Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

AppPIVal
Automatización del Análisis de Seguridad en Aplicaciones Móviles e Integración con Herramientas Externas a través de Mobile Security Framework (MobSF)

Alumno:
Juan Alfonso Velasco Cabrero

Tutores:
Amador Aparicio de la Fuente
Mercedes Martínez González

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres por permitirme cursar estudios universitarios, y agradecer su apoyo durante toda la vida.

En segundo lugar, quiero agradecer a todo mi círculo social, tanto a los amigos de toda la vida como los nuevos que he descubierto en esta estancia en la carrera. Quiero hacer especial mención a todos mis compañeros del Grupo Universitario de Informática, que han hecho que estos cuatro años hayan sido más que un grado universitario. Gracias a todos ellos por hacerme la persona que soy, y poder sentirme orgulloso tanto de mi como de ellos, las personas que me acompañan.

Por último, expreso mi agradecimiento a cada profesor que me haya impartido clase a lo largo de estos 4 años, y especialmente a mis tutores de TFG, Mercedes y Amador, por ofrecer esta oportunidad y las valiosas correcciones hacia mi trabajo.

Resumen

El despliegue de aplicaciones en plataformas como PlayStore o AppStore está mucho más avanzado que las normativas para regular estas, haciendo que existan pocas formas útiles de saber si estas aplicaciones son seguras y protegen nuestra privacidad. En 2015 se desarrolló una herramienta para poder analizar la seguridad de estas aplicaciones, MobSF (Mobile Security Framework). Sin embargo, su uso no está orientado al análisis automático. Este trabajo de fin de grado tiene como objetivo ofrecer una herramienta que automatiza el uso de MobSF, permitiendo el análisis estático de múltiples aplicaciones simultáneamente, automatizar el análisis dinámico, ver análisis recientes y comparar aplicaciones. Además de proporcionar un indicador, *privacy score*, basado en el *security score*, para representar el impacto de estas aplicaciones sobre la privacidad de lo usuarios.

Abstract

The deployment of mobile apps on PlayStore and AppStore has outpaced the development of regulations to control them, making it difficult to know with ease whether these apps are secure and respect user privacy. On 2015 MobSF (Mobile Security Framework) was created, as a tool to analyse the security of these applications. However, its use is not well-suited for automated or large-scale analysis. This final degree project aims to develop a tool that automates the use of MobSF, enabling the simultaneous static analysis of multiple applications, automates dynamic analysis, enables to see recent scans and enables to compare aplications. Besides introducing a new indicator, *the privacy score*, based on the existing *security score*, to assess the impact of these apps over user's privacy.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El presente Trabajo de Fin de Grado aborda la creciente inquietud en torno a la seguridad y la privacidad en el desarrollo de aplicaciones móviles. En un contexto digital cada vez más centrado en el uso de dispositivos móviles, garantizar la protección de los datos personales y asegurar la resistencia de las aplicaciones frente a vulnerabilidades se ha vuelto fundamental. No obstante, tanto la normativa como las buenas prácticas suelen evolucionar más lentamente que el ritmo de la innovación tecnológica, lo que hace necesaria una gestión proactiva y eficaz de estos riesgos.

1.1. Motivación

Hoy en día el desarrollo de aplicaciones móviles está mucho más avanzado que las leyes y políticas que deberían controlar este desarrollo. Entre los diversos aspectos rezagados se encuentran la seguridad y la privacidad.

Existe una percepción generalizada de que muchas empresas desarrolladoras de software, tanto grandes como pequeñas, recolectan datos de los usuarios con fines analíticos o comerciales. Además, muchas veces las empresas quieren sacar un software rápido para facturar antes, dejando de lado aspectos clave como la seguridad de la aplicación.

Por suerte, en el ámbito de la informática siempre surgen grupos que buscan mejorar aspectos de este. Uno de ellos es el equipo detrás de la plataforma MobSF [1], que permite analizar un archivo de aplicación en distintos formatos como ZIP o APK (aquellos detrás de las aplicaciones móviles Android) entre otros, para obtener una puntuación que indique el nivel de seguridad de la aplicación. MobSF es un servicio muy potente, ofreciendo tanto análisis estático como dinámico, así como comparación entre aplicaciones, y descarga de resultados tanto en formato .pdf como .json.

Su uso con interfaz web es bastante intuitivo, sin embargo, no está pensado para análisis

masivos o automatizados. Por suerte, MobSF, cuenta con una API REST [2], con la que podemos usar su servicio desde herramientas externas.

1.2. Objetivos

El objetivo principal de este trabajo automatizar los usos de MobSF, tanto análisis dinámico como el análisis estático permitiendo múltiples aplicaciones simultáneamente. Para completar este objetivo se ofrece una herramienta denominada *AppPiVal*. A continuación se definen los objetivos secundarios:

- Ofrecer un indicador denominado *privacy score* añadido a los resultados ofrecidos por MobSF, debido a que se centra solo en la seguridad dejando de lado la privacidad. Con ello, se ofrece un resultado más completo, con ambos aspectos reflejados. Este indicador está basado en el *security score* que ofrece MobSF.
- Proporcionar un mecanismo de despliegue reproducible, automatizado y portable para el entorno completo de análisis, prescindiendo así de instancias de MobSF ajenas.

1.3. Estructura de la memoria

A continuación se especifica como se organiza la memoria:

- **Capítulo 1. Introducción:** Una breve descripción del proyecto en la que se indican los aspectos fundamentales y objetivos.
- **Capítulo 2. Trabajo relacionado:** Explicación de las soluciones actuales, discutiendo sus limitaciones y las aportaciones de AppPiVal.
- **Capítulo 3. Planificación:** Todo lo referido a la planificación inicial en horas de trabajo por días de la semana y las fechas previstas de finalización de tareas y del propio proyecto. También incluye un plan de riesgos.
- **Capítulo 4. Análisis:** Estudio de las opciones actuales, así como sus desventajas frente a la opción desarrollada en este trabajo. Además, se explican los requisitos y los Casos de Uso relacionados.
- **Capítulo 5. Diseño:** Explicación lógica del desarrollo de AppPiVal, así como del entorno de trabajo utilizado.
- **Capítulo 6. Implementación:** Aborda el desarrollo e instalación del entorno controlado, así como la propia herramienta AppPiVal.
- **Capítulo 7. Pruebas:** Describe el proceso de lanzado de pruebas para verificar que AppPiVal funcione tal y como se indica.

- **Capítulo 8. Conclusiones:** Una breve reflexión sobre los resultados obtenidos tanto en la ejecución de las pruebas como en el propio proyecto. También incluye una serie de mejoras futuras que se podrían realizar para ampliar el alcance del proyecto.

Capítulo 2

Trabajo relacionado

Actualmente, existen cuatro alternativas al realizar una búsqueda relacionada con *"MobSF automation"*. A continuación se describen estas soluciones, así como limitaciones comunes y las aportaciones de AppPiVal.

2.1. Automation-MobSF

El repositorio *Automation-MobSF*[1], desarrollado por el usuario ZachGeo, implementa un servicio que automatiza el análisis de MobSF.

El sistema funciona mediante un script `.sh` que permanece activo y, en intervalos regulares de dos minutos, revisa una carpeta predefinida en busca de archivos `.apk`. En caso de encontrar nuevas aplicaciones, lanza su análisis automáticamente a través de la API de MobSF.

Este sistema presenta una solución simple y funcional, con un enfoque orientado a la monitorización continua. Sin embargo, realiza los análisis de forma completamente secuencial, sin aprovechar capacidades de paralelización o multitarea, lo que limita su escalabilidad en escenarios de análisis masivo.

2.2. Automation of APK Analysis with MobSF

En el blog técnico *Mobile Security: Automation of APK Analysis with MobSF*[2] se describe un procedimiento para automatizar análisis utilizando MobSF a través de scripts y comandos personalizados.

[1] https://github.com/ZachGeo/Automation-MobSF.git
[2] https://wiki.elvis.science/index.php?title=Mobile_Security:_Automation_of_APK_Analysis_with_MobSF

A diferencia de otras herramientas, este enfoque requiere ejecutar manualmente la herramienta para cada aplicación a analizar, lo que lo convierte en una solución poco escalable para entornos donde se desee procesar un volumen elevado de aplicaciones. Además, al igual que otros trabajos similares, el análisis es secuencial, sin posibilidad de ejecutar múltiples procesos en paralelo.

2.3. MobSF-Automation

El usuario pentestguy mantiene el repositorio *MobSF-Automation*[3], una solución dockerizable que permite automatizar el análisis de una aplicación móvil utilizando MobSF.

El sistema permite parametrizar el origen y destino de los archivos, así como la URL de MobSF y su clave API. Una vez lanzado el contenedor, se inicia el análisis del archivo especificado.

Aunque se trata de una herramienta fácilmente integrable y portable, también presenta la limitación de trabajar de forma secuencial. Esto impide su uso eficiente en entornos donde se requiere análisis masivo o simultáneo de múltiples aplicaciones.

2.4. Script propio de MobSF

El propio grupo que desarrolló MobSF, cuenta con *mass_static_analysis.py*[4], script que automatiza el análisis estático para un conjunto masivo de aplicaciones.

Este sistema itera en los ficheros compatibles de una carpeta y los analiza secuencialmente uno a uno.

Aunque es una solución válida, presenta de nuevo la limitación de trabajar de forma secuencial. Además, solo automatiza este tipo de análisis, dejando de lado el análisis dinámico o la comparación entre aplicaciones.

2.5. Limitaciones comunes y aportación

Las herramientas analizadas comparten una serie de limitaciones comunes:

- El análisis se realiza de forma secuencial, lo que restringe la capacidad de procesar grandes volúmenes de aplicaciones de manera eficiente.
- En algunos casos, la ejecución debe realizarse manualmente o no permite una integración sencilla con sistemas externos.

[3] https://github.com/pentestguy/MobSF-Automation.git
[4] https://github.com/MobSF/Mobile-Security-Framework-MobSF/blob/master/scripts/mass_static_analysis.py

- No contemplan mecanismos para el tratamiento estructurado y reutilización de los resultados, ni lógica para generar indicadores derivados, como un Privacy Score.
- Solo uno de ellos, *Automation-MobSF* automatiza el análisis dinámico mediante *web scrapping* [3]

AppPiVal propone un enfoque distinto, centrado en la automatización del análisis estático de múltiples aplicaciones mediante `asyncio`, así como la automatización del análisis dinámico con su propio entorno emulado. Estas características lo convierten en una propuesta más flexible, escalable y orientada a entornos reales de análisis masivo y explotación de datos de seguridad y privacidad.

Capítulo 3

Planificación

En este capítulo se presenta la planificación general del proyecto. Para ello, se analizan las características clave del trabajo, se define la metodología adoptada para su desarrollo, se establece una planificación inicial con las principales tareas a realizar y se identifican los posibles riesgos que podrían afectar al correcto desarrollo del proyecto.

3.1. Características del proyecto

El proyecto está orientado a cumplir con los objetivos explicados en la sección 1.2. Pueden distinguirse varias etapas relevantes que permiten estructurar el trabajo realizado.

- **Estudio de tecnologías:** Análisis detallado de la API de MobSF, eje central del proyecto, evaluando sus capacidades para la automatización de análisis estáticos y dinámicos. Estudio de técnicas de concurrencia en Python y sus distintas librerías para estructurar el proyecto.
- **Prototipo inicial:** Primer prototipo funcional de la herramienta, centrado en los aspectos esenciales del análisis estático automatizado. Menú por línea de comandos, el sistema de configuración a través de ficheros YAML.
- **Expansión de funcionalidades:** Una segunda iteración para añadir nuevas capacidades, como la recuperación de análisis previos, la comparación entre aplicaciones y la automatización del análisis dinámico.
- **Desarrollo entorno docker:** Generar un archivo docker-compose.yml que contenga todo lo necesario para poder ejecutar AppPiVal sin depender de instancias externas de MobSF o emuladores.
- **Pruebas en entorno controlado:** Diseñar un entorno aislado para la ejecución de pruebas, garantizando la repetibilidad de los análisis. Analizar un dataset de aplicacio-

nes para evaluar su comportamiento en términos de seguridad y privacidad, capturando métricas relevantes para su posterior interpretación.

3.2. Metodología empleada

Dado el enfoque dual del proyecto, se ha optado por emplear una metodología que permita avanzar de forma iterativa y adaptable. Concretamente, se utilizarán el desarrollo incremental basado en prototipos.

3.2.1. Desarrollo incremental basado en prototipos

Para la parte técnica, se seguirá un enfoque iterativo orientado a la construcción progresiva de prototipos funcionales. Esta metodología permite validar funcionalidades clave de forma temprana, realizar ajustes a medida que surgen nuevos requisitos y minimizar el impacto de posibles errores estructurales.

Cada iteración del proyecto permitirá incorporar nuevas funcionalidades, como la automatización del análisis estático o dinámico, la configuración mediante ficheros YAML o la generación del *Privacy Score*, evaluando en cada paso su integración técnica y usabilidad.

3.3. Planificación

El desarrollo del proyecto se llevará a cabo en un total de 300 horas distribuidas desde el 13 de enero de 2025 hasta el 16 de junio de 2025. La distribución inicial de horas durante la semana es la siguiente.

Día	Horario	Horas
Lunes	19:30 - 21:00	1,5
Martes	19:30 - 21:00	1,5
Miércoles	19:30 - 21:00	1,5
Jueves	-	-
Viernes	-	-
Sábado	11:30 - 14:00 17:30 - 20:00	5
Domingo	10:30 - 14:00	3.5
Total	-	13h semanales

Tabla 3.1: Distribución de horas de trabajo

Teniendo en cuenta la cantidad de horas de trabajo establecidas, se estiman las fechas de cumplimiento de objetivos que se muestran en la figura 3.1. En ella se indican el conjunto

de tareas con sus correspondientes subtareas incluyendo una estimación para cada una de ellas. La herramienta empleada para organizar el conjunto de tareas mostradas en la figura es Microsoft Project [4].

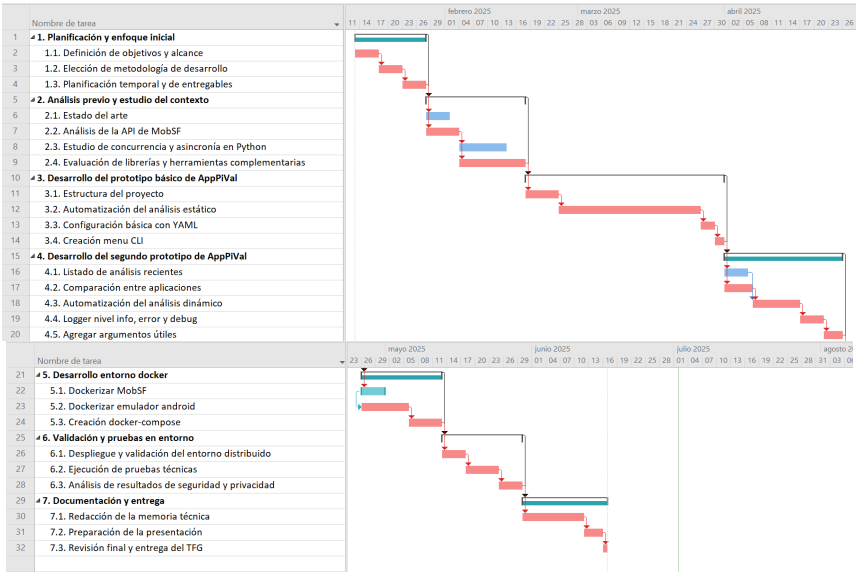

Figura 3.1: Planificación de las tareas del proyecto

Partiendo de la figura 3.1, los principales hitos del proyecto junto a las correspondientes fechas quedarían de la siguiente forma:

Hito	Fecha
Elección y entendimiento de tecnologías a usar	17/02/2025
Primer prototipo de AppPiVal	31/03/2025
Segundo prototipo de AppPiVal	25/04/2025
Archivo docker-compose.yml	12/05/2025
Obtención de resultados y su análisis	29/05/2025
Fin del proyecto	16/06/2025

Tabla 3.2: Hitos del proyecto con las correspondientes fechas

3.4. Riesgos

En esta sección, se elaborará el plan de riesgos que podrían influir en la gestión del proyecto. Para lograr esto, se procederá a la identificación de los riesgos pertinentes y se llevará a cabo una caracterización de estos basada en su probabilidad y su impacto. Al

asignar una escala del 1 al 4 a ambos aspectos, es posible desarrollar una matriz de riesgos que refleje el efecto que tendrían en la planificación del proyecto. A continuación, se presenta un ejemplo de una matriz de riesgos:

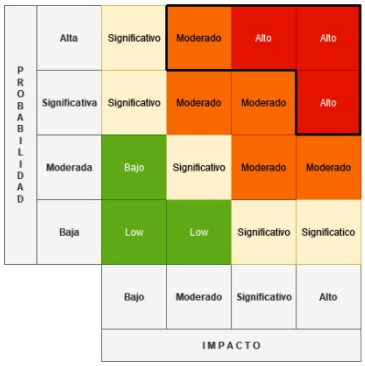

Figura 3.2: Matriz de riesgos (obtenida del libro Software Project Management 5ª edición)

Se otorgará una mayor importancia a aquellos riesgos que se sitúen más cerca de la esquina superior derecha de la matriz. Una vez que se ha establecido un método para clasificar los riesgos, el siguiente paso es proceder a su identificación. Para este proceso, se han considerado factores tanto internos como externos al alumno. A continuación, se enumeran los riesgos identificados hasta el momento:

Nº	**Riesgo**	**Probabilidad**	**Impacto**	**Daño**
1	Estimación de trabajo inferior a la real	3	2	Moderado
2	Incumplimiento de horarios de trabajo	2	1	Bajo
3	Nuevo servicio supera a MobSF	1	3	Significativo
4	API de MobSF cambia de formato	1	3	Significativo
5	Falta disponibilidad MobSF	1	4	Significativo
6	Daños en el ordenador personal	1	1	Bajo
7	Fecha de finalización posterior a la prevista	3	4	Alto
8	Modificación o incumplimiento de alguno de los objetivos	2	3	Moderado
9	Rendimiento de AppPiVal inferior al esperado	2	3	Moderado

Tabla 3.3: Análisis de los riesgos del plan del proyecto

Una vez que se han identificado los riesgos, existen cuatro posibles acciones que se pueden llevar a cabo: aceptar, evitar, reducir y mitigar, o transferir. En gran medida, los riesgos que podrían incidir en la planificación no pueden ser transferidos a otro contexto, ya que todas las tareas son componentes del camino crítico. Por lo tanto, la primera estrategia será evitar dichos riesgos, y como segunda medida, se buscará disminuir la probabilidad y reducir el impacto de cada uno. Si esto no fuera viable, se aceptará su presencia, lo que implicaría un atraso en la culminación del proyecto. Al analizar cada riesgo en particular, se especificarán en la tabla las acciones a implementar para cada uno.

