

Desarrollo de herramientas de seguridad informática para Android

Memoria del Trabajo de Fin de Grado

presentada para optar al Grado de

Ingeniería Informática de Gestión y Sistemas de Información

por

Ander Granado Masid

Director: Pablo González Nalda

2 de septiembre de 2018

Copyright ©2018 ANDER GRANADO MASID. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Agradecimientos

Gracias a mis padres, que siempre han sido un apoyo fundamental a todos los niveles. Sin ellos no habría llegado hasta donde estoy.

Gracias a mi hermana, que junto a mis padres, ha estado siempre ahí, tanto en las buenas como en las malas, soportándome y aguantándome. Apoyándome y dándome fuerzas.

Gracias a Sergio y Alberto, dos grandes amigos que siempre han sabido apoyarme hasta en los momentos más bajos. Ahora que estoy a punto de graduarme, veo más claro que no habría llegado hasta este punto si no llega a ser, en parte, por ellos. Nunca han dudado de que lo conseguiría y me alegra haber podido llegar hasta aquí para demostrárselo.

Gracias a todos los maravillosos compañeros que he conocido durante estos años, tanto a los que se fueron como a los que aguantaron el duro camino. En especial a Julen, Joseba e Iñaki. A Julen por haberme acompañado durante toda la carrera, a la cual hemos hecho frente juntos año tras año. A Joseba e Iñaki por todo el trabajo hecho juntos y por todas las experiencias vividas, que me han marcado tanto a nivel profesional y personal.

Gracias a mis amigos. En especial a Jun, una buena amiga que ha estado siempre apoyándome, aunque en ocasiones me intente distraer de mis estudios. También gracias a Laura, gran amiga y compañera de biblioteca, cuya insistencia en cotillear este informe ha permitido que tenga muchas menos erratas. Gracias a los amigos que han estado en las largas horas de estudio o trabajo y también a los que han estado en los momentos de ocio, fundamentales para coger fuerzas y continuar con el proceso.

Gracias a todos los profesores que me han enseñado, aportado y apoyado durante todo este trayecto. En especial a Pablo, el tutor de este TFG, que me ha servido como guía y ha tenido especial paciencia conmigo durante todo este proceso hasta el final.

Por último, muchísimas gracias a Juana, que se ha convertido en alguien muy especial para mí en muy poco tiempo y ha estado dándome fuerzas para este último sprint de la carrera.

Índice general

Agradecimientos	v
Resumen	xix
I. Alcance del Trabajo	1
1. Descripción, Objetivos y Motivación	3
1.1. Descripción	3
1.2. Objetivos	3
1.3. Motivación	4
2. Viabilidad	5
2.1. Requisitos funcionales del trabajo	5
2.2. Planificación del tiempo	5
2.2.1. EDT	5
2.2.1.1. Fase 1	8
2.2.1.2. Fase 2	9
2.2.2. Agenda del proyecto	9
2.2.3. Tareas	10
2.2.4. Entregables	18
2.2.4.1. Fase 1	18
2.2.4.2. Fase 2	18
2.2.5. Cronograma	18
2.3. Gestión de costos	21
2.3.1. Presupuesto	21

2.4. Gestión de riesgos	24
2.4.1. Explicación y plan de contingencia	24
II. Fase 1: Estado del Arte de la Seguridad Informática	29
3. Introducción	31
4. Conceptos Generales	33
4.1. Seguridad Informática	34
4.2. Seguridad de la Información	34
4.3. Servicios de la Seguridad de la Información	34
4.3.1. CID	35
4.3.2. Otros servicios	35
5. Aplicaciones	37
5.1. Malware	37
5.1.1. Gusanos	39
5.1.2. Troyanos	40
5.1.3. Ransomware	40
5.1.4. Rootkits	41
5.1.5. RATs	41
5.1.6. Spyware	41
5.1.7. Keyloggers	42
5.2. Dispositivos móviles	42
5.2.1. Seguridad en smartphones	43
5.2.1.1. Seguridad en iOS	43
5.2.1.2. Seguridad en Android	44
5.3. Internet of Things	46
5.3.1. Aplicaciones	46
5.3.1.1. Smart homes	46
5.3.1.2. Smart Cities	47
5.3.1.3. Monitorización medioambiental	47
5.3.1.4. Sanidad	47
5.3.1.5. Smart Business	47
5.3.1.6. Seguridad y vigilancia	48
5.3.2. Seguridad en Internet of Things	48
5.4. Cloud Computing	48
5.4.1. Tipos de servicios	49
5.4.1.1. Software as a Service	49
5.4.1.2. Platform as a Service	49
5.4.1.3. Infrastructure as a Service	49
5.4.2. Seguridad en Cloud Computing	50

6. Pentesting	51
6.1. Objetivos	51
6.2. Partes	52
6.3. Recogida de información	53
6.3.1. Internal Footprinting	54
6.3.2. External Footprinting	54
6.3.2.1. Active Footprinting	54
Escaneos DNS	54
Fingerprinting	55
SMTP	55
6.3.2.2. Passive Footprinting	55
Whois	55
Hacking con buscadores	55
Social network engineering	56
6.4. Análisis de vulnerabilidades	56
6.4.1. Pruebas	56
6.4.1.1. Activas	56
6.4.1.2. Pasivas	57
6.4.2. Validación	57
6.4.3. Investigación	57
6.5. Explotación de vulnerabilidades	57
6.5.1. Ataques de contraseñas	58
6.5.1.1. Fuerza bruta	58
6.5.1.2. Por diccionario	59
6.5.2. Exploits	60
6.5.3. Ataques a redes	60
6.5.3.1. Sniffing	61
6.5.3.2. Spoofing	62
6.5.3.3. Hijacking	62
7. Conclusiones	63
7.1. La seguridad informática y los usuarios	64
7.2. Herramientas de seguridad informática para usuarios	65
III. Fase 2: Desarrollo de la aplicación	67
8. Introducción	69
9. Tecnologías y herramientas	71
9.1. Kali Linux	71
9.2. Nmap	72
9.3. Android	72
9.3.1. Android SDK	72

9.3.2. Kotlin	73
9.3.2.1. Ventajas de Kotlin	74
Control de referencias nulas	74
Atributos de una clase	75
Singletons	76
Data Classes	77
Expresiones de rango	77
9.3.3. Android Studio	77
9.4. Otras herramientas	77
9.4.1. Git	78
9.4.2. L ^A T _E X	78
10. Desarrollo de la aplicación	79
10.1. Interfaz gráfica	79
10.2. Navegación	81
10.2.1. Escaneos de red	83
10.2.2. Escaneos de host	86
10.2.3. Ajustes	89
10.3. Aspectos gráficos de Material Design	90
10.3.1. Color	90
10.3.2. Layout	92
10.3.2.1. ConstraintLayout	93
10.3.2.2. LinearLayout	96
10.3.3. Tipografía	97
10.3.4. Iconografía	98
10.4. Conclusiones	98
11. Implementación de la aplicación	101
11.1. Estructura del proyecto	101
11.1.1. Estructura de paquetes de código	102
11.2. Integración de Nmap	103
11.2.1. Instalación	105
11.2.2. Ejecución	108
11.2.3. Leer datos	110
11.3. Persistencia de datos	113
11.3.1. Diseño de la base de datos	114
11.3.2. Implementación de la base de datos	114
11.4. Obtención de información de medios externos	120
11.4.1. Obtención del fabricante de un nodo de la red	120
11.5. Hilos, Tareas y paralelización	125
11.6. Interfaz gráfica	131
11.6.1. Ciclo de vida de Activities y Fragments	131
11.6.2. Navegación	135

IV. Análisis y conclusiones del Trabajo	141
12. Conclusiones	143
12.1. Problemas durante el desarrollo	144
12.2. Trabajo futuro	145
V. Apéndices	147
Bibliografía	149

