M. Ruiz, et al., "AndroCIES: Automatización de la certificación de seguridad para aplicaciones Android", *XVII Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2022)* vol. 265, pp. 192-197, 2022.
NICS Lab. Publications: https://www.nics.uma.es/publications

AndroCIES: Automatización de la certificación de seguridad para aplicaciones Android

Manuel Ruiz, Rubén Ríos, Rodrigo Román, Antonio Muñoz
NICS Lab, Universidad de Málaga
Campus de Teatinos s/n, 29071, Málaga
{mrr, ruben, roman, amunoz}@lcc.uma.es

Juan Manuel Martínez, Jorge Wallace
DEKRA Testing and Certification, S.A.U.
Málaga TechPark, Severo Ochoa, 2&6, 29590, Málaga
{jorge.wallace, juanmanuel.martinez}@dekra.com

***Resumen*—El auge de las plataformas móviles está impulsando el desarrollo de un gran número de aplicaciones, muchas de las cuales salen al mercado sin las convenientes comprobaciones de seguridad. Recientemente, Google está apostando por hacer este problema más visible y concienciar a los usuarios de la necesidad de instalar aplicaciones verificadas por laboratorios independientes. Sin embargo, la certificación de aplicaciones suele ser una tarea ardua y no exenta de errores. Por ello, en este trabajo, presentamos la herramienta AndroCIES, que es capaz de automatizar en gran medida las evaluaciones necesarias para la certificación de aplicaciones móviles, reduciendo en torno a un 20 % el tiempo empleado en este proceso.**

***Index Terms*—certificación, seguridad, aplicaciones, análisis estático**

I. Introducción

Desde su primer lanzamiento público en septiembre de 2008, el sistema operativo para dispositivos móviles, Android, ha recibido una gran acogida por parte de los fabricantes, de los usuarios y de la comunidad de desarrolladores de software. En 2020, había más de 1000 millones de usuarios de Android [1] y su proveedor oficial de aplicaciones, *Google Play*, actualmente cuenta con más de 2,5 millones de apps en su tienda online [2]. Este auge se debe, en gran parte, a su desarrollo accesible basado en un lenguaje de programación común (Java) y la enorme disponibilidad de librerías externas.

Debido a la enorme popularidad de Android, no es difícil encontrar un sinfín de aplicaciones de baja calidad y mal diseñadas desde el punto de vista de la ciberseguridad. De hecho, según [2], casi el 40 % de las aplicaciones existentes en Google Play son de baja calidad. La existencia de este tipo de aplicaciones no sólo afecta negativamente a la seguridad de los usuarios, sino también a la reputación de sus desarrolladores [3]. A fin de evitarlo, Google ha puesto en marcha en su tienda oficial el programa de Evaluación de Seguridad de Aplicaciones Móviles (MASA por sus siglas en inglés), con el objetivo de proporcionar a los usuarios un mecanismo fiable que les permita reconocer fácilmente aquellas aplicaciones que han sido validadas por laboratorios de seguridad independientes [4].

El programa MASA se basa fundamentalmente en los criterios de seguridad establecidos por las normas OWASP (Open Web Application Security Project) para dispositivos móviles. OWASP ha definido una serie de requisitos de seguridad para aplicaciones móviles (MASVS [5]) además de unos casos de prueba (MSTG [6]), que pueden realizarse a través de diversas herramientas para el análisis del código fuente de las aplicaciones (análisis estático) y del comportamiento durante su ejecución (análisis dinámico).

Sin embargo, cabe destacar que, a pesar de existir guías de evaluación y herramientas que permiten realizar gran parte de las pruebas indicadas en tales guías, el proceso sigue siendo una tarea muy manual, que consume gran cantidad de tiempo y es propenso a errores. Así pues, se hace necesaria la creación de herramientas que permitan automatizar lo más posible este proceso de evaluación de aplicaciones móviles, facilitando la tarea de los laboratorios de certificación al tiempo que se reduce la posibilidad de pasar por alto problemas de seguridad, más aún cuando Google Play reconocerá en su tienda a aquellas aplicaciones que hayan sido validadas respecto a un conjunto de requisitos de MASVS.

I-A. Motivación y objetivos

Entre las líneas de negocio de la empresa DEKRA Testing and Certification (en adelante DEKRA) se encuentra la certificación de seguridad de aplicaciones móviles [7]. Su dilatada experiencia en este ámbito y la creciente demanda de este servicio por parte de sus clientes, les ha hecho detectar la necesidad de mejorar su proceso de certificación de aplicaciones. A pesar de contar con diversas herramientas y guías, como la OWASP MSTG, su proceso de certificación requiere de una gran intervención humana. Las herramientas disponibles en el mercado, aunque ofrecen información muy valiosa, no permiten dar un claro veredicto para todos los casos de prueba que requiere el estándar. En otras ocasiones, esta información se encuentra repartida en diferentes partes de la salida de una o varias herramientas. Es aquí donde se hace necesario contar con expertos capaces de interpretar los resultados aportados por las herramientas para así poder ofrecer un veredicto final sobre el caso de prueba. Esto es un tarea tediosa y propensa a errores.