Nº	**Riesgo**	**Acción a realizar**
1	Estimación de trabajo inferior a la real.	Preguntar a los tutores, acelerar tareas y priorizar objetivo principal.
2	Incumplimiento de horarios de trabajo.	Adaptar el horario de trabajo para cumplir con las horas establecidas.
3	Nuevo servicio supera a MobSF.	Analizar, si tiene, su API y plantear adaptación de AppPiVal.
4	API de MobSF cambia de formato.	Analizar los cambios, adaptar y mejorar AppPiVal.
5	Falta disponibilidad MobSF.	Analizar errores del despliegue de MobSF y reiniciar.
6	Daños en el ordenador personal.	Copias con control de versiones en la nube.
7	Fecha de finalización posterior a la prevista.	Realizar la entrega y defensa en una fecha posterior.
8	Modificación o incumplimiento de alguno de los objetivos.	Fijar bien los objetivos con los tutores al principio del proyecto.
9	Rendimiento de AppPiVal inferior al esperado.	Generar semáforos para controlar el uso de los recursos del sistema.

Tabla 3.4: Plan de actuación ante cada uno de los riesgos

A continuación se realiza una descripción detallada de los riesgos y de las acciones a realizar para cada uno de ellos:

1. **Estimación de trabajo inferior a la real:**
   La estimación de tiempo puede verse afectada por desconocimiento de las tecnologías a utilizar o dificultades inesperadas, lo que podría provocar que el tiempo estimado esté alejado de la realidad y ocasione retrasos en el cumplimiento de los objetivos.

   Acciones: Consultar con los tutores para validar las estimaciones realizadas. Priorizar las tareas más críticas y acelerar actividades cuando sea necesario. Considerar la posibilidad de dedicar horas adicionales para mantener los plazos previstos y delegar carga a trabajo futuro

2. **Incumplimiento de horarios de trabajo:**

La simultaneidad del proyecto con otras responsabilidades académicas, prácticas curriculares o extracurriculares puede dificultar el cumplimiento estricto del horario de trabajo previsto.

Acciones: Adaptar el horario de trabajo para compensar las horas destinadas a otras actividades, realizando trabajo fuera del horario habitual si fuese necesario para no afectar el desarrollo del proyecto.

3. **Nuevo servicio supera a MobSF:**
   MobSF es una herramienta consolidada, potente y de código abierto con una comunidad activa y soporte constante, por lo que es poco probable que surja un servicio que la supere ampliamente en el corto plazo. No obstante, existe el riesgo de que nuevas soluciones con funcionalidades o integraciones avanzadas puedan hacer recomendable su adopción o complementariedad.

   Acciones: Mantenerse informado sobre novedades y avances en herramientas de análisis de seguridad móvil. Evaluar periódicamente nuevas soluciones para valorar su integración o sustitución. Adaptar AppPiVal para aceptar los cambios si el nuevo servicio ofece API.

4. **API de MobSF cambia de formato:**
   Las APIs de proyectos maduros como MobSF suelen mantener estabilidad en sus formatos y endpoints para preservar la compatibilidad con clientes y usuarios. Aunque no es habitual, es posible que en actualizaciones mayores se produzcan cambios que afecten la integración.

   Acciones: Vigilar las notas de versión y actualizaciones de MobSF. Si cambia, estudiar si este afecta a los CU planteados y adaptar AppPiVal.

5. **Falta disponibilidad MobSF:**
   Ejecutar MobSF localmente dentro de un contenedor Docker, como se explicará en en capítulo 6 reduce significativamente el riesgo de falta de disponibilidad, ya que en caso de fallo basta con reiniciar el contenedor o la máquina anfitriona para restaurar el servicio rápidamente. Sin embargo, pueden darse problemas transitorios por errores en el despliegue, fallos en la máquina o conflictos de red.

   Acciones: Implementar monitorización básica para detectar caídas o errores del contenedor MobSF. Automatizar reinicios del contenedor cuando se detecten fallos. Mantener copias de seguridad de la configuración y datos para facilitar una rápida restauración.

6. **Daños en el ordenador personal:**
   Un fallo o daño en el equipo personal podría afectar de forma significativa al desarrollo del proyecto.

   Acciones: Realizar copias de seguridad periódicas con control de versiones almacenadas en repositorios remotos para asegurar la integridad y disponibilidad del código y documentos.

7. **Fecha de finalización posterior a la prevista:**
   El proyecto puede requerir más tiempo del inicialmente planificado, lo que podría impedir la presentación y defensa en la convocatoria ordinaria.

   Acciones: En caso de retrasos significativos, planificar la entrega y defensa en convocatoria extraordinaria, asegurando una comunicación adecuada con los tutores.

8. **Modificación o incumplimiento de alguno de los objetivos:**
   Cambios en los objetivos del proyecto o incumplimientos pueden afectar el alcance y calidad del trabajo final.

   Acciones: Definir claramente los objetivos con los tutores al inicio del proyecto y gestionar cualquier modificación de forma controlada, evaluando su impacto y trasladando posibles cambios importantes a trabajos futuros si fuera necesario.

9. **Rendimiento de AppPiVal inferior al esperado** Debido a que vamos a ejecutar AppPiVal junto a un entorno docker, es muy posible que el rendimiento de los análisis sea inferior al esperado.

   Acciones: Se generarán unos semáforos para controlar la subida de los archivos de aplicación y la entrada de nuevos análisis controlando mejor el uso de recursos.

3.5. Seguimiento de la Planificación y Gestión de Riesgos

La planificación inicial del proyecto, detallada en las secciones previas, ha servido como guía fundamental para la ejecución del Trabajo de Fin de Grado. Durante el desarrollo del proyecto, se ha realizado un seguimiento continuo para evaluar el progreso y gestionar cualquier desviación o imprevisto.

En términos generales, el cronograma establecido se mantuvo dentro de los plazos previstos para la mayoría de los hitos. No obstante, **el hito correspondiente a la obtención del archivo** `docker-compose.yml` experimentó un ligero retraso de una semana respecto a la fecha inicialmente programada. Este desvío se debió principalmente a la complejidad de obtener un emulador Android dockerizado que satisficiera las características requeridas por MobSF, lo que llevó a la necesidad de crear una imagen personalizada y a la exhaustiva documentación de su proceso de construcción. Adicionalmente, la elaboración de la memoria ha tomado más tiempo del esperado, resultando en un retraso de dos semanas en la entrega final del TFG.

Para mitigar estos desajustes, se implementaron las siguientes acciones:

- Se reasignaron temporalmente los esfuerzos, dedicando más tiempo a las tareas afectadas para acelerar su finalización.
- Se tomó la decisión de retrasar la fecha de entrega del TFG con el fin de asegurar el cumplimiento integral de todos los hitos y la calidad del trabajo planteado.

Gracias a estas medidas, el impacto en el cronograma global del proyecto fue gestionado, permitiendo alcanzar los objetivos propuestos y entregar un trabajo completo.

Respecto al plan de riesgos:

- Se materializó el riesgo relacionado con el rendimiento de AppPiVal. Como se explicará en la sección 5.4, la subida masiva de archivos de aplicación o el análisis de un volumen excesivo de estos, puede ser perjudicial para el rendimiento del sistema. Sin embargo, gracias a la implementación de semáforos asíncronos en el diseño, este riesgo pudo ser mitigado y su impacto reducido, sin comprometer la calidad del resultado final.

Este seguimiento activo no solo ha asegurado el cumplimiento de los objetivos y el cronograma (con las adaptaciones necesarias), sino que también ha garantizado la robustez de la solución desarrollada, AppPiVal, frente a los desafíos inherentes a proyectos de esta naturaleza.

Capítulo 4

Análisis

En este capítulo se presenta los requisitos funcionales, no funcionales y los casos de Uso que se contemplan, así como el modelo de datos.

4.1. Requisitos

En esta sección se describen los requisitos necesarios para el desarrollo e implementación de la herramienta de automatización AppPiVal. Se incluyen tanto los requisitos funcionales, que detallan las características y comportamientos esperados, como los requisitos no funcionales, que establecen condiciones de rendimiento, seguridad y escalabilidad. Además, se identifican las posibles restricciones técnicas que pueden influir en el desarrollo del proyecto.

4.1.1. Requisitos funcionales

- **RF01 - Análisis estático automatizado:** AppPiVal debe permitir ejecutar automáticamente análisis estáticos sobre archivos de aplicación mediante MobSF.
- **RF02 - Análisis dinámico automatizado:** AppPiVal debe permitir ejecutar automáticamente análisis dinámicos sobre archivos de aplicación mediante MobSF.
- **RF03 - Análisis estático masivo de aplicaciones:** AppPiVal debe aceptar carpetas con múltiples archivos de aplicación y procesarlos de forma concurrente.
- **RF04 - Extracción estructurada de resultados:** AppPiVal debe permitir guardar los resultados obtenidos del análisis en la ruta proporcionada, en sub-directorios dependiendo de los formatos seleccionados, JSON, PDF.
- **RF05 - Gestión de errores del análisis:** AppPiVal debe permitir registrar y reportar errores en los análisis, como fallos de ejecución de MobSF o conflictos de dependencias.

- **RF06 - Control por línea de comandos:** AppPiVal debe poder ejecutarse desde la interfaz de línea de comandos (CLI) con diferentes parámetros como *verbose*, o *modo menú interactivo*.
- **RF07 - Configuración personalizable:** AppPiVal debe permitir configurar parámetros como número de hilos concurrentes, rutas de logs, o el endpoint de MobSF desde un mismo archivo.
- **RF08 - Generación *privacy_score*:** AppPiVal debe generar un índice de privacidad, *Privacy Score* basado en los resultados del análisis. El *privacy score* es un indicador para cuantificar el impacto en la privacidad a partir del *Security Score* [1], siendo un modelo lineal inverso que establece una relación entre estos dos elementos.
- **RF09 - Comparación apps:** AppPiVal debe permitir comparar los resultados de dos análisis anteriores.
- **RF10 - Análisis recientes:** AppPiVal debe permitir mostrar el histórico de análisis anteriores.
- **RF11 - Control sobre estado MobSF:** AppPiVal debe permitir mostrar el estado del entorno para el análisis.

4.1.2. Requisitos no funcionales

- **RNF01 - Registro persistente de ejecución:** AppPiVal debe mantener logs de ejecución accesibles para auditoría.
- **RNF02 - Independencia del interfaz gráfico:** El sistema debe funcionar completamente desde la línea de comandos sin necesidad de interfaz de usuario.
- **RNF03 - Configuración sencilla:** La herramienta debe utilizar configuración externa en formato YAML.
- **RNF04 - Robustez ante fallos de red o servicios externos:** Debe manejar adecuadamente los errores producidos.
- **RNF05 - Entorno propio:** Se debe conseguir un entorno funcional sin dependencias externas como MobSF.
- **RNF06 - Portabilidad:** Se debe poder ejecutar en cualquier entorno gracias a la portabilidad *docker*.

4.2. Casos de Uso

A continuación se presentan los casos de uso que derivan de los requisitos anteriormente explicados, que servirán como base para las pruebas realizadas en el capítulo 7.

[1] Indicador que varía entre 0 y 100, y se basa en la detección de configuraciones inseguras, permisos excesivos y vulnerabilidades en el código y los componentes de la aplicación.

4.2.1. Diagrama de Casos de Uso

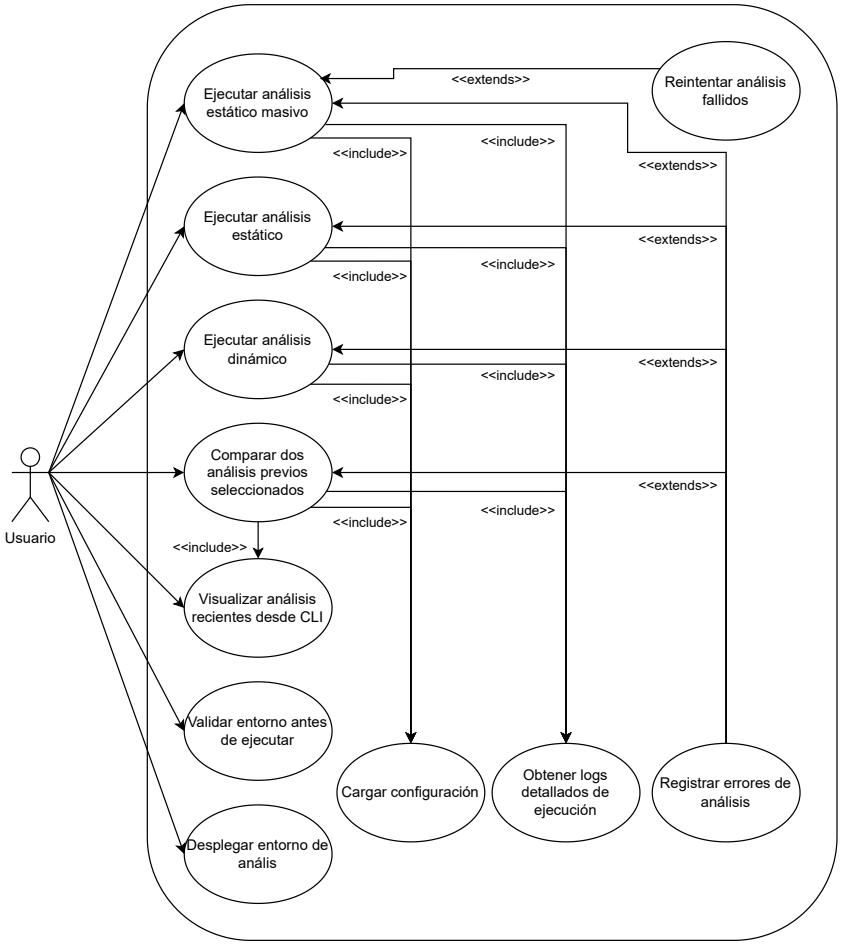

Figura 4.1: Diagrama de Casos de Uso del sistema

4.2.2. Descripción Casos de Uso

Los casos de uso son la descripción de una acción o actividad que puede ser realizada dentro del sistema. El objetivo de la identificación y descripción de los casos de uso es definir las posibles acciones más relevantes que forman parte del sistema y como este se debe comportar durante su realización, para ello especificaremos el flujo normal del caso de uso, la realización exitosa de una acción o actividad, y aquellos flujos alternativos ante imposibilidades de realización o fallos, tanto del sistema como de los actores, a la hora de realizar cada uno de los casos de uso.

En esta sección se describirán los casos de uso identificados en la Figura 4.1, detallando su flujo principal, condiciones, excepciones y requisitos asociados.

CU01	Ejecutar análisis estático sobre una archivo APK.
Descripción:	El usuario proporciona un archivo APK para que AppPiVal realice un análisis estático.
Actor:	Usuario
Pre-condiciones:	Los contenedores (MobSF, DjangoQ) están activos y en estado healthy, y el archivo APK existe y es accesible.
Post-condiciones:	Los resultados se encuentran en los subdirectorios de la carpeta proporcionada en el archivo `config.yml`, y se registra las operaciones en el archivo de logs.
Requisitos asociados:	RF01, RF04, RF06, RF08, RNF01, RNF02, RNF03
Flujo principal:	1. El usuario ejecuta AppPiVal desde CLI con el parámetro -f/–file indicando archivo APK. 2. AppPiVal **incluye CU07 - Cargar configuración desde archivo YAML** `config.yml`. 3. AppPiVal sube la aplicación a MobSF. 4. AppPiVal empieza el análisis estático contra MobSF. 5. AppPiVal descarga y almacena los resultados establecidos en el archivo `config.yml`.
Excepciones:	- MobSF no responde (timeout o error 500). - El APK es inválido o no puede analizarse. - Error de escritura en disco al guardar los resultados.
Notas:	Este caso de uso activa de forma implícita CU08 - Guardar logs de ejecución.

Tabla 4.1: Análisis del caso de uso CU01

CU02	Ejecutar análisis dinámico sobre una archivo APK.
Descripción:	El usuario elige una app disponible para el análisis dinámico y esta se analiza automáticamente.
Actor:	Usuario
Pre-condiciones:	Los contenedores (MobSF, AVD) están activos y en estado healthy, y existe una aplicación disponible para análisis dinámico.
Post-condiciones:	Se almacenan los resultados dinámicos y el evento queda registrado en el log.
Requisitos asociados:	RF02, RF04, RF06, RNF01, RNF02
Flujo principal:	1. El usuario ejecuta AppPiVal desde CLI con el parámetro -i/–interactive. 2. AppPiVal **incluye CU07 - Cargar configuración desde archivo YAML** `config.yml`. 3. AppPiVal muestra el menú interactivo. 4. El usuario elige la opción análisis dinámico. 5. AppPiVal muestra las Apps disponibles para el análisis dinámico. 6. El usuario elige la aplicación. 7. AppPiVal inicia el análisis dinámico contra MobSF. 8. AppPiVal descarga y almacena los resultados.
Excepciones:	- No hay APKs disponibles con análisis estático. - El AVD no responde o no se puede iniciar. - MobSF no puede conectarse con el AVD.
Notas:	Este caso de uso activa de forma implícita CU08 - Guardar logs de ejecución.

Tabla 4.2: Análisis del caso de uso CU02

CU03	Ejecutar análisis estático masivo sobre varios archivos APK.
Descripción:	El usuario inicia el análisis estático sobre múltiples APKs para que AppPiVal los analice concurrentemente.
Actor:	Usuario
Pre-condiciones:	Los contenedores (MobSF, DjangoQ) están activos y en estado healthy y la carpeta indicada en config.yml contiene varios archivos .apk.
Post-condiciones:	Se generan múltiples resultados estructurados y se registran logs de cada análisis, exitoso o fallido.
Requisitos asociados:	RF03, RF04, RF06, RF07, RF08, RNF01, RNF02, RNF03
Flujo principal:	1. El usuario ejecuta AppPiVal. 2. AppPiVal **incluye CU07 - Cargar configuración desde archivo YAML** `config.yml`. 3. AppPiVal inicia análisis concurrente de las APK contra MobSF. 4. AppPiVal realiza *polling* a MobSF hasta que el análisis se ha completado. 5. AppPiVal descarga y almacena los resultados establecidos en el archivo `config.yml`.
Excepciones:	- Carpeta vacía o ruta inválida. - Timeout o error de red en conexión con MobSF. - Excepción durante una tarea asíncrona.
Notas:	Este caso de uso activa de forma implícita CU08 - Guardar logs de ejecución.

Tabla 4.3: Análisis del caso de uso CU03

CU04	Visualizar análisis recientes desde CLI
Descripción:	El usuario elige opción ver análisis recientes en el menu CLI.
Actor:	Usuario
Pre-condiciones:	Existen análisis previos almacenados.
Post-condiciones:	Se muestra el listado de análisis previos disponibles.
Requisitos asociados:	RF06, RF10
Flujo principal:	1. El usuario ejecuta AppPiVal desde CLI con el parámetro -i/–interactive. 2. AppPiVal **incluye CU07 - Leer configuración desde YAML**. 3. AppPiVal muestra el menú interactivo. 4. El usuario elige la opción *visualizar análisis recientes*. 5. AppPiVal muestra las aplicaciones escaneadas.
Excepciones:	- No hay análisis anteriores registrados. - Error al acceder a los datos almacenados.