Índice de figuras

2.1.	EDT completo	7
2.2.	EDT de la Fase 1	8
2.3.	EDT de la Fase 2	9
2.4.	Cronograma de la fase inicial	18
2.5.	Cronograma de la Fase 1	19
2.6.	Cronograma de la Fase 2	20
2.7.	Cronograma de la elaboración de la memoria	21
2.8.	Recursos de trabajo y materiales	21
4.1.	Confidencialidad, Integridad y Disponibilidad	35
5.1.	Infografía sobre el malware en 2016 [17]	39
5.2.	Nuevas muestras de familias de ransomware prominentes [13]	41
5.3.	Esquema de cifrado de un archivo en iOS	44
5.4.	Las diferentes capas que componen la arquitectura de Android	45
6.1.	Logo de Penetration Testing Execution Standard	52
6.2.	Principales protocolos usados en las cuatro capas de TCP/IP	61
10.1.	Mockup inicial de la GUI de la aplicación	80
10.2.	Lista de clases con las Activities y Fragments implementados en la aplicación	81
10.3.	Diagrama de flujo entre los diferentes elementos de la interfaz	82
10.4.	Navbar principal de la aplicación	83
10.5.	Diseño de ScanListFragment para mostrar los diferentes escaneos realizados	84
10.6.	Diferencia en NodeListFragment para un escaneo guardado y uno en curso	85
10.7.	Diseño del NodeInfoActivity	86
10.8.	Diseño del ScanDirectionFragment	87
10.9.	Diseño del DirectionScanActivity y su pestaña con BasicInfoFragment . .	88

10.10. Diseño del resto de pestañas del DirectionScanActivity	88
10.11. Menú contextual para acceder a los ajustes	89
10.12. Diferentes categorías dentro de los ajustes	89
10.13. Ajustes de la sección sobre los escaneos y Nmap	90
10.14. Ejemplo de distinción de colores primario y secundario en Material Design	91
10.15. Colores finales elegidos para la aplicación	91
10.16. Definición de los colores en el fichero colors.xml del proyecto en Android Studio	92
10.17. Elementos del ConstraintLayout del NodeInfoActivity	94
10.18. Vistas de diseño y de blueprint del editor de interfaces gráficas de Android Studio	95
10.19. Layout y márgenes en un blueprint del NodeInfoActivity	96
10.20. Diferentes opciones aplicadas a la tipografía original	97
10.21. Icono de la aplicación en formato redondeado	98
11.1. Estructura básica de archivos del proyecto	102
11.2. Estructura de paquetes de código del proyecto	103
11.3. Distribución de lenguajes en el repositorio de Nmap	104
11.4. Los diferentes binarios para cada arquitectura usados en la aplicación	106
11.5. Arquitectura de Room e interrelación entre sus diferentes componentes	115
11.6. Arquitectura de Room e interrelación entre sus diferentes componentes	116
11.7. Ciclo de vida de un Activity de Android	132
11.8. Ciclo de vida de un Fragment de Android	134

Índice de tablas

2.1.	Calendario de días festivos oficial (de Álava)	10
2.2.	Recursos materiales (software)	21
2.3.	Recursos de trabajo	22
2.4.	Recursos materiales (hardware)	22
2.5.	Recursos materiales (software)	22
2.6.	Costo de recursos de trabajo	22
2.7.	Costo de recursos materiales	23
2.8.	Amortizaciones de hardware y software	23
2.9.	Total presupuesto	23
2.10.	Enumeración de riesgos del proyecto	24
6.1.	Número de diferentes combinaciones de contraseñas posibles para diferentes conjuntos	59
6.2.	Coste computacional de diferentes ataques de fuerza bruta para diferentes conjuntos	59

Índice de códigos fuente

1.	Función que instala Nmap en el dispositivo	106
2.	Función para arrancar el proceso con la shell que ejecutará Nmap	108
3.	Función que crea el comando a ejecutar de Nmap	108
4.	Función con todo el proceso de ejecución de un escaneo en Nmap	109
5.	Extracto de la implementación de un parser de un XML de Nmap	111
6.	Fichero XML con la información de un escaneo estándar de Nmap	111
7.	Data Classes para la información de un scan en Nmap	112
8.	Data Classes para la información de un host en Nmap	112
9.	Data Classes para la información de un puerto en Nmap	113
10.	Implementación de la clase AppDatabase	116
11.	Implementación de diferentes clases Entity	117
12.	Ejemplo de una interfaz DAO para interactuar con la base de datos	118
13.	Ejemplo de obtención de información de la base de datos Room	119
14.	Ejemplo de inserción de información en la base de datos Room	120
15.	Código que obtiene la dirección física en base a una dirección IP	121
16.	Extracto del fichero que relaciona direcciones MAC on fabricantes	122
17.	Clase abstracta para un recurso descargable	122
18.	Obtención el fabricante en base a la dirección MAC	123
19.	Obtención información del objetivo de un puerto concreto	124
20.	Creación del proceso, la entrada y salida para Nmap	125
21.	Ejecución de la descarga y lectura de un archivo en un Thread	126
22.	Ejemplo de la estructura de una AsyncTask	127
23.	Implementación de metodos de una AsyncTask en la clase NetworkScan . .	128
24.	Implementación de parte de la tarea asíncrona SequentialNetworkScan . .	129
25.	Implementación de un NetworkScan dentro de un Activity	130
26.	Selección de un Fragment u otro en función del valor seleccionado en el Navbar	135

27.	Referencias a cada uno de los Fragments definidas dentro del MainActivity	135
28.	Evento para lanzar una nueva Activity para escanear una red	136
29.	Creación del NodeListActivity y recepción de parámetros	136
30.	Creación del NodeListFragment y uso del RecyclerView	137
31.	Definición de un RecyclerView para usar con una lista de nodos	138

Resumen y Organización de la memoria

La memoria presentada a continuación se divide en cinco partes.

En la primera parte, denominada *Alcance del Trabajo*, se presenta un resumen sobre en que consiste el proyecto realizado, enumerando objetivos y aportando una primera visión inicial.

En la segunda parte, *Estado del Arte de la Seguridad Informática*, se muestra uno de los dos bloques de los que consta el proyecto. Un estado del arte exhaustivo sobre la Seguridad Informática, sus aplicaciones y el Pentesting.

En la tercera parte, *Desarrollo de la Aplicación*, se muestra el desarrollo del segundo bloque del proyecto, que reside en una aplicación que implementa una solución a un problema concreto, basándose en todo lo aprendido en el primer bloque.

En la cuarta parte, *Análisis y conclusiones del Trabajo*, se recopilan las conclusiones extraídas tras desarrollar todo el proyecto.

En la ultima, *Apéndices*, se incluye la bibliografía utilizada y referencias al contenido desarrollado

I

Alcance del Trabajo

1

Descripción, Objetivos y Motivación

1.1. Descripción

A lo largo de este Trabajo de Fin de Grado se desarrolla un estudio sobre el campo de la seguridad informática, más concretamente sobre las amenazas y técnicas de seguridad informática actuales. En base a todo ese estudio se desarrolla una aplicación para dispositivos móviles que busca, de manera sencilla para un usuario medio, proporcionar soluciones a tareas recurrentes dentro del campo de la seguridad informática, basándose en herramientas ya existentes. Estas herramientas, usadas por pentesters, analistas forenses o hackers de sombrero blanco permiten elaborar operaciones de todo tipo, desde escanear una red inalámbrica hasta romper el cifrado de un archivo para acceder a la información que contiene.

Gran parte de estas herramientas son gratuitas [1] o incluso de software libre [2], lo que otorga la posibilidad de que dichas utilidades mejoren continuamente. El mayor problema de este tipo de herramientas suelen ser su público objetivo. Normalmente este tipo de soluciones están diseñadas para profesionales del sector, profesionales tanto con conocimientos de seguridad informática como de programación o administración de sistemas. La mayoría de estas herramientas suelen ser desde grandes librerías o frameworks completos, con cierta dificultad de uso, hasta pequeños scripts CLI (Command Line Interface). Debido a esto, cierta tarea como escanear una red, que para un experto en ciberseguridad o un administrador de sistemas se convierte en 5 segundos tecleando un comando, para un usuario medio se convierte en un auténtico quebradero de cabeza.

1.2. Objetivos

El objetivo de este proyecto es doble. Por una parte se busca realizar un análisis del campo de la seguridad informática, un *estado del arte* del área que permita vislumbrar cuales son las amenazas existentes, los diferentes campos de enfoque y las técnicas actuales para

securizar sistemas para, a partir de ahí, concretar las necesidades más importantes dentro de ese campo. Finalmente, basándose en toda la información recogida se acabará desarrollando una aplicación para Android que nos proporcione ciertas utilidades.

Por otra parte, también se busca que la aplicación a elaborar sirva tanto para usuarios experimentados en la materia como para un público general. Para ello, un buen diseño de la interfaz gráfica (GUI, Graphical User Interface) o diferentes principios de experiencia de usuario (UX, User Experience) jugarán un papel fundamental. De esta manera lograremos una transición entre herramientas accesibles solo para unos pocos a herramientas aptas para el público general.