Esta necesidad ha sido materializada en un proyecto de investigación conjunto entre DEKRA y la Universidad de Málaga, denominado CIES. Entre los objetivos de este proyecto se encontraba el desarrollo de una herramienta de evaluación que permitiese a DEKRA reducir el tiempo y esfuerzo empleado en la evaluación de aplicaciones móviles por el equipo de certificación. Más concretamente, la herramienta está diseñada para facilitar la automatización de pruebas de certificación frente a estándares actuales y futuros, como el OWASP MASVS, que es el estándar de facto de la industria y el elegido por Google Play.

A tal fin, la herramienta debería combinar, clasificar y categorizar la información procedente de varias herramientas de análisis, en función de los casos de prueba establecidos por los diferentes estándares. Idealmente la herramienta debería

permitir presentar un veredicto automatizado o, en su defecto, agruparía toda la información necesaria para que el experto pudiera realizar un veredicto con facilidad. De esta forma, no sería necesario que el experto ejecutara varias herramientas y recopilara la información resultante de su ejecución, sino que se limitaría a seguir paso a paso la información expuesta para realizar la verificación del estándar, ahorrando tiempo y esfuerzo en el proceso.

I-B. Estructura del documento

El resto del artículo se organiza según la estructura descrita a continuación. En primer lugar, la sección II, se analiza brevemente el estado del arte y de la técnica, entrando en detalle en los principales estándares actuales y en desarrollo. A continuación, la sección III introduce los requisitos y necesidades de la herramienta desarrollada. En la sección IV se describe la arquitectura de la herramienta en base a las secciones anteriores, mientras que en la sección V se aborda la implementación realizada y su funcionamiento. En la sección VI, se muestran las ventajas que otorga la herramienta. Para finalizar, en la sección VII, se muestran las conclusiones del trabajo y posibles líneas de investigación y desarrollo futuro.

II. Trabajos relacionados

II-A. Estándares de seguridad en móviles

Existen diversos estándares que definen los requisitos mínimos de seguridad y privacidad que una aplicación móvil debe cumplir. Entre ellos, destacan el OWASP Mobile Application Security Verification Standard (MASVS) [5] y el ioXt Mobile Application Profile [8]. El OWASP MASVS es un estándar que establece requisitos de seguridad para aplicaciones móviles y se utiliza en conjunto con la OWASP Mobile Security Testing Guide (MSTG) [6], un manual para el análisis de la seguridad de aplicaciones e ingeniería inversa. En la MSTG se describen procesos técnicos para verificar los casos de prueba necesarios para cumplir con el OWASP MASVS. El ioXt Mobile Application Profile también es un estándar dedicado a la certificación de seguridad de aplicaciones móviles. Dentro de este estándar se definen también una serie de especificaciones de seguridad que otorgan al fabricante los derechos para usar la marca de ioXt Compilance.

Ambos estándares constan de requisitos similares, pero presentan diferencias entre ellos. Aunque ambos son muy completos, el ioXt Mobile Application Profile define requisitos de un nivel más abstracto que el OWASP MASVS, centrándose más en el diseño y la arquitectura que la implementación. Además, el OWASP MASVS presenta una guía de verificación, la ya mencionada OWASP MSTG, lo que le permite centrarse en problemas más concretos. Dado el carácter específico de OWASP MASVS y al interés de Google Play por este dentro de su MASA [4], en este trabajo se ha priorizado el OWASP MASVS y la OWASP MSTG aunque, como se detalla en la sección V, AndroCIES también implementa parte del estándar ioXt.

El OWASP MASVS está compuesto por dos niveles de seguridad. El nivel L1 cubre las buenas prácticas respecto a la seguridad en el desarrollo de aplicaciones. Abarca requisitos básicos relacionados con la calidad del código, el manejo de datos sensibles, y la interacción con el entorno Android. Se ajusta a todas las aplicaciones. Por otra parte, el nivel L2 va dirigido a aplicaciones críticas que necesitan unos requisitos de seguridad más estrictos. En este nivel, la seguridad debe ser parte de la arquitectura de la aplicación, y debe existir un modelo de amenaza. Está orientado a aplicaciones que manejan información altamente sensible, como aplicaciones de banca electrónica.