Tabla 4.4: Análisis del caso de uso CU04

CU05	Comparar dos análisis previos seleccionados
Descripción:	El usuario elige la opción de comparar dos aplicaciones.
Actor:	Usuario
Pre-condiciones:	Existen al menos dos resultados de análisis anteriores.
Post-condiciones:	Se genera y guarda un informe de comparación.
Requisitos asociados:	RF06, RF09, RF10
Flujo principal:	1. El usuario ejecuta AppPiVal desde CLI con el parámetro -i/–interactive. 2. AppPiVal **incluye CU07 - Cargar configuración desde archivo YAML** `config.yml`. 3. AppPiVal muestra el menú interactivo. 4. El usuario elige la opción *comparar aplicaciones*. 5. AppPiVal muestras las aplicaciones escaneadas recientemente. 6. El usuario elige dos aplicaciones a analizar. 7. AppPiVal ejecuta la comparación. 8. AppPiVal guarda los resultados.
Excepciones:	- Menos de dos resultados disponibles. - Error al acceder a los datos o al escribir el informe.
Notas:	Este caso de uso activa de forma implícita CU08 - Guardar logs de ejecución.

Tabla 4.5: Análisis del caso de uso CU05

CU06	Reintentar análisis fallidos
Descripción:	Permite reintentar el análisis de aquellas aplicaciones cuyo análisis anterior no se completó con éxito.
Actor:	Usuario
Pre-condiciones:	El usuario ejecuta AppPiVal con el parámetro `--retry`
Post-condiciones:	Los análisis se ejecutan nuevamente y se actualizan los resultados.
Requisitos asociados:	RF03, RF05, RNF04
Flujo principal:	1. AppPiVal detecta que el análisis de una aplicación ha fallado. 2. AppPiVal vuelve a lanzar el análisis para esa aplicación.
Excepciones:	- El archivo de log está vacío. - No se puede establecer conexión con MobSF.
Notas:	Este caso de uso activa de forma implícita CU08 - Guardar logs de ejecución y CU09 - Registrar log de errores de ejecución.

Tabla 4.6: Análisis del caso de uso CU06

CU07	Cargar configuración desde archivo YAML `config.yml`.
Descripción:	AppPiVal carga su configuración mediante el archivo YAML, `config.yml`
Actor:	Usuario
Pre-condiciones:	El archivo `config.yml` está bien formado y en la ruta esperada.
Post-condiciones:	AppPiVal utiliza la nueva configuración.
Requisitos asociados:	RF07, RNF03
Flujo principal:	1. AppPiVal comprueba que el archivo tiene formato correcto. 2. AppPiVal lee el archivo al iniciar. 3. AppPiVal confirma que utilizará la configuración cargada. 4. AppPiVal utiliza la nueva configuración.
Excepciones:	- El archivo no existe. - YAML mal formado o claves faltantes.

Tabla 4.7: Análisis del caso de uso CU07

CU08	Registrar log de ejecución.
Descripción:	AppPiVal guarda un log de una operación en el archivo `AppPiVal.log`.
Actor:	Sistema
Pre-condiciones:	-
Post-condiciones:	El log de error queda registrado en el archivo.
Requisitos asociados:	RNF01
Flujo principal:	1. AppPiVal realiza una acción, como subir un archivo, o guardar un reporte pdf. 2. AppPiVal escribe un log con información como tipo de operación, hora y archivo afectado en `AppPiVal.log`.
Excepciones:	- Fallo al escribir en el archivo de log. - Falta de permisos o espacio en disco.

Tabla 4.8: Análisis del caso de uso CU08

CU09	Registrar log de errores de ejecución.
Descripción:	AppPiVal detecta un error durante un análisis (por ejemplo, caída de MobSF o APK corrupta) y lo registra adecuadamente.
Actor:	Sistema
Pre-condiciones:	AppPiVal se ejecuta para analizar varias apps.
Post-condiciones:	El error queda registrado en el log y el análisis se marca como fallido.
Requisitos asociados:	RF05, RNF01, RNF04
Flujo principal:	1. AppPiVal detecta un fallo en la ejecución, como MobSF no accesible o timeout. 2. Se registra el error en los logs con detalles. 3. Se continúa con los análisis restantes.
Excepciones:	- Fallo al escribir en el archivo de log. - Falta de permisos o espacio en disco. - Si se activa la flag `-v/--verbose`, los logs se printean tb por pantalla.

Tabla 4.9: Análisis del caso de uso CU09

CU10	Validar estado entorno
Descripción:	AppPiVal verifica si el entorno está preparado antes de ejecutar un análisis.
Actor:	Usuario
Precondiciones:	Docker debe estar instalado y activo.
Postcondiciones:	Si el entorno no está listo, se notifica al usuario con un mensaje de error detallado y se detiene la ejecución.
Requisitos asociados:	RF06, RF11
Flujo principal:	1. El usuario ejecuta AppPiVal con el parámetro `--check`. 2. AppPiVal comprueba contenedores, puertos y configuraciones. 3. Se muestra el informe del estado del entorno.
Excepciones:	- Docker no está instalado. - Algún contenedor necesario no está en estado healthy.

Tabla 4.10: Análisis del caso de uso CU10

CU11	Desplegar entorno de análisis
Descripción:	El usuario despliega los servicios MobSF, AVD y DjangoQ usando docker-compose para preparar el entorno de análisis.
Actor:	Usuario
Precondiciones:	Docker y docker-compose instalados. Red configurada.
Postcondiciones:	El entorno está activo y listo para usarse desde AppPiVal.
Requisitos asociados:	RNF05, RNF06
Flujo principal:	1. El usuario ejecuta docker-compose up. 2. Se levantan los tres servicios. 3. El sistema realiza healthchecks para comprobar disponibilidad.
Excepciones:	- Docker no está instalado. - Algún contenedor necesario no está en estado healthy.

Tabla 4.11: Análisis del caso de uso CU11

4.3. Modelo de datos

En esta sección se describe el modelo de almacenamiento adoptado por el sistema propuesto. El diseño del modelo de datos está condicionado por dos factores principales: por un lado, el funcionamiento interno de MobSF y su arquitectura de persistencia; por otro, las decisiones de diseño adoptadas en la herramienta AppPiVal para la gestión y conservación de los resultados obtenidos durante los análisis.

4.3.1. Persistencia de datos en MobSF

MobSF incorpora un mecanismo interno de almacenamiento que le permite conservar el estado de los análisis realizados, así como sus resultados, mediante una base de datos relacional SQLite. Esta base de datos se utiliza para registrar información sobre cada aplicación analizada, incluyendo identificadores, hashes, tipos de análisis realizados y metadatos asociados.

En el contexto de este trabajo, MobSF se ha desplegado dentro de un contenedor Docker. Dado que el sistema de ficheros de un contenedor es efímero por defecto, se ha montado un volumen persistente externo sobre el directorio interno de datos de MobSF. Este volumen está vinculado a una ruta del sistema anfitrión mediante la configuración especificada en el archivo `docker-compose.yml` 6.2.1. De esta forma, se garantiza la permanencia de los datos incluso tras la detención o reinicio del contenedor.

Esta decisión asegura que la base de datos `db.sqlite3`, así como otros archivos relevantes generados por MobSF (como informes PDF, archivos temporales o resultados intermedios), permanezcan disponibles entre sesiones, y puedan ser accedidos por AppPiVal o por otros procesos auxiliares en el entorno.

4.3.2. Modelo de almacenamiento en AppPiVal

AppPiVal adopta un enfoque semiestructurado para la gestión de sus datos de salida, priorizando la flexibilidad y la facilidad de acceso a los resultados. El objetivo de este modelo es organizar eficientemente la información generada, facilitando su consulta, comparación y análisis posterior.

Para una mejor comprensión de las entidades centrales del sistema y sus interrelaciones, se presenta a continuación un modelo de datos conceptual, que describe la relación fundamental entre las **Aplicaciones** procesadas y los **Resultados** obtenidos, tal y como se ilustra en la figura 4.2.

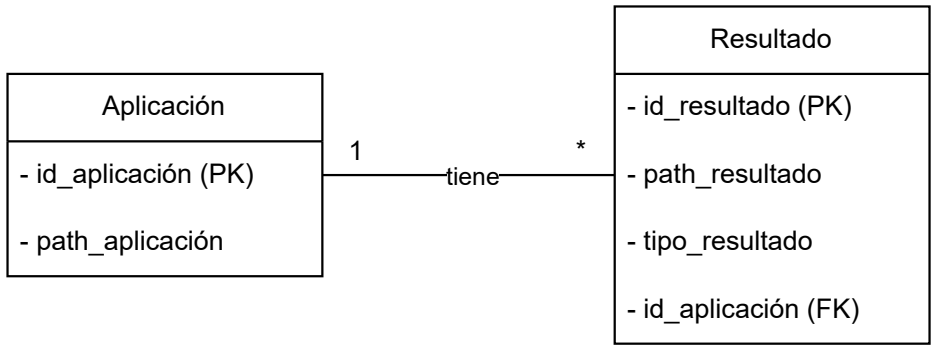

Figura 4.2: Modelo de datos de AppPiVal

Este modelo simplificado establece una relación **uno a muchos** (1:N):

- Una aplicación puede generar cero o muchos resultados de análisis.
- Cada Resultado individual siempre está asociado a una única aplicación de origen.

En este modelo conceptual:

- Cada `Aplicación` se identifica unívocamente por su `id_aplicación` (considerada su clave primaria[2]), que normalmente coincide con el nombre del archivo de la aplicación. Posee además un atributo `path_aplicación` que indica el directorio donde se encuentra.
- Cada `Resultado` tiene su propio `id_resultado` (clave primaria), junto con atributos como `path_resultado` (su ubicación de almacenamiento) y `tipo_resultado` (ej., JSON, PDF). Además, incluye un `id_aplicación` como clave foránea[3], que establece la relación con la `Aplicación` de la que se derivó.

La entidad aplicación representa el archivo bajo análisis y la entidad resultado engloba los diferentes informes y datos obtenidos de dicho análisis. Aunque AppPiVal no utiliza una base de datos relacional interna dedicada para estas dos entidades, esta representación conceptual es clave para entender la lógica de vinculación y organización de la información.

La persistencia real de los datos de AppPiVal se basa en una estructura de sistema de archivos organizada lógicamente. Esta organización física de directorios y archivos es fundamental para el funcionamiento de AppPiVal y se representa en la figura 4.3. A continuación, se detalla esta estructura:

[2]Primary Key o PK
[3]Foreign Key o FK

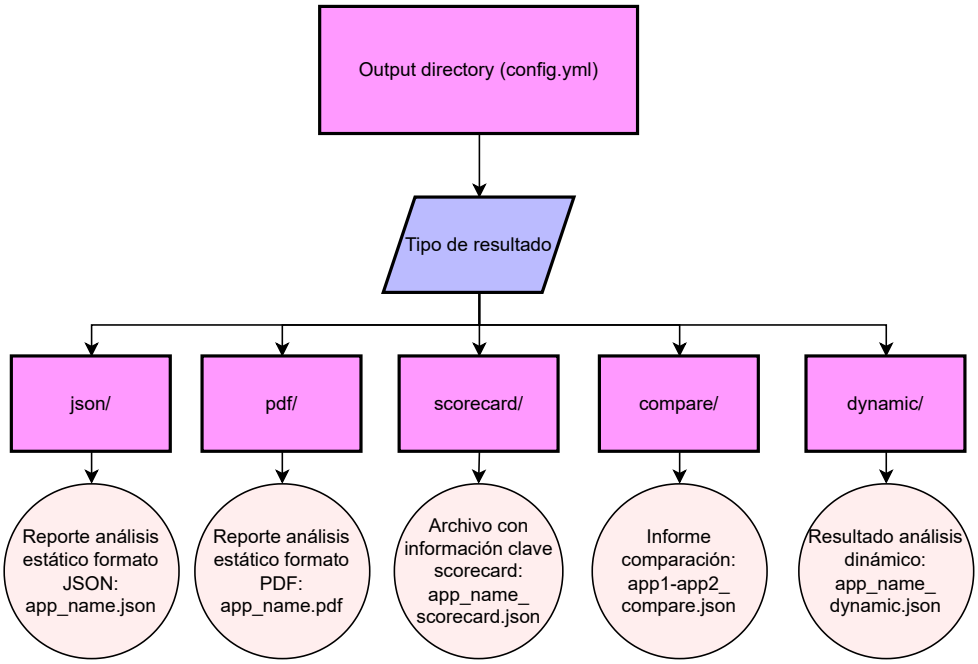

Figura 4.3: Organización lógica de directorios de AppPiVal

- El directorio base para el almacenamiento de todos los resultados es configurable a través del campo `output_directory` en el archivo de configuración `config.yml`.
- Dentro de este directorio raíz, AppPiVal organiza los datos creando subdirectorios específicos para cada *tipo de resultado* generado, lo que permite una categorización clara y un acceso eficiente:
  - `json/`: Contiene los informes JSON detallados, generados y devueltos directamente por MobSF.
  - `scorecard/`: Incluye los archivos `scorecard.json`, que son extracciones estructuradas y resumidas de información clave para una evaluación rápida.
  - `pdf/`: Almacena los informes completos en formato PDF, descargados desde la interfaz web de MobSF.
  - `dynamic/`: Contiene los resultados y artefactos específicos obtenidos de los análisis dinámicos.
  - `compare/`: Almacena los informes JSON resultantes de las operaciones de comparación realizadas entre dos aplicaciones.
- Cada archivo de resultado se nombra utilizando un identificador único de la aplicación analizada (normalmente derivado de su nombre original) y el tipo de resultado en los casos scorecard, compare, dynamic, seguido de la extensión de archivo correspondiente, lo que permite vincular unívocamente el resultado a su aplicación.

Este enfoque, centrado en una organización jerárquica de archivos, permite a AppPiVal gestionar eficazmente la gran variedad de formatos de salida de los análisis. La facilidad de manipulación de archivos y la clara estructura de directorios facilitan la consulta directa y la integración con otras herramientas. Además, la modularización de la lógica de guardado en el código fuente asegura una fácil adaptabilidad del sistema para futuras integraciones con otras soluciones de almacenamiento o bases de datos documentales, si fuera necesario.

Capítulo 5

Diseño

5.1. Tecnologías utilizadas

El lenguaje elegido para la construcción de AppPiVal es Python, debido a que en los últimos años se ha convertido en uno de los lenguajes más utilizados [5]. Además, es ampliamente recomendado para tareas de scripting y automatización [6], así como para programación web y desarrollo de herramientas para redes [7]. Otro factor clave es que parte de MobSF está desarrollado con Django, un framework de Python, lo que facilita la integración con su API.

5.1.1. Concurrencia y paralelismo en Python

Como se mencionó en el apartado 1.1, MobSF ofrece una API que permite su uso desde software externo. Para aprovechar todo su potencial en el análisis masivo de aplicaciones, es necesario realizar múltiples peticiones de forma simultánea, lo que requiere una estrategia de concurrencia eficiente.

Python ofrece tres enfoques principales para la ejecución concurrente y/o paralela:

- **Threading (hilos)**:
  Permite la ejecución de múltiples hilos dentro de un mismo proceso. Es útil para tareas que requieren concurrencia sin necesidad de procesos independientes. Sin embargo, en Python su uso está limitado por el Global Interpreter Lock (GIL), lo que impide la ejecución real en paralelo de múltiples hilos.
- **Multiprocessing (procesos)**:
  Ejecuta múltiples procesos independientes, cada uno con su propio intérprete de Python. Esto evita las restricciones del GIL y permite aprovechar múltiples núcleos de CPU. Es adecuado para tareas computacionalmente intensivas.

- **Multitasking con Asyncio**[1]:
  Basado en un event loop, permite manejar múltiples tareas de manera asíncrona sin necesidad de bloquear la ejecución del programa.

El problema del GIL en Python

Uno de los mayores inconvenientes de Python en términos de concurrencia es la presencia del Global Interpreter Lock (GIL) [8]. Este mecanismo impide que más de un hilo ejecute código Python a la vez dentro de un mismo proceso. El GIL existe porque Python es un lenguaje interpretado y su intérprete no está diseñado para ejecución paralela en múltiples hilos de CPU.

Esta limitación hace que el enfoque de threading sea ineficiente para tareas que requieren alto rendimiento en paralelo. Aunque los hilos pueden ejecutarse de manera concurrente, el GIL los obliga a turnarse, impidiendo un verdadero paralelismo en tareas de CPU intensivo.

Por otro lado, el GIL es una restricción por proceso, no por sistema. Es decir, si se ejecutan múltiples procesos independientes, cada uno con su propio intérprete de Python, el GIL no es un problema. Por tanto, el uso de multiprocessing resulta viable cuando se requiere paralelismo real.

Elección de Multitasking con Asyncio

Dado que la ejecución de AppPiVal está dominada por operaciones de *I/O intensivo*(llamadas HTTP a MobSF y almacenamiento de datos), se opta por multitasking con Asyncio en lugar de threading o multiprocessing.

En resumen:

- Threading no es viable debido a la limitación del GIL, que restringe la ejecución concurrente real en Python.
- Multiprocessing introduce una sobrecarga innecesaria, ya que el problema no es el uso de CPU, sino la eficiencia en operaciones de entrada/salida.
- *Asyncio* permite la concurrencia eficiente sin bloquear la ejecución, ya que está diseñado para manejar muchas tareas de I/O de forma asíncrona.

Al utilizar *asyncio* se pueden lanzar múltiples peticiones en paralelo sin bloquear la ejecución del programa, optimizando el rendimiento en el procesamiento masivo de aplicaciones con MobSF.

[1] https://docs.python.org/3/library/asyncio.html

5.1.2. Librerías asíncronas utilizadas

La concurrencia en AppPiVal se basa en el uso de la librería *asyncio*, que permite la ejecución de tareas de forma asíncrona. Como se ha comentado anteriormente, se utilizará MobSF a través de su API con peticiones http. Sin embargo, no podemos utilizar *request*, librería por defecto en python para peticiones web, ni las escrituras normales de ficheros, ya que sus operaciones son síncronas. Por ello, se han utilizado a mayores las siguientes librerías:

aiohttp

aiohttp[2] es una librería asíncrona para realizar peticiones HTTP basada en *asyncio*. Su elección responde a varias razones:

- **Compatibilidad con Asyncio:** Diseñada específicamente para el modelo de ejecución de *asyncio*, lo que permite un manejo eficiente de múltiples peticiones HTTP concurrentes.
- **Bajo consumo de recursos:** Utiliza un event loop en lugar de múltiples hilos, reduciendo la sobrecarga de memoria y CPU.
- **Soporte para sesiones persistentes:** Permite reutilizar conexiones HTTP con `ClientSession`, optimizando el tiempo de respuesta en peticiones repetitivas a la API de MobSF.

aiofiles

aiofiles[3] es una librería asíncrona para operaciones de lectura y escritura de archivos. Su uso en AppPiVal es clave para el almacenamiento de resultados sin bloquear la ejecución del programa. Sus ventajas incluyen:

- **Operaciones de archivo sin bloqueo:** Al igual que `aiohttp` con las peticiones HTTP, *aiofiles* permite manejar la entrada/salida de archivos sin bloquear el event loop de `asyncio`.
- **Optimización en procesamiento masivo:** Facilita la escritura de múltiples resultados de análisis en archivos sin afectar el rendimiento de la aplicación.

5.1.3. Comparación entre aiohttp y httpx

Aunque *httpx* es otra librería popular para realizar peticiones HTTP de forma asíncrona, se ha optado por *aiohttp* en este proyecto por las siguientes razones:

[2] https://docs.aiohttp.org/en/stable/
[3] https://pypi.org/project/aiofiles/

- **Mejor rendimiento en tareas asíncronas puras:** *aiohttp* ha sido diseñado desde cero para *asyncio*, mientras que *httpx* es una adaptación de *requests* con soporte opcional para *asyncio*.
- **Mayor control sobre conexiones y sesiones:** *aiohttp* ofrece un control más detallado sobre la reutilización de conexiones y la gestión de sesiones HTTP, optimizando el uso de recursos.
- **Menor sobrecarga:** En pruebas comparativas [9], *aiohttp* suele tener menor consumo de memoria en aplicaciones que requieren un alto volumen de peticiones concurrentes.