1.3. Motivación

A día de hoy la informática es una industria fundamental dentro de la sociedad en general y de las vidas de las personas en particular. Los ordenadores personales son la herramienta fundamental de trabajo en una gran cantidad de áreas, además de una herramienta que se encuentra en prácticamente cualquier hogar. Los denominados Smartphones, en conjunción a Internet, se han convertido en la principal herramienta de comunicación. La revolución causada por la industria llega hasta tal punto que una organización de la relevancia de la ONU declara y define Internet como un derecho humano *por ser una herramienta que favorece el crecimiento y el progreso de la sociedad en su conjunto* [3].

Teniendo en cuenta toda la información que transmitimos, almacenamos y procesamos, sería lógico pensar que la seguridad de dicha información es vital. El área de la seguridad informática se encarga de ofrecer los mecanismos necesarios para que nuestra información no se vea comprometida y nuestros dispositivos permanezcan seguros y con la menor cantidad de vulnerabilidades posible. Un área que resulta cada vez más importante, especialmente teniendo en cuenta la llegada de nuevos campos como el Internet of Things (IoT), ya que pasamos de tener no solo nuestros ordenadores o Smartphones conectados a Internet, sino a dotar de conexión a Internet a otros dispositivos como nuestro coche o nuestra lavadora, con todos los riesgos que ello conlleva.

Aunque el campo de la seguridad informática crece a pasos agigantados, sería un error enfocar la seguridad informática de manera unilateral. Para garantizar en la mayor medida posible la seguridad de los sistemas, el usuario debe tomar un papel activo. Si los usuarios disponen de herramientas a su alcance para llevar a cabo tareas que permitan hacer más seguros sus sistemas, jugarán un papel activo, y no pasivo, dentro de la seguridad de su información. Dotar al usuario de la capacidad, que no de la necesidad, de hacer más seguros sus sistemas sirve para complementar el papel de los expertos y profesionales de la seguridad informática.

2

Viabilidad

Este capítulo tiene como objetivo realizar, en base a los objetivos marcados en el capítulo anterior, elaborar un análisis de viabilidad del proyecto, analizando las diferentes tareas que contiene, calculando los correspondientes gastos y sus inherentes riesgos, que pueden afectar, atrasando o incluso impidiendo, la realización del proyecto.

2.1. Requisitos funcionales del trabajo

Los requisitos funcionales de la aplicación (RF a partir de ahora) se elaboran en base a los objetivos descritos y son los siguientes:

- Elaborar un estado del arte que analice el campo de la seguridad informática, sus aplicaciones y analice el área del *pentesting*.
- Elaborar una aplicación que permita obtener información sobre redes y nodos de la red.
- Integrar correctamente herramientas de terceros para lograr que la aplicación resulte lo más escalable posible.
- Aplicar principios sobre experiencia de usuario (UX) y sobre el diseño de interfaces gráficas (GUI) para que la aplicación sea lo más sencilla y cómoda de usar.

2.2. Planificación del tiempo

2.2.1. Estructura de Descomposición del Trabajo

El EDT (Estructura del Desglose del Trabajo o Estructura de Descomposición del Trabajo) es un sistema jerárquico que permite organizar las diferentes tareas de un proyecto. Es una

técnica ampliamente usada para gestionar todo tipo de proyectos, especialmente proyectos de software.

A la hora de planificar el tiempo se ha tenido en cuenta un enfoque en dos fases, a las cuales denominaremos Fase 1 y Fase 2. Esto es debido al carácter del proyecto. Por una parte, para elaborar la aplicación anteriormente mencionada, resulta fundamental realizar un estudio sobre el campo de la seguridad informática, más concretamente sobre el área del pentesting, para poder llegar a encontrar las mejores herramientas y técnicas que permitan desarrollarla. Este estudio, bien desglosado y fundamentado, llevará a la obtención de un elaborado estado del arte, que será el principal objetivo de la Fase 1. Además, dicha fase contiene un periodo de aprendizaje y familiarización con diversos conceptos y tecnologías, que también quedarán reflejados.

En la Fase 2, en función de lo aprendido en la Fase 1, se elaborará la aplicación en base a los criterios de implementar diversas utilidades junto a un experiencia de usuario (UX) óptima, que vendrá acompañada de un buen diseño de una interfaz gráfica (GUI).

El EDT completo quedaría tal y como se muestra en la Figura 2.1.

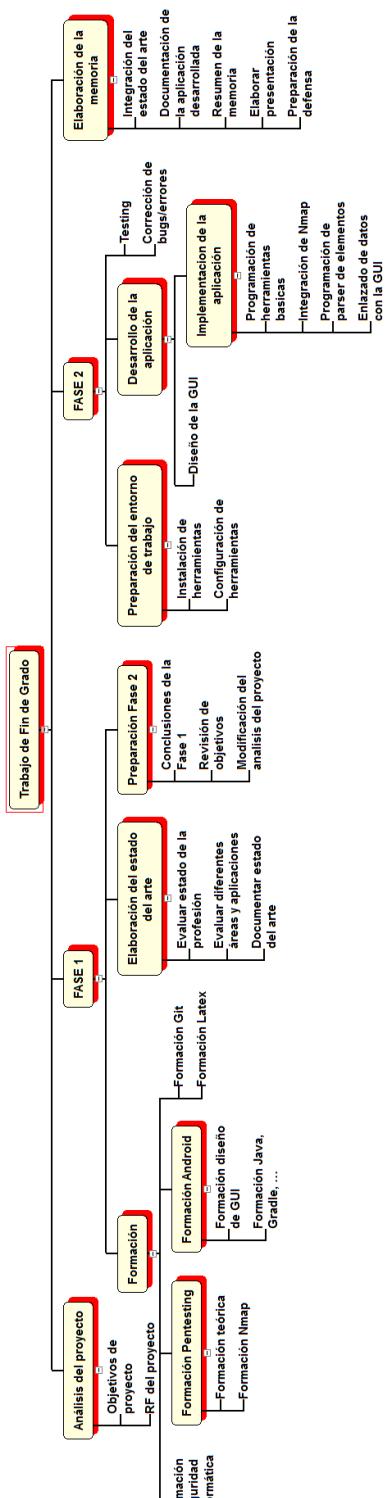


Figura 2.1.: EDT completo

2.2.1.1. Fase 1

El EDT para la Fase 1 quedaría como se muestra en la Figura 2.2.

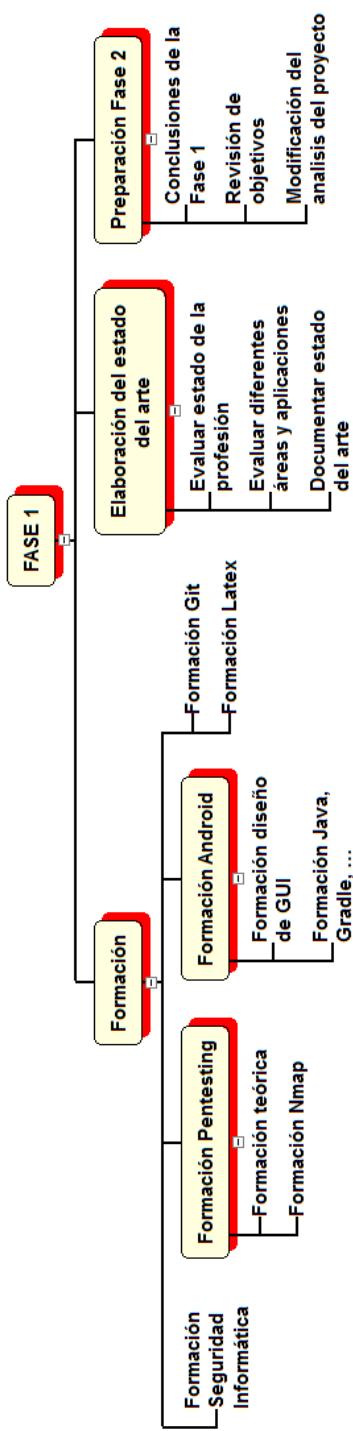


Figura 2.2.: EDT de la Fase 1

2.2.1.2. Fase 2

El EDT para la Fase 2 quedaría como se muestra en las Figura 2.3.

