicioso (Gohari et al., 2021; Muzaffar et al., 2022).

2.4 Variables y medidas

La variable independiente fue la versión del sistema operativo Android con dos niveles: Android 8 (Dispositivo A) y Android 15 (Dispositivo B), que representan distintos grados de endurecimiento de la plataforma y de sus mecanismos de seguridad (Muzaffar et al., 2022).

Las variables dependientes estáticas se definieron a partir de la ingeniería inversa de la APK MagisTV e incluyeron: Primero, un vector binario de permisos declarados en el AndroidManifest. xml; Segundo, el conteo y la categorización de llamadas a APIs sensibles, agrupadas en familias funcionales relevantes para la detección explicable de malware; y Tercero, la presencia de indicios de ofuscación y, cuando procedió, de componentes nativos (.so) con potencial asociación a vulnerabilidades conocidas (CVE/CVSS) (Chen et al., 2021; Palma et al., 2023; Sanna et al., s. f.).

Las variables dependientes dinámicas se derivaron de las trazas de ejecución y abarcaron características de flujo de red obtenidas a partir de archivos PCAP (número de flujos, volumen de tráfico, puertos y protocolos utilizados, duración de conexiones) y frecuencias de system calls y eventos relevantes en logcat, de acuerdo con enfoques que combinan tráfico y llamadas al sistema como observables complementarios del comportamiento malicioso (Gohari et al., 2021; Manzil & Manohar Naik, 2023). A partir de estos indicadores se calcularon diferencias absolutas y relativas entre Android 8 y Android 15, con el fin de contrastar la hipótesis (H1) sobre el impacto de la versión del sistema en la observabilidad y explicabilidad del riesgo asociado a la APK MagisTV (Muzaffar et al., 2022).

2.5 Técnicas e instrumentos de recolección

En el análisis estático, la recolección de datos se basó en la extracción automatizada de permisos, componentes y llamadas a APIs sensibles de la APK MagisTV utilizando MobSF, Apktool y Androguard. Los reportes generados por estas herramientas se exportan a formatos tabulares (por ejemplo, archivos CSV de permisos y APIs) que permitieron cuantificar y comparar de forma estructurada los indicadores estáticos entre Android 8 y Android 15 (Palma et al., 2023; Pathak et al., 2024).

Para el análisis dinámico, se generaron capturas de tráfico de red (PCAP) mediante tcpdump y Wireshark, complementadas con mitmproxy cuando fue necesario inspeccionar tráfico HTTP(S), mientras que ADB/logcat se empleó para registrar eventos del sistema durante sesiones de uso reproducibles inducidas con monkey/UIAutomator. Cuando el dispositivo lo permitió, se recogió información de system calls asociadas a la ejecución de la aplicación, siguiendo trabajos que combinan tráfico y llamadas al sistema como fuentes complementarias de observación del comportamiento malicioso (Gohari et al., 2021; Manzil & Manohar Naik, 2023; Muzaffar et al., 2022).

2.6 Procesamiento y representación de datos

Los archivos PCAP obtenidos en cada sesión se procesaron con herramientas de extracción de características de flujo (por ejemplo, CICFlowMeter) para generar vectores que resumen, por flujo y por sesión, métricas como número de paquetes, volumen de datos, duración de conexiones y puertos/protocolos utilizados (Gohari et al., 2021). Las system calls y eventos relevantes registrados mediante logcat se agregaron en forma de frecuencias por tipo de llamada y por escenario (Android 8 vs Android 15), constituyendo descriptores compactos del comportamiento dinámico de la aplicación (Manzil & Manohar Naik, 2023).

En paralelo, los resultados del análisis estático se representaron como vectores y tablas: un vector binario de permisos declarados en el AndroidManifest.xml, listas de APIs sensibles agrupadas por categoría funcional y marcadores cualitativos de ofuscación y presencia de código nativo. Esta representación homogénea de indicadores estáticos y dinámicos facilitó la elaboración de comparaciones directas entre versiones del sistema y la integración de ambas perspectivas en el análisis final (Gao et al., 2021; Muzaffar et al., 2022; Palma et al., 2023).

2.7 Modelado y estrategias de fusión

Dado que el objetivo principal del estudio no es proponer un nuevo clasificador generalista, sino analizar de forma explicable el comportamiento de una APK específica en dos versiones de Android, el “modelado” se planteó como una integración

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

analítica de tres vistas: Primero, una vista estática basada en permisos y APIs sensibles; Segundo, una vista dinámica basada en características de flujo de red y system calls; y Tercero, una vista estructural opcional, en la que se organizaron relaciones App-API/API-API como grafos conceptuales para apoyar la interpretación de fragmentos críticos de código (Gao et al., 2021; Palma et al., 2023).

En lugar de optimizar modelos de aprendizaje automático a gran escala, se exploraron esquemas sencillos de fusión temprana (comparación conjunta de indicadores estáticos y dinámicos) y fusión tardía (contraste de hallazgos procedentes de cada vista), tomando como referencia taxonomías recientes que muestran las ventajas de combinar fuentes estáticas, dinámicas y estructurales en la caracterización de malware Android (Muzaffar et al., 2022; Xu et al., 2023).

2.8 Evaluación, métricas y análisis comparativo

La evaluación se centró en comparar, de manera descriptiva y explicable, los indicadores estáticos y dinámicos obtenidos por la APK MagisTV en Android 8 y Android 15. Se elaboraron tablas y gráficos que recogen diferencias absolutas y relativas en el perfil de permisos y APIs sensibles, en los patrones de tráfico de red (número de flujos, volumen de datos, destinos y puertos) y en la intensidad/diversidad de system calls y eventos registrados por logcat entre ambos entornos (Gohari et al., 2021; Manzil & Manohar Naik, 2023).

Aunque no se planteó como objetivo principal la construcción de un modelo predictivo, en algunos casos se ensayaron reglas simples de decisión (por ejemplo, presencia simultánea de determinados permisos y patrones de tráfico) para ilustrar su capacidad de señalizar comportamiento malicioso. Para estas reglas, cuando fue pertinente, se calcularon métricas básicas (como tasas de acierto o de falsos positivos/negativos) a modo ilustrativo, tomando como referencia la forma en que la literatura evalúa sistemas de detección de malware basados en características estáticas y dinámicas (Muzaffar et al., 2022; Palma et al., 2023).