El nivel L1 de MASVS consta de un total de siete categorías dedicadas a diferentes aspectos de la aplicación: arquitectura, almacenamiento, criptografía, autenticación, comunicación, plataforma y código. Cada una de estas categorías consta de varios requisitos, algunos de estos son de alto nivel mientras que otros son más específicos. En total hay 47 requisitos definidos en el nivel L1 de MASVS. Google MASA considera únicamente un subconjunto de estos requisitos.

II-B. Estado del arte

El análisis de seguridad de aplicaciones se ha tratado en la literatura desde muchos enfoques diferentes pero los predominantes son el análisis estático y el análisis dinámico. El análisis estático consiste en examinar el código fuente de la aplicación, aunque en ocasiones suele partirse de los propios binarios, sin llevar a cabo ningún tipo de ejecución. Durante este análisis, se buscan malas prácticas de seguridad, que pueden ser intencionadas (p.ej., la aplicación incluye un código malicioso) o no intencionadas (p.ej., el propio código incluye credenciales). En cambio, el análisis dinámico se centra en analizar el comportamiento de la aplicación durante su ejecución sobre un dispositivo real o emulado. En este tipo de análisis es importante lanzar un número suficiente de ejecuciones que abarque el mayor número de flujos de ejecución posibles dentro de la aplicación.

El análisis de seguridad de aplicaciones se ha tratado en numerosas ocasiones en la literatura, por ello, nos centraremos en hacer referencia a los trabajos de revisión más recientes y relevantes hasta la fecha. En cuanto al análisis estático, Li et al. [9] realizan una revisión de la misma desde el punto de vista de la detección de fallos no intencionados, mientras que Pan et al. [10] y Jusoh et al. [11] la estudian desde el punto de vista del análisis de malware. Por otra parte, el análisis dinámico también ha recibido gran atención por parte de la comunidad científica. Cabe destacar el trabajo de Kong et al. [12], donde igualmente se recoge una revisión de la literatura.

De los dos tipos de análisis mencionados, el análisis estático es el más prestado a la automatización ya que presenta una menor complejidad al basarse únicamente en archivos fuente y no necesitar la presencia de una plataforma de ejecución.

II-C. Estado de la técnica

Debido al auge y popularidad del sistema operativo Android, se han desarrollado numerosas herramientas destinadas al análisis de sus aplicaciones.

Algunas de estas herramientas de análisis (p.ej., Ostorlab [17], Kryptowire [18] y NowSecure [19]) son servicios privados en la nube ofrecidos por empresas, mientras que otras (p.ej., AndroShield [13], AndrotomistLite [14], MARA Framework [15] y MobSF [16]), son proyectos de código abierto que puede utilizar cualquiera. Las herramientas ofrecidas como servicios cloud privados, aunque bastante completas,

Tabla I
COMPARACIÓN DE HERRAMIENTAS DE ANÁLISIS ESTÁTICO PARA APLICACIONES ANDROID

	Análisis							Clasificación de	Formato de
	Permisos	Manifest	Código	Certificado	URLs	Librerías externas	SaaS	severidad	salida
AndroShield [13]	✗	✓	✓	✗	✗	✗	✗	✓	Página web
AndrotomistLite [14]	✗	✓	✓	✓	✗	✗	✗	✗	Fichero txt
MARA Framework [15]	✓	✓	✓	✓	✓	✗	✗	✓	Ficheros txt y json
MobSF [16]	✓	✓	✓	✓	✓	✓	✗	✓	Base de datos
Ostorlab [17]	✓	✓	✓	✓	✓	✓	✓	✓	Página Web
Kryptowire [18]	✓	✓	✓	✓	✓	✓	✓	✓	Página Web
NowSecure [19]	✓	✓	✓	✓	✓	✓	✓	✓	Página Web

no han sido consideradas para la construcción de nuestra herramienta de automatización al ser difícil su integración y evitar así la dependencia de servicios de terceros.

Las aplicaciones de código abierto son fácilmente integrables y adaptables, permitiendo ser ejecutadas de forma local. Sin embargo, no todas ofrecen una funcionalidad suficiente para abarcar los casos de prueba establecidos por los estándares revisados en la sección II-A. AndroShield es la que, desde nuestro punto de vista, ofrece una funcionalidad más limitada. Esta herramienta permite analizar tanto el código fuente como el Android Manifest[1], pero con un nivel de detalle insuficiente. AndrotomistLite es una herramienta ligeramente más avanzada que la anterior. La parte de análisis estático está compuesta por APKProfiler [20], una herramienta capaz de descompilar las aplicaciones y extraer información del Manifest, el código y el certificado de la misma. A continuación tenemos MARA Framework, que consta de un conjunto de 14 herramientas capaces de cubrir muchos aspectos de interés, como el análisis de permisos, del Manifest, de código, de certificados, y varios más. La principal desventaja que presenta MARA Framework es la forma en que presenta los resultados, ya que simplemente aporta las salidas de las herramientas por separado, cada una en su propio formato. Finalmente, la herramienta MobSF, aunque también hace uso de varias herramientas entre las cuales se encuentra algunas de las presentes en MARA Framework, con los datos obtenidos añade un análisis propio.

