App móvil – Analizador de vulnerabilidades de servidores web

Gonzalo Ibáñez Rocamora
Máster universitario de Ciberseguridad y Privacidad
Análisis de datos

Joan Caparrós Ramirez
Cristina Pérez Solà

31 de mayo de 2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*App móvil – Analizador de vulnerabilidades de servidores web*
Nombre del autor:	*Gonzalo Ibáñez Rocamora*
Nombre del consultor/a:	*Joan Caparrós Ramirez*
Nombre del PRA:	*Cristina Pérez Solà*
Fecha de entrega (mm/aaaa):	05/2022
Titulación:	*Máster universitario de Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Análisis de datos*
Idioma del trabajo:	*Castellano*
Palabras clave	*app ciberseguridad Android*

Resumen del Trabajo (máximo 250 palabras): *La finalidad de este trabajo es desarrollar una aplicación móvil Android que permita realizar un escaneo de puertos y vulnerabilidades web. En la introducción del trabajo se describe la justificación del trabajo, el contexto, la metodología, la planificación, el presupuesto y los riesgos. Por otro lado, se realiza una fase de investigación de las diferentes tecnologías y técnicas utilizadas en el proyecto. Posteriormente, se realiza el desarrollo de la aplicación de Android, se muestran las conclusiones del proyecto y trabajos futuribles.*

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

En la actualidad, internet se ha convertido en una herramienta esencial e indispensable para la sociedad, y todos los servicios y aplicaciones que manejan usuarios o empresas deben garantizar la máxima seguridad posible para disminuir los riesgos de privacidad a los que puedan ser sometidos. Debido a que el uso de herramientas de internet se ha visto incrementado en los últimos años, lamentablemente, también se presenta un incremento de ataques cibernéticos que causan grandes daños económicos y morales. Es por eso por lo que, a medida que las técnicas usadas con fines ilegales van aumentando su capacidad para comprometer servicios y sistemas, también deben mejorar las técnicas y herramientas para la prevención y mitigación de éstas.

Hoy en día, es necesario disponer de herramientas que ayuden a propietarios de sitios web, servidores o empresas a obtener información de sus sistemas y saber, en la medida de lo posible, el grado de seguridad de sus sistemas o servicios. El proyecto que se desarrolla en este trabajo consiste en desarrollar un software con soporte en un sistema operativo móvil que detecte posibles vulnerabilidades que podrían ser aprovechadas por un atacante o ciberdelincuente, a partir de una dirección IP o servicio web.

Normalmente, para realizar una auditoría en una aplicación web se requiere de cierto conocimiento, experiencia y destreza, ya que se deben tener en cuenta una gran multitud de tecnologías y herramientas utilizadas para el desarrollo de la aplicación que requieren seguridad. Con este proyecto se busca la automatización de un conjunto de procesos necesarios para examinar y detectar vulnerabilidades más comunes en una aplicación web solamente con la inserción de la dirección IP en la aplicación desarrollada en el proyecto. Para poder desarrollar el proyecto es necesario implementar una aplicación móvil que tenga soporte en sistemas operativos Android, una aplicación web (laboratorio de pruebas) para verificar que es totalmente funcional y servicios que escaneen vulnerabilidades y recojan toda la información posible sobre ellas.

1.2. Objetivos del Trabajo

El objetivo global de este proyecto es el desarrollo de una aplicación móvil Android, basada en la detección y el análisis de vulnerabilidades de una determinada dirección IP o servicio web a través de un escaneo realizado a partir de ésta. A su vez, el objetivo global del proyecto se divide en una serie de objetivos primarios:

- Implementar herramientas y técnicas de ciberseguridad a través de un soporte móvil. En este caso, el soporte se corresponde a un dispositivo Android.
- Escaneo de los puertos de un host, con el fin de obtener información de los puertos que estén abiertos y puedan ser comprometidos, así como el servicio que se ejecuta en cada puerto abierto.
- Escanear e identificar vulnerabilidades de un servicio web.
- Aprender sobre el consumo de los API REST y su implementación en el desarrollo de una aplicación Android.
- Cumplir y respetar los tiempos de cada entrega para cumplir con la metodología del proyecto con el fin de acelerar y facilitar el flujo de trabajo.
- Mitigar los problemas, fallas y riesgos correspondientes al desarrollo del proyecto en la medida de lo posible.
- Crear un prototipo que cumpla con las expectativas esperadas desde el inicio del proyecto.

A su vez, el desarrollo del proyecto conlleva cumplir una serie de objetivos secundarios:

- Investigar y estudiar sobre los API REST existentes o herramientas que permitan satisfacer y lograr el objetivo global del proyecto.
- Aprender sobre el uso de contenedores para su posterior implementación en el proyecto en caso de que se necesite.
- Plasmar los resultados obtenidos de los escaneos de una forma clara y precisa en la aplicación Android.
- Recurrir, si es necesario, a la inclusión de hosting web con el fin de dar soporte a los API REST.

1.3. Enfoque y método seguido

En este proyecto se establece una metodología de desarrollo Agile. Esta metodología permite la constante revisión en el desarrollo, lo que conlleva a unamejora de la calidad y de la productividad, un mayor compromiso, motivación y organización por parte del desarrollador con una gestión más flexible del proyecto. La filosofía de esta metodología de trabajo es la mejora continua del modelo a desarrollar mediante un ciclo de desarrollo, testeo y mejora constantehasta el fin del proyecto. Siguiendo esta metodología de trabajo, el proyecto se divide en varias etapas, donde cada etapa es revisada y testeada constantemente, con el fin de corregir errores, minimizar y evitar la acumulaciónde riesgos y fallos en las etapas posteriores. Una vez que se revisa una etapa del proyecto, se vuelven a considerar los puntos a mejorar y se incluyen junto a los objetivos de la siguiente etapa. En este caso, la metodología de trabajo Agile seguida en el proyecto presenta la siguiente estructura:

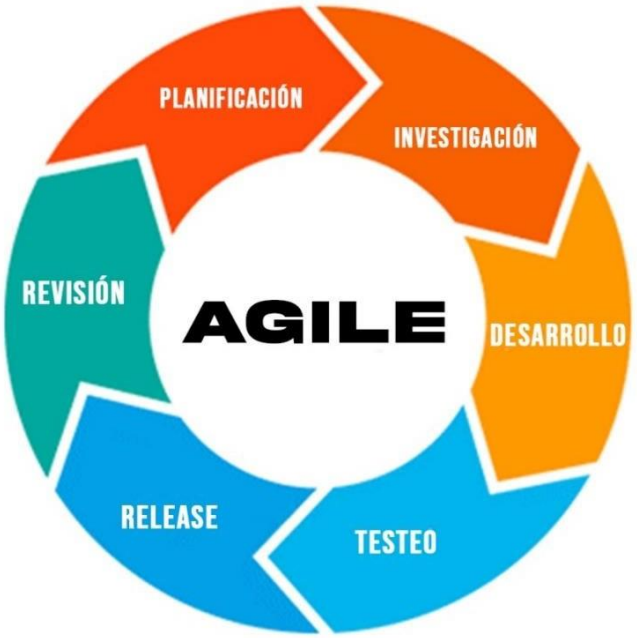

Figura 1. Metodología Agile en el proyecto. (Fuente: Elaboración propia)

Como se puede observar en la figura anterior, la metodología Agile de este proyecto se divide en las siguientes etapas:

- **Planificación**. Consiste en planificar, barajar y detallar las distintas tareas y subtareas que se llevan a cabo para la realización del proyecto. En esta etapa también se incluyen los tiempos planificados para cada tarea y el tiempo total empleado en el desarrollo del proyecto, todo incluido en un diagrama de Gantt.
- **Investigación**. Esta fase está dedicada a la investigación de las distintas herramientas y tecnologías utilizadas en el proyecto, con el fin de implementarlas de la manera más adecuada posible en su desarrollo.
- **Desarrollo**. A partir de la realización de la investigación de las diferentes herramientas y tecnologías, se lleva a cabo el desarrollo del proyecto, tanto de la infraestructura como de la implementación de todas las tecnologías propuestas en la planificación y estudiadas en la fase de investigación, a través de diferentes técnicas y métodos que son detallados, a posteriori, en el presente documento.
- **Testeo**. En esta fase se pone a prueba el desarrollo realizado en etapa anterior y se consideran puntos a mejorar y problemas surgidos en el desarrollo de forma inesperada.
- **Release**. Es la fase donde se detectan las fallas, deficiencias y riesgos de la aplicación, y se realiza la documentación de la memoria para su posterior revisión y entrega.
- **Revisión**. Una vez realizada la memoria, se obtiene una respuesta por parte del tutor del proyecto con todos los puntos débiles para tener en cuenta de cara a la próxima revisión.

1.4. Planificación del Trabajo

De acuerdo con lo definido en los objetivos del proyecto, en la fase de planificación se detallan todas las tareas y subtareas necesarias para cumplir con las expectativas de este trabajo. Por otro lado, se planifican los tiempos de cada entrega en un diagrama de Gantt. El diagrama de Gantt es una herramienta que permite organizar todas las tareas y subtareas del proyecto en determinadasfranjas de tiempo, incluyendo: la fecha de inicio y finalización del proyecto, el tiempo que ocupa realizar cada tarea y, la fecha de inicio y finalización de cada etapa del proceso de realización del proyecto. Este

diagrama permite tener una gestión del tiempo adecuada y dota al proceso de realización del proyecto de una organización extra.

En el diagrama de Gantt del proyecto se han definido cuatro intervalos de tiempo, donde cada intervalo comienza con la etapa de planificación del proyecto, y finaliza con la etapa de revisión, de acuerdo con la metodología seguida (Agile). Respecto a las tareas principales y subtareas a realizar, se plantean las siguientes:

1. Desarrollo de la interfaz gráfica de la aplicación.
    1.1. Diseñar un logo acorde a la aplicación del proyecto.
    1.2. Implementar elementos como botones, campos de texto o vistas dentro de la aplicación móvil para su posterior combinación con las funcionalidades propuestas.
    1.3. Conseguir una correcta interacción entre los distintos elementos de la aplicación
2. Disponer de un servicio web vulnerable para poder testear la aplicación de una forma correcta y legal.
    2.1. Investigación de sitios web vulnerables que cumplan con los requisitos propuestos.
    2.2. Investigación de contenedores Docker para alojar el sitio web vulnerable en un servidor local, en el caso de que se requiera.
3. Implementación de los API REST que permitan el escaneo de puertos y vulnerabilidades a través de peticiones.
    3.1. Investigar sobre peticiones HTTP para realizar consultas a los API REST.
    3.2. Buscar una API REST que permita un escaneo de puertos fiable.
    3.3. Aprender sobre el funcionamiento de *Nmap Online.*
    3.4. Implementar Nmap Online en la aplicación Android.
    3.5. Buscar una API REST que permita un escaneo de vulnerabilidades web.
    3.6. Aprender sobre el funcionamiento de *Arachni.*
    3.7. Instalar Docker y testear el correcto funcionamiento del API REST de *Arachni.*
    3.8. Implementar el API REST de *Arachni* en la aplicación Android.

4. Gestionar los resultados de cada escaneo. Esta tarea tiene como finalidad filtrar la información proporcionada por los API REST.
    4.1. Filtrar, obtener y mostrar los resultados del escaneo de puertos.
    4.2. Filtrar, obtener y mostrar los resultados del escaneo de vulnerabilidades web.
5. Testeo final de la aplicación.
    5.1. Comprobación del correcto escaneo de puertos sobre otros hosts.
    5.2. Comprobación del correcto escaneo de vulnerabilidades sobre otros servicios web.
6. Redacción de la memoria final del proyecto. La duración de esta tarea es transversal, ya que, se elabora en todo el transcurso del desarrollo del proyecto.

Por otro lado, respecto a la gestión y organización del proyecto, es necesario establecer el tiempo que se dedica a cada tarea incluida en el diagrama de Gantt (se incluyen los periodos de descanso):

- Desarrollo de la interfaz gráfica de la aplicación: *20 horas*.
- Disponer de un servicio web vulnerable: *25 horas.*
- Implementación de los API REST: *75 horas.*
- Gestión de resultados: *30 horas*
- Testeo de la aplicación: *35 horas.*
- Redacción de la memoria final: *115 horas.*

En total, se emplean trescientas horas para desarrollar el proyecto según los objetivos propuestos. A continuación, se muestra el diagrama de Gantt propuesto para cumplir la metodología:

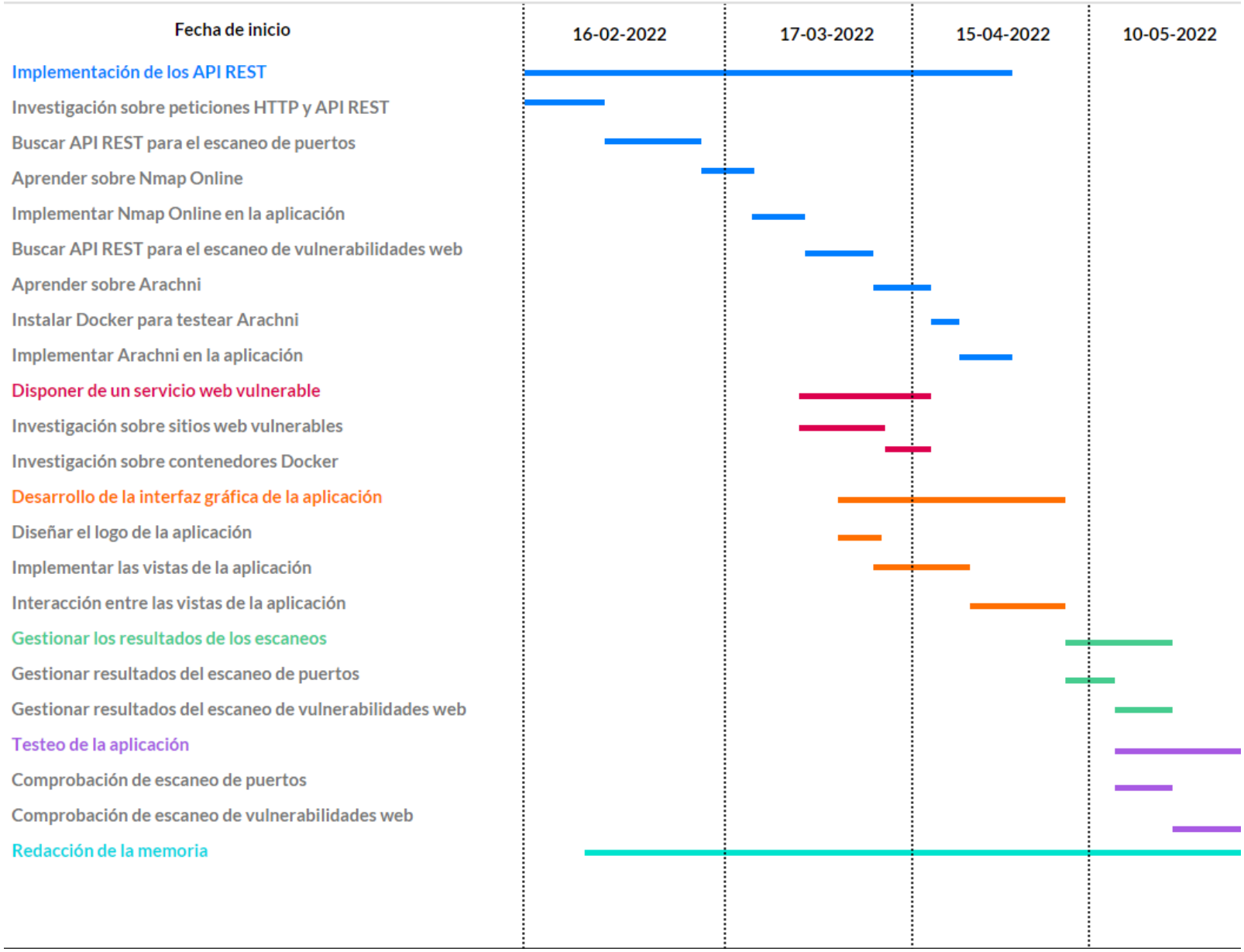

Figura 2. Diagrama de Gantt del proyecto. (Fuente: Elaboración propia)

1.5. Revisión del estado del arte

De acuerdo con la planificación del proyecto, se realiza una breve investigación sobre las herramientas utilizadas en cada tarea.

Herramientas para el escaneo de puertos

En el ámbito de la ciberseguridad, a la hora de realizar una prueba de penetración o de poner a prueba sistemas y servicios, siempre se hace uso de procesos y técnicas que ayudan a recolectar la máxima cantidad de información posible sobre el objetivo (fase de escaneo), para posteriormente proceder con el ataque. Una de esas técnicas es el escaneo de puertos del dispositivo. Los puertos son las interfaces de un dispositivo utilizados para enviar y recibir datos, y los ciberdelincuentes pueden aprovechar estos puertos para comprometerlo. Existen diferentes herramientas que permiten un escaneo de puertos, y entre las más destacables se encuentran Netcat, Nmap o TCP Port Scanner. *Nmap* apareció en 1997 y es el escáner de puertos más utilizado hoy en día en las auditorías de seguridad. Esta herramienta permite llevar a cabo tareas como:

- Identificar los puertos abiertos, cerrados o filtrados de un host.
- Los servicios que corren en cada puerto, con su correspondiente versión.
- El sistema operativo y versión que corre en el sistema host.

Además de ello, permite realizar escaneos con configuraciones muy concretas, por lo que, es una herramienta muy versátil para este tipo de trabajos.

Herramientas de detección de vulnerabilidades web

Las vulnerabilidades de las aplicaciones web son unas de las amenazas más presentes en las empresas hoy en día, por lo que, es necesario saber a qué tipo de vulnerabilidades está expuesto un sitio web con el fin de mitigarlas. Existe una gran cantidad de vulnerabilidades web, como las inyecciones o las denegaciones de servicio, entre otras. De esta forma, al igual que existen vulnerabilidades en sitios web, existen herramientas que permiten la detección e identificación de estas vulnerabilidades. Hoy en día, hay un sinfín de herramientas que permiten realizar estas acciones como: Arachni, Nikto, WFuzz, OWASP Zap, etc.

Muchas de las herramientas mencionadas anteriormente no son del todo completas, por lo que, en este proyecto se escoge Arachni, ya que es el escáner más completo que hace posible cumplir este objetivo del proyecto. Se trata de un escáner versátil, ya que tiene soporte en Mac, Windows y Linux y es capaz de detectar las diez vulnerabilidades más importantes a nivel web según OWASP.

API REST y peticiones HTTP

Una API REST permite a una aplicación o servicio acceder a un recurso dentro de otra aplicación o servicio. La aplicación que solicita el acceso se denomina cliente y la que ofrece el recurso se denomina servidor. La interacción entre las distintas aplicaciones se hace mediante solicitudes HTTP, donde cada solicitud del cliente obtiene una respuesta por parte del servidor. Existen diferentes métodos de peticiones HTTP, pero en este proyecto solamente se requiere de la utilización de dos de ellos: GET y POST. En el método GET, los datos que se envían al servidor se escriben en la dirección URL y se utilizan para solicitar un recurso, y en el método POST, los parámetros se envíanal servidor dentro de la solicitud HTTP y se utilizan para proporcionar datos al servidor con el propósito de obtener una respuesta.

Dispositivos móviles Android como una herramienta de ciberseguridad

En la actualidad, el uso de un terminal móvil se ha convertido indispensable en el día a día de la mayoría de los seres humanos, sobre todo los dispositivos móviles que corren en sistemas operativos Android e iOS. Los dispositivos móviles han ido evolucionando a lo largo de la historia hasta el punto de tener una capacidad de procesamiento y de cómputo superior a la de algunos equipos portátiles. Es por ello por lo que, algunos profesionales de la ciberseguridad emplean estos terminales para desempeñar tareas de pruebas de penetración, obteniendo los mismos resultados que usando un ordenador convencional. Normalmente, las herramientas destinadas al pentesting tienen soporte en computadoras, por lo que, en este proyecto se intenta aprovechar la capacidad computacional de un dispositivo móvil adaptando características que brindan algunas de estas herramientas, como el consumo de éstas mediante el uso de los API REST.

Android se trata de un sistema operativo Open-Source basado en el kernel de Linux desarrollado por Google y está destinado principalmente a dispositivos móviles, tablets o Google TV. En un principio, la intención de este sistema operativo era la implementación en cámaras digitales, pero debido a la poca amplitud del mercado, se opta por crear un sistema operativo que compita con Symbian y Windows Mobile, que, en esa época, son los principales sistemas operativos de móviles. En 2005, es cuando la empresa Google compra Android Inc. y comienza el desarrollo de éste basándose en el kernel de Linux y añadiendo funciones como la pantalla táctil además de incluir botones físicos. Hasta la actualidad, como se puede observar, este sistema operativo crece y adquiere tanta importancia que es, junto a Apple, el sistema operativo más usado en dispositivos móviles. (Ramirez, 2022)

Con el fin de poder desarrollar una aplicación destinado a este sistema operativo, la opción más viable es trabajar sobre el entorno de desarrollo Android Studio, utilizando los lenguajes de programación JAVA y XML, que permiten la utilización de librerías que brindarán facilidad y eficacia en el desarrollo para cumplir con los objetivos y las expectativas del proyecto.

1.6. Recursos y presupuesto

En cuanto a los recursos utilizados y el presupuesto requerido para la elaboración de este proyecto, se tienen en cuenta herramientas, tanto software como hardware. Hay recursos que requieren de una renovación anual, por lo tanto, en el presupuesto se divide en el precio base y el precio anual. En cuanto al hardware utilizado se tiene:

- Ordenador personal: utilizado para la fase de investigación y fase de desarrollo, así como para redactar la memoria del proyecto. Dentro del coste del ordenador personal se incluye el teclado y el ratón utilizado.
- Servidor: es conveniente separar los procesos o servicios a los que recurre la aplicación del ordenador personal, como los API REST, en caso de que se tengan que alojar en un servidor local. Este servidor tiene que estar activo siempre que la aplicación esté en funcionamiento. El coste de este servidor es equivalente al precio de un ordenador de gama baja, puesto que no requiere de gran capacidad computacional.
- Pantallas: en el ordenador personal se han utilizado dos pantallas para un desarrollo más rápido del proyecto. Para evitar tener que introducir una tercera pantalla para el servidor, se recurre a una de las dos que usa el ordenador personal.

Por otro lado, se hace referencia al software utilizado en este proyecto:

- Suscripción anual en el API utilizada para el escaneo de puertos: al principio se opta por la versión gratuita de esta aplicación, pero tiene una limitación de cinco escaneos diarios al día. Es por ello por lo que, es conveniente pagar por una membresía mensual que proporciona diez mil escaneos al día y una serie de ventajas que no tiene el plan gratis.
- Dominio web: la aplicación utilizada para el escaneo de vulnerabilidades web, la cual se aloja en un servidor local, requiere de un dominio en la internet pública para su funcionamiento.

Al principio se piensa en contratar un sistema de hosting y dominio web, pero posteriormente se opta por usar una herramienta que expone a internet un servicio alojado en un servidor. Esta aplicación tiene un plan gratuito, pero cada vez que se inicia cambia el dominio asociado al servicio, por lo tanto, se opta por pagar un plan mensual que permite dominios fijos para cada servicio.

- Conexión a internet: se ha utilizado conexión a internet con fibra óptica en todo el proceso de desarrollo del proyecto.

Una vez detallados todos los recursos requeridos, se adjuntan dos tablas: una para el precio base y otra para el precio anual de las membresías:

Ordenador personal	**700 €**
Servidor	**200 €**
Pantallas	**120 € / unidad**
Total	**1.140 €**

Tabla 1. Presupuesto base del proyecto.

API REST	**95,15 €**
Dominio web	**230,66 €**
Conexión a internet	**420 €**
Total	**745,81 €**

Tabla 2. Presupuesto anual del proyecto.

El coste total del proyecto sería de 1140 euros fijos, y renovar los servicios utilizados supone un coste de 745,81 euros al año.

1.7. Análisis de riesgos

El desarrollo del proyecto puede desviar la planificación establecida, ya que, pueden aparecer ciertos problemas o riesgos que se deben tener en cuenta:

Riesgo 1. Desconocimiento sobre los API REST. Debido a la falta de experiencia en aplicaciones de este tipo por parte del alumno, puede que sea necesario dedicar más tiempo de lo previsto en la planificación a la investigación sobre estas tecnologías.

Mitigación: En el caso de que suceda se debe quitar importancia a aspectos que no tengan demasiada relación con las funcionalidades principales del proyecto.

Riesgo 2. Incompatibilidades en el consumo de los API mediante Android Studio. Puede darse el caso de que se tengan que barajar distintas técnicas o librerías para consumir los API REST desde Android.

Mitigación: Empezar realizando peticiones sencillas, hasta llegar a realizar las peticiones requeridas para cumplir los objetivos del proyecto.

Riesgo 3. Tiempo dedicado a la estética de la aplicación. La estética de una aplicación puede mejorarse siempre, por lo que, puede ser que le dedique más tiempo del que se le debería dedicar.

Mitigación: Lo primero es implementar la aplicación con las funciones deseadas, y si sobra tiempo, dedicárselo a la estética de la aplicación.

Riesgo 4. Problemas en la recogida de los resultados. Es probable que haya que filtrar los resultados de las respuestas de los API REST para mostrar solamente información necesaria para el usuario.

Mitigación: Analizar detenidamente las respuestas de los API REST y estudiar sobre el manejo de los datos en formato JSON.

2. Investigación

Se procede a realizar un estudio y análisis tanto, de todas las herramientas utilizadas en el proyecto, como de la infraestructura y diseño de la aplicación. Para ello, se consideran relevantes los siguientes aspectos para la investigación:

- Infraestructura general de la aplicación
- Herramientas para el testeo de la aplicación.
- Herramientas para el escaneo de puertos.
- Herramientas para el escaneo de vulnerabilidades web.
- Análisis del diseño de la aplicación.

En primer lugar, se plantea un esquema referente a la infraestructura del proyecto. Posteriormente, se investigan las distintas herramientas que existen para realizar un escaneo de puertos y de vulnerabilidades web con el fin de elegir las que más se adapten a los objetivos del proyecto. Una vez realizado lo anterior, se procede a investigar y elegir una herramienta que permita testear las aplicaciones de escaneo de puertos y vulnerabilidades web. Finalmente, se realiza un análisis del diseño de la aplicación antes de proceder con la fase de desarrollo.