En la figura 5.1 se muestra una tabla incluida en el repositorio de prueba comparativas [9], donde se comparan las librerías HTTPX, Requests y AIOHTTP, ampliamente utilizadas para realizar peticiones HTTP en Python. La tabla recoge de forma estructurada las funcionalidades más relevantes para desarrolladores, incluyendo compatibilidad con asincronía, manejo de cookies, soporte para HTTP/2, autenticación, entre otros aspectos.

HTTPX destaca por su equilibrio entre capacidades síncronas y asíncronas, soporte para HTTP/2 y una amplia gama de funcionalidades integradas. Requests, aunque sigue siendo muy popular por su simplicidad y compatibilidad con operaciones síncronas, carece de soporte nativo para asincronía y HTTP/2. Por su parte, AIOHTTP se posiciona como la opción más eficiente en términos de rendimiento, aunque limita su uso a entornos completamente asíncronos y no proporciona algunas comodidades como la decodificación automática de JSON.

Esta tabla ayuda a seleccionar la librería más adecuada según las necesidades específicas del entorno de desarrollo y los requisitos de la aplicación.

Feature	HTTPX	Requests	AIOHTTP
Async compatible	Yes	No	Yes
Sync compatible	Yes	Yes	No
Automatic JSON decoding	Yes	Yes	No
HTTP/2 support	Yes	No	No
Cookies	Yes	Yes	Yes
Redirects	Yes	Yes	Yes
Authentication	Yes	Yes	Yes
Custom headers	Yes	Yes	Yes
Streaming responses	No	No	No
Size	Large	Smaller	Smaller
Performance	Good	Good	Excellent

Figura 5.1: Tabla comparativa librería peticiones web python

Dado que AppPiVal necesita realizar numerosas peticiones a la API de MobSF de manera eficiente, *aiohttp* es la mejor opción para garantizar un rendimiento óptimo.

5.1.4. Otras librerías utilizadas

requests[4], para poder realizar peticiones en aquellas operaciones síncronas, y no tener que depender de *aiohttp*.

PyYAML[5], para poder interactuar fácilmente con los archivos en formato YAML, como el archivo de configuración `config.yml`.

argparser[6], librería para poder leer los argumentos de CLI, así como generar una documentación sobre los argumentos disponibles, accesible con la flag `-h/--help`

logging[7], librería para controlar logs de ejecución.

5.2. Estructura modular de AppPiVal

La herramienta AppPiVal ha sido diseñada con una arquitectura modular en Python, organizada en torno a paquetes diferenciados según funcionalidad. Esta estructura presente en la figura 5.2 favorece la sostenibilidad, la ampliación futura y la separación de responsabilidades. A continuación se describe cada módulo:

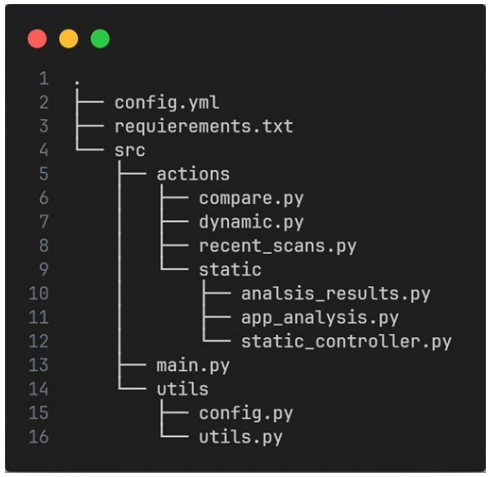

Figura 5.2: Estructura de AppPiVal

[4] https://pypi.org/project/requests/
[5] https://pyyaml.org/wiki/PyYAMLDocumentation
[6] https://docs.python.org/es/3/library/argparse.html
[7] https://docs.python.org/3/library/logging.html

- **main.py**: Punto de entrada a la aplicación. Gestiona el menú CLI y la ejecución automática.
- **actions/static/**: Contiene el controlador del análisis estático y su lógica, como subida de aplicaciones, análisis y el tratamiento de resultados.
- **actions/dynamic.py**: Gestiona el análisis dinámico.
- **actions/compare.py**: Permite comparar dos aplicaciones previamente analizadas.
- **actions/recent_scans.py**: Lista escaneos previos para su consulta.
- **utils/utils.py**: Utilidades comunes, como menú interactivo, argumentos CLI o logger.
- **utils/config.py**: Lectura de la configuración de config.yml.

5.2.1. Configuración por archivo yaml

El archivo `config.yml` centraliza la configuración, indicando parámetros de ejecución como número tareas concurrentes o ficheros abiertos simultáneamente, rutas de entrada/salida, url de MobSF y su API key. Esta estrategia permite desacoplar lógica del sistema y configuración contextual, facilitando el despliegue en distintos entornos.

5.2.2. Argumentos

Los argumentos disponibles son los siguientes:

- -h/–help. Muestra información sobre las herramienta y su uso.
- -c/–check. Muestra si MobSF está activo y se puede empezar el análisis
- -d/–debug. Activa el modo debug de los logs, añadiendo a estos información de cada petición que se realiza a MobSF y cada acción que se realiza.
- -i/–interactive. Activa el modo iterativo, mostrando un menú con las opciones disponibles.
- -f/–file. Indica una ruta al archivo a analizar.
- -r/–retry. Indica a MobSF repetir los análisis fallidos.
- -v/–verbose. Activa los logs en pantalla. Esté o no activo, los logs son guardados en el archivo AppPiVal.log en la raíz del proyecto.

Estos son parseados y gestionados por la librería *argparser*

5.3. Flujo de ejecución lógico

AppPiVal puede ejecutarse de forma automática o en modo interactivo. El flujo de ejecución depende del modo de inicio seleccionado por el usuario:

- **Modo automático**: En la figura 5.3 se muestra el flujo de ejecución cuando el usuario ejecuta AppPiVal sin el parámetro `-i/--interactive`. Se inicia directamente el análisis estático múltiple siguiendo los siguientes pasos:
  1. Se inicializa la configuración establecida en el archivo `config.yml`.
  2. Se inicializa el logger[8] y se parsean los argumentos.
  3. Se obtienen las aplicaciones a analizar dentro del directorio establecido en `config.yml`.
  4. Se crea una tarea de asyncio por aplicación a analizar[9], y se lanzan con `asyncio.gather()`
  5. Por cada tarea, se realiza el análisis estático, subiendo la aplicación a MobSF, esperando el análisis hasta que este se complete.
  6. Por último, se descargan los resultados indicados en `config.yml`.
- **Modo interactivo**: En la figura 5.4 se muestra el flujo de ejecución cuando el usuario ejecuta AppPiVal con el parámetro `-i/--interactive`, con los siguientes pasos:
  1. Se inicializa la configuración establecida en el archivo `config.yml`.
  2. Se inicializa el logger y se parsean los argumentos.
  3. Se presenta el menú de opciones en terminal. El usuario puede elegir entre las siguientes acciones:
     - **Análisis estático múltiple:** Se sigue el flujo comentado en la figura 5.3, teniendo en cuenta que ya se han inicializado la configuración logger y parseado los argumentos.
     - **Análisis dinámico:** Se muestran las aplicaciones en pantalla disponibles para este. Una vez el usuario elije una opción, empieza el análisis contra MobSF, guardando el resultado en la ruta establecida en la configuración.
     - **Mostrar escanéos recientes:** Se muestran los escanéos recientes hechos contra la instancias de MobSF, con información relevante de cada aplicación.[10]
     - **Comparar aplicaciones:** Se muestras los escanéos recientes. Tras elegir dos de ellos, se inicia la comparación de las dos aplicaciones contras MobSF, guardando los resultados en la ruta establecida.

  Tras acabar con la acción seleccionada, AppPiVal notifica al usuario, pudiendo volver al menú.

[8] Encargado de guardar e imprimir los logs

[9] Si se ejecuta AppPiVal con el parámetro `-f/--file` estos dos pasos se omiten, creando una única tarea para esta aplicación

[10] Es decir, los escanéos mostrados son aquellos que MobSF tiene en ese momento guardados en su base de datos.

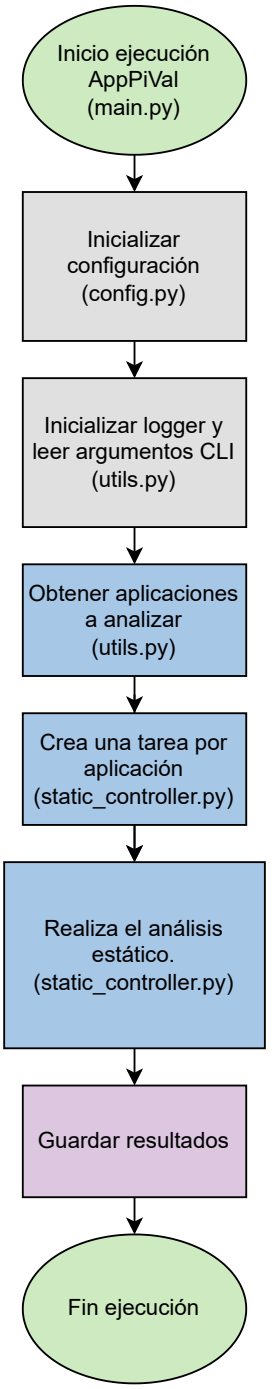

Figura 5.3: Flujo de ejecución de AppPiVal sin flag -i/–interactive

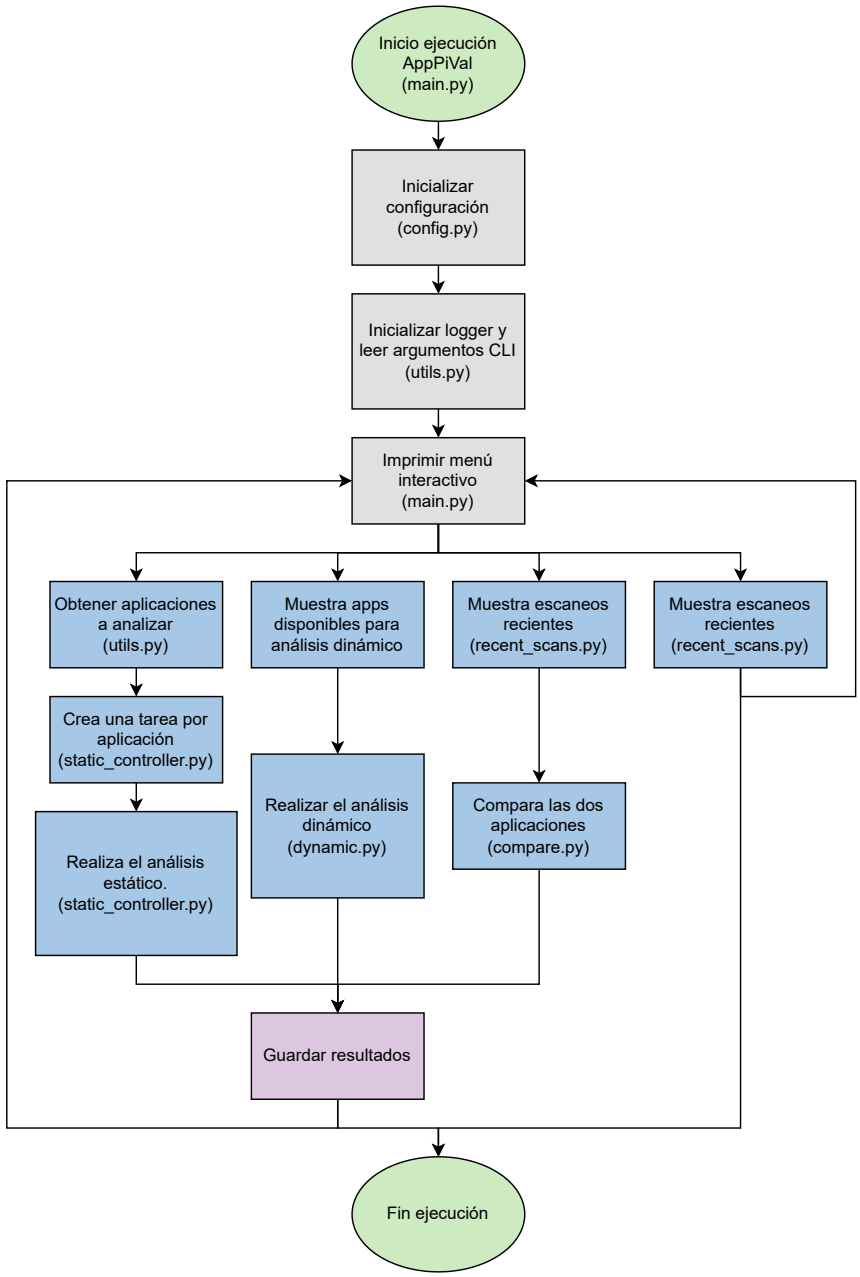

Figura 5.4: Flujo de ejecución de AppPiVal con flag -i/–interactive

5.4. Diseño de concurrencia para análisis estático

El análisis estático múltiple de AppPiVal está diseñado para ejecutarse de manera concurrente, aprovechando la asincronía nativa de Python a través del módulo `asyncio` y las librerías `aiohttp` y `aiofiles`. Esto permite analizar múltiples aplicaciones de forma paralela utilizando una única sesión HTTP persistente.

El modelo de ejecución se basa en la generación de una lista de tareas asíncronas, cada una correspondiente a una aplicación a analizar, que son gestionadas mediante `asyncio.gather`. Cada tarea se encarga de subir el archivo de aplicación a MobSF, empezar el análisis, realizar un polling hasta que el análisis se ha completado, y descargar los resultados en los formatos y la ruta establecidos en el archivo de configuración.

Cabe destacar que existen dos tipos de semáforos con su valor indicado en el archivo `config.yml`.

- **Semáforo *analysis*:** Este semáforo indica cuantas peticiones de inicio de análisis se envían a MobSF, debido a que una carga excesiva de análisis concurrentes puede ralentizar el rendimiento de los análisis. Hay que aclarar que MobSF establece un timeout de 1 hora para cada análisis que gestiona, mismo valor que se usa internamente para el límite al hacer polling.

- **Semáforo *open_files*:** Este semáforo indica cuantos archivos se pueden estar leyendo a la vez, debido a que la API de MobSF necesita que el archivo se envíe como tal, no solo como un descriptor de archivo. Sin este semáforo, una carga excesiva de archivos podría sobrecargar la RAM y hacer que el SO matara el proceso.

5.5. Infraestructura conceptual de despliegue con Docker

Aunque AppPiVal ha sido diseñado para funcionar en cualquier entorno que disponga de Python y acceso a MobSF, se ha propuesto una infraestructura basada en contenedores para facilitar su despliegue y uso en entornos controlados.

Esta infraestructura incluye tres contenedores principales:

- **MobSF**: instancia principal del motor de análisis.

- **Emulador AVD**: dispositivo virtual Android para análisis dinámico.

- **DjangoQ**: sistema de colas para la gestión distribuida de tareas.

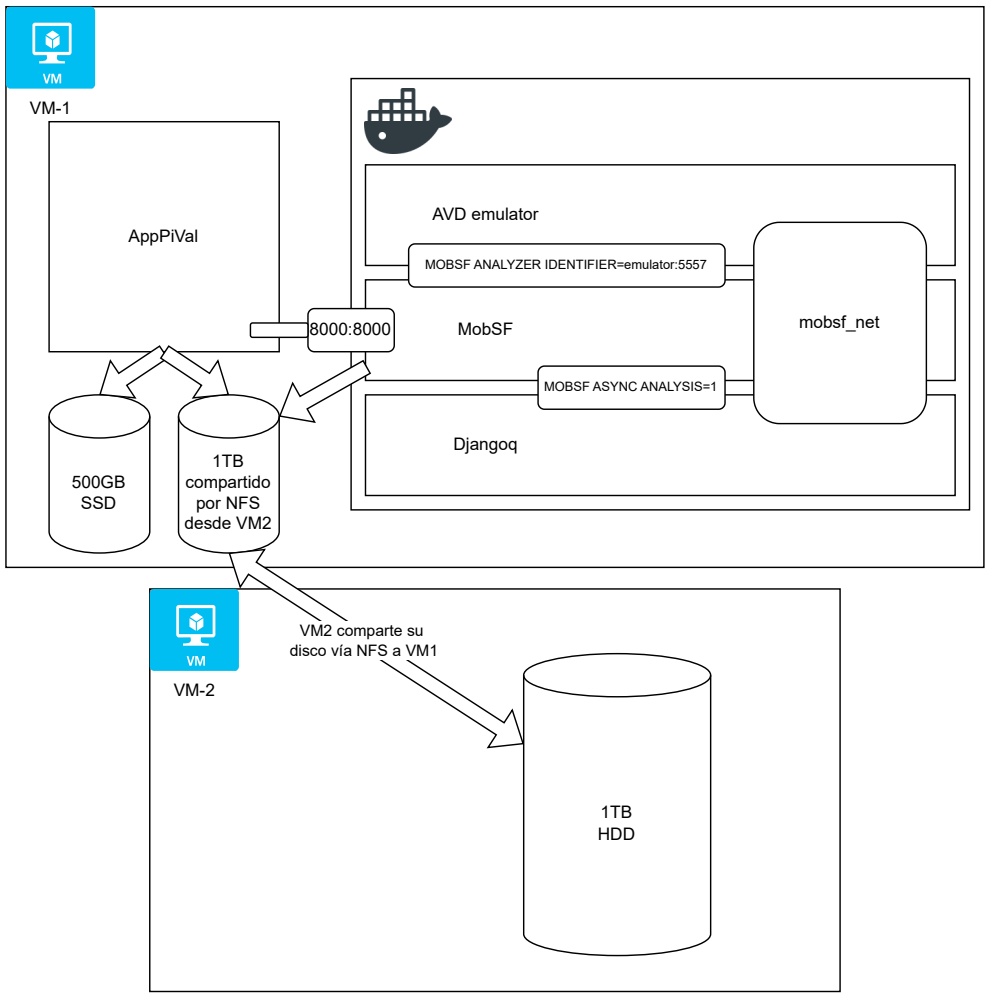

Figura 5.5: Arquitectura AppPiVal

La arquitectura representada en la figura 5.5 representa la infraestructura conceptual de despliegue de AppPiVal utilizando contenedores Docker, utilizando dos máquinas virtuales. Este enfoque facilita la portabilidad, escalabilidad y control del entorno de análisis. La arquitectura se compone de los siguientes elementos:

- **Máquina Virtual 1:** Es el entorno anfitrión donde se ejecuta AppPiVal y se despliega toda la infraestructura dockerizada.
- **Contenedores Docker:** Dentro de la primera máquina virtual se definen tres contenedores principales, conectados entre sí a través de una red Docker llamada mobsf_net:

    - **MobSF:** Es la instancia principal del motor de análisis estático y dinámico. Expone el puerto 8000:8000 para su acceso. Está configurado con la variable de entorno MOBSF_ASYNC_ANALYSIS=1, que permite la ejecución de análisis de forma asíncrona gracias al conteneodr DjangoQ.
    - **AVD Emulator:** Contenedor que simula un dispositivo Android para realizar análisis dinámicos de aplicaciones. Se comunica con MobSF mediante el identificador que se indica a MobSF mediante MOBSF_ANALYZER_IDENTIFIER=emulator:5557.
    - **DjangoQ:** Sistema de colas encargado de gestionar tareas distribuidas de análisis de forma asíncrona. Está integrado con MobSF para automatizar la ejecución de tareas.