En resumen, la herramienta MobSF cubre prácticamente todos los aspectos de análisis estático abordables por los estándares de seguridad en aplicaciones móviles. Además, la información de salida generada se recoge en una única base de datos y, como ventaja adicional, realiza un juicio de severidad para alguna de la información presentada. Sin embargo, de las herramientas de código abierto que se mencionan, casi ninguna realiza un análisis de las librerías externas que utilizan las aplicaciones. MobSF es la única que realiza este análisis pero de manera muy superficial. Más adelante, en la sección IV, mostraremos cómo es posible solucionar esta limitación de MobSF. En la Tabla I se ofrece un resumen de la comparación realizada.

III. ANÁLISIS DE REQUISITOS

El objetivo principal de este trabajo es la construcción de una herramienta que permita automatizar el proceso de certificación de seguridad de aplicaciones móviles basadas en Android. Tanto este objetivo como los requisitos que se detallan a continuación han sido obtenidos a lo largo de varias reuniones entre los miembros del equipo de investigación y desarrollo de la Universidad de Málaga y el equipo de certificación de DEKRA.

- **Completitud**: la herramienta debe ser capaz de evaluar todos los casos de prueba del MASVS L1 que DEKRA considera más relevantes, los cuales están alineados con lo establecido por el MASA de Google Play.
- **Extensibilidad**: la herramienta debe ser capaz de permitir la evaluación de estándares actuales y futuros con relativa facilidad en base a los casos de prueba establecidos en tales estándares sin necesidad de rediseñar la herramienta.
- **Portabilidad**: la herramienta debe ser capaz de ser desplegada en diferentes sistemas operativos y entornos, físicos o virtualizados.
- **Persistencia**: la herramienta debe ser capaz de garantizar la persistencia de los resultados obtenidos tras analizar las aplicaciones.
- **Soporte multi-usuario**: la herramienta debe ser capaz de ofrecer el servicio de análisis a múltiples expertos a la vez.
- **Usabilidad**: la herramienta debe ofrecer una interfaz de usuario sencilla, que permita al experto analizar los resultados forma clara y concisa.

Con todo esto, objetivo y requisitos, se procede al diseño de la herramienta, que detallaremos en la siguiente sección.

IV. DISEÑO

Durante el diseño de la herramienta se ha adoptado una serie de decisiones que permiten acometer los diversos requisitos planteados en la sección anterior. Respecto a la completitud del análisis, se han añadido dos herramientas dedicadas exclusivamente al análisis de librerías externas denominadas LibScout [21] y OWASP Dependency Check [22]. Ambas siguen un funcionamiento similar: realizan un perfil de las librerías que se están utilizando, recogen los identificadores y los comparan con sus bases de datos para ver si se detecta alguna vulnerabilidad. Además, también se ha realizado un fork propio [23] de la herramienta MobSF donde se añaden algunas reglas de detección para el código y una extensión del análisis del Android Manifest.

Para permitir la modificación y creación de nuevos estándares, se ha adoptado una arquitectura modular. Esta arquitectura puede ver reflejada en la Figura 1, dentro del contenedor amarillo de mayor tamaño. Partiendo desde arriba, el módulo de Front-End se encargará de recoger los informes proporcionados por los módulos dedicados a cada uno de los estándares.

[1]En el Manifest se definen datos de interés para analizar la seguridad de una aplicación Android como los permisos o las características hardware necesarias para su ejecución.