2.2. Infraestructura general

Para que la aplicación móvil tenga las funcionalidades principales descritas en los objetivos de este proyecto, hace uso de servicios externos para realizar los procesos de escaneo de puertos y vulnerabilidades web. Los servicios externos utilizados en el proyecto proceden de terceros, ya que, el tiempo de desarrollo de estos excedería el tiempo establecido en la fase de planificación. Estos servicios deben estar alojados en la nube para que la aplicación se pueda comunicar con ellos mediante peticiones HTTP. En cuanto a la infraestructura del proyecto, se dispone de un dispositivo móvil con la aplicación móvil a desarrollar en este proyecto y, por otro lado, los servicios externos correspondientes al escaneo de puertos y vulnerabilidades web. A continuación, en la figura tres, se muestra la infraestructura de este proyecto:

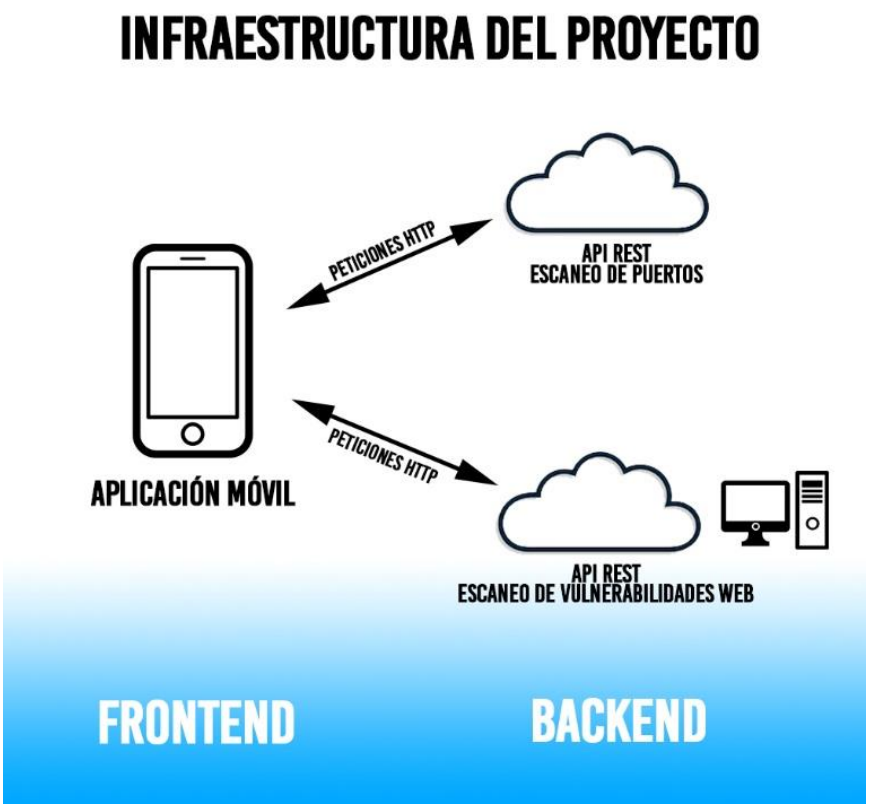

Figura 3. Diagrama de Gantt del proyecto. (Fuente: Elaboración propia)

2.3. Estudio de herramientas para el testeo

Para poder testear el funcionamiento de la aplicación durante su desarrollo, se procede a investigar sobre diferentes alternativas de aplicaciones web vulnerables, barajando la facilidad en cuanto a implementación y las vulnerabilidades que contienen. En un principio se estudia la posibilidad de que el sitio web vulnerable para el testeo sea OWASP Juice Shop: una aplicación web vulnerable que abarca vulnerabilidades de todo el OWASP Top 10. Se adjunta en la figura cuatro el entorno de la web mencionada:

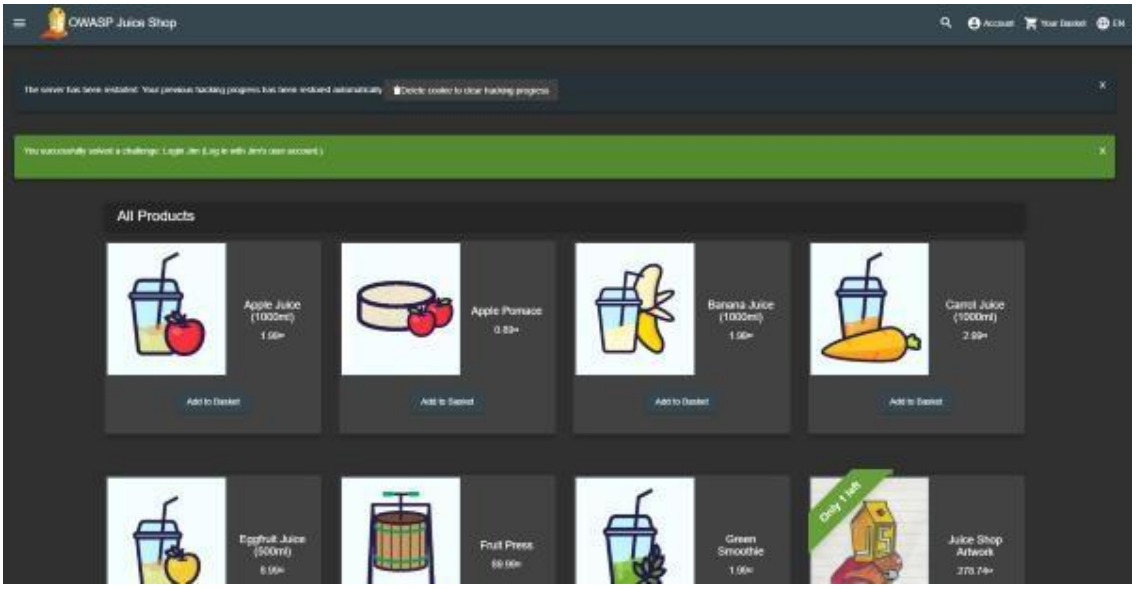

Figura 4. OWASP Juice Shop. (Fuente: Elaboración propia)

OWASP Top 10 es un estándar destinado a desarrolladores y profesionales de la seguridad web, con el fin de garantizar que un servicio web se encuentre seguro frente a las diez vulnerabilidades web más comunes que suelen aprovechar los ciberdelincuentes para comprometer un servicio web. Cada año, se actualiza el listado de vulnerabilidades del estándar. A continuación, se muestra una figura que contiene de la evolución de las vulnerabilidades que recoge el estándar, enumeradas desde la A1 hasta la A10,desde 2017 hasta 2021:

OWASP Top 10 2017		change	OWASP Top 10 2021 proposal	
A1	Injections	**as is**	A1	Injections
A2	Broken Authentication	**as is**	A2	Broken Authentication
A3	Sensitive Data Exposure	**down 1**	A3	Cross-Site Scripting (XSS)
A4	XML eXternal Entities (XXE)	**down 1 + A8**	A4	Sensitive Data Exposure
A5	Broken Access Control	**down 1**	A5	Insecure Deserialization **(merged with XXE)**
A6	Security Misconfiguration	**down 4**	A6	Broken Access Control
A7	Cross-Site Scripting (XSS)	**up 4**	A7	Insufficient Logging & Monitoring
A8	Insecure Deserialization	**up 3 + A4**	A8	**NEW:** Server Side Request Forgery (SSRF)
A9	Known Vulnerabilities	**as is**	A9	Known Vulnerabilities
A10	Insufficient Logging & Monitoring	**up 3**	A10	Security Misconfiguration

Figura 5. Vulnerabilidades Top 10 OWASP. (Fuente: https://blog.segu- info.com.ar/)

La implementación del servicio web de OWASP Juice Shop tiene una deficiencia en cuanto a complejidad y comodidad: es obligatorio correr el servicio en un servidor local para poder empezar a realizar pruebas de penetración sobre ella y un posible alojamiento en la internet pública. En un principio se opta por elegir esta opción, recurriendo al uso de contenedores, pero debido a que existen otros sitios web que permiten testear la aplicación sin necesidad de recurrir a contenedores ni servidores locales, finalmente se descarta.

Como el propósito del proyecto es crear una aplicación móvil que permita un escaneo de puertos y vulnerabilidades web y no de levantar un servicio web vulnerable, se estudian otras opciones más viables:

- SecurityTweets: es un servicio web desarrollado a partir de tecnologías como nginx, Python, Flask y CouchDB. Este sitio contiene seis de las diez vulnerabilidades del estándar *OWASP Top 10: A3, A5, A6, A8, A9 y A10*. El sitio web ya se encuentra alojado, por lo que no se tiene que hacer ninguna configuración previa para realizar el testeo desde la aplicación móvil.
- Acuart: este servicio web está desarrollado a partir de tecnologías como Apache, PHP y MySQL. Al igual que el anterior, contiene muchas de las vulnerabilidades del estándar OWASP Top 10 y tampoco requiere de configuraciones a la hora de proceder con el testeo debido a que ya se encuentra alojado. En la figura 6 se visualiza el entorno de este servicio:

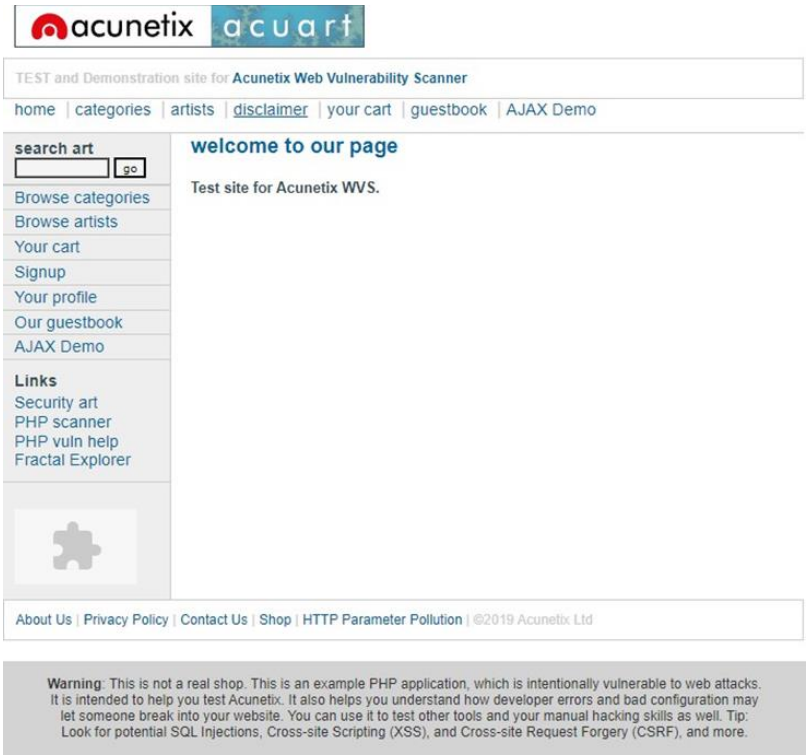

Figura 6. Sitio web vulnerable Acunetix. (Fuente: Elaboración propia)

Finalmente se escoge el sitio web Acunetix Acuart para el testeo de la aplicación debido a las vulnerabilidades que contiene, aunque también se podría hacer uso del sitio SecurityTweets como alternativa.

2.4. Estudio de herramientas de escaneo de puertos

Para el proceso de escaneo de los puertos de un sistema existen diferentes tecnologías a tener en cuenta: *Nmap, TCP Port Scanner, Angry IP Scan, Zenmap o Network Scanner (listado y características básicas).* De todas las herramientas listadas, se escoge Nmap, ya que, es la más conocida y usada y, dispone de un API REST bastante completo para poder realizar los escaneos mediante peticiones HTTP. Nmap es una herramienta Open-Source destinada a realizar auditorías de seguridad a través de paquetes IP con el fin de obtener información de una red, como equipos, puertos, servicios, o sistemas operativos, entre otros, que existen en ella con el fin de realizar una gestión y controlar la seguridad de esta. Esta herramienta usa diferentes tipos de paquetes: TCP, UDP, SCTP, ICMP. El motivo de su uso para este proyecto es identificar los dispositivos, servicios y sistemas operativos que se encuentran en una red determinada para poder realizar, con el fin de aumentar la seguridad de ésta. A continuación, se testea el funcionamiento de Nmap en la figura 7:

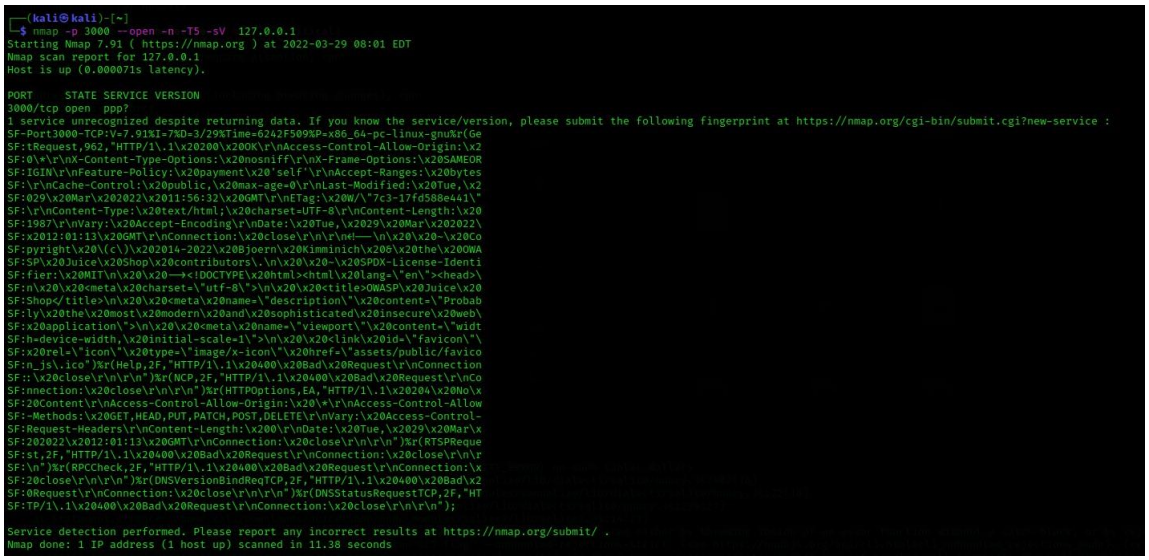

Figura 7. Escaneo de puertos con Nmap. (Fuente: Elaboración propia)

Esta herramienta dispone de comandos para establecer distintas opciones y filtros aplicables al escaneo como filtrar puertos, mostrar más información sobre cada puerto, detectar la versión de los servicios asociados a cada puerto o, duración del escaneo, entre otros.

Todos estos comandos se pueden encontrar en la documentación oficial de la herramienta. Nmap tiene, a su vez, una API REST *(Nmap Online*) para desarrolladores que permite escanear direcciones IP mediante consultas HTTP. Este API REST funciona a través de una clave API y permite al usuario realizar las acciones que se recogen en la tabla 3:

Acciones	**Descripción**	**Endpoint**
Iniciar un nuevo escaneo	Permite realizar un nuevo escaneo con parámetros de configuración específicos.	*POST /v01/start_scan*
Verificar el estado del escaneo	Permite verificar si el escaneo ha finalizado o sigue realizándose	*POST /v01/check_scan_status*
Resultados del escaneo	Permite obtener los resultados del escaneo realizado	*POST /v01/scan_result*

Tabla 3. Endpoints de Nmap Online.