2.9 Reproducibilidad y consideraciones éticas

Para favorecer la reproducibilidad, se fijaron semillas aleatorias cuando procedió (por ejemplo, en la generación de estímulos con monkey), se versionaron los scripts utilizados para automatizar la instalación, ejecución y recolección de datos, y se mantuvo una estructura de carpetas estandarizada que agrupa, para cada escenario (Android 8 y Android 15), los reportes estáticos, matrices de características, archivos PCAP y registros de logcat/system calls. Además, se documentó el hash SHA-256 de la APK MagisTV y las versiones de las herramientas empleadas, en línea con las buenas prácticas descritas en estudios sobre generación de datasets y experimentos de malware Android (Muzaffar et al., 2022; Pathak et al., 2024).

Las pruebas se realizaron en un entorno doméstico controlado en Pasaje (Ecuador), utilizando dispositivos físicos dedicados y una red de pruebas mediante un punto de acceso configurado en Kali Linux. La APK se ejecutó exclusivamente con fines académicos y de investigación, evitando cualquier distribución o uso que vulnerara derechos de autor o políticas de plataformas oficiales. Se siguieron buenas prácticas éticas en el manejo de tráfico y registros, descartando cualquier dato que pudiera corresponder a información personal real y limitando el análisis a los flujos generados por la propia aplicación bajo condiciones controladas (Gohari et al., 2021).

3. Resultados y Discusión

3.1 Resultados del análisis estático guiado por ingeniería inversa

3.1.1 Evaluación global de seguridad con MobSF

El análisis estático automatizado de la APK de IPTV MagisTV (versión 6.4.2, paquete com.msandroid.mobile), ejecutado con MobSF, arrojó una puntuación de seguridad de 52/100, que sitúa a la aplicación en un nivel de riesgo medio. La tarjeta de puntuación de MobSF reportó 2 hallazgos de severidad alta, 14 de severidad media y 1 hotspot asociado a 9 permisos críticos, junto con 2 ítems clasificados como seguros (ausencia de rastreadores de privacidad y desactivación de configuración remota).

Los hallazgos de mayor severidad se relacionan con la posibilidad de instalar la aplicación en versiones de Android vulnerables o sin parches y con la habilitación explícita de tráfico en texto claro. Entre los hallazgos de nivel medio destacan advertencias sobre el esquema de firma, actividades con modos de lanzamiento potencialmente inseguros y posibles secretos embebidos en el código. En conjunto, estos resultados señalan que el riesgo se concentra en la configuración de la aplicación y en el uso de permisos sensibles, más que en técnicas avanzadas de evasión. Los principales hallazgos del análisis estático con MobSF se sintetizan en la Tabla 2, y el detalle de severidad de los hallazgos reportados por MobSF se ilustra en la Figura 6.

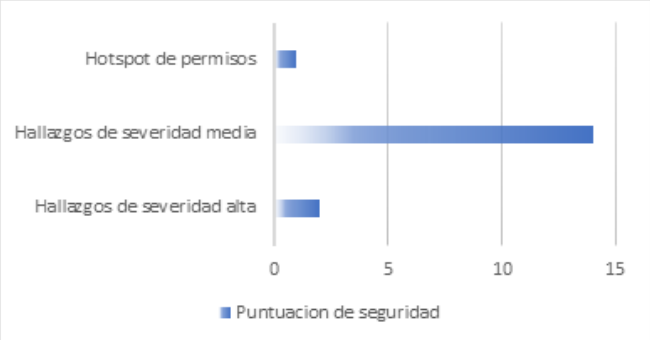

Figura 6. Puntuación de seguridad: Detalle de severidad en los hallazgos de MobSF.
Fuente: Los autores.

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Tabla 2. Resumen del análisis estático de MagisTV v6.4.2 (estático) con MobSF.
Fuente: Los autores.

Categoría	Indicador	Valor
Puntuación global de seguridad	Security score (0-100)	52
Evaluación de riesgo	Riesgo global	Medio
Severidad alta	Issues "High"	2
Severidad media	Issues "Medium"	14
Hotspot de permisos	Issues "Hotspot" asociados a permisos críticos	1
Permisos críticos	Permisos marcados como críticos	9
Elementos seguros	Ítems clasificados como "Secure"	2
Trackers de privacidad	Trackers detectados	0

Desde la perspectiva de la discusión, este gráfico inicial muestra que MagisTV dispone de capacidades técnicas suficientes para impactar en la confidencialidad y en la superficie de exposición del dispositivo. La combinación de permisos críticos y configuraciones laxas justifica profundizar en el detalle del manifiesto, de la estructura interna del código y, más adelante, del comportamiento dinámico.

3.1.2 Perfil de permisos y potencial de abuso

Las distintas herramientas de análisis confirmaron la declaración de 23 permisos en el archivo AndroidManifest.xml de MagisTV. De ellos, 13 corresponden a permisos de nivel normal, 9 a permisos de nivel peligroso y 1 a un permiso de tipo propietario o no estándar. Este reparto se traduce en un predominio de permisos considerados rutinarios por la plataforma, acompañado de un bloque significativo de permisos de alto impacto.

Entre los permisos solicitados destacan los orientados a conectividaddered(INTERNET,ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, CHANGE_NETWORK_STATE, CHANGE_WIFI_STATE, CHANGE_WIFI_MULTICAST_STATE), almacenamiento (READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, MANAGE_EXTERNAL_STORAGE, lectura de imágenes y audio), captura de contenido (CAMERA), notificaciones y estado del dispositivo (POST_NOTIFICATIONS, WAKE_LOCK, VIBRATE, GET_TASKS) e identificación/atribución (AD_ID, BIND_GET_INSTALL_REFERRER_SERVICE). La sección de permisos abusados de MobSF indica, además, coincidencia con un subconjunto relevante de permisos catalogados como comunes en aplicaciones maliciosas. El conteo y la clasificación por nivel de los permisos solicitados se presentan en la Tabla 3.

Tabla 3. Conteo y clasificación de permisos por nivel y categoría funcional en MagisTV.
Fuente: Los autores.