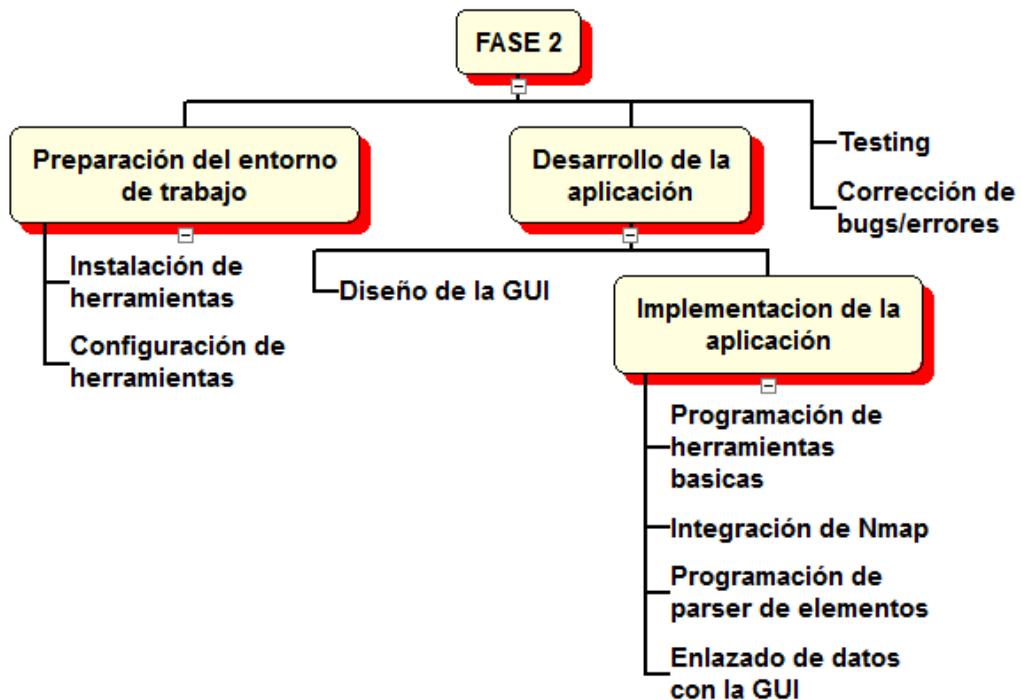


Figura 2.3.: EDT de la Fase 2

2.2.2. Agenda del proyecto

El proyecto se llevará a cabo durante varios meses, comenzando en abril. Se trabajará a media jornada (4 horas) de lunes a viernes, con las siguientes excepciones. Primero, se tendrá en cuenta el calendario de festivos oficiales para aplicar algunas jornadas festivas en cuyos días no se trabajará. Esos días quedan reflejados en la Tabla 2.1.

Fecha	Evento
1 de Enero	Año nuevo
6 de Enero	Día de Reyes
19 de Marzo	San José
29 de Marzo	Jueves Santo
30 de Marzo	Viernes Santo
2 de Abril	Lunes de Pascua
28 de Abril	San Prudencio
1 de Mayo	Día del trabajo
25 de Julio	Santiago Apóstol
5 de Agosto	Virgen Blanca
15 de Agosto	Asunción de la Virgen
12 de Octubre	Fiesta Nacional de España
6 de Diciembre	Día de la constitución
8 de Diciembre	Inmaculada Concepción
25 de Diciembre	Navidad

Tabla 2.1.: Calendario de días festivos oficial (de Álava)

En base a dicho calendario, la fecha estimada de finalización del proyecto es del 16 de enero del 2018.

2.2.3. Tareas

El EDT completo de tareas, al que se añaden la fase inicial de objetivos del proyecto, y toda la fase final de elaboración de la memoria, presentación y la defensa quedaría de la siguiente manera.

0. Análisis del proyecto

0.1 Objetivos del proyecto

0.2 RF del proyecto

1. FASE 1

1.1 Formación

1.1.1 Formación seguridad informática

1.1.2 Formación Pentesting

1.1.2.1 Formación Teórica

1.1.2.2 Formación Nmap

1.1.3 Formación Android

1.1.3.1 Formación Diseño de GUI

- 1.1.3.2** Formación Java, Gradle, ...
- 1.1.4** Formación Git
- 1.1.5** Formación L^AT_EX
- 1.2** Elaboración del estado del arte
 - 1.2.1** Evaluar estado de la profesión
 - 1.2.2** Evaluar diferentes áreas y aplicaciones
 - 1.2.3** Documentar estado del arte
- 1.3** Preparación de la Fase 2
 - 1.3.1** Conclusiones de la Fase 1
 - 1.3.2** Revisión de objetivos
 - 1.3.3** Modificación del análisis del proyecto
- 2.** FASE 2
 - 2.1** Preparación del entorno de trabajo
 - 2.1.1** Instalación de herramientas
 - 2.1.2** Configuración de herramientas
 - 2.2** Desarrollo de la aplicación
 - 2.2.1** Diseño de la GUI
 - 2.2.2** Implementación de la aplicación
 - 2.2.2.1** Programación de herramientas básicas
 - 2.2.2.2** Integración de Nmap
 - 2.2.2.3** Programación de parser de elementos
 - 2.2.2.4** Enlazado de datos con la GUI
 - 2.2.3** Testeo
 - 2.2.4** Corrección de bugs/errores
- 3.** Elaboración de la memoria
 - 3.1** Integración del estado del arte
 - 3.2** Documentación de la aplicación desarrollada
 - 3.3** Resumen de la memoria
 - 3.4** Elaborar presentación
 - 3.5** Preparación de la defensa
- 4.** Reuniones periódicas

A continuación se explica mediante una breve definición cada tarea, además de especificar su duración en horas.

Número: 0.1.

Nombre: Objetivos del proyecto.

Descripción: Definir los objetivos que tiene que cumplir el TFG.

Trabajo estimado: 5 horas.

Número: 0.2.

Nombre: RF del proyecto.

Descripción: Definir, en base a los objetivos del proyecto, los Requisitos Funcionales (RF) concretos del proyecto.

Trabajo estimado: 5 horas.

Número: 1.1.1.

Nombre: Formación seguridad informática.

Descripción: Familiarizarse con el amplio entorno de la seguridad informática y comprender las diferentes áreas, objetivos y el estado de dicho campo.

Trabajo estimado: 30 horas.

Número: 1.1.2.1.

Nombre: Formación Teórica.

Descripción: Familiarizarse con los conceptos de Pentesting, las diferentes técnicas usadas y las diferentes fases del proceso de Pentesting.

Trabajo estimado: 20 horas.

Número: 1.1.2.2.

Nombre: Formación Nmap.

Descripción: Familiarizarse con el entorno de Nmap, cómo implementarlo, usarlo para obtener información y de qué formas se puede obtener información estructurada y organizada para su posterior uso.

Trabajo estimado: 10 horas.

Número: 1.1.3.1.

Nombre: Formación Diseño de GUI.

Descripción: Aprender a usar herramientas de diseño de GUI, diferentes patrones de diseño en sistemas Android, y el uso de IDEs o herramientas para desarrollar dichas GUIs.

Trabajo estimado: 10 horas.

Número: 1.1.3.2.

Nombre: Formación Java, Gradle,

Descripción: Aprender sobre el uso de Java para desarrollar aplicaciones Android, diferentes clases, utilidades o conceptos recurrentes en la programación para Android.

Trabajo estimado: 15 horas.

Número: 1.1.4.

Nombre: Formación Git.

Descripción: Aprender el uso de dicho sistema de control de versiones para llevar un control riguroso del desarrollo del proyecto y de la aplicación.

Trabajo estimado: 5 horas.

Número: 1.1.5.

Nombre: Formación L^AT_EX.

Descripción: Aprender diferentes conceptos de L^AT_EX para elaborar tanto el estado del arte como el propio informe de la manera más clara y elegante posible.

Trabajo estimado: 5 horas.

Número: 1.2.1.

Nombre: Evaluar estado de la profesión.

Descripción: Analizar los diferentes campos de la profesión, las necesidades mas demandadas y los diferentes perfiles de profesionales dentro del campo.

Trabajo estimado: 5 horas.

Número: 1.2.2.

Nombre: Evaluar diferentes áreas y aplicaciones.

Descripción: Evaluar las necesidades concretas a nivel técnico, las aplicaciones más usadas y las virtudes y carencias de éstas.

Trabajo estimado: 10 horas.

Número: 1.2.3.

Nombre: Documentar estado del arte.

Descripción: Elaborar la documentación en base a toda la información recogida para obtener un elaborado estado del arte.

Trabajo estimado: 5 horas.

Número: 1.3.1.

Nombre: Conclusiones de la Fase 1.