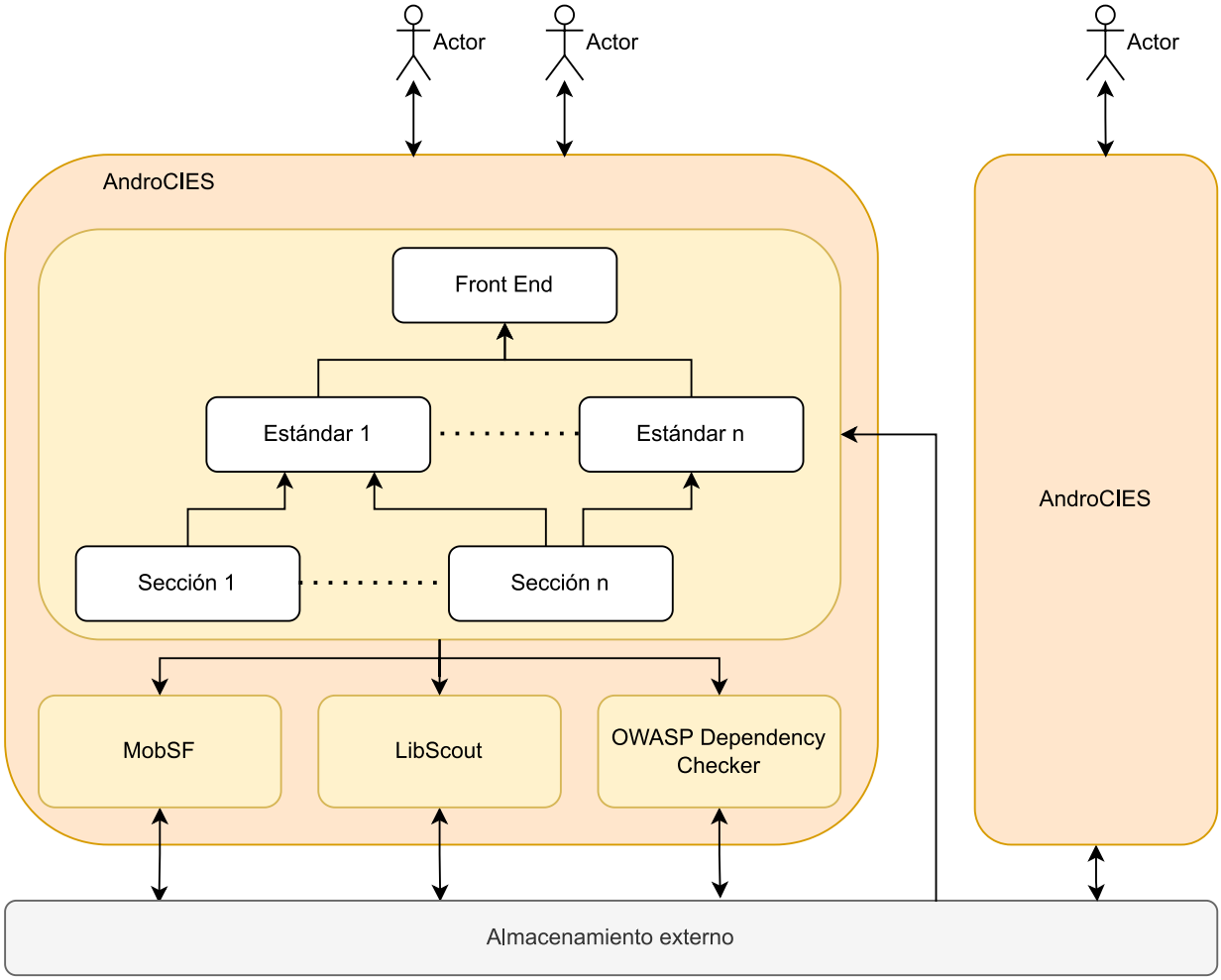

Figura 1. Estructura de la herramienta

Estos, a su vez, procesan la información transmitida por los módulos de sus secciones. Las secciones son los módulos encargados de analizar la información y, si la complejidad lo permite, proporcionar un veredicto. Si alguno de los estándares tuviera secciones que definen requisitos similares, los módulos pueden ser reutilizados.

Para conseguir la portabilidad del sistema, se ha diseñado un despliegue basado en contenedores. A la hora de dividir las herramientas en contenedores, debido a las dependencias, al tamaño y a la disponibilidad de las mismas, se ha decidido crear dos contenedores. El primero contendría la herramienta desarrollada, así como las herramientas de LibScout y OWASP Dependency Check, mientras que el segundo estaría formado por la herramienta MobSF. Sin embargo, debido a la volatilidad de los mismos, ha sido necesario crear una zona de almacenamiento externo para garantizar la persistencia de los datos. Ambos contenedores tendrán acceso a dicho almacenamiento.

Para garantizar la facilidad de uso y una experiencia multiusuario, se ha decidido una interfaz accesible mediante navegador web. Esta interfaz será abordada utilizando una arquitectura del tipo Modelo-Vista-Controlador. De esta forma, se eliminan las dependencias entre la información generada por la herramienta y la muestra de los datos al usuario, permitiendo la reutilización de la información en otros sistemas. Además, cabe mencionar que debido al desacoplamiento el almacenamiento externo, es posible lanzar dos instancias de la herramienta accediendo a las mismas bases de datos. Esto permite dedicar dos máquinas distintas al análisis y ver en cada una de ellas los resultados obtenidos de ambas, lo que facilita la experiencia de varios usuarios de forma paralela.