Indicador	Conteo (n)
Permisos totales declarados	23
Permisos de nivel normal	13
Permisos de nivel peligroso	9
Permisos de tipo propietario/no estándar	1
Permisos de red	6
Permisos de almacenamiento	5
Permisos de cámara y multimedia	1
Permisos de notificación y estado del dispositivo	9
Permisos de identificación y atribución	2

Nota: Conteos obtenidos del AndroidManifest.xml de MagisTV v6.4.2 (estático) mediante MobSF; las categorías funcionales corresponden a una agrupación analítica de los permisos declarados.

La combinación de permisos de red, almacenamiento, cámara, notificaciones e instalación configura un escenario donde la aplicación podría, si el código así lo implementa, acceder de forma intensiva a datos locales, generar tráfico persistente hacia servidores externos y participar en procesos de instalación o seguimiento publicitario. Por sí solo, este perfil no prueba un comportamiento malicioso, pero sí delimita un espacio de riesgo que será contrastado con el análisis dinámico.

3.1.3 Estructura de componentes y bibliotecas nativas (Apktool)

El análisis del AndroidManifest.xml mediante Apktool reveló una arquitectura interna extensa. La aplicación declara 61 actividades, 27 servicios, 18 receptores de difusión y 4 proveedores de contenido, lo que refleja una aplicación con numerosas pantallas, flujos de navegación y tareas auxiliares. La actividad principal se marca como exportada y actúa como punto de entrada, incluyendo un deep link asociado a un esquema y dominio específicos. Otros componentes exportados se relacionan con la recepción de notificaciones o interacciones con servicios en la nube.

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

La mayoría de servicios y receptores figuran como no exportados, pero una parte importante pertenece a SDKs de terceros orientados a mensajería, analítica, publicidad y transmisión de contenidos. En el nivel de aplicación, el manifiesto configura android:allowBackup=”false” y, al mismo tiempo, android:usesCleartextTraffic=”true”, lo que protege parcialmente contra copias de seguridad no autorizadas, pero mantiene habilitado el uso de tráfico sin cifrado fuerte, ya señalado en la evaluación de riesgo de MobSF.

En el directorio lib/ se identificaron 21 bibliotecas nativas (.so) para armeabi-v7a y otras 21 para arm64-v8a. Entre ellas figuran librerías de reproducción multimedia, gestión de fallos, comunicaciones de red y componentes asociados a SDKs de analítica. El hecho de que gran parte de la funcionalidad se delegue en código nativo y módulos externos complica el análisis y abre la puerta a vulnerabilidades o comportamientos que no son directamente visibles en el código Java. La estructura de componentes y la identificación de librerías nativas se presentan en la Tabla 4.

Tabla 4. Resumen de componentes de aplicación y librerías nativas mediante análisis estático.
Fuente: Los autores.

Tipo de elemento	Descripción general	Cantidad
Actividades	Pantallas y flujos de interfaz	61
Servicios	Procesos en segundo plano y tareas auxiliares	27
Receptores (broadcast receivers)	Recepción de eventos y notificaciones del sistema	18
Proveedores de contenido	Exposición estructurada de datos internos	4
Bibliotecas nativas .so (armeabi-v7a)	Librerías para arquitectura de 32 bits	21
Bibliotecas nativas .so (arm64-v8a)	Librerías para arquitectura de 64 bits	21

Desde el punto de vista interpretativo, esta estructura sugiere una aplicación fuertemente modularizada, donde buena parte del comportamiento depende de servicios residentes y de componentes nativos o de terceros. Esta complejidad incrementa la superficie de ataque y refuerza la necesidad de estudiar qué módulos se activan efectivamente en cada versión de Android durante el análisis dinámico.

3.1.4 Análisis puntual de APIs y patrones sensibles con JADX

La APK de MagisTV se cargó en la interfaz gráfica de JADX para inspeccionar el código Java descompilado y buscar de forma dirigida ciertas APIs asociadas a riesgo elevado. El objetivo era identificar indicios de recolección de identificadores del dispositivo, ejecución de comandos del sistema o uso de canales de comunicación de bajo nivel.

En primer lugar, se buscó el patrón getDeviceId. La herramienta localizó tres coincidencias en clases de la librería androidx.appcompat.app, donde la llamada aparece como keyEvent.getDeviceId() integrada en la lógica de tratamiento de eventos de teclado. En este contexto, el método se utiliza para obtener el identificador del dispositivo de entrada, no del módulo de telefonía, por lo que se interpreta como funcionalidad de compatibilidad de interfaz y no como recolección deliberada de identificadores persistentes del terminal. La evidencia de la API getDeviceId identificada en el análisis con JADX se muestra en la Figura 7.

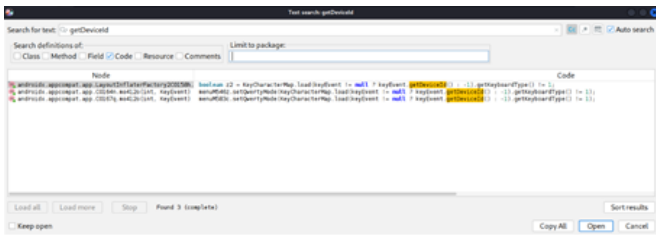

Figura 7. Evidencia de la API getDeviceId en JADX: Búsqueda y descarte de uso malicioso de identificadores.
Fuente: Los autores.

A continuación, se realizaron búsquedas de otros métodos y cadenas: getSubscriberId, getSimSerialNumber, Runtime.exec, ProcessBuilder, HttpURLConnection, Socket, okhttp, REQUEST_INSTALL_PACKAGES y pm.installPackage. En todos los casos, JADX reportó cero coincidencias en el código Java descompilado. Esto sugiere que la aplicación no invoca directamente, al menos en esta capa, APIs típicamente asociadas a obtención de identificadores de línea, ejecución de comandos del sistema, uso explícito de sockets de bajo nivel o gestión directa de instalaciones desde el código de alto nivel. Los resultados de la búsqueda dirigida de APIs y patrones sensibles se presentan en la Tabla 5.

Estos resultados matizan el perfil de riesgo: aunque el manifiesto otorga a la aplicación amplias capacidades, el código Java visible no revela llamadas evidentes a algunas APIs consideradas especialmente intrusivas. Esto no excluye que funciones sensibles se ejecuten en bibliotecas nativas o SDKs externos, pero sí indica que, a este nivel, los patrones clásicos de spyware o ejecución de comandos no son predominantes.