2.4.1. Testeo del API REST de Nmap

Con la finalidad de testear el API REST de Nmap se intenta buscar una forma de verificar su correcto funcionamiento sin necesidad de implementarlo en la aplicación móvil del proyecto. Para ello, existen varias alternativas como *Postman, Insomnia REST Client* o *Paw*, pero finalmente se elige Postman por experiencia previa. Postman es una herramienta que permite realizar peticiones HTTP, y normalmente se usa para verificar que el consumo de un API funciona como se espera. Esta herramienta dispone de una interfaz gráfica intuitiva que permite realizar peticiones HTTP de todo tipo, permitiendo introducir: la dirección del API REST con el tipo de petición HTTP que se desea realizar, parámetros de consulta, autorizaciones en caso de que el API las requiera y un cuerpo (para peticiones POST). En este capítulo solamente se espera el correcto funcionamiento de la API REST de Nmap, ya que, en la fase de desarrollo del proyecto se detalla cada petición con el fin de realizar un nuevo escaneo, verificar su estado y recoger los resultados. Siguiendo la documentación de Nmap, se realiza una petición POST con el fin de crear un nuevo escaneo, tal y como se muestra en la figura 8:

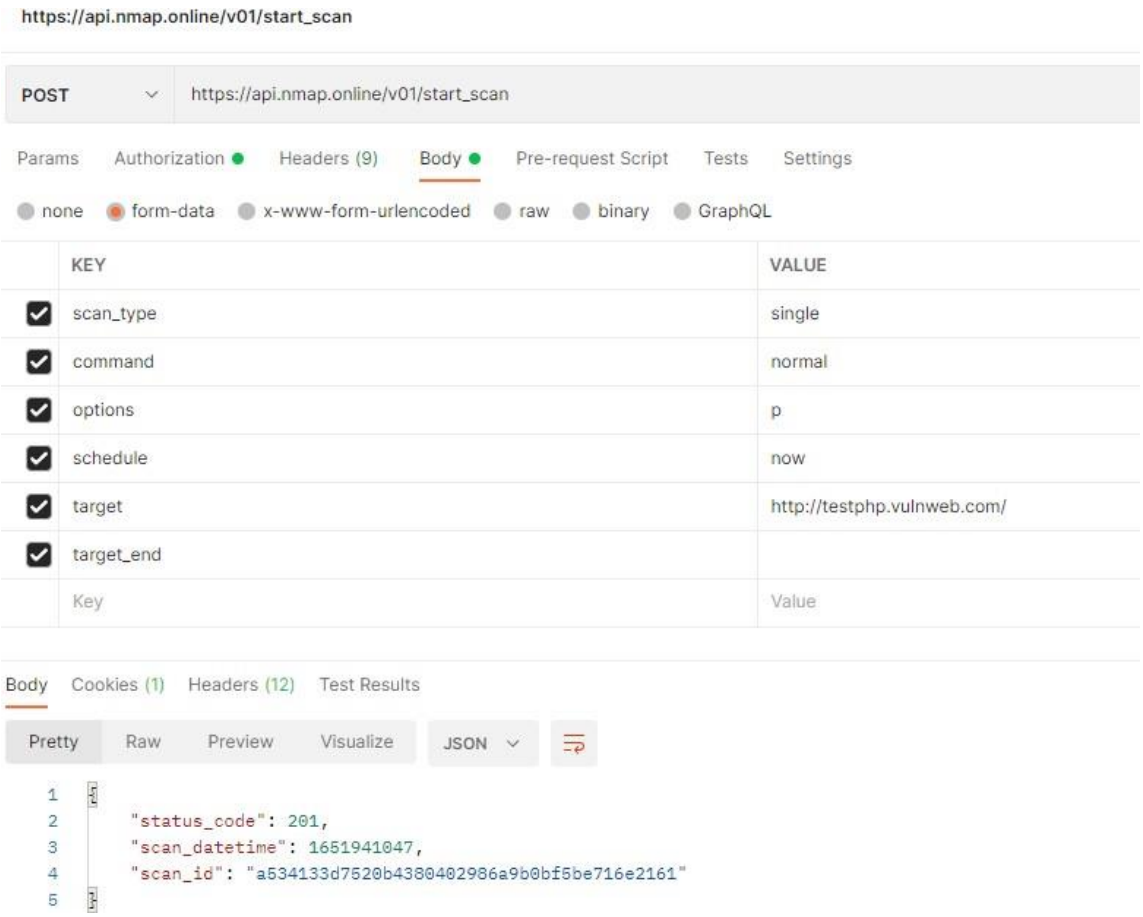

Figura 8. Petición HTTP al API REST de Nmap mediante Postman. (Fuente: Elaboración propia)

Como se puede apreciar en la figura anterior, la petición funciona correctamente, ya que, se debe obtener como respuesta la identificación del escaneo iniciado, para posteriormente verificar su estado y obtener los resultados.

2.5. Estudio de herramientas de escaneo de vulnerabilidades web

Existen múltiples herramientas que permiten escanear vulnerabilidad es en un servicio web como *Nikto* o *Arachni*. En este proyecto se usa Arachni, ya que, dispone de un API REST, que es una característica esencial para el desarrollo de la aplicación, al contrario de Nikto. Este escáner es gratuito, es Open-Source y multiplataforma y, está desarrollado con Ruby. A través de él, se permite evaluar la seguridad de aplicaciones web modernas, adaptándose a las necesidades del entorno mediante cuatro tipos de implementación simple y sin dependencias de bases de datos, servicios, configuraciones y bibliotecas. A su vez, esta herramienta incluye como soporte una API

REST, permitiendo realizar consultas HTTP con el fin de obtener información sobre los escaneos mediante el envío de comandos hacia ésta. En la figura 9 se adjunta la interfaz gráfica de Arachni:

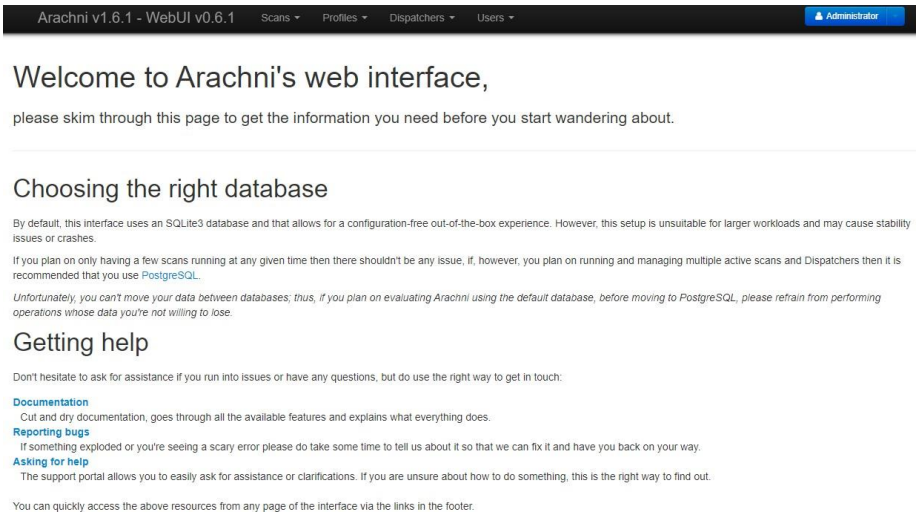

Figura 9. Interfaz gráfica de Arachni. (Fuente: https://developer.android.com/)

En cuanto a las vulnerabilidades que cubre el escaneo de esta herramienta, se incluyen desde las vulnerabilidades más usuales en aplicaciones web, como: inyecciones SQL y NoSQL, Path Traversal o inclusión de archivos, entre otras. En este proyecto se hará uso de la API REST que proporciona esta herramienta, la cual permite realizar las siguientes acciones mediante consultas HTTP (Laskos, s. f):

Acción	**Descripción**	**Endpoint**
Listar escaneos realizados	Cada escaneo tiene asociado un identificador que permite al usuario ver los escaneos realizados anteriormente	GET */scans*
Realizar nuevo escaneo	Permite crear un escaneo por parte del usuario con diferentes configuraciones	POST */scans*
Verificar el estado del escaneo	Permite al usuario ver en qué estado se encuentra un escaneo mediante su identificador.	*GET* /scans*/:id*
Listar los resultados del	Permite devolver los	*GET /scans/:id/report*

escaneo	resultados de las vulnerabilidades encontradas durante el escaneo	

Tabla 4. Endpoints del API REST de Arachni.

2.5.1. Testeo del API REST de Arachni

A diferencia de Nmap, para poder utilizar Arachni y poder tener acceso a su API REST, se debe ejecutar en un servidor. Para ello, existen dos maneras de hacerlo: ejecutar Arachni en una máquina virtual o, contenizarlo. Si se ejecuta la herramienta en una máquina virtual se pueden dar problemas de incompatibilidades dependiendo del sistema operativo, además de tener que estar pendiente de instalar actualizaciones si la herramienta se actualiza. Por otro lado, existe un contenedor Docker en el repositorio DockerHub que permite correr el servicio sin los problemas que se pueden dar en el caso de ejecutarlo en una máquina virtual. Un contenedor permite probar aplicaciones en varios entornos,es decir, puede ejecutar cualquier cosa en su interior y no dependen de ningún sistema operativo, lo que los hace ser más ligeros y portátiles. Poseen múltiples beneficios como(NetApp, s. f):

- Menos sobrecarga, ya que no incluyen imágenes del sistema operativo.
- Versatilidad. Las aplicaciones se pueden ejecutar en diferentes hardware y sistemas operativos.
- Crear entornos aislados para desarrollar sin interferencias.
- Inicio instantáneo debido a sus pocas dependencias.

Por otro lado, los contenedores tienen un funcionamiento parecido a las máquinas virtuales como VirtualBox, ya que la máquina se ejecuta dentro de un ordenador, pero las máquinas virtuales contienen en su interior mucha más información como el sistema operativo o los discos duros, lo que las hace bastante más lentas que los contenedores. (Corbetti, 2021)

Para instalar Docker en Windows 10 es necesario descargar la aplicación Docker Desktop del sitio web oficial. Una vez instalada, dentro de ella se puede acceder al repositorio oficial de Docker (DockerHub), en el que existe un contenedor Docker con la infraestructura del API REST de Arachni disponible para descargar y empezar a utilizarla. Al iniciar el contenedor descargado, ya se puede acceder a él mediante el puerto 7331 del servidor local (*http:///localhost:7331/*), de forma que ya está todo listo y configurado para realizar peticiones a la API mediante Postman para verificar su

correcto funcionamiento. La figura 10 ilustra Arachni siendo ejecutado en un contenedor Docker:

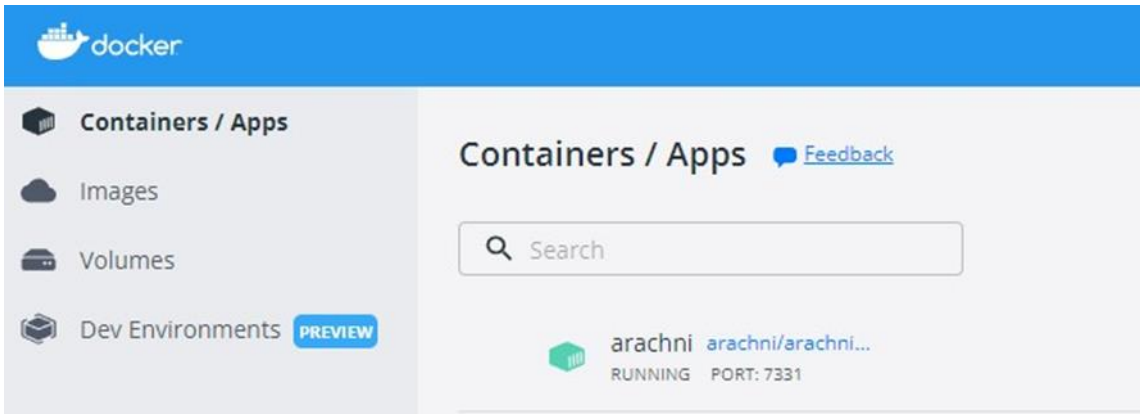

Figura 10. Docker con Arachni funcionando. (Fuente: Elaboración propia)

Ya se tiene el API REST de Arachni corriendo en un contenedor Docker, a través del puerto 7331 para realizar el primer escaneo de prueba. Se intenta realizar una petición al API REST mediante Postman a través de la dirección 0.0.0.0:7331 pero genera un error que especifica que no se pueden enviar peticiones a un localhost, por lo que se debe buscar una alternativa rápida para poder testear el correcto funcionamiento del API REST: *ngrok*.