- **Almacenamiento local:** La VM dispone de un volumen de 500GB, usado para el sistema operativo, y resultados parciales o volátiles.
- **Máquina virtual 2:** Incluye un volumen externo de 1 TB que comparte a la primera vía NFS, lo que permite obtener una gran capacidad para resultados, y poder hacer permanente los datos del contenedor MobSF.

En conjunto, esta arquitectura modular y dockerizada permite ejecutar AppPiVal en entornos controlados con alta disponibilidad, facilitando el análisis masivo, automatizado y con resultados permanentes de aplicaciones móviles.

La arquitectura completa, como la configuración NFS, o la configuración docker mediante un archivo `docker-compose.yml`, se desarrollará en el capítulo 6.

Capítulo 6

Implementación

El presente capítulo describe el proceso de implementación del entorno de pruebas diseñado para automatizar análisis de seguridad y privacidad sobre aplicaciones móviles. Dicha implementación se apoya en una arquitectura distribuida compuesta por dos máquinas virtuales configuradas para trabajar de forma conjunta y eficiente.

A continuación, se detalla la instalación y configuración del sistema de almacenamiento compartido, seguido del despliegue de los servicios clave mediante Docker y Docker Compose: el emulador Android, la instancia principal de MobSF y el clúster de tareas DjangoQ. Cada sección incluye ejemplos de código representativos, las dificultades técnicas encontradas y las soluciones adoptadas durante el desarrollo. Esta implementación constituye la base funcional sobre la que se ha construido la herramienta AppPiVal.

6.1. Instalación y configuración NFS

Para satisfacer los requisitos de almacenamiento del entorno de pruebas, se ha optado por una arquitectura distribuida que combina dos máquinas virtuales con características complementarias. La Máquina virtual 1 dispone de un disco SSD de 500GB, ofreciendo una alta velocidad de lectura y escritura, fundamental para ejecutar análisis dinámicos de forma ágil. Por otro lado, la Máquina virtual 2 cuenta con un disco HDD de 1TB, lo que aporta una gran capacidad de almacenamiento para guardar resultados de análisis y datos temporales de forma persistente.

Dado que MobSF requiere tanto velocidad como espacio en disco, se ha decidido compartir el almacenamiento disponible en la máquina de 1TB con la de 250GB mediante un sistema de archivos en red (NFS). De este modo, se maximiza el aprovechamiento de ambos recursos, delegando la capacidad al HDD y manteniendo la ejecución del análisis en la máquina más rápida.

A continuación, se describen las características básicas de ambas máquinas mediante las

salidas de los comandos `df -h` (información de almacenamiento) e `ip a` (información de red), esenciales para configurar correctamente el servicio NFS.

6.1.1. Máquina virtual 1

En la figura 6.1 se muestra la salida del comando `df -h`, listando los sistemas de archivos montados en esta máquina. Como podemos ver, el sistema raíz está montado sobre */dev/sda2*, ofreciendo casi los 500GB de espacio de esta máquina virtual.

```
usuario@virtual:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           1,6G  1,1M  1,6G   1% /run
/dev/sda2       473G  4,3G  449G   1% /
tmpfs           7,9G     0  7,9G   0% /dev/shm
tmpfs           5,0M     0  5,0M   0% /run/lock
tmpfs           1,6G     0  1,6G   0% /run/user/1000
```

Figura 6.1: Almacenamiento Máquina virtual 2

En la figura 6.2 podemos ver la salida del comando `ip a` de la máquina virtual 1, útil para conocer la dirección IP de esta máquina, para utilizarla más tarde en la configuración del servicio NFS. Se observa que la interfaz de red activa es *ens18*, con una dirección IP privada del rango 10.0.20.1/18 y máscara de red correspondiente a 255.255.192.0. Se aprecia que la dirección es asignada dinámicamente y que la interfaz se encuentra en estado activo (state UP).

```
usuario@virtual:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
    link/ether 08:00:27:09:20:01 brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    inet 10.0.20.1/18 brd 10.0.63.255 scope global dynamic ens18
       valid_lft 1437sec preferred_lft 1437sec
```

Figura 6.2: Configuración de red Máquina virtual 1

6.1.2. Máquina virtual 2

En la figura 6.3 se muestra la salida del comando `df -h`, listando los sistemas de archivos montados en esta máquina. Como podemos ver, tenemos montado en `/data` el sistema de archivos */dev/vdb1*, disco de 1TB de capacidad que utilizaremos en el servicio NFS.

```
usuario@virtual:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           392M 1016K  391M   1% /run
/dev/vda2       9,8G  4,0G  5,4G  43% /
tmpfs           2,0G     0  2,0G   0% /dev/shm
tmpfs           5,0M     0  5,0M   0% /run/lock
/dev/vdb1       1,0T   50G  974G   5% /data
tmpfs           392M     0  392M   0% /run/user/1000
```

Figura 6.3: Almacenamiento Máquina virtual 2

En la figura 6.4 podemos ver la salida del comando `ip a` ejecutado en la máquina virtual 2, útil para conocer la dirección IP de esta máquina, para utilizarla más tarde en la configuración del servicio NFS. En ella se observa que la interfaz de red activa es ens18, la cual ha recibido dinámicamente la dirección IP 10.0.60.3/18, dentro de una red privada clase A. Esta configuración indica que la máquina está conectada a una red interna con máscara de subred 255.255.192.0, lo que permite hasta 16.382 hosts en la misma subred, una configuración típica en entornos virtualizados para facilitar la escalabilidad y la segmentación lógica del tráfico.

```
usuario@virtual:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
    link/ether 08:00:27:09:60:03 brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    inet 10.0.60.3/18 brd 10.0.63.255 scope global dynamic ens18
       valid_lft 1121sec preferred_lft 1121sec
```

Figura 6.4: Configuración de red Máquina virtual 2

6.1.3. Configuración en la máquina servidora

Vamos a usar la *máquina virtual 2* como servidor NFS, ya que dispone de 1TB de almacenamiento, el cual está montado sobre `/data` y será el sistema de ficheros que compartiremos

a la *máquina virtual 1* mediante NFS. Para ello seguimos los siguientes pasos

1. Instalar el servidor NFS ejecutando:

   ```
   $ sudo apt update
   $ sudo apt install nfs-kernel-server
   ```

2. Configurar `/etc/exports`

   `$ sudo vim /etc/exports` y agregar la siguiente línea
   `/data 10.0.20.1(rw,sync,root_squash,subtree_check,fsid=0)`

   - rw → Queremos tanto leer como escribir
   - sync → Queremos que los cambios se apliquen al momento, lo que previene corrupción
   - root_squash → No permite que el root del cliente actúe como root en el servidor.
   - subtree_check → Verifica que el path esté dentro de la carpeta exportada.
   - fsid=0 → Trata a /data como raiz del path para la exportación.

3. Aplicar cambios ejecutando:

   ```
   $ sudo exportfs -ra
   $ sudo systemctl restart nfs-kernel-server
   ```

6.1.4. Configuración en la máquina cliente

Una vez tenemos la configuración del servidor lista, podemos montar este sistema de archivos que comparte con nosotros siguiendo los siguientes pasos:

1. Instalar soporte NFS ejecutando:

   ```
   $ sudo apt update
   $ sudo apt install nfs-common
   ```

2. Crear punto de montaje ejecutando:

   ```
   $ sudo mkdir -p /mnt/AppPiValData
   ```

3. Montar manualmente ejecutando:

   ```
   $ sudo mount 10.0.60.3:/ /mnt/AppPiValData
   ```

4. Montaje automático al arrancar

   `$ sudo nano /etc/fstab` y agregamos la siguiente línea
   `10.0.60.3:/ /mnt/AppPiValData nfs defaults 0 0`

Como vemos, al montar tanto manualmente o configurando el archivo `/etc/fstab`, indicamos la carpeta `/` de la máquina servidor, no la carpeta `/data`. Esto se debe a la opción *fsid=0* que activamos en `/etc/exports`.

6.1.5. Seguridad del Servidor NFS

La seguridad en un entorno NFS (Network File System) es crucial, ya que permite compartir sistemas de archivos a través de una red, lo que inherentemente introduce vectores de riesgo si no se configura correctamente. Asegurar un servidor NFS implica una combinación de configuraciones a nivel de exportación, control de acceso a nivel de red y gestión adecuada de permisos en el sistema de archivos subyacente.

Para asegurar esta seguridad se han tomado las siguientes medidas:

- El archivo `/etc/exports` es la primera línea de defensa para controlar quién puede acceder a qué recursos. Por ello, se ha compartido la carpeta /data solo a la ip del cliente.
- Activar la opción `root_squash`, haciendo que el usuario `root` del cliente obtenga un usuario sin privilegios dentro del servidor. Para que **usuarios no privilegiados** puedan escribir dentro del servidor existen dos alternativas:
  - El UID y el GID del usuario que necesita acceder a los recursos sea consistente entre el cliente y servidor.
  - Cambiando la propiedad de la carpeta al usuario *nobody:nogroup* (lo que permite al usuario mapeado por `root_squash` escribir si los permisos lo permiten).

Como se verá más adelante en el diseño del docker de MobSF, se ha mapeado los volúmenes para hacer que sus datos sean persistentes. Para aislar estos, se ha utilizado la primera opción como protección haciendo que una carpeta destino `/mnt/AppPiValData/MobSF` sea propiedad de 9901:9901, UID y GID que utiliza el user de MobSF dentro de su contenedor.

6.2. Despliegue con Docker Compose

Para implementar el entorno descrito en el diseño, se ha utilizado Docker Compose como herramienta de orquestación, facilitando la creación, despliegue y conexión de los distintos servicios.

6.2.1. Archivo docker-compose.yml

El fichero `docker-compose.yml` define los servicios, redes, volúmenes, variables de entorno y dependencias entre los contenedores. A continuación, se muestra su estructura principal:

```
services:
  emulator:
    build: ./emulator
    container_name: emulator
    privileged: true
    restart: unless-stopped
    networks:
      - mobsf_net
    healthcheck:
      test: ["CMD", "bash", "-c", "adb -s emulator-5554 shell getprop sys.
          boot_completed | grep -q 1"]
      interval: 15s
      timeout: 5s
      retries: 10
      start_period: 180s

  mobsf:
    image: opensecurity/mobile-security-framework-mobsf:latest
    container_name: mobsf
    depends_on:
      emulator:
        condition: service_healthy
    ports:
      - "8000:8000"
    environment:
      - MOBSF_ASYNC_ANALYSIS=1
      - MOBSF_ANALYZER_IDENTIFIER=emulator:5557
    volumes:
      - /mnt/AppPiValData/MobSF:/home/mobsf/.MobSF
    restart: unless-stopped
    networks:
      - mobsf_net
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000"]
      interval: 15s
      timeout: 5s
      retries: 10

  djangoq:
    image: opensecurity/mobile-security-framework-mobsf:latest
    container_name: djangoq
    depends_on:
      - mobsf
    command: scripts/qcluster.sh
    volumes:
      - /mnt/AppPiValData/MobSF:/home/mobsf/.MobSF
    restart: unless-stopped
    networks:
      - mobsf_net
    healthcheck:
      disable: true
```

```
networks :
  mobsf_net :
    driver :  bridge
```

Listing 6.1: archivo docker-compose.yml

Configuración de Redes

Se define una red `mobsf_net` de tipo `bridge`. Esta red interna permite que todos los contenedores de la aplicación (`emulator`, `mobsf` y `djangoq`) se comuniquen entre sí de forma aislada y segura, sin exponer sus puertos directamente al host a menos que se especifique explícitamente. Esto asegura una comunicación eficiente y privada entre los componentes.

Contenedor emulador AVD

La imagen de este contenedor se construye de forma personalizada utilizando el `Dockerfile` 6.2.1. Este `Dockerfile` incluye las dependencias necesarias para ejecutar el emulador de Android y el servidor `adb` (Android Debug Bridge) dentro del contenedor. El proceso de inicio del emulador dentro del contenedor se gestiona a través de un script `entrypoint.sh` 6.2.1, que configura y lanza la instancia del AVD.

```
FROM ubuntu:22.04

ENV DEBIAN_FRONTEND=noninteractive
ENV ANDROID_SDK_ROOT=/opt/android-sdk
ENV PATH="${ANDROID_SDK_ROOT}/cmdline-tools/latest/bin:${ANDROID_SDK_ROOT}/
    platform-tools:${ANDROID_SDK_ROOT}/emulator:${PATH}"

RUN apt-get update && apt-get install -y \
    git unzip curl wget openjdk-17-jdk \
    libgl1-mesa-dev libpulse-dev libx11-dev libxcb1 \
    socat net-tools adb qemu-kvm vim\
    && rm -rf /var/lib/apt/lists/*

RUN mkdir -p ${ANDROID_SDK_ROOT}/cmdline-tools \
 && cd ${ANDROID_SDK_ROOT}/cmdline-tools \
 && wget https://dl.google.com/android/repository/commandlinetools-linux
     -10406996_latest.zip -O sdk.zip \
 && unzip sdk.zip -d tmp \
 && mv tmp/cmdline-tools ${ANDROID_SDK_ROOT}/cmdline-tools/latest \
 && rm -rf tmp sdk.zip



RUN yes | sdkmanager --sdk_root=${ANDROID_SDK_ROOT} --licenses

RUN sdkmanager --sdk_root=${ANDROID_SDK_ROOT} \
    "platform-tools" "emulator" "platforms;android-30" "system-images;android
        -30;default;x86_64"

COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
```

```
29
30 CMD ["/entrypoint.sh"]
```

Listing 6.2: Archivo Dockerfile del emulator android

```
1
2  AVD_NAME=mobsf_emulator
3  AVD_PATH="/root/.android/avd/${AVD_NAME}.avd"
4  adb start-server
5
6  if [ ! -d "$AVD_PATH" ]; then
7      echo "Creando AVD por primera vez..."
8      echo "no" | avdmanager create avd -n ${AVD_NAME} -k "system-images;android
           -30;default;x86_64" --device "pixel"
9  fi
10
11 emulator -avd ${AVD_NAME} -no-window -gpu swiftshader_indirect -no-audio -no-
       boot-anim -writable-system -no-snapshot -port 5554 &
12
13 adb wait-for-device
14
15 adb root
16 adb shell avbctl disable-verification
17 adb disable-verity
18 adb reboot
19
20 while true; do
21     if adb devices | grep -w "device" | grep -q "emulator"; then
22         echo "Emulador disponible"
23         break
24     fi
25     echo "Esperando ADB..."
26     sleep 5
27 done
28
29 adb root
30 adb remount
31 adb shell "su 0 mount -o rw,remount /system"
32
33 socat TCP-LISTEN:5557,fork TCP:127.0.0.1:5555 &
34
35 tail -f /dev/null
```

Listing 6.3: Archivo entrypoint.sh del emulator android

La configuración del servicio `emulator` en `docker-compose.yml` incluye los siguientes detalles:

- `container_name: emulator`: Asigna un nombre específico al contenedor (`emulator`), facilitando su identificación y referencia en la red Docker.
- `privileged: true`: Esta directiva es crucial para el emulador. Otorga al contenedor permisos extendidos sobre el dispositivo host, lo cual es necesario para que el emulador pueda interactuar correctamente con el kernel de Linux subyacente y para que las funcionalidades de virtualización de hardware sean accesibles. Sin este permiso, el emulador no podría arrancar o funcionar de forma estable.

- `restart: unless-stopped`: Configura la política de reinicio del contenedor. El emulador se reiniciará automáticamente si se detiene por cualquier motivo (ej., un fallo interno), a menos que sea detenido explícitamente por un comando del usuario.
- `networks: - mobsf_net`: Conecta el emulador a la red interna `mobsf_net`, permitiendo su comunicación directa con el contenedor de MobSF.
- `healthcheck`: Define una comprobación de salud rigurosa para asegurar que el emulador está completamente iniciado y operativo antes de que MobSF intente interactuar con él.
  - `test: [ÇMD", "bash", c", .ªdb -s emulator-5554 shell getprop sys.boot_completed | grep -q 1"]`: Ejecuta un comando `adb` dentro del emulador para verificar que el sistema operativo Android ha completado su arranque (el valor de la propiedad `sys.boot_completed` es `1`).
  - `interval: 15s`, `timeout: 5s`, `retries: 10`: Especifican que la comprobación se realiza cada 15 segundos, con un tiempo máximo de 5 segundos, reintentando hasta 10 veces en caso de fallo.
  - `start_period: 180s`: Proporciona un período inicial de gracia de 3 minutos. Durante este tiempo, las comprobaciones de salud fallidas no contarán para el número de reintentos, permitiendo al emulador un tiempo suficiente para arrancar completamente sin que Docker Compose lo marque como no saludable prematuramente.

Contenedor MobSF

Se utiliza la imagen oficial de MobSF (`opensecurity/mobile-security-framework-mobsf`), garantizando el uso de una versión estable y actualizada de la herramienta. Como bien se ha indicado, se expone el puerto 8000 para permitir el acceso a su interfaz web y API desde el host. Se define un volumen persistente en `/mnt/AppPiValData/MobSF`, mapeado a `/home/mobsf/.MobSF` dentro del contenedor. Esto es crucial para asegurar la persistencia de los resultados de análisis, bases de datos y configuraciones de MobSF a través de los reinicios o recreaciones del contenedor.

Además, se configuran las siguientes variables de entorno:

- `MOBSF_ASYNC_ANALYSIS=1`: Habilita el modo de análisis asíncrono en MobSF. Esto es esencial para que AppPiVal pueda enviar múltiples tareas de análisis a una cola y procesarlas concurrentemente o en segundo plano, en lugar de esperar la finalización de cada una.
- `MOBSF_ANALYZER_IDENTIFIER=emulator:5557`: Especifica el emulador que MobSF debe utilizar para realizar los análisis dinámicos. Este identificador corresponde al emulador configurado en el servicio `emulator`.

El servicio `mobsf` también incluye:

- `container_name: mobsf`: Asigna el nombre `mobsf` al contenedor principal de MobSF.
- `depends_on: emulator: condition: service_healthy`: Establece una dependencia crucial. El contenedor `mobsf` no se iniciará hasta que el servicio `emulator` esté en un estado `healthy`. Esto evita que MobSF intente conectarse o utilizar un emulador que aún no está completamente operativo, previniendo errores de inicio.
- `restart: unless-stopped`: Configura la política de reinicio automático del contenedor de MobSF.
- `networks: - mobsf_net`: Conecta MobSF a la red interna `mobsf_net`, permitiendo la comunicación con el emulador y el servicio de cola DjangoQ.
- `healthcheck`: Se ha añadido un healthcheck con `curl` para verificar su disponibilidad antes de lanzar el servicio `djangoq`. Este healthcheck asegura que el servidor web de MobSF está respondiendo en el puerto 8000, con una frecuencia de 15 segundos, un timeout de 5 segundos y hasta 10 reintentos.