3.1.5 Análisis con Androguard y densidad de clases

Androguard se empleó para corroborar, sobre el bytecode Dalvik, los hallazgos anteriores y obtener métricas adicionales sobre la estructura interna. Tras ejecutar AnalyzeAPK(), la herramienta validó el paquete, procesó el manifiesto, añadió el fichero DEX y construyó referencias cruzadas. Las funciones de listado de permisos, actividades y servicios devolvieron resultados coherentes con los obtenidos por MobSF, Apktool y JADX, lo que refuerza la consistencia del recuento de componentes y del perfil de permisos.

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

Tabla 5. Resultados de búsqueda dirigida de APIs/patrones sensibles en JADX.
Fuente: Los autores.

API/patrón buscado	Resultado en JADX	Interpretación rápida
getDeviceId	3 coincidencias en clases androidx.appcompat.*	Asociado a dispositivos de entrada, no a identificadores de línea
getSubscriberId	0 coincidencias	No se observa obtención directa del IMSI
getSimSerialNumber	0 coincidencias	No se observa lectura del número de serie de la SIM
Runtime.exec	0 coincidencias	No se detecta ejecución directa de comandos del sistema
ProcessBuilder	0 coincidencias	No se observa construcción de procesos externos
HttpURLConnection	0 coincidencias	No se identifican usos directos de esta API
Socket	0 coincidencias	No se observan sockets de bajo nivel en el código Java
Okhttp	0 coincidencias	No se detectan referencias textuales a esta librería HTTP
REQUEST_INSTALL_PACKAGES	0 coincidencias	El permiso se declara en el Manifest, pero no aparece en el código
Pm.installPackage	0 coincidencias	No se observa invocación directa al gestor de paquetes

Se programaron consultas sobre el objeto de análisis para localizar APIs como getDeviceId en la clase de telefonía, Runtime.exec o getRunningTasks. Al igual que en el análisis con JADX, estas búsquedas no devolvieron referencias directas, lo que sugiere que, en el bytecode analizado, dichas rutinas no se utilizan de forma explícita. Por otra parte, un script que recorre el conjunto de clases informó de aproximadamente 2147 clases presentes en la APK, incluyendo tanto clases de soporte de bibliotecas estándar como numerosas clases con nombres muy breves u opacos, compatibles con patrones de ofuscación o empaquetado. La búsqueda dirigida de APIs críticas en el bytecode mediante Androguard se muestra en la Figura 8, y el escaneo de clases y los indicios compatibles con ofuscación se presentan en la Figura 9.

Desde el punto de vista interpretativo, la alta densidad de clases y la presencia de nombres poco descriptivos confirman que MagisTV es una aplicación compleja y parcialmente ofuscada, donde la lógica relevante para la seguridad se reparte entre múltiples módulos. Esto dificulta el análisis estático y refuerza la conveniencia de observar el comportamiento efectivo en ejecución para caracterizar la aplicación de forma más precisa.

Figura 8. Búsqueda dirigida de APIs críticas en el bytecode utilizando la herramienta de Androguard.
Fuente: Los autores.

Figura 9. Resultado del escaneo de clases: conteo total y presencia de nombres opacos compatibles con ofuscación.
Fuente: Los autores.

3.2 Resultados del análisis dinámico en Android 8 y Android 15

3.2.1 Actualización forzada de la aplicación

Al intentar ejecutar la versión 6.4.2 de MagisTV, la aplicación forzó inmediatamente un flujo de actualización hacia la versión 6.5.2, sustituyendo el binario instalado y bloqueando la ejecución de versiones anteriores. Los registros del sistema muestran la descarga de la nueva APK y el evento de actualización del paquete en primer plano, tras lo cual la versión previa deja de estar disponible para el usuario. Este comportamiento condiciona el entorno experimental, ya que obliga a considerar la versión 6.5.2 como referencia efectiva para las pruebas dinámicas, y evidencia un mecanismo de control del proveedor que limita la posibilidad

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

de mantener versiones antiguas para el análisis comparativo. El comportamiento observado en la actualización forzada se resume en la Tabla 6, y el mecanismo de actualización forzada observado en ambos entornos se ilustra en la Figura 10.

Tabla 6. Resumen del comportamiento de actualización forzada. Fuente: Los autores.

Aspecto observado	Resultado
Versión inicial	6.4.2
Versión tras actualización	6.5.2
Tipo de actualización	Forzada, en primer plano
Posibilidad de ejecutar 6.4.2	No, el binario queda sustituido
Impacto en el experimento	Las pruebas dinámicas se realizan sobre 6.5.2

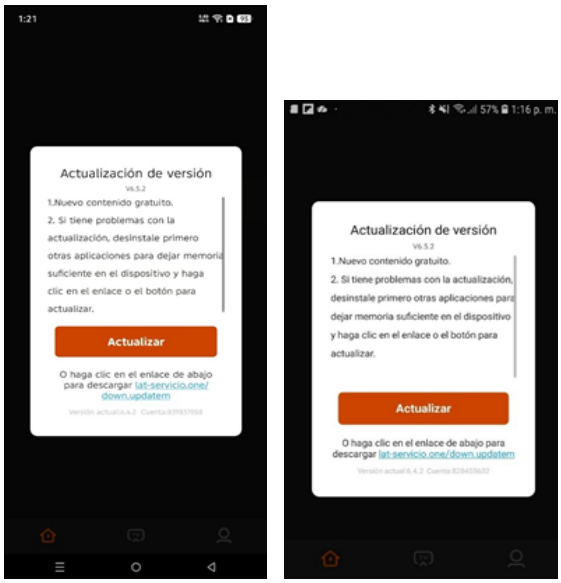

Figura 10. Mecanismo de actualización forzada de la APK que define la versión efectiva para las pruebas dinámicas. (a) Android 15; (b) Android 8.
Fuente: Los autores.

Durante el experimento, la actualización obligatoria se produjo en 12 de noviembre de 2025 a las 15:00; a partir de ese momento, todas las ejecuciones y observaciones dinámicas se realizaron sobre MagisTV v6.5.2 (dinámico), al ser la versión efectiva en ejecución tras el reemplazo del binario instalado.