La arquitectura general del sistema se puede ver en la Figura 1. En ella se representan dos instancias de AndroCIES conectadas al mismo almacenamiento externo. Se observan los componentes internos de AndroCIES, como MobSF, LibSCcout, OWASP Dependency Check y la parte desarrollada en JavaScript encargada de la recogida de la información y el análisis de datos.

V. Implementación y funcionamiento

Tras las consideraciones e ideas presentadas en las secciones III y IV, se ha decidido realizar la implementación con NodeJS. NodeJS es un framework de ejecución de código JavaScript que facilita la implementación de la interfaz web y la modularidad. Además, para la interfaz web se han utilizado plantillas dinámicas de HTML mediante Handlebars. Para la contenerización se ha utilizado Docker y como orquestador se utiliza Docker Swarm, facilitando así el despliegue de la aplicación.

El funcionamiento es el siguiente:

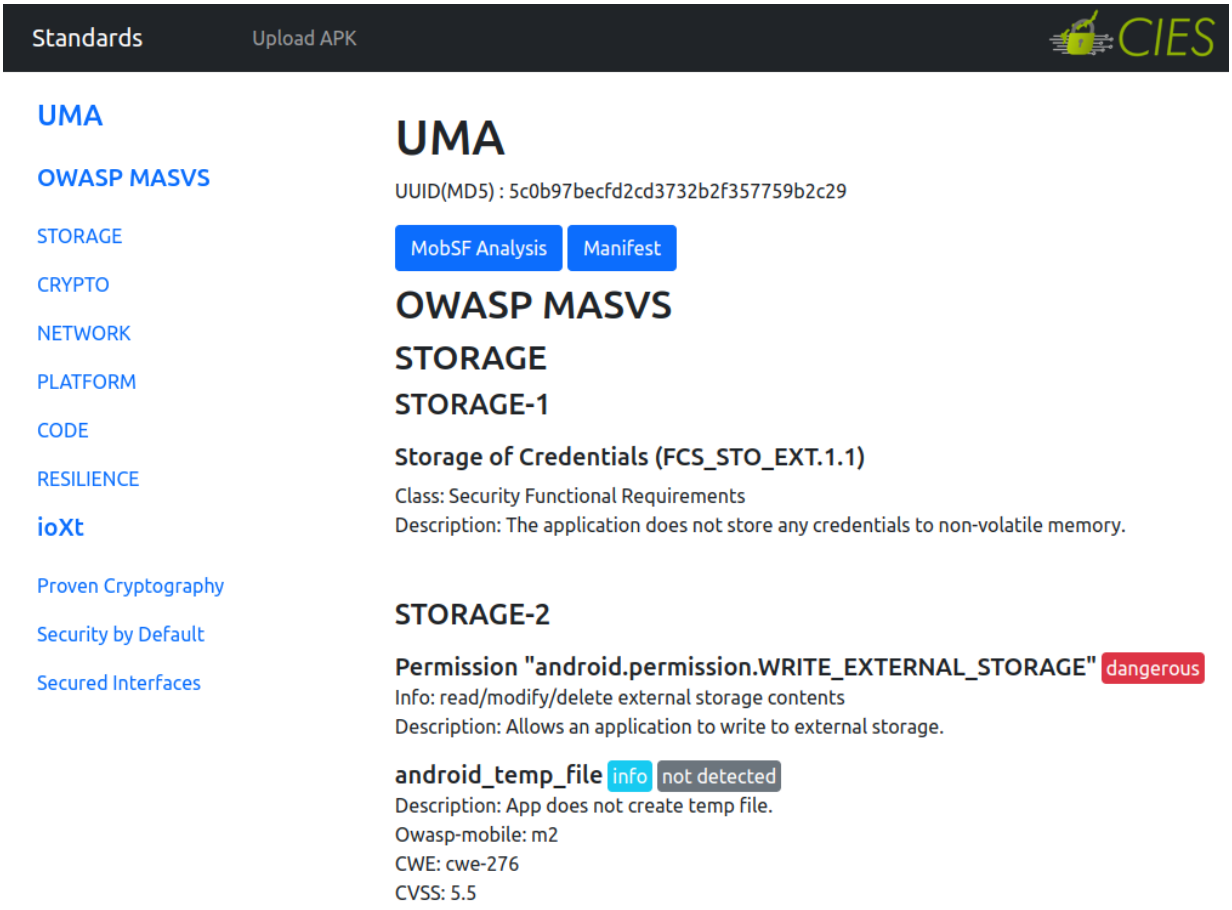

Figura 2. Interfaz de la herramienta

1. El usuario se conecta a la interfaz web y selecciona las aplicaciones sobre las que quiere realizar el análisis.
2. AndroCIES se encarga de enviar las aplicaciones seleccionadas al resto de herramientas de análisis y comenzar el proceso.
3. Las herramientas proporcionan los resultados del análisis en su propio formato. MobSF almacena el resultado en una base de datos, mientras que LibScout y OWASP Dependency Check lo almacenan en ficheros JSON.
4. AndroCIES recoge la información a través de los módulos de entrada, donde se procesa la información.
5. Los módulos de entrada pasan la información a sus módulos superiores, los de estándar, quienes organizan la información en el formato de salida.
6. Los módulos de estándar pasan la información generada al módulo de Front-End que será el encargado de mostrarlo por pantalla.
7. El usuario obtiene el informe generado por pantalla.

La mayoría de módulos de sección obtienen la información de la base de datos de MobSF. Principalmente se extraen datos de la sección del análisis de código, pero también participan otras como la parte de análisis de permisos, de certificado, del Android Manifest, y de los elementos exportados, entre otros. Las herramientas LibScout y OWASP Dependency Check se especializan en el análisis de librerías externas, y son utilizadas por una sola sección. Además, también existen módulos propios de la herramienta dedicados a analizar el archivo de Android Manifest para ampliar la información proporcionada por MobSF.

En la Figura 2 se muestra la interfaz gráfica de la herramienta desarrollada. En la parte superior existe un menú que permite la navegación entre las diferentes vistas de la herramienta. Una de ellas permite seleccionar las aplicaciones a analizar, mientras que la otra, la mostrada en la figura, presenta los informes generados. En la parte izquierda se pueden ver las diferentes secciones de los estándares implementados. En concreto, se ve el estándar OWASP MASVS y sus diferentes categorías, seguido del estándar ioXt y algunas de sus categorías. Este último