Contenedor DjangoQ

Este contenedor es una instancia dedicada de MobSF configurada para ejecutar su sistema de gestión de colas, Django-Q. Tal como se indica en la configuración, el comando principal que se ejecuta al iniciar el contenedor es `python manage.py qcluster`. Este comando lanza el *cluster* de Django-Q, que es responsable de procesar las tareas asíncronas enviadas por el servicio MobSF principal, como los análisis dinámicos o la generación de informes.

El servicio `djangoq` utiliza la misma imagen base de MobSF y se conecta al volumen y red compartidos para acceder a los datos y al servicio MobSF:

- `container_name: djangoq`: Asigna un nombre claro al contenedor de la cola.
- `depends_on: mobsf: condition: service_started`: Su ejecución depende de que el servicio `mobsf` esté activo. Esta directiva asegura que el contenedor de la cola no intente iniciarse antes de que MobSF esté listo para recibir tareas. La condición `service_started` es suficiente aquí, ya que Django-Q solo necesita que el proceso de MobSF esté iniciado, no necesariamente 'healthy' en el sentido de su interfaz web.
- `environment`: Se configuran variables de entorno para MobSF y Django-Q:
  - `MOBSF_HOME=/home/mobsf/.MobSF`: Es crucial que apunte al mismo volumen persistente (`/mnt/AppPiValData/MobSF`) que el servicio `mobsf`. Esto permite que ambos contenedores compartan la misma base de datos SQLite (donde Django-Q gestiona las tareas y MobSF guarda sus resultados) y la configuración.
  - `MOBSF_USE_CELERY_QUEUE=1`: Activa explícitamente la integración de MobSF con su sistema de colas.
- `volumes`: Se montan dos volúmenes:

  - `- /mnt/AppPiValData/MobSF:/home/mobsf/.MobSF`: Asegura el acceso al volumen de datos persistente compartido con el servicio `mobsf`.
  - `- ./MobSF/MobSF/settings.py:/usr/src/MobSF/MobSF/settings.py`: Este volumen permite sobreescribir el archivo de configuración `settings.py` interno de MobSF con una versión local. Esto es útil para personalizar la configuración de Django-Q (ej., broker de cola, ajustes de caché) sin tener que reconstruir la imagen Docker completa de MobSF.

- `restart: unless-stopped`: Configura el reinicio automático del servicio de cola si se detiene inesperadamente.
- `networks: - mobsf_net`: Conecta el servicio `djangoq` a la red interna `mobsf_net`, permitiendo que MobSF le envíe tareas y que ambos componentes se comuniquen.

6.3. Implementación funcional de AppPiVal

El desarrollo de AppPiVal se ha realizado en Python 3.12, siguiendo la estructura modular definida en el diseño. A continuación se detallan las funciones clave implementadas:

Gestión del menú CLI

La interacción por consola se gestiona mediante el menú, ubicado en `utils.py` y `main.py`:

```
def main() -> None:
    """
    Main function that executes the required analysis (Default: standard
        static analysis).
    """
    if args.check:
        if not server_up():
            return

    if args.interactive:
        while True:
            match main_menu():
                case 1:
                    asyncio.run(static_analysis())
                case 2:
                    dynamic_analysis()
                case 3:
                    display_recent_scans(0)
                case 4:
                    compare_apps()
                case 0:
                    print("Exiting...\n")
                    break
                case _:
                    print("Invalid option. Please try again.")
            input("All is done. To go back to menu press [INTRO]\n")
    elif not args.interactive:
```

```
27          if args.file:
28              asyncio.run(static_analysis(Path(args.file)))
29          else:
30              asyncio.run(static_analysis())
```

Listing 6.4: Función main

Función principal de análisis estático

El análisis masivo se lanza mediante la función `main_analysis()`, que gestiona la creación de tareas asíncronas y su ejecución mediante `asyncio.gather`:

```
1  async def static_analysis(app = None) -> None:
2      """
3      Standar async analysis main function, that creates the async tasks for
           each app
4      """
5      if app:
6          logger.info(
7              f"Starting static analysis on app {app}."
8          )
9          try:
10             async with ClientSession() as session:
11                 await main_analysis(app, session)
12
13         except Exception as err:
14             logger.error(f"{err}")
15     else:
16         logger.info(
17             f"Starting static analysis on apps in {config.project.apps_folder}
                    with {config.semaphores.open_files} open files simultaneously
                    and with {config.semaphores.analysis} concurrent analysis."
18         )
19         try:
20             async with ClientSession() as session:
21                 await gather(
22                     *[
23                         main_analysis(app, session)
24                         for app in get_apps(config.project.apps_folder)
25                     ]
26                 )
27
28         except Exception as err:
29             logger.error(f"{err}")
```

Listing 6.5: Función asíncrona de análisis estático

Automatización del análisis dinámico

El análisis dinámico se realiza mediante la función `dynamic_analysis()`, que se conecta con el contenedor AVD y ejecuta:

- Instalación de root_ca
- Configuración del global_proxy
- Tests tls
- Activity Tester
- Descarga de resultados

Comparación y recuperación

Las funciones `compare_apps()` y `show_recent_scans()` permiten visualizar escaneos anteriores y generar informes de comparación sin repetir análisis.

Gestión de errores y logs

Toda la herramienta cuenta con un sistema de registro centralizado. Este se gestiona con la librería `logging`, teniendo siempre la salida en un archivo AppPiVal.log, y pantalla si se activa la flag -v/–verbose:

```
logger.info(
    f" \
    Starting static analysis on apps in {config.project.apps_folder}\n \
    with {config.semaphores.open_files} open files simulteniously and \n \
    with {config.semaphores.analysis} concurrent analysis\n"
)
logger.debug(f"Method api/v1/scan on {app_hash} STARTED")
logger.error(f"Error on {err}")
```

Listing 6.6: Ejemplos logger

Archivo de Configuración `config.yml`

Para ofrecer flexibilidad en la configuración de la ejecución de AppPiVal sin modificar el código fuente, la aplicación utiliza un archivo de configuración externo en formato YAML, denominado `config.yml`. Este archivo permite a los usuarios definir rutas de entrada y salida, ajustar parámetros de concurrencia, configurar la conexión con la API de MobSF y seleccionar los formatos de salida de los resultados.

El formato del archivo `config.yml` es el siguiente:

```
project :
  apps_folder :  ""
  output_folder :  ""

semaphores :
  analysis :
```

```
  open_files :

api :
  endpoint :  ""
  key :  ""

output :
  scorecard :  false
  pdf :  false
  json :  false
```

Listing 6.7: Formato archivo config.yml

A continuación, se describen en detalle cada una de las secciones y sus parámetros:

Sección `project` Esta sección define las rutas de directorios fundamentales para la operación de AppPiVal, especificando dónde encontrar las aplicaciones a analizar y dónde almacenar los resultados.

- `apps_folder`: Cadena de texto. Ruta absoluta o relativa al directorio que contiene los archivos de aplicaciones que AppPiVal debe procesar. Por ejemplo: `/home/usuario/apps_a_analizar`. La aplicación iterará sobre todos los archivos de aplicación encontrados en esta carpeta.
- `output_folder`: Cadena de texto. Ruta absoluta o relativa al directorio donde AppPiVal guardará todos los resultados de los análisis, incluyendo informes, logs y cualquier otro artefacto generado. Por ejemplo: `/home/usuario/resultados_apppival/"`. Los resultados se guardarán como se explicó en la sección 4.3

Sección `semaphores` Esta sección es vital para el control de la concurrencia y la gestión de recursos durante el proceso de análisis, permitiendo ajustar el rendimiento y evitar la sobrecarga del sistema o de MobSF.

- `analysis`: Número entero. Define el número máximo de análisis de MobSF que pueden ejecutarse concurrentemente. Dado que MobSF puede ser intensivo en recursos, este semáforo permite limitar la carga simultánea para evitar cuellos de botella o inestabilidad. Un valor de `5`, por ejemplo, significa que AppPiVal no enviará más de 5 APKs a MobSF para análisis simultáneos en cualquier momento. Este parámetro mitiga el riesgo de sobrecarga de MobSF, como se mencionó en la Tabla 3.4 (Riesgo 9).
- `open_files`: Número entero. Especifica el número máximo de descriptores de archivo que AppPiVal puede mantener abiertos simultáneamente para la lectura de APKs. Esto ayuda a gestionar la memoria y los recursos del sistema operativo, especialmente cuando se procesan grandes volúmenes de aplicaciones. Un valor de `25` significa que no se intentará abrir más de 25 archivos de aplicación a la vez para su procesamiento inicial.

Sección `api` Esta sección contiene los parámetros necesarios para que AppPiVal se conecte y autentique con la API REST de MobSF, permitiendo la comunicación programática para el envío de APKs y la recuperación de resultados.

- `endpoint`: Cadena de texto. La URL base de la API de MobSF. Por ejemplo: `"http://localhost:8000/api/"`. Debe apuntar a la dirección donde el contenedor MobSF expone su API.
- `key`: Cadena de texto. La clave de autenticación (*API Key*) para acceder a la API de MobSF. Esta clave se genera en la configuración de la instancia de MobSF y es necesaria para autorizar las solicitudes de AppPiVal.

Sección `output` Esta sección controla los tipos de informes adicionales que AppPiVal descargará tras los análisis y guardará en el `output_folder` especificado.

- `scorecard`: Booleano (`true` o `false`). Si se establece en `true`, AppPiVal guardará el resultado del análisis, agregando el privacy_score a este json.
- `pdf`: Booleano (`true` o `false`). Si se establece en `true`, AppPiVal descargará el informe en formato PDF para cada análisis completado, utilizando la funcionalidad de MobSF.
- `json`: Booleano (`true` o `false`). Si se establece en `true`, AppPiVal guardará el reporte para cada análisis en formato JSON.

El uso de `config.yml` centraliza la configuración, mejora la portabilidad de la aplicación y permite a los usuarios adaptar AppPiVal a diferentes entornos y necesidades sin modificar el código, contribuyendo a la robustez y facilidad de mantenimiento del sistema.

Capítulo 7

Pruebas

En este capítulo se desarrollan las pruebas correspondientes a los casos de uso 4.2 del sistema AppPiVal.

7.1. Prueba PR01 – Análisis estático

En esta prueba se validará el análisis estático unitario de una aplicación. Para ello, se ejecutará AppPiVal con el parámetro `-f/--file` seguido de una ruta hacia el archivo .apk a analizar. También se activará la flag `-v/--verbose`, para poder ver los logs de ejecución de forma directa. También se descargarán los tres tipos de resultados *scorecard*, *json* y *pdf*.

Caso de uso asociado: CU01 – Ejecutar análisis estático sobre una archivo APK.

Objetivo: Verificar que el sistema realiza correctamente el análisis estático de una APK individual.

Pasos:

1. Ejecutar AppPiVal con la flag `-f app-PR1.apk` y con `-v/--verbose`.
2. Se carga configuración `config.yml`.
3. Se sube el archivo a MobSF.
4. MobSF analiza la aplicación.
5. Una vez analizado, se descargan los resultados.

Resultado esperado: Obtenemos tres archivos correspondientes a los tres tipos de resultados en la ruta establecida en el archivo `config.yml`.

7.1.1. Ejecución

Para esta prueba, contamos con el contenido de `config.yml` mostrado en la figura 7.1. Como estamos ejecutando solo un análisis, indicando el archivo de aplicación por parámetros, no es necesario indicar ningún directorio en `apps_folder`, pero si es necesario indicar donde queremos que AppPiVal deje los resultados. En este caso tampoco nos importa los valores de los semáforos, ya que al solo analizar una aplicación, estos no provocan ningún efecto. En cuanto a los valores de la sección api, tenemos tanto el endpoint de MobSF, que en estas pruebas al estar dockerizado en la misma máquina virtual, podemos acceder por localhost, y la clave que este nos ofrece. En cuanto a los resultados del análisis a descargar, tenemos todos seleccionados.:

```yaml
project:
  apps_folder: ""
  output_folder: "./reports"

semaphores:
  analysis: 5
  open_files: 25

api:
  endpoint: "http://localhost:8000"
  key: "cb754e8ed75dde6797f081c342dcecfd510ce588e6f2452ae34a459b47303a80"

output:
  scorecard: true
  pdf: true
  json: true
```

Figura 7.1: Contenido de `config.yml`

Tras configurar el archivo `config.yml` ejecutamos el análisis ejecutando `python3 main.py -v -f app-PR1.apk`. Una vez ejecutado podemos ver los logs de ejecución en terminal como se muestra en la figura 7.2 gracias al parámetro `-v/--verbose`.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -v -f app-PR1.apk
[INFO] 16:13:32 - Starting static analysis on app app-PR1.apk.
[INFO] 16:13:40 - App app-PR1.apk: UPLOADED
[INFO] 16:13:40 - App app-PR1.apk: Scanning
[INFO] 16:13:52 - App app-PR1.apk: SCANNED SUCCESSFULLY
[INFO] 16:13:55 - App app-PR1.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app-PR1.apk_scorecard.json
[INFO] 16:14:01 - App app-PR1.apk: JSON RESULTS on /home/usuario/AppPiVal/AppPiVal/reports/json/app-PR1.apk.json
[INFO] 16:14:13 - App app-PR1.apk: PDF RESULTS on /home/usuario/AppPiVal/AppPiVal/reports/pdf/app-PR1.apk.pdf
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.2: Ejecución AppPiVal

En la figura 7.3 podemos ver que tenemos guardado en ./reports los 3 tipos de archivos, cada uno en su subdirectorio correspondiente. Podemos comprobarlo ejecutando `tree ../reports`[1].

[1]Hay que tener en cuenta que como el archivo `config.yml` se encuentra en la raiz del proyecto (es decir

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ tree ../reports/
../reports/
├── json
│   └── app-PR1.apk.json
├── pdf
│   └── app-PR1.apk.pdf
└── scorecard
    └── app-PR1.apk_scorecard.json

3 directories, 3 files
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.3: Contenido `../reports/`

Resultado obtenido: Obtenemos tres archivos correspondientes a los tres tipos de resultados en la ruta establecida en el archivo `config.yml`.

7.2. Prueba PR02 – Análisis dinámico desde menú CLI

En esta prueba se validará el análisis dinámico de una aplicación. Para ello, se ejecutará AppPiVal con el parámetro `-i/--interactive`. También se activará la flag `-v/--verbose`, para poder ver los logs de ejecución de forma directa. Una vez con el menú interactivo, elegimos la opción *Dynamic Analysis*.

Caso de uso asociado: CU02 – Ejecutar análisis dinámico sobre una archivo APK.

Objetivo: Validar que el usuario puede acceder al menú CLI y lanzar un análisis dinámico sobre una aplicación previamente analizada.

Pasos:

1. Ejecutar AppPiVal con la flag `-i/--interactive` y con `-v/--verbose`.
2. Seleccionar opción *Dynamic Analysis*.
3. Elegir una aplicación disponible.
4. Esperar y validar que se lanza y completa el análisis dinámico.

Resultado esperado: Obtenemos un reporte del análisis dinámico en la ruta establecida en el archivo `config.yml`

por encima de src/). Si indicamos un directorio con un path relativo, este se resolverá desde la ubicación del proyecto, en el subdirectorio reports, en este caso `/home/usuario/AppPiVal/AppPiVal/reports/`, y como estamos dentro de src, por esto ejecutamos `tree ../reports`

7.2.1. Ejecución

En cuanto a `config.yml`, tenemos la misma configuración que la prueba anterior 7.1. En esta prueba solo nos interesa la carpeta de resultados `output_folder` y los valores de *endpoint* y *key* de la sección api.

Para ejecutar el análisis dinámico, ejecutamos `python3 main.py -v -i`, para obtener el menú interactivo. Como vemos en la figura 7.4, tras obtener el menú interactivo, podemos elegir la opción 2, para realizar un análisis dinámico.

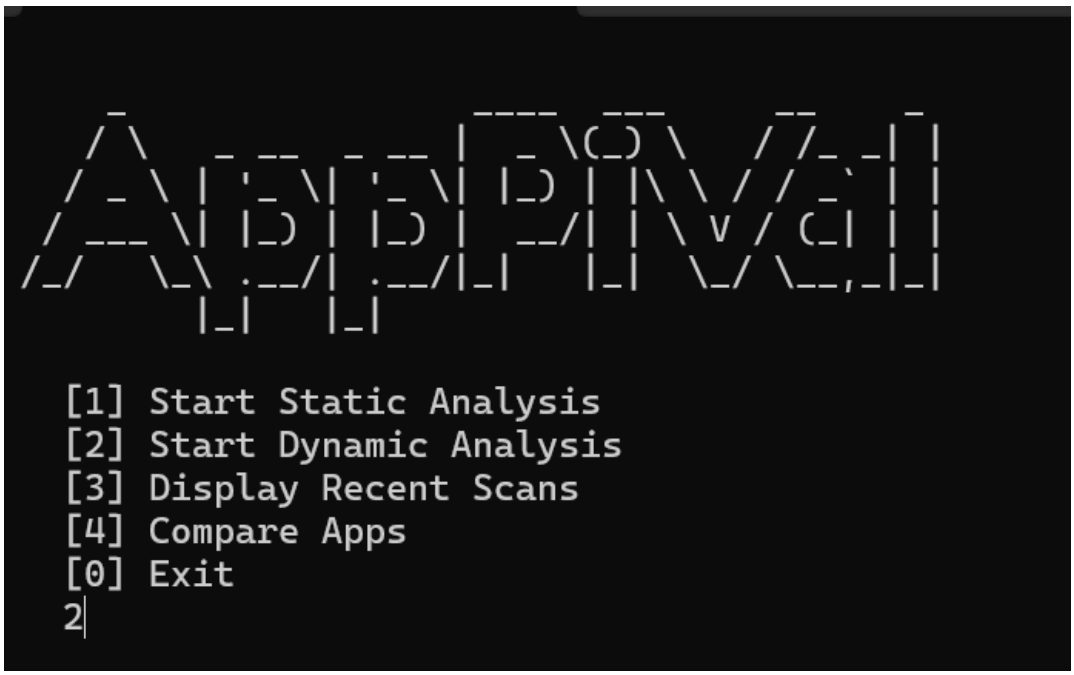

Figura 7.4: Ejecución AppPiVal -v -i

Esto nos mostrará una lista de las aplicaciones disponibles para el análisis dinámico, figura 7.5, y podemos elegir una seleccionando su número ID.

```
  2

Apps prepared to a dynamic analysis:
ID    APP_NAME                                  VERSION_NAME          PACKAGE_NAME
1     Rhythm Hop                                1.0.7                 com.dash.dancing.smash.game.tiles.circles.beat.piano.rhythm.hop
2     Delhi Study Circle                        1.4.44.1              co.thor.blfrv
3     AppLock                                   1.0.6                 com.privacy.applock.plus
4     Seterra                                   2.3.9                 com.seterra
5     Beem It                                   4.0.1+16064           it.beem.app
6     Ster-Kinekor                              5.04.800              nz.co.vista.android.movie.sterkinekor
7     Got-Cha                                   1.5                   com.app.gotyew
8     Finanční kalkulačky                       2.1.6.1               cz.psc.android.financnikalkulacky
9     Missio Dei                                5.16.0                com.subsplash.thechurchapp.s_J5GXD9
10    Email for Hotmail Outlook                 1.4                   com.mail.hotmail.outlook.email
11    Terago                                    33.3.17.4253          uk.co.terago.drivers1
12    MedHub                                    1.3.4                 com.medhub.mobile
13    Proton Mail                               3.0.7                 ch.protonmail.android
14    Fantasy                                   2.6.6                 com.nft.creator.nftartmaker.crypto
15    Tressette                                 1.1.1                 com.WhatWapp.Tressette
16    Invitations                               4.8.4                 com.desygner.invitations
17    Vishnu Puran Full Episode(Hindi)          4.0                   ads.google.deletenow
18    Skype                                     8.36.0.52             com.skype.raider
19    Protect                                   3.2.0.3877            com.lacoon.security.fox
20    1000 English verbs                        1.7                   com.kkk.english_words
21    Murri Pusat                               1.01                  id.co.androjex.apps.android5e4a1c2549875
22    매쓰플랫                                      3.9.1                 iclass.mathflat.parent.student
23    iStudents                                 2.8.3                 com.subsplash.thechurchapp.internationalstudentsinc
24    Binance                                   2.39.2                com.binance.dev
25    All Messages Recovery                     3.0                   com.app.recoverdeletedmesasges.viewdeleted.messagerecovery.restoredeletedmessages
26    Opencho                                   1.0.5                 com.app.Opencho
27    Christmas HD Wallpapers                   1.2                   com.my.my.blockremover

Options:
[Number] Analyze app
[Q] Exit
Select an option: 23
```