No fue posible repetir el análisis estático sobre MagisTV v6.5.2 (dinámico), debido a que dicha versión no se encontraba disponible como archivo APK independiente en el momento del experimento y fue obtenida únicamente mediante actualización in-app; por tanto, no se dispone del APK v6.5.2 para calcular su SHA-256 ni para ejecutar nuevamente el pipeline estático. Como evidencia complementaria, se registró que, tras la actualización, la aplicación instalada alcanzó un tamaño aproximado de 170 MB. En consecuencia, la correlación estático-dinámico se interpreta considerando que la evidencia estática reportada corresponde a MagisTV v6.4.2 (estático), mientras que la evidencia dinámica corresponde a MagisTV v6.5.2 (dinámico), lo que introduce una discontinuidad de trazabilidad por versión que se discute como una limitación metodológica.

3.2.2 Prueba de arranque

En la prueba de arranque se midieron los tiempos hasta la aparición de la pantalla de bienvenida y de la actividad principal. En Android 8, el sistema registra la creación del proceso MagisTV y la actividad de splash, seguida del renderizado de la actividad principal en un intervalo reducido. En Android 15 se observa un patrón similar, con tiempos de dibujo de la ventana inicial en el orden de cientos de milisegundos y una transición rápida hacia la pantalla principal. En ambos dispositivos el arranque es fluido y sin errores críticos, aunque en Android 15 se registran eventos adicionales de validación de conectividad y de gestión de rendimiento que no aparecen en Android 8. Los resultados de la prueba de arranque se presentan en la Tabla 7.

Tabla 7. Resultados de la prueba de arranque. Fuente: Los autores.

Versión Android	Tiempo hasta splash	Tiempo splash-pantalla principal	Errores críticos
Android 8	2,0 s	3,2 s	No observados
Android 15	0,4 s	3,1 s	No observados

3.2.3 Prueba de inicio de sesión

En la prueba de inicio de sesión se ejercitó el flujo desde la pantalla de cuenta hasta la actividad de autenticación. El Android 8, logcat refleja la creación de la actividad LoginAty y la colocación de MagisTV en primer plano, acompañadas de tráfico de red asociado a la validación de credenciales. En Android 15 la transición entre AccountAty y LoginAty se registra con mayor detalle, incluyendo tiempos de muestra de la nueva ventana y mensajes de gestión de interfaz. En ninguno de los dispositivos se observan errores de autenticación ni cierres inesperados durante la prueba. Los resultados de las pruebas de inicio de sesión se presentan en la Tabla 8.

Tabla 8. Resultados de las pruebas de inicio de sesión. Fuente: Los autores.

Indicador	Android 8	Android 15
Actividades implicadas	LoginAty	AccountAty, LoginAty
Tiempo de cambio de actividad	0.015 s	0.145 s
Errores visibles para el usuario	No observados	No observados
Tráfico de red asociado	Bajada: 1.433 MB; Subida: 0.064 MB (PCAP)	Bajada: 0.127 MB; Subida: 0.078 MB (PCAP)

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

3.2.4 Prueba de navegación por menús

La prueba de navegación por menús simuló un uso típico: desplazamiento por categorías, selección de elementos y retorno a la pantalla principal. En Android 8 se observa el paso de MainAty a PlayAty, con ocultamiento de la actividad principal y creación de superficies de renderizado específicas para la reproducción, así como incrementos sostenidos en los contadores de tráfico Wi-Fi para MagisTV. En Android 15, además de la secuencia MainAty a PlayAty, el sistema registra cambios frecuentes en el pipeline gráfico y en el estado del reproductor multimedia, coherentes con la carga de listas, imágenes y vistas previas. Los resultados de la prueba de navegación por menús se presentan en la Tabla 9.

Tabla 9. Resultados de la prueba de navegación por menús. Fuente: Los autores.

Aspecto	Android 8	Android 15
Actividades principales	MainAty, PlayAty	MainAty, PlayAty, vistas internas adicionales
Patrón de tráfico	Bajada: 225.211 MB; Subida 2.224 MB (PCAP)	Bajada: 281.597 MB; Subida: 5.087 MB (PCAP)
Estabilidad de la interfaz	0 cierres inesperados	0 cierres inesperados; mayor instrumentación gráfica

3.2.5 Prueba de reproducción de canal (streaming)

En el escenario se mantuvo un canal de IPTV activo durante 8 minutos en cada plataforma. En Android 8, los contadores de tráfico por aplicación y la captura de red evidenciaron 125.43 MB de tráfico de bajada y 1.29 MB de subida, sin errores de reproducción. En Android 15, aunque el sistema no expone contadores por aplicación en el log, la captura de red registró 124.15 MB de bajada y 2.10 MB de subida, igualmente sin interrupciones. En ambos casos, el tráfico se concentró predominantemente en flujos descendentes y el servicio se mantuvo estable, sin mensajes de error visibles, lo cual sugiere un comportamiento compatible con un cliente de IPTV convencional bajo carga sostenida. Los resultados de la prueba de reproducción de canal se presentan en la Tabla 10.

Tabla 10. Resultados de la prueba de reproducción de canal. Fuente: Los autores.

Versión Android	Duración (min)	Tráfico bajado	Tráfico de subida	Errores de reproducción
Android 8	8 minutos	125.43 MB (PCAP)	1.29 MB (PCAP)	No observados
Android 15	8 minutos	124.15 MB (PCAP)	2.10 MB (PCAP)	No observados

3.2.6 Pruebas de uso en segundo plano

En la prueba de uso en segundo plano se dejó MagisTV abierta, pero sin interacción, mientras se observaba su comportamiento. En Android 8, los contadores del sistema muestran que la aplicación sigue generando tráfico de red en background, con incrementos continuos de bytes enviados y recibidos durante el intervalo de observación, lo que indica mantenimiento de flujos activos pese a no estar en primer plano. En Android 15, aunque los contadores por aplicación no son visibles en el log, la captura de red evidencia que todavía se producen conexiones en ventanas breves, mientras el sistema alterna estados de congelación y descongelación del proceso para aplicar políticas de ahorro de energía. Los resultados de la prueba de uso en segundo plano se presentan en la Tabla 11.

Tabla 11. Resultados de las pruebas de uso en segundo plano. Fuente: Los autores.