Ngrok es una herramienta que permite compartir los recursos de un servidor. Su uso es bastante sencillo comparado con la función que realiza y, al realizar una petición con Postman a la dirección que proporciona *Ngrok*, no se genera ningún tipo de problema. Para poder lanzar esta herramienta, basta con descargarla desde el sitio web oficial y ejecutar la herramienta proporcionando el servicio y el puerto al que se quiere aplicar. En este caso, como el API REST de Arachni corre en el puerto 7331 con un serivicio HTTP, basta con introducir la siguiente instrucción después de instalar *Ngrok*: *ngrok http 7331.* Una vez realizado esto, en la figura 11 se observa que ya se tiene el puerto 3000 con el servicio HTTP activo en una dirección URL:

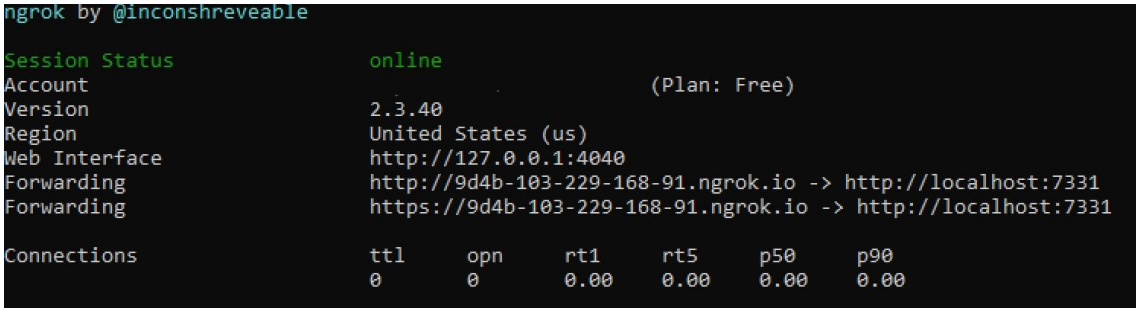

Figura 11. Funcionamiento de Ngrok. (Fuente: Elaboración propia)

Para poder realizar un nuevo escaneo, siguiendo la documentación oficial de Arachni, se debe realizar una petición HTTP de tipo POST al endpoint /scans del puerto del servidor local, donde solamente es obligatoria la URL objetivo (sitio web vulnerable Acunetix) en el cuerpo de la petición y una autenticación básica con las credenciales *arachni:password*. En la fase de desarrollo se realizan configuraciones más específicas y acordes a los resultados que se quieren obtener, pero para verificar que el API REST funciona correctamente, se testea con la configuración más básica (debe responder con el identificador del escaneo iniciado). De igual forma que se hace con Nmap Online, en la figura 12 se testea el funcionamiento del API REST de Arachni con una petición de inicio de escaneo

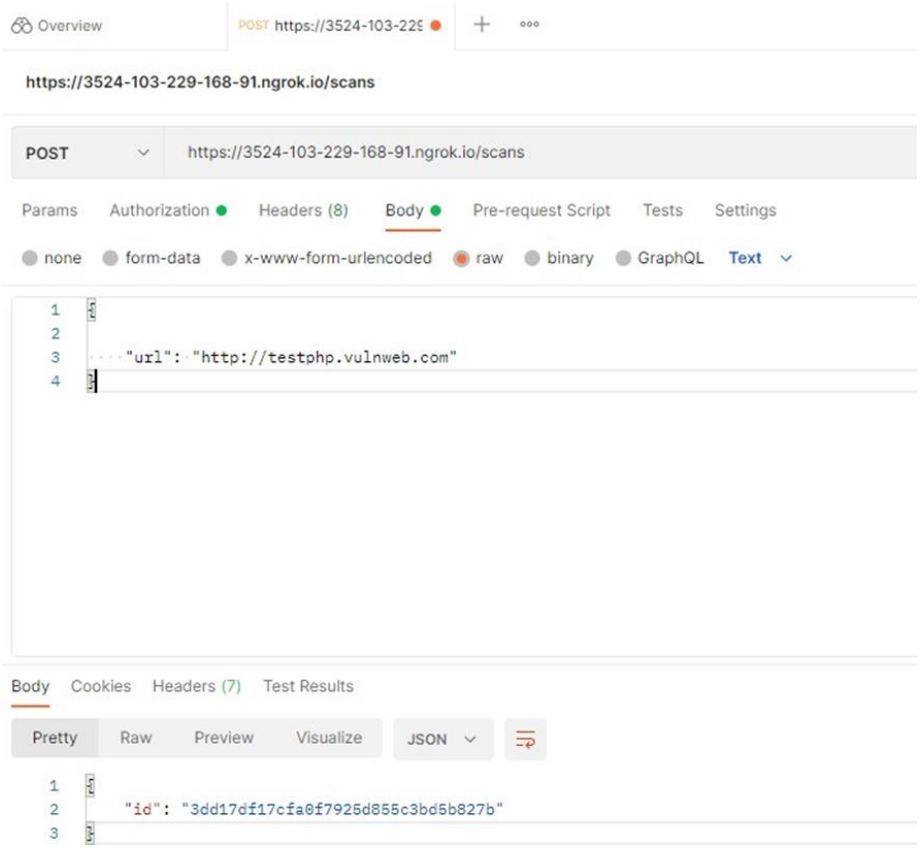

Figura 12. Petición HTTP al API REST de Arachni mediante Postman. (Fuente: Elaboración propia)

Como se puede apreciar, se realiza una petición HTTP de tipo POST al endpoint */scans,* incluyendo en el cuerpo de la petición la URL del sitio web de Acunetix. El API REST de Arachni funciona correctamente, ya que, en la respuesta de la petición se observa el identificador del escaneo iniciado.

2.6. Análisis del diseño de la aplicación

Antes de comenzar el desarrollo de la interfaz gráfica en el entorno Android Studio se crea un prototipo (*mockup*) de las diferentes pantallas de la aplicación y la interacción que existe entre ellas. Para ello, se plantea la elaboración de tres pantallas, una dedicada al menú principal, una destinada al escaneo de puertos y otra dedicada al escaneo de vulnerabilidades web. En la figura 13 se muestra la pantalla del menú principal, formada por tres botones, un texto y una imagen:

Figura 13. Mockup del menú principal de la aplicación. (Fuente: Elaboración propia)

A continuación, se realiza el prototipo de las pantallas correspondientes al escaneo de puertos y vulnerabilidades web. Se muestran en las figuras 14 y 15:

Figura 14. Mockup de la pantalla de escaneo de puertos. (Fuente: Elaboración propia)

Figura 15. Mockup de la pantalla de escaneo de vulnerabilidades web. (Fuente: Elaboración propia)

Cuando se realizan los esquemas correspondientes a cada una de las pantallas, se realiza un diagrama de interacción entre ellas y los elementos que contienen en su interior. Este diagrama se presenta en la figura 16:

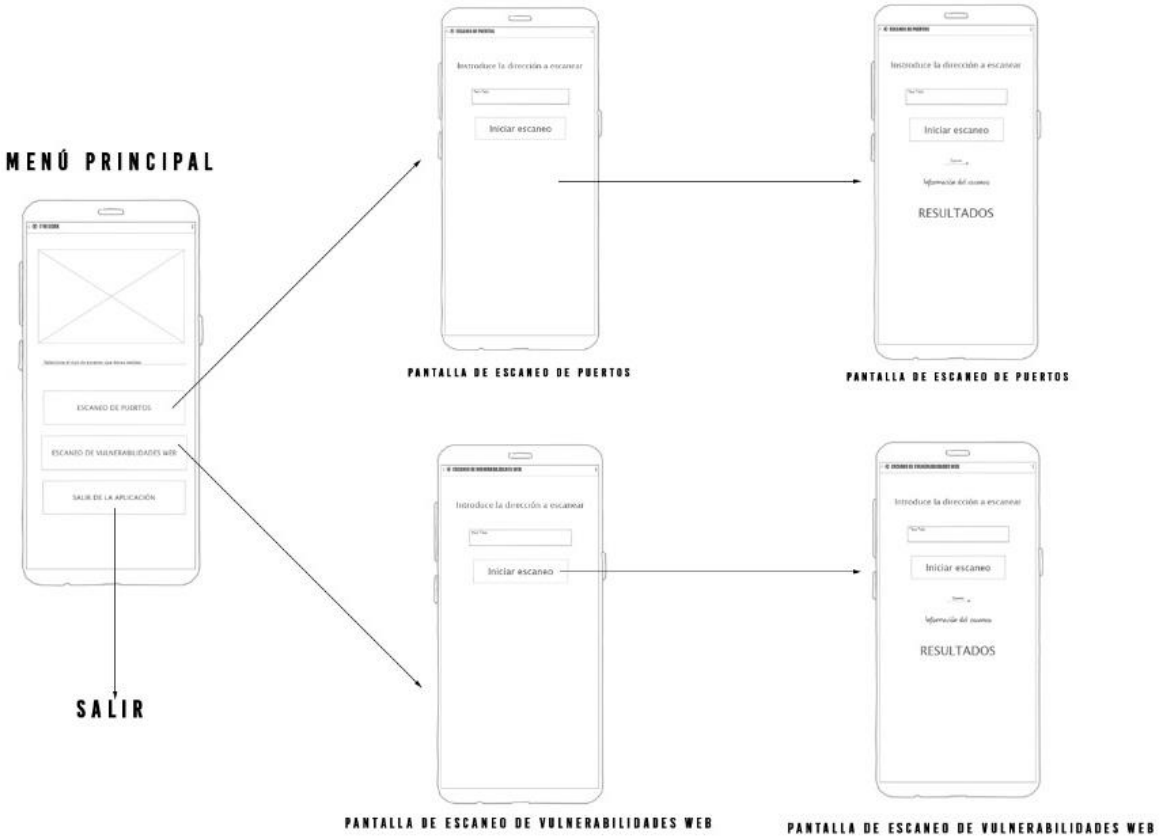

Figura 16. Diagrama de flujo del prototipo de la aplicación. (Fuente: Elaboración propia)

Una vez presentado el prototipo inicial, se procede a detallar los elementos visuales de la aplicación (vistas) y las funciones que tienen cada uno de ellos. El elemento gráfico más presente en la aplicación es el botón, el cual, tiene distintas funciones en el proyecto:

1. Permitir la navegación entre las distintas actividades de la aplicación.
2. Iniciar un nuevo escaneo, tanto de puertos como de vulnerabilidades web.
3. Salir de la aplicación.

A su vez, se utilizan distintos textos, que se utilizan para: informar al usuario de las opciones de navegación de la aplicación, proporcionar instrucciones para que la aplicación sea más intuitiva y volcar los resultados de los distintos escaneos por pantalla. Por otro lado, se dispone de una barra de progreso en forma de círculo, con el fin de que, a la hora del escaneo de puertos y vulnerabilidades web, el usuario tenga la impresión de que la aplicación está trabajando en ello.

2.6.1. Logotipo de la aplicación

A la vez que se plantea el prototipo referente al diseño de la aplicación se realiza un logotipo para que se muestre en el menú principal. Este logotipo se crea en el software de edición fotográfica Photoshop debido a la experiencia del alumno en él. La figura 17 ilustra el logo creado para la aplicación:

Figura 17. Logotipo de la aplicación. (Fuente: Elaboración propia)

3. Desarrollo del proyecto

En este capítulo se procede a desarrollar la arquitectura general de la aplicación (incluyendo la interfaz gráfica) y la implementación del código, tanto del backend como del frontend.

3.1. Arquitectura general de la aplicación

A la par que se realiza la investigación de las distintas herramientas y alternativas usadas en el proyecto, se construye la arquitectura de la aplicación móvil. Dentro de la estructura general, se dispone de tres actividades diferentes con sus tres correspondientes clases:

- Actividad principal: se corresponde con el menú de inicio, en el cual, se elige el tipo de escaneo que se quiere realizar (escaneo de puertos o escaneo de vulnerabilidades web) a través de dos botones, dónde cada botón redirecciona la aplicación a una actividad diferente. También dispone un botón para salir de la aplicación. Se muestra su interfaz gráfica en la figura 18:

Figura 18. Diseño del menú principal. (Fuente: Elaboración propia)

- Actividad dedicada al escaneo de puertos: si se desea acceder a esta actividad, en la actividad principal debe pulsar el botón referente al escaneo de puertos. Esta actividad contiene un campo de texto para introducir la dirección a escanear, un botón para iniciar el escaneo y un botón para volver a la actividad principal. Posteriormente, se vuelcan los resultados en la misma actividad. En la figura 19 se muestra la interfaz gráfica de esta actividad antes de realizar el escaneo:

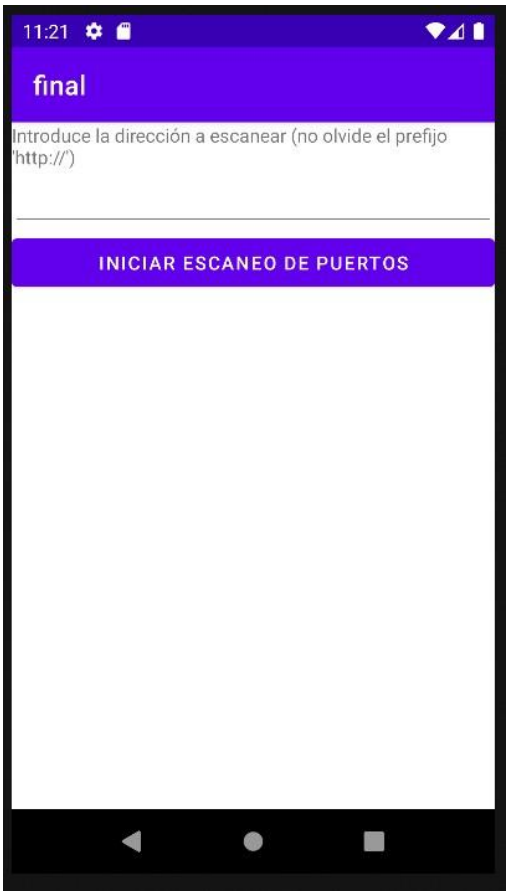

Figura 19. Diseño de la actividad de escaneo de puertos. (Fuente: Elaboración propia)

Una vez realizado el escaneo, la actividad muestra los resultados obtenidos tal y como se visualiza en la figura 20:

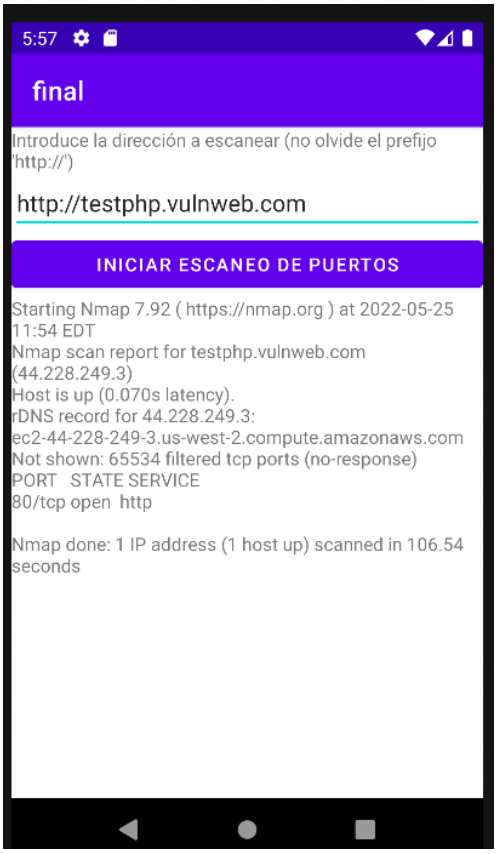

Figura 20. Resultados del escaneo de puertos en la aplicación. (Fuente: Elaboración propia)