Figura 7.5: Lista aplicaciones disponibles

Tras elegir la aplicación, el análisis dinámico empieza, ejecutando varias operaciones como se explicó en la subsección 6.3, obteniendo los logs de ejecución como se muestra en la figura 7.6.

```
[INFO] 16:15:32 - Dynamic analysis tarted on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:15:32 - root_ca installed on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:15:33 - global_proxy set on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:15:33 - tls tests started on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:17:13 - tls tests ended on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:17:13 - Starting Activity tester on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:18:11 - Activity tester ended on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:18:38 - Stopping analysis on 3d8cafc18d6773319ca71618d1e6a1b5
[INFO] 16:18:42 - Results on /home/usuario/AppPiVal/AppPiVal/reports/dynamic/3d8cafc18d6773319ca71618d1e6a1b5_dynamic.json
All is done. To go back to menu press [INTRO]
```

Figura 7.6: Logs ejecución análisis dinámico

Una vez el análisis dinámico termina, podemos ver que se nos guarda el reporte del análisis en el directorio que hemos indicado en `config.yml` y en el subdirectorio `dynamic`. Podemos verificarlo ejecutando `tree ../reports/`, figura 7.7.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ tree ../reports/
../reports/
├── dynamic
│   └── 3d8cafc18d6773319ca71618d1e6a1b5_dynamic.json
├── json
│   └── app-PR1.apk.json
├── pdf
│   └── app-PR1.apk.pdf
└── scorecard
    └── app-PR1.apk_scorecard.json

4 directories, 4 files
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.7: Contenido `../reports`

Resultado obtenido: Obtenemos un reporte del análisis dinámico en la ruta establecida en el archivo `config.yml`.

7.3. Prueba PR03 – Análisis estático masivo

En esta prueba se validará el análisis estático sobre varias aplicaciones. Para ello, se ejecutará AppPiVal solo con la flag `-v/--verbose`, que accede por defecto al análisis estático múltiple. También se descargarán los tres tipos de resultados *scorecard*, *json* y *pdf*. Para poder hacer una prueba controlada, se analizarán 5 apps simultáneamente.

Caso de uso asociado: CU03 – Ejecutar análisis estático masivo sobre varios archivos APK.

Objetivo: Validar que el sistema analiza concurrentemente varias APKs.

Pasos:

1. Ejecutar AppPiVal con la flag `-v/--verbose`.
2. Se carga configuración `config.yml`.
3. Se crea una tarea por aplicación a analizar.
4. Cada aplicación se sube a MobSF.
5. MobSF analiza la aplicación.
6. Una vez analizadas, se descargan los resultados.

Resultado esperado: Por cada aplicación, hay un archivo de los formatos y directorios establecido en `config.yml`

7.3.1. Ejecución

Para esta prueba, vamos a analizar 5 aplicaciones simultáneamente, para poder ver con claridad los logs de ejecución.

Para esta prueba tenemos la configuración de `config.yml` mostrada en la figura 7.8. Como directorio de entrada, elegimos `./apps`, donde tenemos las cinco aplicaciones a analizar, y para el directorio de resultados, el mismo que en las dos pruebas anteriores. En esta ocasión los valores de los semáforos si nos importan. En este caso tenemos un valor de 3 en el semáforo de análisis, para que en los logs de ejecución se observe correctamente esta característica técnica. En cuanto al valor de *open_files* al ser un número de aplicaciones pequeño no sería relevante, por lo que en esta prueba se descarta. En cuanto a los valores de la api, tenemos los mismos que en las dos pruebas anteriores, y para los resultados, vamos a elegir solo *scorecard*, para disminuir los logs de ejecución.

```yaml
project:
  apps_folder: "./apps"
  output_folder: "./reports"

semaphores:
  analysis: 3
  open_files: 25

api:
  endpoint: "http://localhost:8000"
  key: "cb754e8ed75dde6797f081c342dcecfd510ce588e6f2452ae34a459b47303a80"

output:
  scorecard: true
  pdf: false
  json: false
```

Figura 7.8: Contenido `config.yml`

Podemos acceder al modo análisis estático múltiple mediante el menú interactivo, aunque por defecto, es el modo que AppPiVal ejecuta, por lo que ejecutaremos solo `python3 main.py -v` como se muestra en la figura 7.9 junto a los logs de ejecución. Como vemos, todas las aplicaciones se han subido, analizado y descargado los resultados de forma simultánea. Además vemos como, en este caso, al tener un valor de 3 en el semáforo de análisis, la aplicación *app-PR3-3.apk* no se ha empezado a analizar hasta que una de las tres primeras que han entrado en la fase de análisis acaba esta etapa, al igual que pasa con *app-PR3-2.apk*.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -v
[INFO] 16:24:45 - Starting static analysis on apps in ./apps with 25 open files simultaneously and with 3 concurrent analysis.
[INFO] 16:24:56 - App app-PR3-4.apk: UPLOADED
[INFO] 16:24:56 - App app-PR3-4.apk: Scanning
[INFO] 16:24:56 - App app-PR3-1.apk: UPLOADED
[INFO] 16:24:56 - App app-PR3-1.apk: Scanning
[INFO] 16:24:56 - App app-PR3-5.apk: UPLOADED
[INFO] 16:24:56 - App app-PR3-5.apk: Scanning
[INFO] 16:24:56 - App app-PR3-3.apk: UPLOADED
[INFO] 16:24:56 - App app-PR3-2.apk: UPLOADED
[INFO] 16:25:08 - App app-PR3-5.apk: SCANNED SUCCESSFULLY
[INFO] 16:25:08 - App app-PR3-3.apk: Scanning
[INFO] 16:25:09 - App app-PR3-1.apk: SCANNED SUCCESSFULLY
[INFO] 16:25:09 - App app-PR3-2.apk: Scanning
[INFO] 16:25:11 - App app-PR3-4.apk: SCANNED SUCCESSFULLY
[INFO] 16:25:13 - App app-PR3-5.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app-PR3-5.apk_scorecard.json
[INFO] 16:25:14 - App app-PR3-1.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app-PR3-1.apk_scorecard.json
[INFO] 16:25:15 - App app-PR3-4.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app-PR3-4.apk_scorecard.json
[INFO] 16:25:17 - App app-PR3-3.apk: SCANNED SUCCESSFULLY
[INFO] 16:25:18 - App app-PR3-2.apk: SCANNED SUCCESSFULLY
[INFO] 16:25:18 - App app-PR3-3.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app-PR3-3.apk_scorecard.json
[INFO] 16:25:19 - App app-PR3-2.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app-PR3-2.apk_scorecard.json
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.9: Ejecución AppPiVal -v

Además, vemos como ahora solo se han descargado los resultados de scorecard, que podemos comprobar ejecutando `tree ../reports`, con su salida en la figura 7.10.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ tree ../reports/
../reports/
├── dynamic
│   └── 3d8cafc18d6773319ca71618d1e6a1b5_dynamic.json
├── json
│   └── app-PR1.apk.json
├── pdf
│   └── app-PR1.apk.pdf
└── scorecard
    ├── app-PR1.apk_scorecard.json
    ├── app-PR3-1.apk_scorecard.json
    ├── app-PR3-2.apk_scorecard.json
    ├── app-PR3-3.apk_scorecard.json
    ├── app-PR3-4.apk_scorecard.json
    └── app-PR3-5.apk_scorecard.json

4 directories, 9 files
```

Figura 7.10: Contenido de `../reports`

Resultado obtenido: Tenemos cincos archivos scorecard en el directorio reports, tal y como especificamos en `config.yml`.

7.4. Prueba PR04 – Visualizar análisis anteriores

En esta prueba se validará la visualización de los análisis recientes. Para ello, se ejecutará AppPiVal solo con la flag `-i/--interactive`. Una vez con el menú interactivo, elegimos la opción *Display recent scans*.

Caso de uso asociado: CU04 – Visualizar análisis recientes desde CLI

Objetivo: Validar que el usuario puede acceder al listado de aplicaciones analizadas correctamente.

Pasos:

1. Ejecutar AppPiVal con la flag `-i/--interactive`.
2. Seleccionar opción *Display Recent Scans*.
3. Navegar por la aplicación

Resultado esperado: Se muestra una lista de escaneos previos, incluyendo nombres de apps y fechas.

7.4.1. Ejecución

Vamos a ejecutar AppPiVal con la flag `-i/--interactive` con el comando `python3 main.py -i`, para acceder al menú interactivo, y elegimos la opción *Display Recent Scans* como se muestra en la figura 7.11.

Figura 7.11: Ejecución AppPiVal -i

Tras elegir la opción, podemos ver como se nos muestran las aplicaciones recientemente analizadas, figuras 7.12 y 7.13, en grupos de 15, pudiendo navegar entre todas las disponibles.

```
ID    APP_NAME             MD5                                TIMESTAMP
480                        1c7f72eaf0046fa4947dde2da743d164   2025-05-30T16:18:43.068Z
481   Delhi Study Circle   4d6d17ffa53ab5ea4d95e156287d3098   2025-05-30T16:18:43.066Z
482                        7f5e9c53d894b620e7a0373556a3e3e3   2025-05-30T16:18:43.064Z
483                        3a938a01f87b7f730b5765176a3ff099   2025-05-30T16:18:42.910Z
484                        9bc86d2d5bca578a84fb0d98672b0a21   2025-05-30T16:18:40.300Z
485   Ster-Kinekor         cd8da7809dd56b890dcd30f10b8f705c   2025-05-30T16:18:37.836Z
486   AppLock              34b8fc46cfd1cdd1cd8f4f61ab0bc345   2025-05-30T16:18:37.328Z
487   Beem It              625451e91d10ccf10f9d6a1317ca7383   2025-05-30T16:18:36.908Z
488   Finanční kalkulačky  84165ee7cdc8c5a1aa439e01c089d79a   2025-05-30T16:18:36.469Z
489                        f3091e4dacc51f34055187ccca727f64   2025-05-30T16:18:35.361Z
490                        60f04e7da21de9438e4042817ba9aa79   2025-05-30T16:18:35.289Z
491   Got-Cha              f9a55846ebdbef37b33ea999f09f8fcd   2025-05-30T16:18:35.288Z
492   MedHub               acdaa06aa0b2e0924d2dfe1d2c16e6e6   2025-05-30T16:18:35.280Z
493   Tressette            8ebc41a53c33b41fade5f1f19c541a79   2025-05-30T16:18:31.634Z
494   Email for Hotmail Outlook 3500c7ae6a5689f5115a2b936ce4e86c      2025-05-30T16:18:31.485Z
Options:
[A] Previous page
[S] Next page
[Q] Exit
Option:
```

Figura 7.12: Lista de aplicaciones recientemente analizadas 1

```
ID    APP_NAME             MD5                                TIMESTAMP
510   Murri Pusat          e8af76edee0603680d6efb8f252ba5bb   2025-05-30T16:18:12.498Z
511   All Messages Recovery 270dcc492542bf59a8fa4e7e8eeab4e0   2025-05-30T16:18:11.779Z
512   Christmas HD Wallpapers 0eaefade7714fb391642e511e32e1aaf   2025-05-30T16:18:11.601Z
513   iStudents            3d8cafc18d6773319ca71618d1e6a1b5   2025-05-30T16:18:11.218Z
514   Opencho              db1e37a5bd152e488a2c64b20d157f93   2025-05-30T16:18:11.210Z
Options:
[A] Previous page
[Q] Exit
Option:
```

Figura 7.13: Lista de aplicaciones recientemente analizadas 2

Resultado obtenido: Obtenemos una lista detallada de las aplicaciones analizadas recientemente, con nombre, fecha y MD5.

7.5. Prueba PR05 – Comparación de análisis

En esta prueba se validará la comparación de los aplicaciones. Para ello, se ejecutará AppPiVal solo con la flag `-i/--interactive`. Una vez con el menú interactivo, elegimos la opción *Compare Apps*.

Caso de uso asociado: CU05 – Comparar dos análisis previos seleccionados

Objetivo: Verificar que el sistema permite comparar dos apps escaneadas y genera informe comparativo.

Pasos:

1. Ejecutar AppPiVal con la flag `-i/--interactive`
2. Seleccionar opción *Compare Apps*.
3. Elegir dos apps distintas
4. Descargar informe de comparación.

Resultado esperado: Se crea informe comparativo entre las dos apps seleccionadas.

7.5.1. Ejecución

Para acceder al modo de comparar aplicaciones, ejecutamos `python3 main.py -v -i` para acceder al menú interactivo y escoger la opción *Compare Apps* tal y como se ve en la figura 7.14.

Figura 7.14: Ejecución AppPiVal -i -v

Tras ello, elegimos dos aplicaciones de entre las disponibles a analizar, figura 7.15, viendo los logs de ejecución. [2].

Figura 7.15: Elección de las dos aplicaciones

Una vez las aplicaciones se han comparado podemos ver la descarga de resultados, comprobándolo con la salida del comando `tree ../reports/` en la figura 7.16

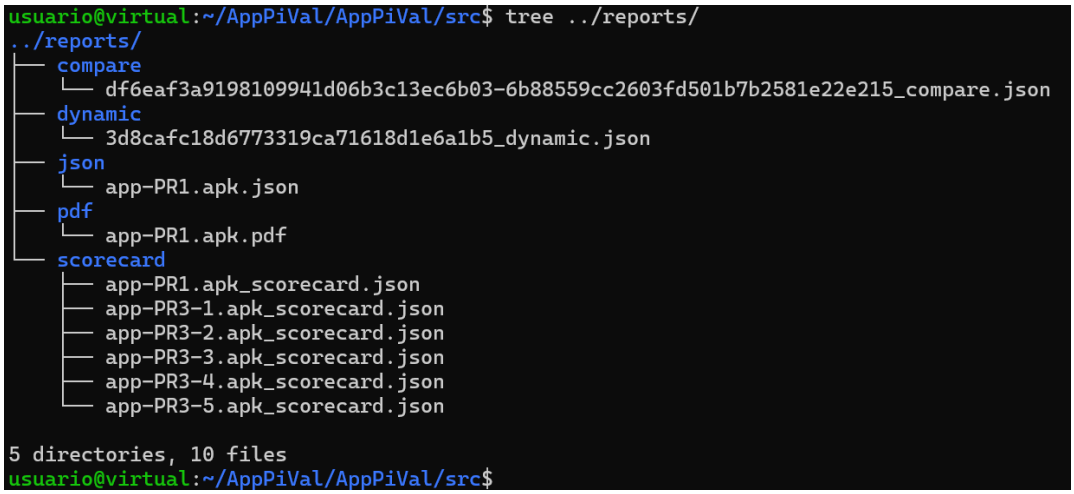

Figura 7.16: Contenido `../reports`

Resultado obtenido: Obtenemos un reporte json de la comparación entre las dos aplicaciones seleccionadas.

7.6. Prueba PR06 – Reintento de análisis fallidos

En esta prueba se validará que AppPiVal puede reintentar los análisis de aplicaciones que hayan fallado. Para ello, se ejecutará AppPiVal con la flag `-r/--retry` y la flag `-v/--verbose`. Se empezará el análisis de 2 aplicaciones, una fallará y su análisis se repetirá hasta su éxito.

Caso de uso asociado: CU06 – Reintentar análisis fallidos

[2]Hay ciertas aplicaciones que no permiten comparación, en ese caso MobSF devuelve un error *500 Server Error: Internal Server Error for url: http://localhost:8000/api/v1/compare*, que si accedemos a los logs del contenedor de MobSF vemos que es porque las aplicaciones no son compatibles para comparar.

Objetivo: Validar que el sistema detecta y relanza automáticamente los análisis fallidos.

Pasos:

1. Ejecutar AppPiVal con la flag `-r/--retry` y `-v/--verbose`.

2. Verificar que solo se relanzan aquellos análisis fallidos anteriormente.

Resultado esperado: Se completan correctamente los análisis que anteriormente fallaron.

7.6.1. Ejecución

Para esta prueba, vamos a realizar un análisis múltiple como en la prueba 7.3 ejecutando AppPiVal con la flag de reintento, es decir, con el comando `python3 main.py -v -r`.

Para esta prueba tenemos la misma configuración de `config.yml` que en la prueba mencionada, figura 7.17.

```yaml
project:
  apps_folder: "./apps"
  output_folder: "./reports"

semaphores:
  analysis: 3
  open_files: 25

api:
  endpoint: "http://localhost:8000"
  key: "cb754e8ed75dde6797f081c342dcecfd510ce588e6f2452ae34a459b47303a80"

output:
  scorecard: true
  pdf: false
  json: false
```

Figura 7.17: Contenido `config.yml`

Para simular un fallo, vamos a empezar el análisis con el docker de MobSF iniciándose, y vemos como, una vez se inicia, ya recibe las peticiones de análisis de aquellas aplicaciones que habían fallado como se ve en los logs de ejecución en la figura 7.18.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -v -r
[INFO] 22:00:26 - Starting static analysis on apps in ./apps with 25 open files simulteniously and with 3 concurrent analysis.
[ERROR] 22:00:26 - Error on UPLOADING app-PR3-3.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:26 - Retrying app app-PR3-3.apk in 10s
[ERROR] 22:00:26 - Error on UPLOADING app-PR3-4.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:26 - Retrying app app-PR3-4.apk in 10s
[ERROR] 22:00:26 - Error on UPLOADING app-PR3-2.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:26 - Retrying app app-PR3-2.apk in 10s
[ERROR] 22:00:26 - Error on UPLOADING app-PR3-1.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:26 - Retrying app app-PR3-1.apk in 10s
[ERROR] 22:00:36 - Error on UPLOADING app-PR3-4.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:36 - Retrying app app-PR3-4.apk in 10s
[ERROR] 22:00:36 - Error on UPLOADING app-PR3-2.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:36 - Retrying app app-PR3-2.apk in 10s
[ERROR] 22:00:36 - Error on UPLOADING app-PR3-3.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:36 - Retrying app app-PR3-3.apk in 10s
[ERROR] 22:00:36 - Error on UPLOADING app-PR3-1.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:36 - Retrying app app-PR3-1.apk in 10s
[ERROR] 22:00:46 - Error on UPLOADING app-PR3-3.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:46 - Retrying app app-PR3-3.apk in 10s
[ERROR] 22:00:46 - Error on UPLOADING app-PR3-4.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:46 - Retrying app app-PR3-4.apk in 10s
[ERROR] 22:00:46 - Error on UPLOADING app-PR3-2.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:46 - Retrying app app-PR3-2.apk in 10s
[ERROR] 22:00:46 - Error on UPLOADING app-PR3-1.apk: [Errno None] Can not write request body for http://localhost:8000/api/v1/upload
[INFO] 22:00:46 - Retrying app app-PR3-1.apk in 10s
[INFO] 22:01:06 - App app-PR3-4.apk: UPLOADED
[INFO] 22:01:06 - App app-PR3-4.apk: Scanning
[INFO] 22:01:06 - App app-PR3-1.apk: UPLOADED
[INFO] 22:01:06 - App app-PR3-1.apk: Scanning
[INFO] 22:01:06 - App app-PR3-3.apk: UPLOADED
[INFO] 22:01:06 - App app-PR3-3.apk: Scanning
[INFO] 22:01:07 - App app-PR3-2.apk: UPLOADED
[INFO] 22:01:17 - App app-PR3-3.apk: SCANNED SUCCESSFULLY
[INFO] 22:01:17 - App app-PR3-2.apk: Scanning
[INFO] 22:01:20 - App app-PR3-1.apk: SCANNED SUCCESSFULLY
[INFO] 22:01:20 - App app-PR3-4.apk: SCANNED SUCCESSFULLY
```

Figura 7.18: Contenido `config.yml`

Resultado obtenido: Se reintenta el análisis de las aplicaciones que fallan en algún punto.