Descripción: Elaborar una serie de conclusiones en función a todo el estudio realizado sobre el campo de la seguridad informática.

Trabajo estimado: 5 horas.

Número: 1.3.2.

Nombre: Revisión de objetivos.

Descripción: Revisión de los objetivos y los Requisitos Funcionales de la aplicación a desarrollar en función a todo lo investigado.

Trabajo estimado: 5 horas.

Número: 1.3.3.

Nombre: Modificación del análisis del proyecto.

Descripción: Modificar la parte de análisis del proyecto realizada anteriormente, antes de comenzar con la Fase 2.

Trabajo estimado: 5 horas.

Número: 2.1.1.

Nombre: Instalación de herramientas.

Descripción: Instalación de todo lo necesario para desarrollar la aplicación.

Trabajo estimado: 5 horas.

Número: 2.1.2.

Nombre: Configuración de herramientas.

Descripción: Configuración de todas las herramientas para que el desarrollo de la aplicación sea lo mas cómodo posible.

Trabajo estimado: 5 horas.

Número: 2.2.1.

Nombre: Diseño de la GUI.

Descripción: Diseñar una interfaz gráfica clara y sencilla de usar para interactuar con las funciones a implementar.

Trabajo estimado: 15 horas.

Número: 2.2.2.1.

Nombre: Programación de herramientas básicas.

Descripción: Programar herramientas básicas para el escaneo de redes.

Trabajo estimado: 10 horas.

Número: 2.2.2.2.

Nombre: Integración de Nmap.

Descripción: Integrar el núcleo de Nmap en la aplicación para poder hacer uso de toda su funcionalidad.

Trabajo estimado: 10 horas.

Número: 2.2.2.3.

Nombre: Programación de parser de elementos.

Descripción: Elaborar un puente entre Nmap y la aplicación para obtener los datos de Nmap y poder usarlos en la aplicación de la manera más organizada posible.

Trabajo estimado: 20 horas.

Número: 2.2.2.4.

Nombre: Enlazado de datos con la GUI.

Descripción: Enlazar los datos con las diferentes vistas a través de diversos controladores, para poder visualizar e interactuar con ellos.

Trabajo estimado: 15 horas.

Número: 2.2.3.

Nombre: Testing.

Descripción: Una vez desarrollada la aplicación, realizar un amplio testeo para comprobar que funciona correctamente.

Trabajo estimado: 10 horas.

Número: 2.2.4.

Nombre: Corrección de bugs/errores.

Descripción: En base a los errores detectados en el testeo, implementar las correcciones a dichos fallos.

Trabajo estimado: 15 horas.

Número: 3.1.

Nombre: Integración del estado del arte.

Descripción: Integrar el estado del arte desarrollado dentro de la memoria.

Trabajo estimado: 5 horas.

Número: 3.2.

Nombre: Documentación de la aplicación desarrollada.

Descripción: Elaborar en base a todo el proceso de desarrollo una documentación clara sobre la aplicación e integrarla en la memoria.

Trabajo estimado: 15 horas.

Número: 3.3.

Nombre: Resumen de la memoria.

Descripción: Terminar la elaboración de la memoria, añadiendo las diferentes secciones necesarias y el formato correspondiente.

Trabajo estimado: 5 horas.

Número: 3.4.

Nombre: Elaborar presentación.

Descripción: Elaborar la presentación en diapositivas que se usará en la defensa ante el tribunal.

Trabajo estimado: 5 horas.

Número: 3.5.

Nombre: Preparación de la defensa.

Descripción: Preparar la defensa ante el tribunal en función a la documentación elaborada.

Trabajo estimado: 10 horas.

Número: 4.

Nombre: Reuniones periódicas.

Descripción: Reuniones periódicas con el director del TFG para llevar un control del desarrollo del proyecto.

Trabajo estimado: 15 horas.

2.2.4. Entregables

2.2.4.1. Fase 1

El entregable de la Fase 1 consistirá en un estado del arte redactado sobre el campo de la seguridad informática que analice cuales son las amenazas existentes, los diferentes campos de enfoque y las técnicas actuales para securizar sistemas, haciendo especial hincapié en el pentesting

2.2.4.2. Fase 2

El entregable de la Fase 2 consistirá en una aplicación elaborada, que sirva como herramienta para el escaneo de redes informáticas. La aplicación, desarrollada para Android, estará correctamente empaquetada, con los posibles fallos corregidos y además dispondrá de una GUI sencilla de usar.

2.2.5. Cronograma

El cronograma completo, con las dos fases, se muestra en las Figuras 2.4, 2.5, 2.6 y 2.7.