Indicador	Android 8	Android 15
Tráfico de red en background	Bajada: 35.774 MB; Subida: 0.366 MB (PCAP)	Bajada: 39.480 MB; Subida: 0.612 MB (PCAP)
Gestión del proceso	Congelación/ descongelación: 0/0	Congelación/ descongelación: 2/1
Servicios de la app (NetService)	Detención por inactividad: 1	Detención por inactividad: 1

3.2.7 Pruebas de cierre forzado y reinicio

Durante el cierre forzado, la aplicación se encontraba reproduciendo contenido en primer plano. En ambos dispositivos, el cierre interrumpe de inmediato la reproducción y termina el proceso asociado a MagisTV. En Android 15, los registros incluyen anotaciones explícitas de cierre forzado y liberación de recursos, mientras que en Android 8 se limita a la finalización del proceso y las actividades. Tras el cierre, la app puede iniciarse de nuevo desde el lanzador sin errores, recreando la secuencia de splash y pantalla principal. Este escenario delimita claramente los intervalos de ejecución a considerar en el análisis de tráfico y eventos. Los resultados de la prueba de cierre forzado y reinicio se presentan en la Tabla 12.

3.2.8 Pruebas de almacenamiento (lectura/escritura)

En la prueba de almacenamiento se observó la actividad de MagisTV sobre el sistema de ficheros durante 4 minutos. En Android 8, el kernel genera un número elevado de eventos de auditoría de seguridad vinculados a accesos a ficheros, lo que sugiere intentos repetidos de lectura o escritura en rutas sujetas a políticas de SELinux. En Android 15 no se registran ese tipo

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

Tabla 12. Resultados de las pruebas de cierre forzado y reinicio.
Fuente: Los autores.

Aspecto	Android 8	Android 15
Efecto de cierre forzado	Fin de proceso y actividades	Fin de proceso + anotaciones de cierre
Capacidad de reinicio	Se reinicia sin errores	Se reinicia sin errores
Impacto en tráfico de red (observación en PCAP)	Corte inmediato de flujos	Corte inmediato de flujos

de eventos, pero sí llamadas frecuentes a utilidades de sistema para consultar el estado del almacenamiento y su capacidad, lo que indica un uso más ajustado a las APIs de alto nivel. Los resultados de las pruebas de almacenamiento se presentan en la Tabla 13.

Tabla 13. Resultados de las pruebas de almacenamiento
Fuente: Los autores.

Indicador	Android 8	Android 15
Duración de la prueba	4.76 minutos	4.02 minutos
Eventos de seguridad por accesos a ficheros	1069 eventos	0 eventos
Consultas al estado de almacenamiento	21 registros	357 registros

3.2.9 Prueba de uso de cámara y micrófono

En este escenario se activaron aquellas rutas de la interfaz que, en principio, podrían conducir al uso de cámara o micrófono (búsqueda, navegación, reproducción, menús, auxiliares). Tanto en Android 8 como en Android 15, los registros se centran en eventos de interfaz, gestión de vistas y decodificación de video, sin trazas explícitas del inicio de servicios de cámara del sistema ni de componentes de captura de audio ligados a grabación. Tampoco se observaron diálogos de solicitud de permiso en tiempo de ejecución asociados a estos recursos. Los resultados de la prueba de uso de cámara y micrófono se presentan en la Tabla 14.

Tabla 14. Resultados de las pruebas de uso de cámara y micrófono.
Fuente: Los autores.

Recurso	Android 8	Android 15
Inicialización cámara	No observada	No observada
Inicialización micrófono	No observada	No observada
Diálogos de permiso	No observados	No observados

3.2.10 Prueba de persistencia tras reinicio (autoarranque)

Para evaluar la persistencia, se reiniciaron completamente ambos dispositivos con MagisTV instalada y sesión reiniciada. Durante el arranque no se observaron procesos ni actividades de MagisTV iniciándose de manera automática: la aplicación aparece listada como app de terceros, pero no se registran ejecuciones de servicios, receptores o actividades propios. MagisTV solo vuelve a ejecutarse cuando el usuario la abre manualmente después del boot. Los resultados de la prueba de persistencia tras reinicio se presentan en la Tabla 15.

Tabla 15. Resultados de la prueba de persistencia tras reinicio.
Fuente: Los autores.

Aspecto	Android 8	Android 15
Autoarranque tras reinicio	No	No
Procesos de la app durante el boot	No observados	No observados
Necesidad de acción del usuario	Sí, apertura manual	Sí, apertura manual

3.2.11 Prueba de interceptación y evasión TLS

Finalmente, se evaluó la capacidad de inspeccionar el tráfico TLS de MagisTV mediante un proxy interceptador. Con mitmproxy configurado como proxy HTTP(S) y su certificado raíz instalado en los dispositivos, la aplicación y sus componentes de red intentan establecer conexiones TLS hacia servicios externos, por ejemplo, dominios de Google, pero el handshake falla sistemáticamente por rechazo del certificado presentado por el proxy. El resultado es que el tráfico capturado corresponde a intentos de conexión TLS abortados, sin exposición del contenido de las peticiones en texto claro. Los resultados de la prueba de interceptación y evasión TLS se presentan en la Tabla 16, y la evidencia de evasión TLS observada durante la interceptación se muestra en la Figura 11.

Tabla 16. Resultados de la prueba de interceptación y evasión TLS.
Fuente: Los autores.

Dispositivo	Resultado de la interceptación	Comportamiento observado
Android 8	Fallida	Handshakes TLS abortados, sin datos en claro
Android 15	Fallida	Patrón análogo, sin diferencias relevantes

Este escenario muestra que, al menos en las versiones analizadas, el tráfico de MagisTV se protege frente a la inspección mediante proxy TLS genérico, por lo que el análisis dinámico de red debe apoyarse principalmente en metadatos de conexión (destinos, puertos, volúmenes) y no en el contenido descifrado de las comunicaciones.