7.7. Prueba PR07 – Cargar configuración YAML

En esta prueba se validará que AppPiVal aplica correctamente la configuración mediante el fichero `config.yml`. Para ello, se ejecutará AppPiVal sin flags.

Caso de uso asociado: CU07 – Cargar configuración desde archivo YAML

Objetivo: Verificar que el sistema detecta, lee y aplica correctamente el archivo `config.yml`.

Pasos:

1. Iniciar AppPiVal
2. Validar lectura del archivo y uso de los parámetros configurados

Resultado esperado: El sistema usa los parámetros leídos en la ejecución posterior.

7.7.1. Ejecución

La carga de configuración la podemos ver en pruebas anteriores, tanto en la descarga de resultados en la ruta seleccionada (Pruebas 7.1, 7.2, 7.3), valor de semáforos (Prueba 7.3) o tipo de resultados a descargar (Pruebas 7.1 y 7.3).

En este caso vamos a poner otros dos ejemplos de ejecución de AppPiVal, el primero con la configuración de la prueba 7.3, y otro eliminando el valor del semáforo *analysis*. Como vemos en la figura 7.19, en la primera ejecución se nos carga la configuración descrita en la prueba 7.3. Sin embargo, con una configuración mal formada, AppPiVal nos avisa, saliendo del proceso, ya que no puede continuar con los análisis.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -v
[INFO] 16:27:47 - Starting static analysis on apps in ./apps with 3 open files simulteniously and with 5 concurrent analysis.
usuario@virtual:~/AppPiVal/AppPiVal/src$ vim ../config.yml
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -v
Carefull!!, config.yml file is misformed or with no values.
You can check format on setup script or on github repository
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.19: Carga configuración `config.yml`

Resultado obtenido: El sistema usa los parámetros leídos del archivo en la ejecución posterior siempre y cuando este esté bien formado.

7.8. Prueba PR08 – Guardar logs de ejecución

En esta prueba se validará que AppPiVal guarda correctamente los logs de ejecución. Para ello, se ejecutará AppPiVal con la flag `-d/--debug` para obtener todos los logs disponibles.

Caso de uso asociado: CU08 – Registrar log de ejecución

Objetivo: Verificar que el sistema genera logs correctos tras operaciones relevantes.

Pasos:

1. Ejecutar cualquier análisis (estático, dinámico o comparación).
2. Inspeccionar el archivo `AppPiVal.log`.

Resultado esperado: El archivo de log contiene operaciones con timestamp y descripción clara.

7.8.1. Ejecución

Para esta prueba vamos a ejecutar un análisis estático unitario, de la misma forma que hacíamos en la prueba 7.1, añadiendo la flag `-d/--debug`. Es decir, vamos a ejecutar `python3`

`main.py -v -d -file app.apk` como se muestra en la figura 7.20.Como podemos ver, se nos printean todos los logs, incluso aquellos de *nivel DEBUG*.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -v -d -f app.apk
[INFO] 14:16:01 - Starting static analysis on app app.apk.
[DEBUG] 14:16:01 - Method api/v1/upload on app.apk STARTED
[DEBUG] 14:16:03 - Method api/v1/upload on app.apk FINISHED
[INFO] 14:16:03 - App app.apk: UPLOADED
[INFO] 14:16:03 - App app.apk: Scanning
[DEBUG] 14:16:03 - Method api/v1/scan on app.apk STARTED
[DEBUG] 14:16:06 - Polling api/v1/search on app.apk for 0 attemp
[DEBUG] 14:16:13 - Method api/v1/scan on app.apk FINISHED
[INFO] 14:16:13 - App app.apk: SCANNED SUCCESSFULLY
[DEBUG] 14:16:13 - Method /api/v1/scorecard on app.apk STARTED
[DEBUG] 14:16:16 - Method /api/v1/scorecard on app.apk FINISHED
[DEBUG] 14:16:16 - Saving app.apk enrich scorecard STARTED
[DEBUG] 14:16:16 - Saving app.apk enrich scorecard FINISHED
[INFO] 14:16:16 - App app.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app.apk.json
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.20: Ejecución AppPiVal -v -d -f app.apk

Podemos verificar que se han guardado en el archivo de logs `AppPiVal.log` ejecutando `tail -n 14 ../AppPiVal.log`, como muestra la figura 7.21

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ tail -n 14 ../AppPiVal.log
[INFO] 14:16:01 - Starting static analysis on app app.apk.
[DEBUG] 14:16:01 - Method api/v1/upload on app.apk STARTED
[DEBUG] 14:16:03 - Method api/v1/upload on app.apk FINISHED
[INFO] 14:16:03 - App app.apk: UPLOADED
[INFO] 14:16:03 - App app.apk: Scanning
[DEBUG] 14:16:03 - Method api/v1/scan on app.apk STARTED
[DEBUG] 14:16:06 - Polling api/v1/search on app.apk for 0 attemp
[DEBUG] 14:16:13 - Method api/v1/scan on app.apk FINISHED
[INFO] 14:16:13 - App app.apk: SCANNED SUCCESSFULLY
[DEBUG] 14:16:13 - Method /api/v1/scorecard on app.apk STARTED
[DEBUG] 14:16:16 - Method /api/v1/scorecard on app.apk FINISHED
[DEBUG] 14:16:16 - Saving app.apk enrich scorecard STARTED
[DEBUG] 14:16:16 - Saving app.apk enrich scorecard FINISHED
[INFO] 14:16:16 - App app.apk: SCORECARD on /home/usuario/AppPiVal/AppPiVal/reports/scorecard/app.apk.json
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.21: Contenido `AppPiVal.log`

Resultado obtenido: El archivo de log contiene los logs de operaciones con timestamp y descripción clara.

7.9. Prueba PR09 – Registrar errores de ejecución

En esta prueba se validará que AppPiVal guarda correctamente los logs de error. Para ello, se ejecutará AppPiVal y se forzará un fallo, como el estado de MobSF o indicar una aplicación no existente.

Caso de uso asociado: CU09 – Registrar log de errores de ejecución

Objetivo: Confirmar que los errores se registran correctamente en el log.

Pasos:

1. Simular fallo de MobSF o fichero aplicación
2. Ejecutar análisis
3. Revisar logs

Resultado esperado: El archivo de log contiene el error con timestamp y descripción clara.

7.9.1. Ejecución

Vamos a simular dos fallos y tras ejecutar AppPiVal, revisar el fichero AppPiVal.log para ver si estos logs se han guardado correctamente

En primer lugar, vamos a detener MobSF, simulando que se ha caído por algún fallo de red o fallo del contenedor. Para ello ejecutamos `docker-compose stop mobsf` para parar el contenedor, y tras ello, ejecutamos AppPiVal mediante `python3 main.py`. Como podemos ver en la figura 7.22, al ejecutar AppPiVal se guardan los logs de error, en este caso indicando que ha fallado la subida de las aplicaciones, ya que es el primer paso en el que AppPiVal interactúa con MobSF.

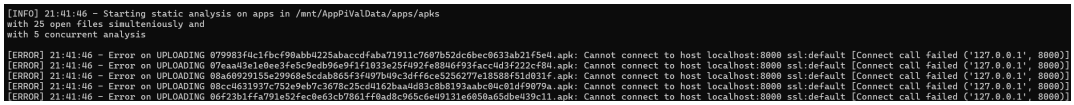

Figura 7.22: Contenido de AppPiVal.log tras error 1

En una segunda prueba, vamos a ejecutar `python3 main.py -f app_inválida.apk`, siendo app_inválida.apk un archivo que no existe. Ahora vemos en la figura 7.23 como el mensaje de error cambia, indicando que el archivo no existe, además de indicarnos que el análisis ha fallado.

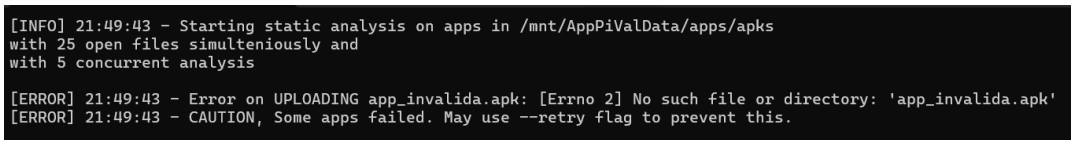

Figura 7.23: Contenido de AppPiVal.log tras error 2

Resultado obtenido: Los logs de error se guardan correctamente en el archivo AppPiVal.log, con el marca de tiempo y descripción.

7.10. Prueba PR10 – Validar estado del entorno

En esta prueba se validará que AppPiVal comprueba el estado del entorno antes de ejecutar un análisis. Para ello, se ejecutará AppPiVal con la flag `-c/--check`.

Caso de uso asociado: CU10 – Validar estado entorno

Objetivo: Verificar que el sistema detecta si los contenedores y servicios necesarios están activos.

Pasos:

1. Ejecutar AppPiVal con la flag `-c/--check`
2. Si el entorno está operativo, el análisis empieza.

Resultado esperado: El sistema informa correctamente de disponibilidad de contenedores.

7.10.1. Ejecución

Teniendo todos los contenedores levantados y en estado healthy, podemos AppPiVal con el comando `python3 main.py -c -v` con su salida en 7.24.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -c -v
MobSF is up and usable
[INFO] 21:15:35 - Starting static analysis on apps in /mnt/AppPiValData/apps/apks
with 25 open files simulteniously and
with 5 concurrent analysis
```

Figura 7.24: Validación exitosa entorno mediante AppPiVal -c

Si queremos probar el otro caso, podemos parar el contenedor de MobSF momentáneamente con `docker-compose stop mobsf`, y ejecutar de nuevo el comando `python3 main.py -c -v` con su salida en 7.25.

```
usuario@virtual:~/AppPiVal/AppPiVal/src$ docker-compose stop mobsf
[+] Stopping 1/1
 ✔Container mobsf  Stopped
usuario@virtual:~/AppPiVal/AppPiVal/src$ python3 main.py -c -v
MobSF is not usable : <urlopen error [Errno 111] Connection refused>
usuario@virtual:~/AppPiVal/AppPiVal/src$
```

Figura 7.25: Validación no exitosa entorno mediante AppPiVal -c

Resultado obtenido: Tanto si el entorno está operativo como si no, AppPiVal avisa, empezando el análisis solo en caso afirmativo.

7.11. Prueba PR11 – Despliegue del entorno de análisis

En esta prueba se validará el entorno personalizado. Para ello, se ejecutará los comando `docker/docker-compose` pertinentes para levantar y preparar el entorno de análisis.

Caso de uso asociado: CU11 – Desplegar entorno de análisis

Objetivo: Verificar que los servicios se levantan correctamente con `docker-compose`.

Pasos:

1. Ejecutar `docker-compose up -d --build`
2. Ejecutar `docker ps` para verificar estado contenedores.

Resultado esperado: Todos los contenedores se levantan correctamente y están en estado healthy.

7.11.1. Ejecución

Para empezar, podemos comprobar que no tenemos ningún contenedor activo. Para ello, podemos ejecutar el comando `docker ps [-a]` con su salida en la figura 7.26. Si tuviéramos algún contenedor activo, podemos ejecutar `docker-compose down`, para "tumbar"todos los contenedores, o indicar el nombre de uno para solo "tumbar.esçe contenedor.

```
usuario@virtual:~/AppPiVal/AppPiVal$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
usuario@virtual:~/AppPiVal/AppPiVal$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
usuario@virtual:~/AppPiVal/AppPiVal$
```

Figura 7.26: Salida de comandos docker ps y docker ps -a

Para levantar los contenedores, ejecutamos el comando `docker-compose up -d --build` con su salida en la figura 7.27. Si ya hemos creado la imagen personalizada del AVD, esta estará cacheada, y tardará menos. De otro modo, el comando para levantar estos contenedores no cambia.

Figura 7.27: Salida de comando docker-compose up -d –build

Una vez el comando `docker-compose up -d -build` termina, damos tiempo al contenedor de MobSF a levantarse correctamente, y tras unos minutos, podemos ejecutar de nuevo el comando `docker ps`, que nos devuelve los tres contenedores junto a su estado, como vemos en la figura 7.28.

Figura 7.28: Salida de comando docker ps tras levantar contenedores

Resultado obtenido: Tenemos los tres contenedores levantados, preparados para empezar los análisis.

Capítulo 8

Conclusiones

8.1. Conclusiones

El desarrollo de AppPiVal ha permitido demostrar que es viable automatizar el análisis de seguridad en aplicaciones móviles mediante MobSF. Además, se ha conseguido realizar análisis estáticos múltiples. Esta característica no se ha trasmitido al análisis dinámico debido a limitaciones con el emulador, aunque se ha conseguido automatizar de forma correcta.

Se ha logrado encapsular esta funcionalidad en una arquitectura portable basada en contenedores, facilitando su despliegue y mantenimiento. Con esto se ha conseguido no tener que depender de servicios externos y poder tener un entorno de análisis completo propio y modificable.

Además, se ha incorporado satisfactoriamente el indicador de privacidad *privacy score* para reflejar esta característica además de la seguridad, lo que enriquece de gran forma los resultados de MobSF.

8.2. Líneas de trabajo futuro

Entre las posibles mejoras destacan:

- **Generar una forma de representación para el indicador:** Actualmente el indicador *privacy score*, solo es un campo del archivo scorecard formato json. Sin embargo, sería interesante tener una forma visual de representarlo, como si lo tiene actualmente el indicador *security score*, haciendo el reporte visual, además del archivo json, mucho más informativo y completo.
- **Modo servicio *standby*:**. La primera solución presentada en la sección 2.1, funcionaba con un script que permanecía activo en el *backgroud*. Aunque este aspecto se

puede resolver con tecnologías como screen[1], sería interesante dotar a AppPiVal de esta característica

- **Mejorar el análisis dinámico:**. Por como está montado el emulador y la automatización del análisis dinámico, hay un endpoint de MobSF */api/v1/android/start_activity* que no se ha podido implementar, y es realmente útil para ejecutar actividades específicas dentro de la aplicación.

[1] https://www.gnu.org/software/screen/manual/screen.html

Glosario

AVD *Android Virtual Device*. Entorno de emulación que reproduce el comportamiento de un dispositivo Android físico. Permite ejecutar, probar y analizar aplicaciones móviles sin necesidad de un terminal real. En AppPiVal, el AVD se utiliza para realizar análisis dinámicos automatizados mediante integración con MobSF.

CLI Interfaz de Línea de Comandos (*Command Line Interface*). Medio de interacción con un sistema mediante la introducción de comandos en texto. Es común en herramientas técnicas y permite mayor control, automatización y eficiencia en entornos como scripts o despliegues.

Docker Plataforma de contenedorización de código abierto que permite empaquetar aplicaciones junto con todas sus dependencias en unidades portables llamadas contenedores. Facilita la ejecución consistente de software en entornos aislados, independientemente del sistema operativo host.

Docker Compose Herramienta oficial de Docker que permite definir, configurar y ejecutar múltiples contenedores de forma conjunta mediante un único archivo `docker-compose.yml`. Es especialmente útil para entornos con arquitecturas distribuidas o servicios interdependientes.

NFS *Network File System*. Sistema de archivos distribuido que permite a un usuario acceder a archivos y directorios ubicados en ordenadores remotos como si estuvieran en un sistema de archivos local. En el contexto de este trabajo, NFS se utiliza para montar directorios compartidos entre la máquina anfitriona y las máquinas virtuales o contenedores, facilitando el intercambio de datos y la persistencia de la información.

Polling Técnica en la que un sistema o programa consulta periódicamente a otro (o a un recurso) para comprobar si hay nuevos datos, eventos o cambios de estado disponibles. A menudo se utiliza cuando la comunicación asíncrona no es posible o deseable, pero puede consumir recursos de forma ineficiente si las comprobaciones son muy frecuentes y los cambios son poco comunes.

YAML Lenguaje de serialización de datos de lectura sencilla, diseñado para ser altamente legible por humanos. Es comúnmente utilizado para archivos de configuración, intercambio de datos entre lenguajes y serialización de objetos.

Apéndice A. Repositorio de Código

En este apéndice se proporciona la información necesaria para acceder al código fuente desarrollado en el presente Trabajo de Fin de Grado.

Todo el código fuente de este trabajo esta alojados en un repositorio público. Se puede acceder al mismo a través de la siguiente URL:

https://github.com/Obi-Juan-NoSeEnoje17/AppPiVal.git

Se recomienda encarecidamente a los lectores interesados en explorar el código, realizar pruebas o contribuir, que visiten el repositorio. En él encontrarán la estructura completa del proyecto, así como un archivo README.md con instrucciones detalladas sobre cómo clonar el repositorio, configurar el entorno y ejecutar el código.

Bibliografía

[1] Mobile Security Framework (MobSF) Team. Mobile security framework (mobsf). Herramienta para análisis de seguridad de aplicaciones móviles.

[2] Mobile Security Framework (MobSF) Team. Mobile security framework - api documentation. Documentación de API REST.

[3] Richard Lawson. *Web scraping with Python*. Packt Publishing Ltd, 2015.

[4] Microsoft Corporation. Software de administración de proyectos — microsoft project, January 2025. Visitado el 30 de enero de 2025.

[5] TIOBE Software. Tiobe index, 2025. Accessed: 2025-03-15.

[6] Kinsta. What is a scripting language? definition, examples, and uses, 2025. Accessed: 2025-03-15.

[7] DesignGurus. Which language is best for network engineer?, 2025. Accessed: 2025-03-15.

[8] David Beazley. Understanding the python gil. In *PyCON Python Conference. Atlanta, Georgia*, pages 1–62, 2010.

[9] Oxylabs. httpx vs requests vs aiohttp: Which http client should you choose?, 2022.