se encuentra aún en desarrollo. En el informe generado, lo primero que se observa es el nombre de la aplicación analizada (UMA) junto a un identificador único y dos enlaces con información adicional: el informe generado por MobSF sin procesar y el archivo *AndroidManifest.xml*. A continuación, se muestran los resultados para cada una de las secciones del OWASP MASVS, en este caso, los dos primeros apartados de la primera categoría. De esta forma, la información relevante que aparecería dispersa en otras herramientas, es agrupada y mostrada de manera amigable en la sección correspondiente. Por ejemplo, en la sección *Storage-2*, se detecta la existencia de permisos de escritura en el almacenamiento externo y no se detecta de creación de archivos temporales. Esta información aparecería separada en las secciones de análisis de permisos y análisis de código en la herramienta MobSF, lo cual dificulta la realización de un

veredicto sobre este requisito.

VI. Evaluación y Resultados

Tras la implementación de la herramienta, se realizó un primer análisis de seguridad a varias aplicaciones reconocidas de la tienda oficial Google Play. El objetivo de este análisis era comprobar la efectividad de la herramienta a la hora de evaluar los casos de prueba del OWASP MASVS. Estas aplicaciones fueron analizadas de manera independiente por el grupo de investigación de la Universidad de Málaga y por parte de DEKRA, utilizando sus procesos tradicionales. Dichas aplicaciones pertenecen principalmente a las categorías de almacenamiento de fotos, traducción de texto, gestión de contactos y gestión de conexiones VPN.

Los resultados fueron muy satisfactorios ya que los resultados obtenidos por ambos equipos fueron prácticamente idénticos.

Tras esta primera validación, el equipo de certificación de DEKRA ha estado utilizando AndroCIES para el análisis de seguridad de otras aplicaciones móviles. En promedio, se ha determinado una reducción de un 20 % del tiempo empleado para la evaluación respecto al estándar OWASP MASVS utilizando la metodología definida en la OWASP MSTG. Esta es una reducción considerable, teniendo en cuenta el margen de mejora existente si se continúa el trabajo en la misma línea.

VII. Conclusión

El auge en el desarrollo de aplicaciones móviles y el interés de Google en una Play Store más segura y transparente está empujando a los laboratorios especializados a mejorar sus procesos de certificación de seguridad de aplicaciones. En este trabajo se ha realizado un análisis de los principales aspectos que son necesarios para cumplir con los principales estándares de seguridad propuestos por la industria y se han estudiado herramientas que permiten evaluar, aunque no de manera completa, el cumplimiento de esos estándares. A partir de este estudio se ha desarrollado AndroCIES, un herramienta capaz de realizar de manera automática estas pruebas y presentar los resultados de forma intuitiva y organizada a los expertos, reduciendo así el tiempo dedicado al proceso de verificación del estándar OWASP MASVS en torno a un 20 %.