3.3 Captura de eventos del sistema y tareas en segundo plano

Para completar las pruebas dinámicas, se capturaron estadísticas del programador de tareas y del gestor de paquetes del sistema, filtradas para el paquete com.msandroid.mobile. En el volcado

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

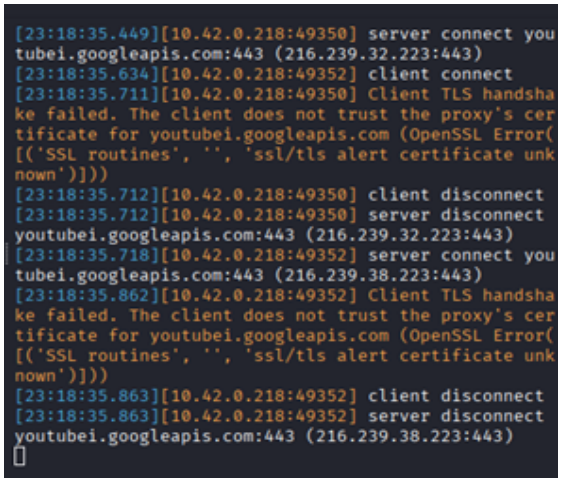

Figura 11. Evidencia de evasión TLS: falla en el handshake debido a la desconfianza en el certificado del proxy.
Fuente: Los autores.

de trabajos programados se observan ejecuciones breves de servicios asociados a plataformas de mensajería de terceros, que se inician y detienen en distintos momentos, pero el resumen indica cero trabajos en segundo plano en ejecución en el instante de la captura.

El análisis de las tablas de resolución de actividades, receptores y servicios revela que MagisTV se integra de forma intensa con el ecosistema de eventos del sistema. Registra receptores para cambios de batería, conexión y desconexión de alimentación, estado de almacenamiento, conectividad de red, presencia del usuario y señales de arranque y cambio de hora, así como servicios dedicados a notificaciones push y a la proyección de contenido a otros dispositivos. Esta red de receptores implica que, aunque la aplicación no se autoarranque de forma visible tras el boot, dispone de múltiples puntos de enganche para activarse en respuesta a eventos del entorno. El resumen de receptores de eventos del sistema y servicios asociados se presenta en la Tabla 17.

El gestor de paquetes confirma que en ambas plataformas MagisTV declara un conjunto amplio de permisos, pero muestran diferencias en cómo el sistema aplica dichos permisos en tiempo de ejecución: Android 8 permite un mayor margen de acceso a almacenamiento externo, mientras que Android 15 refuerza restricciones y canaliza gran parte de la actividad a través de APIs de alto nivel. Esta asimetría ayuda a explicar por qué en pruebas anteriores Android 8 genera un volumen elevado de auditorías de acceso a ficheros, mientras que en Android 15 la actividad de la aplicación se refleja principalmente como consultas de estado y eventos controlados por la propia plataforma.

3.4 Limitaciones.

Este estudio se basa en un diseño de caso único centrado en una aplicación IPTV MagisTV, por lo que la generalización de los resultados debe interpretarse con cautela. Aunque el enfoque híbrido se aplicó de forma controlada en dos entornos (Android 8 y Android 15), los patrones observados pueden variar en otras familias de aplicaciones o muestras con técnicas de evasión diferentes.

Una limitación metodológica relevante deriva de la actualización obligatoria detectada durante la ejecución dinámica: el análisis estático se efectuó sobre el APK obtenido MagisTV v6.4.2 (estático), mientras que la observación dinámica se realizó sobre la versión efectiva en ejecución tras la actualización, MagisTV v6.5.2 (dinámico), el 12-11-25 a las 15:00 pm. Dado que la versión v6.5.2 se obtuvo únicamente mediante actualización in-app y no se dispuso del APK como artefacto independiente, no fue posible calcular su huella criptográfica SHA-256 ni replicar el análisis estático sobre dicha versión, lo que introduce una discontinuidad de trazabilidad por versión entre evidencia estática y dinámica.

Finalmente, el análisis dinámico se encuentra condicionado por restricciones típicas de instrumentación en entornos móviles, como la visibilidad parcial de tráfico cifrado y el alcance del estímulo automatizado sobre funciones efectivamente expuestas por la aplicación. En consecuencia, algunos comportamientos pueden no manifestarse en ausencia de disparadores específicos, por lo que se prioriza la interpretación conjunta de evidencias bajo los escenarios controlados definidos.

Tabla 17. Resumen de receptores de eventos del sistema y servicios asociados.
Fuente: Los autores.

Elemento del sistema	Android 8	Android 15	Comentario
Receptores de batería	Sí	Sí	Activación condicionada al estado de carga/batería
Receptores de almacenamiento	Sí	Sí	Monitorean umbrales de espacio disponible
Receptores de conectividad/red	Sí	Sí	Permiten reaccionar a cambios de red y presencia de usuario
Receptores de arranque/tiempo	Sí	Sí	Usados para programar trabajos y diagnósticos
Servicios de mensajería push	Sí	Sí	Canal potencial para notificaciones y activación en segundo plano
Servicios de proyección/red local	Sí	Sí	Soporte para casting y reproducción remota

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

4. Conclusiones

El estudio evidenció que un enfoque híbrido guiado por ingeniería inversa, que combina análisis estático del Manifest, de las llamadas a API y de la estructura interna de la APK con observaciones dinámicas de tráfico de red, registros del sistema y comportamiento en ejecución, permite caracterizar de manera entendible y explicable el perfil de riesgo de una aplicación de Android que si se recurriera únicamente a una de estas perspectivas.

La aplicación MagisTV en el análisis estático mostró una superficie de permisos amplia, una arquitectura compleja con numerosos componentes y un uso intensivo de bibliotecas nativas, lo que, en abstracto, habilita escenarios de riesgo elevados; sin embargo, las pruebas dinámicas en escenarios de uso representativos (arranque, autenticación, navegación, reproducción, segundo plano, almacenamiento, persistencia y eventos del sistema) mostraron un comportamiento consistente con el de un cliente de IPTV: no se observan usos efectivos de cámara o micrófono, no se detectó autoarranque tras el reinicio y el tráfico permaneció encapsulado en canales TLS resistentes a la inspección mediante proxy genérico. Esta combinación de hallazgos confirma la utilidad del enfoque híbrido para depurar percepciones alarmistas basadas solo en permisos y, al mismo tiempo, para documentar de manera trazable la superficie de exposición real.

La comparación entre Android 8 y Android 15 con la misma APK puso de manifiesto que la versión del sistema operativo condiciona tanto la forma en que se activan los mecanismos de seguridad como los observables disponibles para la detección; mientras Android 8 expone de manera más directa la actividad de la aplicación, con tráfico continuo en segundo plano y un volumen elevado de auditorías por acceso a ficheros; Android 15 aplica políticas más estrictas sobre almacenamiento y procesos, canaliza las operaciones a través de APIs de alto nivel y modula la ejecución mediante congelación y descongelación de aplicaciones; estas diferencias sostienen la hipótesis planteada: integrar indicadores estáticos y dinámicos en un marco comparativo mejora la capacidad de explicar el comportamiento y la superficie de riesgo frente al análisis estático aislado, y sugiere que los sistemas de detección deberían incorporar explícitamente la versión de Android como variable de contexto.