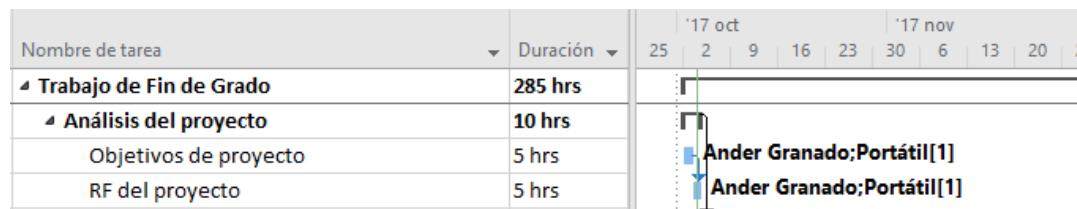


Figura 2.4.: Cronograma de la fase inicial

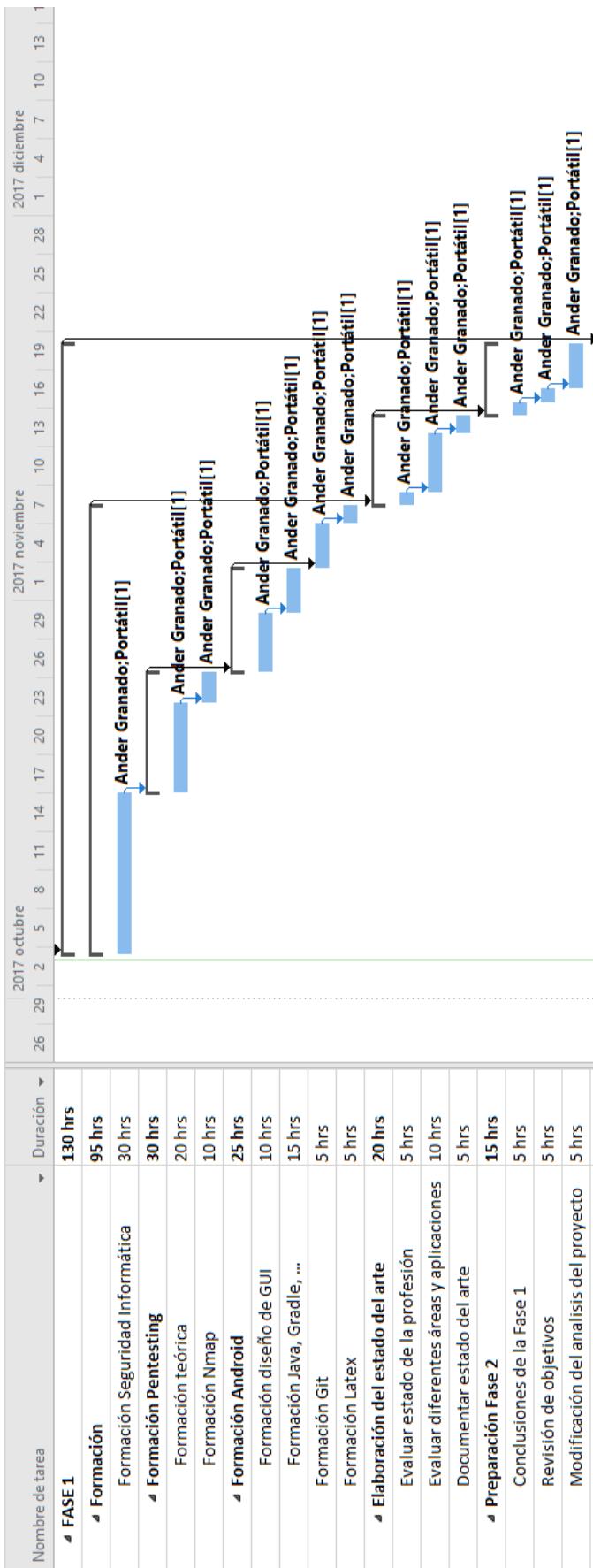


Figura 2.5.: Cronograma de la Fase 1

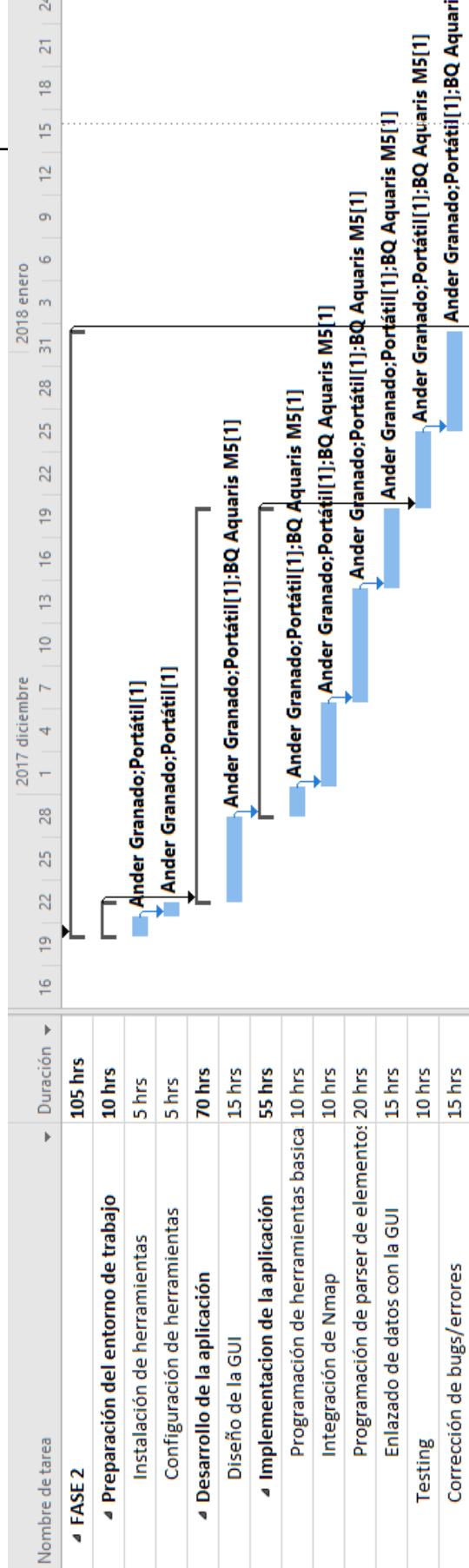


Figura 2.6.: Cronograma de la Fase 2

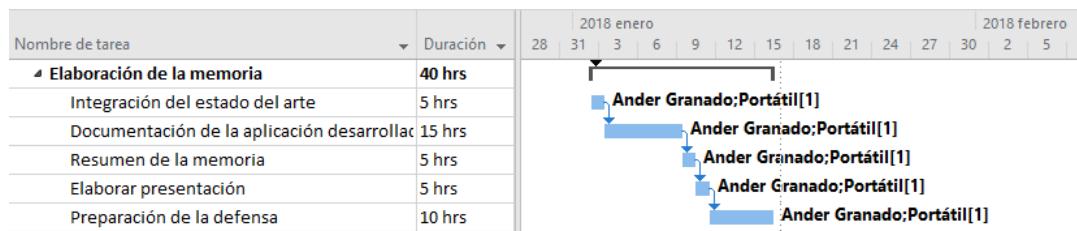


Figura 2.7.: Cronograma de la elaboración de la memoria

2.3. Gestión de costos

A la hora de elaborar un análisis de costos, con el objetivo de obtener un presupuesto, primero es necesario identificar todos los recursos que influyen en ese presupuesto. Por una parte, tendremos los recursos de trabajo, es decir empleados, y por otra parte los recursos materiales, tanto de software como de hardware. En la Figura 2.8 se muestran tanto los recursos de trabajo como los recursos materiales de hardware.

	Nombre del	Tipo	Etiqueta de	Iniciales	Grupo	Capacidad	Tasa	Tasa horas	Costo/U.	Acumu	Calendario base
1	Ander Granado	Trabajo		A		100%	14.00 €/hr	0.00 €/hr	0.00 €	Prorratoe	Calendario Estudiantil
2	Portátil	Material		P			0.00 €	0.00 €	1.50 €	Prorratoe	
3	BQ Aquaris M5	Material		BQ			0.00 €	0.00 €	1.50 €	Prorratoe	

Figura 2.8.: Recursos de trabajo y materiales

Por otra parte, en la Tabla 2.2 se muestra los recursos materiales de software utilizados.

Concepto	Coste
Windows 10	135,00 €[4]
Project 2016	1.369,00 €[5]
WBS Chart Pro	187,50 €[6]
Ubuntu 16.04	0,00 €
TeX Studio	0,00 €
Android Studio	0,00 €
Nmap	0,00 €
Git	0,00 €
GitHub	0,00 €

Tabla 2.2.: Recursos materiales (software)

2.3.1. Presupuesto

El análisis de costes del proyecto se refleja en toda la información que aparecen entre la Tabla 2.3 y la Tabla 2.9. Para ello se tienen en cuenta varios puntos. El primero, se tiene en

cuenta un salario base de 14 €/h, correspondiente un salario estándar de un analista programador. Para los recursos materiales, se tiene en cuenta un costo por uso de 1,50 €. Estos costos por uso vienen dados por el gasto que generan debido al consumo energético, internet y otro tipo de gastos derivados de su utilización. Por otra parte se realiza una separación para los recursos de software, entre los que se incluye el propio Microsoft Project, software usado para la realización de todo el análisis de viabilidad y la gestión del proyecto. Para el cálculo de las amortizaciones se considera un tiempo de amortización de 3 años, el cual traducido en horas vendría a ser 4800 horas ($3 \text{ años} * 200 \text{ días laborables} * 8 \text{ horas} = 4800 \text{ horas}$).

Concepto	Coste
Ander Granado	14,00 €/h

Tabla 2.3.: Recursos de trabajo

Concepto	Coste
Ordenador portátil	700,00 €
BQ Aquaris M5	250,00 €

Tabla 2.4.: Recursos materiales (hardware)

Concepto	Coste	Número de licencias
Windows 10	135,00 €[4]	1
Project 2016	1369,00 €[5]	1
WBS Chart Pro	187,50 €[6]	1
Ubuntu 16.04	0,00 €	1
TeX Studio	0,00 €	1
Android Studio	0,00 €	1
NMap	0,00 €	1
Git	0,00 €	1
GitHub	0,00 €	1

Tabla 2.5.: Recursos materiales (software)

Concepto	Trabajo (h)	Trabajo horas extra	Coste	Coste horas extra	Importe
Ander Granado	285	0	14,00 €/h	0	4.044,00 €

Tabla 2.6.: Costo de recursos de trabajo

Concepto	Unidades	Coste	Importe
Ordenador portátil	1	1,50 €/uso	29 x 1,50 € = 43,50 €
BQ Aquaris M5	1	1,50 €/uso	7 x 1,50 € = 10,50 €
TOTAL			54,00 €

Tabla 2.7.: Costo de recursos materiales

Concepto	Coste unitario	T. de Amort. ¹	C.U.A. ² (€)	T. de uso	Importe
Ordenador portátil	700,00 €	4800 horas	0,145833 €	285 h	41,57 €
BQ Aquaris M5	250,00 €	4800 horas	0,052083 €	25 h	1,30 €
Windows 10	135,00 €	4800 horas	0,028125 €	180 h	5,06 €
M. Project 2016	1.369,00 €	4800 horas	0,285208 €	25 h	7,13 €
WBS Chart Pro	187,50 €	4800 horas	0,039062 €	5 h	0,19 €
Ubuntu 16.04	0,00 €	4800 horas	0,000000 €	105 h	0,00 €
TeX Studio	0,00 €	4800 horas	0,000000 €	40 h	0,00 €
Android Studio	0,00 €	4800 horas	0,000000 €	105 h	0,00 €
NMap	0,00 €	4800 horas	0,000000 €	30 h	0,00 €
Git	0,00 €	4800 horas	0,000000 €	145 h	0,00 €
GitHub	0,00 €	4800 horas	0,000000 €	145 h	0,00 €
TOTAL					55,25 €

Tabla 2.8.: Amortizaciones de hardware y software

Concepto	Importe
Recursos de Trabajo (R.T.)	4.044,00 €
Recursos Materiales (R.M.)	54,00 €
Costo fijo	0,00 €
Amortizaciones	55,25 €
SUMA	4.153,25 €
Gastos generales (10 %)	415,32 €
Beneficio (15 %)	622,99 €
SUBTOTAL	5.191,56 €
IVA (21 %)	1.090,23 €
TOTAL	6281,79 €

Tabla 2.9.: Total presupuesto

Con esto se concluye que, tal como figura en la Tabla 2.9, el coste del proyecto asciende a la cantidad de *seis mil doscientos ochenta y cinco con setenta y nueve euros* (6281,79 €).