A pesar de que los resultados ofrecidos por la herramienta son prometedores, existen varios aspectos susceptibles de mejora, que abordaremos en el futuro. Aunque una posible línea de trabajo es la adición de nuevos estándares para la certificación de aplicaciones móviles, se trata de una tarea relativamente sencilla de acometer debido al diseño modular de nuestra herramienta. Así pues, consideramos que las dos líneas principales de trabajo futuro serían, por un lado, la incorporación de técnicas de análisis dinámico de código y, por otro lado, el otorgar a la herramienta de cierta inteligencia. En este sentido, la inteligencia iría orientada hacia la detección de posibles falsos positivos, así como a aprender de los veredictos realizados por el analista experto que hace uso de la propia herramienta.

Agradecimientos

Este trabajo ha sido financiado en parte por la Corporación Tecnológica de Andalucía (CTA) bajo la subvención 21/1046 y la Consejería de Empleo, Empresa y Comercio de la Junta de Andalucía a través de la Agencia de Innovación y Desarrollo de Andalucía (IDEA) (pendiente aprobación).

Referencias

[1] M. Mena Roa. (2021) Android e ios dominan el mercado de los smartphones. [Online]. Available: https://es.statista.com/grafico/18920/cuota-de-mercado-mundial-de-smartphones-por-sistema-operativo/

[2] AppBrain. (2022) Number of android apps on google play. [Online]. Available: https://www.appbrain.com/stats/number-of-android-apps

[3] H. Wang, H. Li, L. Li, Y. Guo, and G. Xu, "Why are android apps removed from google play? a large-scale empirical study," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2018, pp. 231–242. [Online]. Available: https://doi.ieeecomputersociety.org/

[4] Google. (2022) Mobile application security assessment. [Online]. Available: https://appdefensealliance.dev/masa

[5] OWASP. (2022) Owasp mobile application security verification standard (masvs). [Online]. Available: https://github.com/OWASP/owasp-masvs

[6] ——. (2022) Owasp mobile security testing guide (mstg). [Online]. Available: https://github.com/OWASP/owasp-mstg

[7] DEKRA. (2021) Dekra autorizado por ioxt alliance para realizar evaluaciones de ciberseguridad en aplicaciones móviles. [Online]. Available: https://www.dekra.es/es/dekra-autorizado-por-ioxt-alliance-para-realizar-pruebas-ciberseguridad-aplicaciones-moviles-vpn/

[8] ioXt Alliance. (2020) ioxt mobile application profile. [Online]. Available: https://static1.squarespace.com/static/5c6dbac1f8135a29c7fbb621/t/604aa3fa668a8e3b50630433/1615504379349/Mobile_Application_Profile.pdf

[9] L. Li, T. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, vol. 88, pp. 67–95, 8 2017.

[10] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A systematic literature review of android malware detection using static analysis," *IEEE Access*, vol. 8, pp. 116 363–116 379, 2020.

[11] R. Jusoh, A. Firdaus, S. Anwar, M. Z. Osman, M. F. Darmawan, and M. F. Ab Razak, "Malware detection using static analysis in android: a review of feco (features, classification, and obfuscation)," *PeerJ Computer Science*, vol. 7, 2021.

[12] P. Kong, L. Li, J. Gao, K. Liu, T. Bissyande, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, 3 2019.

[13] A. Amin, A. Eldessouki, M. T. Magdy, N. Abdeen, H. Hindy, and I. Hegazy, "Androshield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach," *Information*, vol. 10, no. 10, p. 326, 2019.

[14] V. Kouliaridis. (2022) Androtomistlite. [Online]. Available: https://github.com/billkoul/AndrotomistLite

[15] C. Kisutsa. (2019) Mobile application reverse engineering and analysis (mara) framework. [Online]. Available: https://github.com/xtiankisutsa/MARA_Framework

[16] A. Abraham. (2022) Mobile security framework (mobsf). [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF

[17] Ostorlab. (2022) Mobile security testing. [Online]. Available: https://www.ostorlab.co/

[18] Kryptowire. (2022) Kryptowire mobile app security testing. [Online]. Available: https://www.kryptowire.com/

[19] NowSecure. (2022) Nowsecure: Deliver secure mobile apps. [Online]. Available: https://www.nowsecure.com/

[20] I. Grevenitis. (2022) An android application decompilation and feature extraction library. [Online]. Available: https://github.com/Giannisgre/APKProfiler

[21] E. Derr. (2019) Libscout. [Online]. Available: https://github.com/reddr/LibScout

[22] OWASP. (2022) Owasp dependency-check. [Online]. Available: https://owasp.org/www-project-dependency-check/

[23] M. Ruiz Ruiz. (2022) Fork mobsf. [Online]. Available: https://github.com/roxax19/Mobile-Security-Framework-MobSF