- Actividad dedicada al escaneo de vulnerabilidades web: para acceder a esta actividad, es necesario pulsar el botón referente al escaneo de vulnerabilidades web de la actividad principal. De igual forma que la actividad dedicada al escaneo de puertos, esta actividad dispone de un campo de texto para introducir la dirección a escanear, un botón para iniciar un escaneo y un botón para regresar a la actividad principal. En la figura 21 se muestra esta actividad:

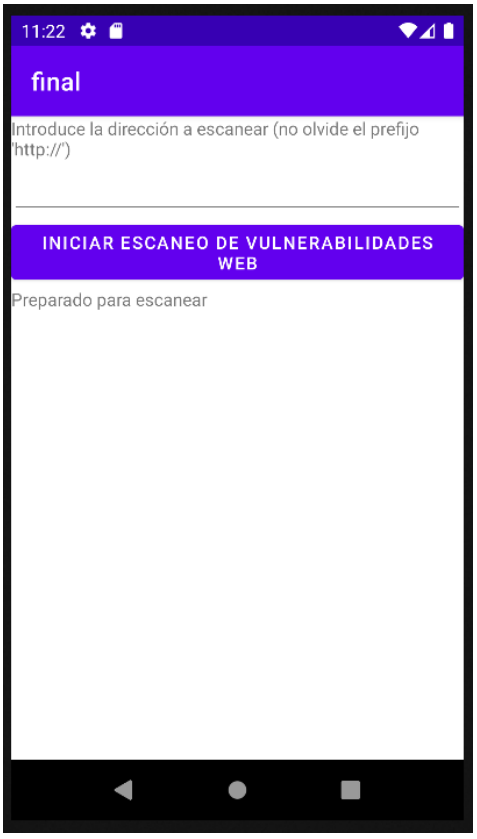

Figura 21. Diseño de la actividad de escaneo de vulnerabilidades web. (Fuente: Elaboración propia)

Una vez realizado el escaneo, la actividad correspondiente al escaneo de vulnerabilidades web muestra los resultados como se observa en la figura 22:

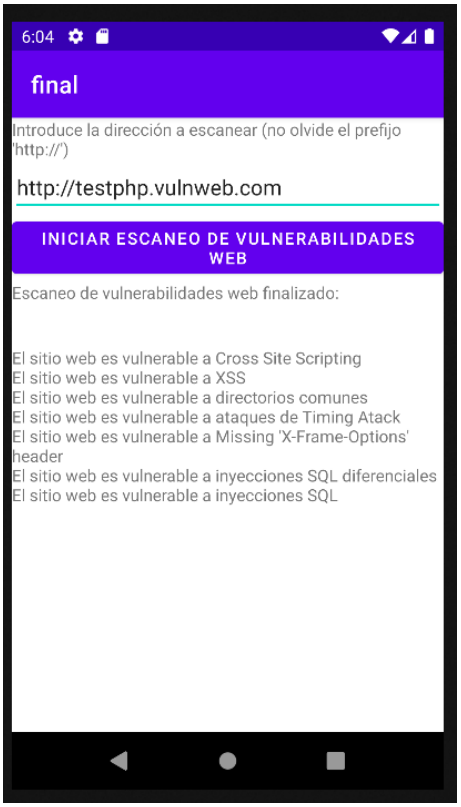

Figura 22. Resultados del escaneo de vulnerabilidades web en la aplicación. (Fuente: Elaboración propia)

Cada actividad, además de tener su correspondiente clase, dispone de código XML para el diseño de cada actividad. Cabe destacar que, para el proceso, tanto de escaneo de puertos como de vulnerabilidades web se emplean hilos independientes para no ralentizar o bloquear el hilo principal de la aplicación.

3.2. Implementación

Para poder entender correctamente el desarrollo de este proyecto en Android Studio, antes se debe de hacer referencia a dos archivos de configuración esenciales para una correcta compilación y uso de librerías: *AndroidManifest.xml y build.gradle*. Posteriormente, se explica y se detalla el código desarrollado para cada actividad.

3.2.1. Archivos de configuración Android y librerías

Todos los proyectos de aplicaciones en Android Studio deben de contener el archivo *AndroidManifest.xml*. Dentro de él se incluyen datos de gran relevancia, como pueden ser (Android, s. f.):

- Nombre del paquete de la aplicación: es de utilidad para compilar el proyecto, y cuando se empaqueta la aplicación, se sustituye por un ID único.
- Actividades, servicios, receptores de emisiones y proveedores de contenido.

- Permisos necesarios para el correcto funcionamiento de la aplicación.

Con el fin de que la aplicación tenga conexión a internet se debe añadir al archivo *AndroidManifest.xml* y queda de la siguiente manera en el proyecto:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.afinal">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Final">
        <activity
            android:name=".Escaneo_web"
            android:exported="false" />
        <activity
            android:name=".Escaneo_puertos"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

En cuanto al archivo *build.gradle*, se trata de un archivo de configuración de la compilación donde se pueden crear configuraciones de compilación personalizadas. Al crear un proyecto en Android Studio, este archivo se genera automáticamente con unos valores predeterminados, y se le pueden añadir, como en el caso del desarrollo de este proyecto, las dependencias de otras herramientas, como *OkHTTP3*. *OkHTTP3* se trata de un cliente HTTP destinado a aplicaciones Android y Java. Es utilizado para enviar solicitudes HTTP y obtener las correspondientes respuestas por parte de los API REST. La librería de OkHTTP3 se tiene que agregar al archivo de configuración, además de importar la librería en cada clase en la que se use. Se adjunta el código del archivo build.gradle:

```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 32
```

```
    defaultConfig {
        applicationId "com.example.afinal"
        minSdk 27
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {
    implementation 'com.squareup.okhttp3:okhttp:3.10.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-
core:3.4.0'
}
```

3.2.2. Hilos (threads)

Antes de comenzar a detallar todo el proceso de desarrollo del código referente a la aplicación del proyecto, se debe entender el funcionamiento de los hilos. Toda aplicación Android, usa un hilo principal para su correcta ejecución, de modo que el código de la aplicación se ejecuta dentro de él (incluyendo la interfaz gráfica). Para no ralentizar ni bloquear la ejecución del hilo principal se pueden incluir otros hilos dentro del código, que ejecuten instrucciones y subprocesos de forma paralela a éste, y una vez finalizados, vuelven para actualizar la información en el hilo principal. Es por ello, que, este proyecto utiliza hilos para realizar los diferentes procesos de escaneo de puertos y de vulnerabilidades web. A continuación, se adjunta una figura dónde se observa de una manera más gráfica la función que tienen los hilos en una aplicación:

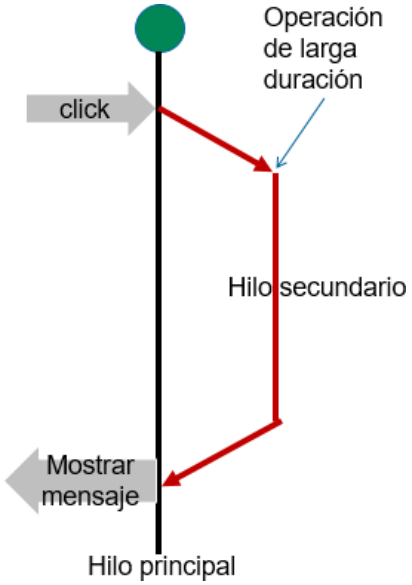

Figura 19. *Hilos en Android.* (Fuente: https://codigoonclick.com/)

3.2.3. Actividad MainActivity

Como se explica anteriormente, esta actividad actúa de menú principal de la aplicación, y a su vez, se divide en dos partes: la parte gráfica (en lenguaje XML) y la parte que permite realizar la interacción entre los distintos elementos gráficos que contiene el archivo XML correspondiente a esta actividad (en lenguaje Java). Este archivo XML queda definido de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="10dp"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="match_parent"
            android:layout_height="269dp"
            app:srcCompat="@drawable/logo" />

        <TextView
            android:id="@+id/texto_menu"
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:text="Selecciona el tipo de escaneo que quiere
realizar:"
            android:textColor="@color/black"
            android:gravity="center"
            />

        <Button
            android:id="@+id/btn_act_puertos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Escaneo de puertos" />

        <Button
            android:id="@+id/btn_act_vuln"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Escaneo de vulnerabilidades web" />

        <Button
            android:id="@+id/btn_salir_app"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Salir de la aplicación" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Como se puede apreciar, todos los elementos de esta pantalla están integrados dentro de una vista *LinearLayout*. Esta vista hace que todos los elementos que están dentro de ella se encuentren ordenados verticalmente y se respeten las proporciones y distancias en cualquier dispositivo Android, independientemente de la resolución de su pantalla. A su vez, en su interior, hay cinco vistas: tres botones, un texto y una imagen (que corresponde con el logotipo de la aplicación). Ahora, el archivo Java correspondiente a esta actividad se dividide en dos partes:

1. Importación de librerías. Es necesaria para el correcto funcionamiento y compilación del código si se hace uso de librerías en él.
2. Clase MainActivity. En ella, se llevan a cabo diferentes acciones:
    - Asociar el diseño XML a la clase MainActivity.
    - Crear los objetos Button y asociarlos a los botones correspondientes del diseño XML.
    - Detectar la pulsación de cada botón por parte del usuario para ir a una nueva actividad o para salir de la aplicación, a través del uso de intenciones.

El código Java perteneciente a esta clase queda definido de la siguiente manera:

```
package com.example.afinal;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Boton de intención a la actividad del escaneo de puertos

        Button Btn_puertos = findViewById(R.id.btn_act_puertos);
        Btn_puertos.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new
Intent(getApplicationContext(),Escaneo_puertos.class);
                startActivity(intent);
            }
        });


        //Boton de intención a la actividad del escaneo de
vulnerabilidades web
        Button Btn_vuln = findViewById(R.id.btn_act_vuln);
        Btn_vuln.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new
Intent(getApplicationContext(),Escaneo_web.class);
                startActivity(intent);
            }
        });

        //Boton de intención a la actividad del escaneo de
vulnerabilidades web
        Button Btn_salir = findViewById(R.id.btn_salir_app);
        Btn_salir.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
            }
        });
    }
}
```

3.2.4. Actividad Escaneo_puertos

Para detallar el desarrollo de código llevado a cabo en la actividad que contiene el proceso de escaneo de puertos, se procede a explicar el archivo XML. Este archivo contiene, de la misma manera que ocurre en la actividad referente al menú principal, una vista *LinearLayout* para que todos los elementos gráficos de la vista estén alienados verticalmente. Dentro de esta vista se encuentra: un texto informativo para el usuario, una caja de texto (*Edit Box*) para que el usuario introduzca la dirección a escanear, un

botón para realizar el escaneo y dos textos que sirven para mostrar al usuario información y los resultados del escaneo, respectivamente. El código correspondiente a este archivo queda de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Escaneo_puertos">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/banner_puertos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Introduce la dirección a escanear (no olvide el
prefijo 'http://')" />

        <EditText
            android:id="@+id/edit_text_puertos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
             />

        <Button
            android:id="@+id/iniciar_puertos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Iniciar escaneo de puertos" />

        <ProgressBar
            android:id="@+id/progressBar"
            style="?android:attr/progressBarStyle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            tools:visibility="gone" />

        <TextView
            android:id="@+id/texto_estado"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text=""

            tools:visibility="gone" />

        <TextView
            android:id="@+id/report_puertos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="" />

        <TextView
            android:id="@+id/nmap_info"
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:text="" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Ahora, se procede a explicar el código referente a la clase vinculada a esta actividad. De igual forma que en la actividad principal, se deben de importar una serie de librerías, entre ellas, la librería *OkHttp3* usada para la realización de peticiones HTTP a los APIs REST, para un correcto funcionamiento de esta actividad:

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.IOException;
import okhttp3.FormBody;
import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import java.util.concurrent.TimeUnit;
import okhttp3.RequestBody;
import okhttp3.Response;
```

Una vez que se importan las librerías se trabaja el código correspondiente a la clase Escaneo_puertos. Esta clase está formada por cuatro funciones:

Función nmap_new_scan()

Esta función está dedicada a realizar una petición HTTP de tipo POST al API REST de Nmap a través del cliente *OkHttp3*, con el fin de realizar un nuevo escaneo. Para ello, a través de un formulario HTTP se establecen una serie de parámetros necesarios para iniciar el escaneo:

- *scan_type*. Hace referencia al tipo de escaneo que se quiera realizar: *single* (solamente escanea una dirección), *range* (escanea un rango de direcciones). En este parámetro se le ha asignado un escaneo de una dirección.
- *command.* En este parámetro se puede configurar un escaneo normal, rápido, de puertos o de detección de sistema operativo, entre otros. En este caso, se realiza un escaneo de puertos para que a la salida no se muestre información irrelevante. Se muestran los diferentes comandos que ofrece *Nmap Online* en la figura 23:

```
normal - Normal Scan (nmap host)
fast - Fast Scan
ping - Ping scan
port - Port scan
osinfo - Scan OS information and Traceroute
osdetect - OS Detection
fcrdns - Forward-confirmed Reverse DNS (Paid users only)
firewall - Firewall Detection (Paid users only)
top5port - Scan top 5 ports (Paid users only)
top20port - Scan top 20 ports (Paid users only)
top20tcp - Scan top 20 TCP ports (Paid users only)
top20udp - Scan top 20 UDP ports (Paid users only)
malware - Detecting malware infections (Paid users only)
```

Figura 23. Comandos de Nmap Online. (Fuente: https://api.nmap.online/)

- *options*. Este parámetro hace referencia a las opciones del escaneo. *Nmap Online* ofrece una gran variedad de opciones, pero en este proyecto, se opta por la opción *p*. Se muestran las diferentes opciones que ofrece *Nmap Online* en la figura 24:

```
pn: No ping (-Pn)
sn: No port scan (host discovery only) (-sn)
f: Fast scan (-F)
p: All ports (-p-)
n: no DNS lookup (-n)
r: reverse-DNS lookup for all hosts (-R)
f8: Fragment IP data into 8 bytes (-f)
ff16: Fragment IP data into 16 bytes (-ff)
v: Verbose output (-v)
vv: More verbosity (-vv)
d: Debugging details (-d)
dd: More details (-dd)
pr: ARP Scan (-PR)
ss: TCP SYN scan (-sS)
st: TCP connect scan (-sT)
su: UDP scan (-sU)
snn: Null Scan (-sN)
sff: FIN Scan (-sF)
sxx: Xmas Scan (-sX)
sa: TCP ACK Scan (-sA)
o: OS Detection (-O)
sc: Default scripts (-sC)
t0: Slowest (paranoid) (-T0)
t1: Sneaky (-T1)
t2: Polite (-T2)
t3: Normal (-T3)
t4: Aggressive (-T4)
t5: Fastest (insane) (-T5)
sv: Service Detection (-sV)
sv2: Service Detection (intensity of 2) (-sV --version-light)
sv9: Service Detection (intensity of 9) (-sV --version-all)
reason: Getting More Details (--reason)
```

Figura 24. Opciones de Nmap Online. (Fuente: https://api.nmap.online/)

- *schedule*. Se puede configurar un escaneo para realizarlo al momento o dejarlo programado, pero en este proyecto, el escaneo se realiza al momento.

- *target.* En este parámetro se introduce la dirección IP que se desea escanear. La dirección para escanear es recogida por el *Edit Box* insertado en el archivo XML

También, se le tienen que pasar parámetros como el *content-type* de la respuesta a la petición (en este caso es *multipart/form-data)* o la clave API obtenida anteriormente.

Al realizar esta petición HTTP POST, Nmap devuelve siete posibles respuestas en formato *json*, pero todas son de códigos de errores a la hora de realizar el escaneo menos la que tiene el código 200. La respuesta con el código 200 incluye el identificador del escaneo, dentro del parámetro *scan_id* en la respuesta en formato json, que posteriormente se guarda en una variable llamada *nmap_scan_id*. Se adjunta el código correspondiente a esta función:

```
private void nmap_new_scan() {
    OkHttpClient client = new OkHttpClient();
    MediaType JSON = MediaType.parse("multipart/form-data;charset=utf-
8");
    JSONObject actualData = new JSONObject();
    RequestBody body = new FormBody.Builder()
            .add("scan_type", "single")
            .add("command", "normal")
            .add("options", "p")
            .add("schedule", "now")
            .add("target", direccion_ip)
            .add("target_end", "")
            .build();

    Request request = new Request.Builder()
            .url("https://api.nmap.online/v01/start_scan")
            .post(body)
            .addHeader("content-type", "multipart/form-data")
            .addHeader("NMAP-API-KEY",
"oswshgi7g1nv4eqh1yxewosnqki6z9njs3nuavo7hrlrs0cz")
            .build();
    try {
        Response response = client.newCall(request).execute();
        JSONObject json = new JSONObject(response.body().string());
        nmap_scan_id = json.getString("scan_id");
        System.out.println(nmap_scan_id);


    } catch (IOException | JSONException e) {
        e.printStackTrace();
    }


}
```

Función getnmapstatus()

Una vez que se obtiene el identificador del escaneo iniciado, se procede a verificar el estado del escaneo. Esta función se encarga de realizar una petición HTTP de tipo POST al API REST de Nmap de la misma forma que la función *nmap_new_scan()*, pero en el formulario solamente se le añade el parámetro *scan_id*, con el valor del identificador del escaneo iniciado. La respuesta a esta petición por parte del API REST también se realiza en formato. *json*, dónde se encuentra el parámetro de interés: *scan_id*. El estado del escaneo consta de tres etapas*: Starting soon, Scanning y Finished*, y una vez que se obtiene se guarda en la variable *nmap_scan_id.* El código correspondiente a la función *getnmapstatus()* es el siguiente:

```
private void getnmapstatus() {
        OkHttpClient client = new OkHttpClient();

        RequestBody body = new FormBody.Builder()

                //Se añade en el cuerpo de la petición el id del escaneo
recibido por la función nmap_new_scan()

                .add("scan_id", nmap_scan_id)
                .build();

        Request request = new Request.Builder()
                .url("https://api.nmap.online/v01/check_scan_status")
                .post(body)
                .addHeader("content-type", "multipart/form-data")
                .addHeader("NMAP-API-KEY",
"oswshgi7g1nv4eqh1yxewosnqki6z9njs3nuavo7hrlrs0cz")
                .build();

        try {
            Response response = client.newCall(request).execute();
            JSONObject json = new JSONObject(response.body().string());
            estado_escaneo = json.getString("scan_status");
            System.out.println (estado_escaneo);
            if (estado_escaneo.contains("Finished")){
                finished = true;
            }

        } catch (IOException | JSONException e) {
            e.printStackTrace();
        }


    }
}
```

Función getnmapresults()

Esta función es la encargada de recoger los resultados del escaneo de puertos una vez que el estado del escaneo sea *Finished.* Para ello, se vuelve a realizar una petición HTTP POST de la misma manera que en la función anterior. Lo único que diferencia esta función de la anterior es el *endpoint* de la petición y la recogida de resultados. Como resultado de esta petición se obtiene una respuesta *json*, que contiene en su interior los resultados del escaneo dentro del parámetro *results*. Los resultados del escaneo se guardan en la variable *resultado*:

```
private void getnmapresults() {
    OkHttpClient client = new OkHttpClient();

    RequestBody body = new FormBody.Builder()
            .add("scan_id", nmap_scan_id)
            .build();

    Request request = new Request.Builder()
            .url("https://api.nmap.online/v01/scan_result")
            .post(body)
            .addHeader("content-type", "multipart/form-data")
            .addHeader("NMAP-API-KEY",
"oswshgi7g1nv4eqh1yxewosnqki6z9njs3nuavo7hrlrs0cz")
            .build();


    try {
        Response response = client.newCall(request).execute();
        JSONObject json = new JSONObject(response.body().string());
        resultado = json.getString("result");
       // System.out.println(resultado.substring(0, 310));
        System.out.println(resultado);
    } catch (IOException | JSONException e) {
        e.printStackTrace();
    }

}
```

Método onCreate

Este método es llamado justo al lanzarse la actividad y es donde se desarrolla el código correspondiente a las principales funciones de la actividad. Al comienzo de este método, se instancian los objetos referentes a los elementos visuales declarados en el archivo XML de la acticidad. Después de ello, el método espera a que se produzca una pulsación en el botón que inicia el escaneo y se inicia un nuevo hilo.

Dentro del hilo iniciado, se llama a la función nmap_new_scan() para comenzar un nuevo escaneo y se muestran por pantalla diferentes textos. Una vez que ha iniciado el escaneo, se activa la barra de progreso para darle a entender al usuario que se está realizando un escaneo y, mediante un bucle while se llama cada cuarenta segundos a la función getnmapstatus(), hasta que la variable que obtiene el estado del escaneo

adquiera el valor “Finished”. Una vez que el escaneo finaliza, se oculta la barra de progreso y se muestra por pantalla el resultado del escaneo. Se adjunta, a continuación, el código correspondiente a este método:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_escaneo_puertos);

    //  Se crean los objetos de los elementos de la interfaz gráfica para
conseguir su interacción

    EditText direccion_escaneo = findViewById(R.id.edit_text_puertos);
    Button Btn_inicio_puertos = findViewById(R.id.iniciar_puertos);
    ProgressBar progress_puertos = findViewById(R.id.progressBar);
    TextView Estado_escaneo = findViewById(R.id.texto_estado);
    TextView resultado_p = findViewById(R.id.report_puertos);
    TextView nmap_info = findViewById(R.id.nmap_info);
    progress_puertos.setVisibility(View.GONE);

    //  Se añade un listener en el botón que inicia el escaneo

    Btn_inicio_puertos.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            direccion_ip = direccion_escaneo.getText().toString();
    //  Una vez que se pulsa el botón para iniciar el escaneo se genera
un hilo independiente

            new Thread(new Runnable() {
                public void run() {




    //  Una vez dentro del hilo, se llama a la función que inicia un
nuevo escaneo

                    nmap_new_scan();

                     runOnUiThread(new Runnable() {
                         @Override
                         public void run() {
                             Estado_escaneo.setVisibility(View.VISIBLE);
                             Estado_escaneo.setText("Iniciando escaneo de
puertos...");
                             Estado_escaneo.setVisibility(View.GONE);;
                             try {
                                 TimeUnit.SECONDS.sleep(4);
                             } catch (InterruptedException e) {
                                 e.printStackTrace();
                             }
                             nmap_info.setText("Escaneando la dirección: "
+ direccion_ip );

progress_puertos.setVisibility(View.VISIBLE);;
                         }});

    //  Posteriormente, en un bucle, se llama a la función que verifica
el estado del escaneo
    //  Se verifica el estado del escaneo cada 20 segundos, hasta que el
```

```
estado del escaneo sea "Finished"

                while (finished==false) {
                    getnmapstatus();
                    nmap_info.setText("Estado del escaneo: " +
estado_escaneo);
                    try {
                        TimeUnit.SECONDS.sleep(40);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }



    // Una vez que el escaneo esté en estado "Finalizado", se llama a la
función getnmapresults() para obtener los resultados del escaneo

                    getnmapresults();
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            nmap_info.setVisibility(View.GONE);;
                            progress_puertos.setVisibility(View.GONE);;
                            resultado_p.setText(resultado);

                        }

                    });

                }

            }).start();


        }
    });

}
```

3.2.5. Actividad Escaneo_web

Esta actividad funciona de manera similar a la actividad Escaneo_puertos. En cuanto al archivo de diseño XML, contiene una vista *LinearLayout* para que todos los elementos gráficos de la vista estén alienados verticalmente. Dentro de esta vista se encuentra: un texto informativo para el usuario, una caja de texto (*Edit Box*) para que el usuario introduzca la dirección a escanear, un botón para realizar el escaneo y dos textos que sirven para mostrar al usuario información y los resultados del escaneo, respectivamente. El código correspondiente a este archivo queda de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Escaneo_web">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/banner_web"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Introduce la dirección a escanear (no olvide el
prefijo 'http://')" />

        <EditText
            android:id="@+id/edit_text_web"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            />

        <Button
            android:id="@+id/iniciar_web"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Iniciar escaneo de vulnerabilidades web" />

        <ProgressBar
            android:id="@+id/progressBar"
            style="?android:attr/progressBarStyle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            tools:visibility="gone" />

        <TextView
            android:id="@+id/texto_estado"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text=""

            tools:visibility="gone" />

        <TextView
            android:id="@+id/report_web"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="" />

        <TextView
            android:id="@+id/report_web_info"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

De la misma forma que sucede en la actividad Escaneo_puertos, la clase Escaneo_web comienza con la importación de todas las librerías necesarias para la correcta compilación de la aplicación. A su vez, contiene en su interior cuatro funciones:

Función arachni_new_scan()

Es la función encargada de realizar un nuevo escaneo a través de una petición HTTP de tipo POST al *endpoint /scans* mediante el uso de la librería OkHTTP3. Al cuerpo del formulario, se le deben añadir algunos parámetros del escaneo como la dirección a escanear o los elementos que se desean auditar en la web a escanear. Como respuesta a esta petición, se obtiene el identificador del escaneo realizado y se guarda en la variable arachni_scan_id para la posterior recogida de resultados. A continuación, se adjunta el código correspondiente a esta función:

```
    private void arachni_new_scan() {
    OkHttpClient client = new OkHttpClient.Builder().build();
    MediaType textPlainMT = MediaType.parse("text/plain; charset=utf-8");
    String input_json = "{\"url\": \"" + direccion_ip + "\", \"checks\":
[\"*\"],\"audit\": {\n" +
            "         \"elements\": [\"link\", \"form\", \"cookie\"]\n" +
            "     }\n}";

    Request request = new Request.Builder().url("https://5e09-103-229-
168-91.ngrok.io/scans")
            .addHeader("Authorization", Credentials.basic("arachni",
"password"))
            .post(RequestBody.create(textPlainMT, input_json)).build();


    try {
        Response response = client.newCall(request).execute();
        JSONObject json = new JSONObject(response.body().string());
        arachni_scan_id = json.getString("id");
        System.out.println(arachni_scan_id);

    } catch (IOException | JSONException e) {
        e.printStackTrace();
    }


}
```

Función getarachnistatus()

Al igual que sucedía con Nmap, con el identificador del escaneo realizado se procede a consultar el estado en el que se encuentra. Mediante una petición HTTP de tipo GET se realiza una petición al endpoint */scans/:identificador* a través de la librería OkHTTP3. Como respuesta a esta petición, se obtiene el estado del escaneo. A continuación, se adjunta el código correspondiente a esta función:

```
private void getarachnistatus() {
    OkHttpClient client = new OkHttpClient();
```

```
    Request request = new Request.Builder()
            .url("https://5e09-103-229-168-91.ngrok.io/scans/" +
arachni_scan_id)
            .addHeader("Authorization", Credentials.basic("arachni",
"password"))
            .build();


    try {
        Response response = client.newCall(request).execute();
        JSONObject json = new JSONObject(response.body().string());
        estado_escaneo_arachni = json.getString("status");
        System.out.println(estado_escaneo_arachni);

    } catch (IOException | JSONException e) {
        e.printStackTrace();
    }

}
```

Función getarachnireport()

Esta función se encarga de recoger los resultados del escaneo a través de una petición HTTP de tipo GET al endpoint */scans/:identificador/report*. Como respuesta a la petición se obtiene un archivo en formato *json* con todos los resultados, pero también contiene información irrelevante. Como solución, se observa que, dentro del archivo de respuesta, las vulnerabilidades encontradas se encuentran dentro del parámetro *issues*, y dentro de ese parámetro, cada vulnerabilidad está definida de la siguiente forma: *"shortname": "vulnerabilidad*".