¹Tiempo de amortización²Coste unitario de amortización

2.4. Gestión de riesgos

En este apartado se identifican y analizan las diferentes amenazas que puedan llegar a impedir el correcto desarrollo del proyecto, haciendo que éste se retrase. Para ello primero se identifican los diferentes riesgos que pueden existir y se indica su peligrosidad. La peligrosidad es un valor cualitativo que indica en qué medida puede afectar ese riesgo al proyecto.

Se han identificado los siguientes riesgos, los cuales se muestran en la Tabla 2.10.

Riesgo	Peligrosidad
Pérdida de información	Alta
Enfermedades	Alta
Dificultades en la implementación de la aplicación	Alta
Dedicación no exclusiva al trabajo	Media
Averías o problemas técnicos con los recursos materiales	Media
Cambios o ampliación de requisitos	Media
Planificación muy optimista	Media

Tabla 2.10.: Enumeración de riesgos del proyecto

2.4.1. Explicación y plan de contingencia

Tras haber identificado los riesgos inherentes al proyecto en la Tabla 2.10, se hace mayor hincapié en los detalles de dicho riesgos, elaborando una descripción más detallada y analizando su probabilidad. La probabilidad de cada riesgo se muestra de manera cualitativa, ya que aportar un valor numérico concreto en este tipo de casos resulta bastante complicado. También, junto a lo mencionado, se añade para cada riesgo un plan de contingencia. El plan de contingencia consiste básicamente en aportar medidas para afrontar dicho riesgo, tanto preventivas (para antes de que ocurra) como correctoras (para en caso de ocurrir).

PÉRDIDA DE INFORMACIÓN

Descripción: Podría darse el caso de que parte de la información, como la memoria del proyecto, el estado del arte o el código de la aplicación se perdieran.

Probabilidad: Baja.

Peligrosidad: Alta.

Medidas preventivas: Uso de herramientas de control de versiones como Git junto a uso de herramientas cloud como GitHub o Dropbox.

Medidas correctoras: Recuperación de la información mediante herramientas de análisis de unidades.

ENFERMEDADES

Descripción: Podría suceder que el único recurso de trabajo contrajera una enfermedad o tuviera un accidente.

Probabilidad: Baja.

Peligrosidad: Alta.

Medidas preventivas: Ninguna.

Medidas correctoras: Usar horas fuera del calendario para corregir el retraso en el proyecto.

DIFICULTADES EN LA IMPLEMENTACIÓN DE LA APLICACIÓN

Descripción: Podría suceder que, durante la implementación de la aplicación, se dieran dificultades a nivel de programación a la hora de cumplir con los Requisitos Funcionales.

Probabilidad: Media.

Peligrosidad: Alta.

Medidas preventivas: Una formación sólida en las herramientas y tecnologías que se van a usar para el desarrollo de la aplicación.

Medidas correctoras: Replantear las tareas posteriores y dedicar un esfuerzo extra para el aprendizaje y refuerzo de las herramientas usadas.

DEDICACIÓN NO EXCLUSIVA AL TRABAJO

Descripción: Podría suceder que, debido a exámenes u otros asuntos personales, el desarrollador no pudiera aportar toda la dedicación que requiere el proyecto.

Probabilidad: Media.

Peligrosidad: Media.

Medidas preventivas: No previsible.

Medidas correctoras: Cambiar calendario de trabajo aumentando las horas para subsanar los retrasos producidos por dicho riesgo.

AVERÍAS O PROBLEMAS TÉCNICOS CON LOS RECURSOS MATERIALES

Descripción: Podría darse el caso de que alguno de los recursos materiales de hardware, como el ordenador o el teléfono móvil, sufren algún tipo de avería.

Probabilidad: Baja.

Peligrosidad: Media.

Medidas preventivas: No previsible.

Medidas correctoras: Uso de otros dispositivos para continuar con el desarrollo del proyecto, adquiridos mediante compra o préstamo.

CAMBIOS O AMPLIACIÓN DE REQUISITOS

Descripción: Podría darse el caso en el que, tras la conclusión de la Fase 1, se cambian Requisitos Funcionales de la aplicación a desarrollar o se añadirán nuevos Requisitos Funcionales.

Probabilidad: Media - Alta.

Peligrosidad: Media.

Medidas preventivas: Revisión constante de los Requisitos Funcionales durante todas las fases del proyecto para minimizar el impacto que pueda causar los cambios en ellos.

Medidas correctoras: Adaptar el calendario las tareas posteriores y el calendario de trabajo actual.

PLANIFICACIÓN MUY OPTIMISTA

Descripción: Puede darse el caso de que la planificación elaborada para el proyecto sea demasiado optimista y no haya tenido en cuenta ciertos aspectos más concretos del desarrollo del proyecto.

Probabilidad: Media.

Peligrosidad: Media.

Medidas preventivas: No previsible.

Medidas correctoras: Adaptar, y en caso de que fuera necesario retrasar, la fecha final del proyecto para dar cabida en la planificación toda esa duración extra no prevista.

II

**Fase 1: Estado del Arte de la Seguridad
Informática**

3

Introducción

La siguiente parte de esta memoria contiene el trabajo realizado durante la denominada *Fase 1* del proyecto. En él se elabora un completo estado del arte sobre el mundo de la seguridad informática. Dicho estado del arte se encuentra dividido en cuatro capítulos.

En el primero, con fines puramente introductorios, se pasa a mencionar una serie de conceptos básicos sobre seguridad informática, un conjunto de definiciones que resulta fundamental comprender sobre este campo de la informática.

En el segundo capítulo, llamado *Aplicaciones de la Seguridad Informática*, se enumeran una serie de amenazas hacia la seguridad, pasando de un enfoque más clásico a profundizar en las áreas con más auge dentro del campo, mostrando sus particularidades, con el objetivo de hacerse una visión global de las diferentes necesidades y de cómo la seguridad informática se aplica en cada campo.

En el tercer capítulo, denominado *Pentesting*, se profundiza en la técnica que su propio nombre indica. Dicha técnica es una de las herramientas fundamentales de la seguridad informática, y permite obtener una mejor visión de como securizar sistemas.

Sin embargo, el objetivo de este estado del arte no es simplemente obtener una visión global del campo de la seguridad informática, sino también ser capaces de extraer una serie de conclusiones con el objetivo final de ofrecer una solución que consiga hacer que el propio usuario, inexperto, pero el principal *target* de cualquiera de las tecnologías de la información, tenga un papel proactivo en la seguridad de sus sistemas y su información.

De esta manera, de una visión vaga y difuminada de un campo con cada vez mayor importancia, se obtendrá una imagen nítida de todo el área, que además permita enfocar la solución a desarrollar.

4

Conceptos Generales

"El único sistema verdaderamente seguro es aquel que se encuentra apagado, enterrado en un bloque de hormigón en una habitación sellada con plomo y vigilada por guardias armados. Incluso entonces tendría mis dudas"

— Gene Spafford

Tal y como Gene Spafford aseguraba con sus palabras [7], la seguridad informática no es una ciencia infalible y el sistema verdaderamente seguro es una quimera, algo que no existe y a su vez resulta inalcanzable. Partiendo de ese punto, la seguridad informática tiene diversos objetivos que surgen de la necesidad de proteger la información y los sistemas informáticos cada vez más en auge. Objetivos como por ejemplo:

- Minimizar y gestionar los riesgos y detectar los posibles problemas y amenazas de seguridad.
- Garantizar la utilización adecuada del sistema y la información.
- Limitar las pérdidas y conseguir a la adecuada recuperación del sistema en caso de un incidente de seguridad.
- Cumplir con el marco legal los requisitos impuestos por los clientes en sus contratos.

Existen numerosas definiciones para el término seguridad informática. En esencia, la seguridad informática es el área de la informática que se enfoca tanto en la protección de sistemas como en la protección de la propia información.

Sin embargo, esa escueta definición resulta insuficiente y se hace necesario distinguir entre dos conceptos. Por una parte, tenemos el concepto de seguridad informática y, por otra parte, el concepto de seguridad de la información. Aunque al principio ambos conceptos pueden parecer sinónimos, se tratan de áreas diferentes.