En términos prácticos, y desde la perspectiva del usuario final, los resultados de este estudio no permiten clasificar a MagisTV como malware en los escenarios ensayados, pero sí la sitúan como una aplicación de riesgo elevado: combina un conjunto amplio de permisos, una fuerte dependencia de servicios externos y un incierto tratamiento interno de datos. En consecuencia, no se considera recomendable su instalación en dispositivos que gestionen información sensible ni en entornos institucionales, en todo caso, su uso debería restringirse a dispositivos personales o de laboratorio, aislados y con controles de seguridad adicionales, por parte de usuarios que sean plenamente conscientes de las implicaciones de privacidad y exposición asociadas.

Como trabajo futuro, se plantea extender la metodología a un conjunto representativo de aplicaciones benignas y maliciosas, con el fin de evaluar su generalización y robustez frente a familias y técnicas de evasión diversas. Para analizar su escalabilidad a grandes volúmenes, se propone automatizar la extracción de características estáticas (Manifest, componentes, llamadas a API y bibliotecas nativas), la ejecución reproducible de escenarios dinámicos mediante orquestación (ADB, UIAutomator/Monkey) y la consolidación de evidencias (logcat y flujos de red derivados de PCAP) en reportes trazables por versión. No obstante, el componente dinámico mantiene un costo temporal e infraestructural significativo; por ello, resulta pertinente explorar esquemas de pre filtrado estático y priorización de casos para reducir el número de ejecuciones instrumentadas.

Contribución de los autores

Contreras Espinoza Anthony Alexander: Conceptualización, Investigación, Metodología. **Honores Tapia Joofre Antonio**: Curación de datos, Análisis formal, Administración del proyecto, Supervisión, Validación. **Valarezo Pardo Milton Rafael**: Administración del proyecto, Supervisión, Validación. **Contreras Alonso Tania Yesminia**: Administración del proyecto, Supervisión, Validación.

Conflictos de interés

Los autores declaran no tener ningún conflicto de interés.

Referencias bibliográficas

Aamir, M., Iqbal, M. W., Nosheen, M., Ashraf, M. U., Shaf, A., Almarhabi, K. A., Alghamdi, A. M., & Bahaddad, A. A. (2024). AMDDLmodel: Android smartphones malware detection using deep learning model. *PLOS ONE, 19(1)*, e0296722. https://doi.org/10.1371/journal.pone.0296722

Almomani, I., Ahmed, M., & El-Shafai, W. (2022). Android malware analysis in a nutshell. *PLOS ONE, 17(7)*, e0270647. https://doi.org/10.1371/journal.pone.0270647

Chen, Y.-C., Chen, H.-Y., Takahashi, T., Sun, B., & Lin, T.-N. (2021). Impact of Code Deobfuscation and Feature Interaction in Android Malware Detection. *IEEE Access, 9,* 123208-123219. https://doi.org/10.1109/ACCESS.2021.3110408

Gao, H., Cheng, S., & Zhang, W. (2021). GDroid: Android malware detection and classification with graph convolutional network. *Computers & Security, 106,* 102264. https://doi.org/10.1016/j.cose.2021.102264

Gohari, M., Hashemi, S., & Abdi, L. (2021). Android Malware Detection and Classification Based on Network Traffic Using Deep Learning. 2021 7th International Conference

e-ISNN 2550-6730 | Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026 | DOI: 10.33936/isrtic.v10i1.8098

INFORMÁTICA Y SISTEMAS
Revista de Tecnologías de la Informática y las Comunicaciones

Facultad de Ciencias Informáticas
Universidad Técnica de Manabí
Av. Urbina y Che Guevara, Portoviejo, Ecuador
revista.iys@utm.edu.ec

on Web Research (ICWR), 71-77. https://doi.org/10.1109/ICWR51868.2021.9443025

Gu, J., Zhu, H., Han, Z., Li, X., & Zhao, J. (2024). GSEDroid: GNN-based Android malware detection framework using lightweight semantic embedding. *Computers & Security, 140*, 103807. https://doi.org/10.1016/j.cose.2024.103807

Manzil, H. H. R., & Manohar Naik, S. (2023). Android malware category detection using a novel feature vector-based machine learning model. *Cybersecurity, 6(1)*, 6. https://doi.org/10.1186/s42400-023-00139-y

Muzaffar, A., Ragab Hassen, H., Lones, M. A., & Zantout, H. (2022). An in-depth review of machine learning based Android malware detection. *Computers & Security, 121,* 102833. https://doi.org/10.1016/j.cose.2022.102833

Palma, C., Ferreira, A., & Figueiredo, M. (2023). Explainable Machine Learning for Malware Detection on Android *Applications*. *Information, 15(1)*. https://doi.org/10.3390/info15010025

Pathak, A., Kumar, Th. S., & Barman, U. (2024). Static analysis framework for permission-based dataset generation and android malware detection using machine learning. *EURASIP Journal on Information Security, 2024(1)*, 33. https://doi.org/10.1186/s13635-024-00182-3

Sanna, S. L., Soi, D., Maiorca, D., Fumera, G., & Giacinto, G. (s. f.). A risk estimation study of native code vulnerabilities in Android applications. Recuperado 29 de octubre de 2025, de https://dx.doi.org/10.1093/cybsec/tyae015

Smmarwar, S. K., Gupta, G. P., & Kumar, S. (2024). Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: A comprehensive review. *Telematics and Informatics Reports, 14,* 100130. https://doi.org/10.1016/j.teler.2024.100130

Xu, Q., Zhao, D., Yang, S., Xu, L., & Li, X. (2023). Android Malware Detection Based on Behavioral-Level Features with Graph Convolutional Networks. *Electronics, 12(23)*. https://doi.org/10.3390/electronics12234817

e-ISNN 2550-6730 | **Vol. 10, Núm. 1 (24-39): Enero-Junio, 2026** | **DOI: 10.33936/isrtic.v10i1.8098**