Por tanto, una vez que se recogen los resultados, se observa dentro del parámetro *issues* y se realizan distintos condicionales con el fin de encontrar dentro de él las vulnerabilidades detectadas. Dentro de cada condicional, si se encuentra una vulnerabilidad dentro del archivo de respuesta, se añade a la variable que alimenta a la vista de texto que contiene el resultado del escaneo en la aplicación. Se adjunta el código correspondiente a esta función:

```
private void getarachnireport() {
    OkHttpClient client = new OkHttpClient();


    Request request = new Request.Builder()
            .url("https://5e09-103-229-168-91.ngrok.io/scans/" +
arachni_scan_id + "/report")
            .addHeader("Authorization", Credentials.basic("arachni",
"password"))
            .build();
```

```
    try {
        Response response = client.newCall(request).execute();
        JSONObject json = new JSONObject(response.body().string());
        report_escaneo_arachni = json.getString("issues");
        System.out.println (report_escaneo_arachni);
        //System.out.println ("Aqui tiene");

        if
(report_escaneo_arachni.contains("\"shortname\":\"xss_script_context\""))
{
            //texto_report_escaneo_arachni = "Cross";
            //System.out.println("Vuln backup files");
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a Cross
Site Scripting\n");
            //cross_site = true;
        }

        if (report_escaneo_arachni.contains("\"shortname\":\"CSRF\"")){
            //texto_report_escaneo_arachni = "Cross";
            //System.out.println("Vuln backup files");
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a Cross-
Site Request Forgery\n");
            //cross_site = true;
        }

        if
(report_escaneo_arachni.contains("\"shortname\":\"unvalidated_redirect_do
m\"")){
            //texto_report_escaneo_arachni = "Cross";
            //System.out.println("Vuln backup files");
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
Unvalidated DOM redirect\n");
            //cross_site = true;
        }

        if (report_escaneo_arachni.contains("\"shortname\":\"xss\"")){
            //texto_report_escaneo_arachni = "Cross";
            System.out.println("Vuln backup files");
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a XSS\n");
            //cross_site = true;
        }


        if (report_escaneo_arachni.contains("\"shortname\":
\"xss_dom_script_context\"")){
            //texto_report_escaneo_arachni = "Cross";
            //System.out.println("Vuln backup files");
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a DOM XSS
in script context\n");
            //cross_site = true;
        }

        if
(report_escaneo_arachni.contains("\"shortname\":\"common_directories\""))
{
            //texto_report_escaneo_arachni = "Cross";
            //System.out.println("Vuln backup files");
```

```
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
directorios comunes\n");
            System.out.println("jeje");
            //cross_site = true;
        }

        if
(report_escaneo_arachni.contains("\"shortname\":\"unencrypted_password_fo
rms\"")){
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
Unencrypted password form\n");

        }

        if
(report_escaneo_arachni.contains("\"shortname\":\"code_injection_timing\"
")){
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a ataques
de Timing Atack\n");
        }
        if
(report_escaneo_arachni.contains("\"shortname\":\"xss_dom_script_context\
""")){
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a ataques
DOM Script Context\n");        }
        if
(report_escaneo_arachni.contains("\"shortname\":\"insecure_cors_policy\""
)){            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a Insecure
'Access-Control-Allow-Origin' header\n");
        }

        if
(report_escaneo_arachni.contains("\"shortname\":\"password_autocomplete\"
")){            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a Password
field with auto-complete\n");
        }
        if
(report_escaneo_arachni.contains("\"shortname\":\"x_frame_options\"")){

            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a Missing
'X-Frame-Options' header\n");
                    }
        if
(report_escaneo_arachni.contains("\"shortname\":\"no_sql_injection\"")){
                        report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
inyecciones No SQL\n");
        }
        if
(report_escaneo_arachni.contains("\"shortname\":\"no_sql_injection_differ
ential\"")){
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
inyecciones No SQL diferenciales\n");        }
```

```
        if
(report_escaneo_arachni.contains("\"shortname\":\"session_fixation\"")){
                        report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a ataques
de fijación de sesiones\n");         }

        if
(report_escaneo_arachni.contains("\"shortname\":\"xpath_injection\"")){

            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a Xpath
injection\n");
        }

        if
(report_escaneo_arachni.contains("\"shortname\":\"sql_injection_different
ial\"")){
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
inyecciones SQL diferenciales\n");         }
        if
(report_escaneo_arachni.contains("\"shortname\":\"os_cmd_injection\"")){
                        report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
inyecciones de comando (CMD Injections)\n");         }
        if
(report_escaneo_arachni.contains("\"shortname\":\"xss_dom\"")){
            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a DOM
Based Cross Site Scripting\n");         }
        if
(report_escaneo_arachni.contains("\"shortname\":\"sql_injection\"")){

            report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a
inyecciones SQL\n");
        }
        if
(report_escaneo_arachni.contains("\"shortname\":\"file_inclusion\"")){
                        report_escaneo_arachni_info =
report_escaneo_arachni_info.concat("El sitio web es vulnerable a File
Inclusion\n");
        }
    } catch (IOException | JSONException e) {
        e.printStackTrace();
    }

}
```

Método onCreate

Este método es llamado justo al lanzarse la actividad y es donde se desarrolla el código correspondiente a las principales funciones de la actividad. Al comienzo de este método, se instancian los objetos referentes a los elementos visuales declarados en el archivo XML de la actividad. Después de ello, el método espera a que se produzca una pulsación en el botón que inicia el escaneo y se inicia un nuevo hilo.

Una vez que se inicia el hilo, se llama a la función arachni_new_scan() y en vez de verificar el estado del escaneo como sucedía con el escaneo de puertos, se añade un retraso de trescientos segundos (media de duración de los escaneos con Arachni). Esto se debe a que, consultando el foro de discusiones de Arachni, se observa que, si se realizan distintas peticiones cada cierto tiempo para consultar información de un mismo identificador, el escaneo dejaba de funcionar correctamente.

Una vez que se han cumplido los trescientos segundos, se llama a la función *getarachnireport()* y se muestran los resultados en una vista de texto de la actividad. Se adjunta el código correspondiente a este método:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_escaneo_web);

    EditText direccion_escaneo = findViewById(R.id.edit_text_web);
    Button Btn_inicio_web = findViewById(R.id.iniciar_web);
    ProgressBar progress_web = findViewById(R.id.progressBar);
    progress_web.setVisibility(GONE);
    TextView arachni_text_report = findViewById(R.id.texto_estado);
    TextView arachni_text_report_info=
(TextView)findViewById(R.id.report_web_info);
    arachni_text_report.setText("Preparado para escanear");
    //  Se añade un listener en el botón que inicia el escaneo web
    Btn_inicio_web.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            direccion_ip = direccion_escaneo.getText().toString();
            //  Una vez que se pulsa el botón para iniciar el escaneo se
genera un hilo independiente

            new Thread(new Runnable() {
                public void run() {
                    arachni_new_scan();
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            progress_web.setVisibility(View.VISIBLE);
                        }});


                    arachni_text_report.setText("Escaneando la dirección:
" + direccion_ip );
                    try {
                        TimeUnit.SECONDS.sleep(300);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                    //getarachnistatus();
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            progress_web.setVisibility(GONE);
                        }});

                    getarachnireport();
```

```
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            arachni_text_report.setText("Escaneo de
vulnerabilidades web finalizado:\n");

arachni_text_report_info.setText(report_escaneo_arachni_info);

                        }

                    });
                }

            }).start();


        }
    });

}
```

4. Conclusiones

En el desarrollo de este proyecto se demuestra que se puede realizar una aplicación móvil Android basada en la detección y análisis de puertos y de vulnerabilidades web mediante un escaneo a través del uso de diferentes herramientas ya existentes a un bajo coste. La infraestructura presentada al principio del proyecto se sigue manteniendo hasta el final del desarrollo, siendo, la alternativa menos compleja y costosa que se ha barajado durante su transcurso.

Las herramientas de escaneo utilizadas en el proyecto son las más populares en el ámbito de la seguridad, además de tener una amplia comunidad, por lo que, siguen teniendo soporte y actualizaciones periódicas. Las actualizaciones en el API REST de Nmap se realizan automáticamente, y en el caso de Arachni, se tiene que actualizar el contenedor cada vez que exista una nueva actualización de la herramienta.

Cumplimiento de objetivos

Los objetivos principales del proyecto propuestos en este documento se han cumplido en su totalidad. Se ha creado un prototipo que cumple con las expectativas esperadas implementando herramientas y técnicas de ciberseguridad en un soporte móvil. El escaneo de puertos permite obtener los puertos abiertos de un host con sus respectivos servicios, y el escaneo de vulnerabilidades web arroja las vulnerabilidades más comunes que puede tener un servicio web. Por otro lado, se ha adquirido conocimiento sobre los API REST, su consumo y su implementación en sistemas operativos Android. En cuanto a la metodología Agile planteada en la introducción de la memoria, se han cumplido y respetado los tiempos en la medida de lo posible, lo que ha supuesto un flujo de trabajo acorde a lo esperado.

Los objetivos secundarios también se han llevado a cabo satisfactoriamente. Se han encontrado API REST que permiten satisfacer el objetivo general del proyecto, como son Nmap Online o Arachni. A su vez, se ha conseguido ejecutar Arachni haciendo uso de contenedores y se ha conseguido plasmar los resultados de los escaneos de una forma clara y precisa.

Riesgos presentados y asumidos

En cuanto a los riesgos presentados en la introducción de este documento, la mayoría se han conseguido solventar. Al finalizar el proyecto, se tiene un conocimiento mucho más amplio sobre los API REST y su funcionamiento, sin tener la necesidad de recortar demasiado tiempo de otros procesos de la fase de planificación. A la hora de implementar el consumo de los API REST en Android, surgen problemas de

compatibilidad entre distintas librerías, que se consiguen solventar mediante la mitigación establecida para ese riesgo: empezar realizando peticiones más sencillas, hasta llegar a las requeridas por los objetivos del proyecto. En cuanto a la recogida de los resultados de los escaneos a través de las respuestas de los API REST, no ha ocurrido ningún problema que no se haya podido solventar dentro del tiempo establecido en la fase de planificación. El único riesgo que no se ha podido mitigar del todo es el que está relacionado con la estética de la aplicación, ya que, es un punto que siempre se puede mejorar y debido a que se ha empleado tiempo de más para otros aspectos más importantes, no se ha podido perfeccionar todo lo que se desea.

5. Trabajo futurible

Mientras que se desarrolla el proyecto surgen ideas de mejora del proyecto que debido al tiempo establecido en la fase de planificación no ha sido posible llevarlas a cabo:

Persistencia de datos

Sería interesante dotar a la aplicación móvil de persistencia de datos mediante una base de datos que almacenara los escaneos realizados por cada usuario. Esta mejora consistiría en añadir una pantalla de inicio de sesión con el fin de que cada usuario tenga disponible un portal con toda la información de cada escaneo realizado para que se pueda visualizar en el momento que se desee.

Opciones de escaneo de puertos

En el caso del escaneo de puertos, Nmap ofrece múltiples configuraciones a la hora de realizar un nuevo escaneo. Podría ser de gran utilidad que, antes de realizar un escaneo, en vez de solo solicitar la dirección a escanear al usuario, se le diera la opción de elegir configuraciones como el tipo del escaneo a realizar o las opciones del escaneo, ya que no todos los usuarios que utilizan esta herramienta de escaneo de puertos utilizan las mismas configuraciones. En relación con la persistencia de datos propuesta anteriormente, se plantea la posibilidad de crear un menú de ajustes con las preferencias del usuario en cuanto a opciones de escaneo.

Tiempo de escaneo de vulnerabilidades web

En el capítulo de desarrollo se comenta que en el API REST de la herramienta que realiza el escaneo de vulnerabilidades web no es aconsejable realizar múltiples peticiones a un identificador de escaneo con el fin de verificar el estado de este, ya que, deja de realizar el escaneo correctamente. Se plantea, como trabajo futurible,

encontrar una forma de determinar que el escaneo ha finalizado sin influir en su funcionamiento, al igual que ocurre con el escaneo de puertos.

Notificaciones

Uno de los elementos más implementados en las aplicaciones móviles son las notificaciones. Sería interesante que, después de iniciar un nuevo escaneo, la aplicación siguiese funcionando en segundo plano, para que el usuario no tenga que esperar dentro de ella. Una vez finalizado el escaneo, se lanzaría una notificación para que el usuario pudiese visualizar resultados del escaneo.

Traducción a otros idiomas

La aplicación móvil desarrollada en el proyecto podría ser utilizada por más usuarios si se incluyese una traducción a otros idiomas. Es una mejora que no conllevaría demasiado tiempo implementarla y aportaría comodidad a usuarios que no sean de habla hispana.

6. Bibliografía

Laskos, T. (s. f.). *REST API · Arachni/arachni Wiki*. GitHub. https://github.com/Arachni/arachni/wiki/REST-API

Arachni Scanner. (2022, 21 febrero). Arachni - Web Application Security Scanner Framework. https://www.arachni-scanner.com/

Nmap - Free Security Scanner For Network Exploration & Security Audits. (s. f.). Nmap. https://nmap.org/docs.html

HostingPlus. (2021, 9 noviembre). *Qué es NMAP y para qué sirve | Blog | Hosting Plus Perú*. https://www.hostingplus.pe/blog/que-es-nmap-y-para-que-sirve/

Android. (s. f.). *Arquitectura de la plataforma | Desarrolladores de Android |*. Android Developers. https://developer.android.com/guide/platform?hl=es-419

https://androidos.readthedocs.io/_/downloads/en/latest/pdf/

Ramírez, I. (2022, 9 enero). *Historia y evolución de Android: cómo un sistema operativo para cámaras digitales acabó conquistando los. . .* Xataka Android. https://www.xatakandroid.com/sistema-operativo/historia-evolucion-android-como-sistema-operativo-para-camaras-digitales-acabo-conquistando-moviles

NetApp. (s. f.). *¿Qué son los contenedores? - Ventajas y casos prácticos | NetApp*.https://www.netapp.com/es/devops-solutions/what-are-containers/

Corbetti, S. (2021, 17 mayo). *Cómo funciona Docker: estructura y funcionamiento en detalle*. The Solving. https://thesolving.com/es/contenerizacion/como-funciona-docker-estructura-y-funcionamiento/

Android Developers. (s. f.). *Descripción general del manifiesto de una app | Desarrolladores de Android |*. https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419