4.1. Seguridad Informática

Existen un gran número de definiciones para el concepto de seguridad informática. Según INTECO (Instituto Nacional de Tecnologías de la Comunicación, ahora INCIBE), la seguridad informática consiste en *la protección de las infraestructuras TIC que soportan un negocio o empresa* [8]. En la norma ISO 7498, en la que se recoge el modelo OSI (modelo de referencia creado en 1980 para arquitecturas de red, en contraposición a la heterogeneidad del modelo TCP/IP), se define la seguridad informática como *una serie de mecanismos que minimizan la vulnerabilidad de bienes y recursos en una organización* [9].

Aunque existen numerosas definiciones para el concepto de seguridad informática, por lo general engloban el carácter de protección de cualquier tipo de recurso tecnológico informático, muchas veces supeditado a su uso en una organización, aunque no es necesariamente obligatorio.

4.2. Seguridad de la Información

Ligado al concepto de seguridad informática se encuentra el concepto de seguridad de la información. Según INTECO, la seguridad de la información consiste en *la protección de los activos de información fundamentales para el éxito de cualquier organización* [8].

En una definición más antigua, elaborada en la norma ISO/IEC 17799 [10] se define la seguridad de la información como *la preservación de la CID*, acrónimo de “Confidencialidad, Integridad y Disponibilidad”. El concepto de CID es uno de los pilares de la seguridad de la información, en el que se ahondará más adelante.

Mientras que la seguridad informática se encarga de proteger las infraestructuras, es decir, se centra en un plano más técnico, la seguridad de la información no tiene porque hacer referencia a ningún tipo de tecnología informática o de comunicación. Los activos de información pueden ser digitales, como correos electrónicos, páginas web, imágenes, bases de datos, etc. pero no tiene por qué ser necesariamente así. También se pueden clasificar como activos de información documentos en papel, contratos, faxes, etcétera.

Al igual que ocurre con la seguridad informática, también resulta habitual tratar el término de la seguridad de la información a nivel empresarial, cuando no está exclusivamente ligado a ese campo.

4.3. Servicios de la Seguridad de la Información

La seguridad de la información proporciona una serie de servicios que tienen como objetivo proteger todo tipo de activos de información, ya sea de una organización o de un usuario particular. La suma y complementación de estos servicios es lo que permite dicha protección de la información.

4.3.1. Confidencialidad, Integridad y Disponibilidad

Dentro de estos servicios, existe el acrónimo CID (*Confidencialidad, Integridad, Disponibilidad*, en inglés CIA (*Confidentiality, Integrity, Availability*)). CID engloba los 3 servicios de la seguridad de la información existentes con mayor relevancia [11].



Figura 4.1.: Confidencialidad, Integridad y Disponibilidad

- **Confidencialidad:** garantiza que la información no se ponga a disposición ni revela a individuos, entidades o procesos no autorizados.
- **Integridad:** garantiza que la información no se modifique malintencionadamente, por parte de individuos o procesos, durante su procesado o su transmisión, y en caso de modificarse, permite detectar dichas modificaciones.
- **Disponibilidad:** garantiza el acceso y utilización de la información y los sistemas de tratamiento de la misma por parte de los individuos, entidades o procesos autorizados cuando lo requieran. Además garantiza la recuperación de la información y de los sistemas de información en caso de posibles incidentes.

4.3.2. Otros servicios

Además de CID, existen otros servicios que complementan el concepto de seguridad informática. Aunque CID consiste en tres servicios que son de obligado cumplimiento para garantizar la seguridad de la información, no resultan suficientes, ya que no cubren todo tipo de casos en los que se puede comprometer la información. Entre el diverso número que existen, seguidamente se enumeran y definen algunos de ellos, considerados de los más relevantes [12].

- **Autenticación:** garantiza que la identidad del creador de un mensaje es legítima. Permite asegurar la autoría de la información creada o modificada.

- **No repudio:** permite, mediante diferentes mecanismos, demostrar la autoría de un mensaje e impide que el usuario niegue esa circunstancia.
- **Autorización:** permite controlar el acceso a cierto sistema o información por parte de un usuario, permitiendo dicho acceso sólo a ciertos usuarios previamente designados, una vez superado el servicio de autenticación.
- **Auditabilidad:** permite registrar y monitorizar la utilización de los distintos recursos del sistema por parte de los usuarios para garantizar el correcto uso del sistema y de su información.
- **Anonimato:** permite garantizar el anonimato de los usuarios que acceden a los recursos y consumen determinados tipos de servicios, preservando así su privacidad. Puede entrar en conflicto con otros ya mencionados, como la autenticación o la auditoría del acceso a los recursos.
- **Protección a la réplica:** impide que se haga uso de ataques de repetición que engañen al sistema provocando operaciones y modificaciones de la información no deseadas.

5

Aplicaciones de la Seguridad Informática

"Las organizaciones gastan millones de dólares en firewalls y dispositivos de seguridad, pero tiran el dinero porque ninguna de estas medidas cubre el eslabón más débil de la cadena de seguridad: la gente que usa y administra los ordenadores"

— Kevin Mitnick

Con el objetivo de garantizar todos esos servicios mencionados en el capítulo anterior, se han desarrollado una gran cantidad de áreas dentro de la propia seguridad informática, que permiten cumplir con estos objetivos.

Las áreas de investigación y los campos de trabajo dentro de la seguridad informática avanzan a un ritmo acelerado debido al propio avance de la informática. El desarrollo de áreas como los smartphones o el Internet of Things (IoT) hace necesario nuevas técnicas que permitan garantizar la seguridad de la información a todos los niveles. Áreas como el ransomware, los wearables, los automóviles o el ciberspying son algunas en las que más hincapié se está haciendo en los últimos años [13].

Teniendo eso en cuenta, durante este capítulo se describirán diferentes tipos de amenazas, tras lo cual se elaborará un análisis en mayor profundidad de las principales áreas que preocupan en el momento actual a la seguridad informática, que son los smartphones, el Internet of Things y el Cloud Computing.

5.1. Malware

La seguridad informática siempre se ha relacionado con el concepto de virus informático. Si bien uno de los objetivos de la seguridad informática es evitar que este tipo de software consiga acceder a información o dañar sistemas, los virus informáticos no son la única manera que existe para comprometer la seguridad de la información. Sin embargo, el cam-

po de la seguridad informática se desarrolla después de que surgieran estas primeras piezas de malware. Es precisamente debido al surgimiento de estas que se comprende la necesidad de desarrollar toda una serie de técnicas para garantizar la seguridad de la información almacenada y procesada en estos sistemas.

El concepto de malware es relativamente novedoso. El origen de programas capaces de replicarse y distribuirse se remonta hacia 1949, cuando el mismísimo Von Neumann expuso *La Teoría y Organización de Autómatas Complejos* [14], en la cual habla sobre pequeños autómatas capaces de replicarse por sí solos. Dos décadas después, concretamente en 1971, nace el denominado como primer virus informático de la historia, llamado *Creeper* [15] (enredadera en inglés). Dicho virus no se puede considerar malware como tal ya que no causaba daño a los sistemas en los que se replicaba, simplemente mostraba un inofensivo mensaje a los usuarios ("Soy una enredadera... ¡atrápame si puedes!"). Para contrarrestar dicho virus surgió *Reaper* (segadora en inglés), el cual a día de hoy es denominado como el primer antivirus de la historia. Sin embargo, no es hasta 1984 cuando Frederick B. Cohen acuña por primera vez el término virus informático en uno de sus estudios, definiéndolo como "*Programa que puede infectar a otros programas incluyendo una copia posiblemente evolucionada de sí mismo*" [16], realizando una analogía con el mundo de la biología.

No es hasta la década de los 80, en la cual los ordenadores personales empezaron a emerger en los hogares, cuando se comienzan a desarrollar virus informáticos a los que sí que podemos considerar maliciosos. Ejemplos famosos son el virus *Viernes 13* en 1987 o el gusano *Happy* en 1999 [16]. Desde la década de los 80 y hasta la actualidad se han ido desarrollando este tipo de virus informáticos, los cuales podemos clasificar como malware.

Cabe mencionar que el malware en sí no tiene porque tener la capacidad para replicarse; se puede considerar malware cualquier pieza de software que busque un uso malintencionado o malicioso del sistema.

Dentro de la categoría de malware se han ido desarrollando diferentes piezas de software que tienen sus particularidades. Dependiendo de su funcionamiento o de los usos para los que esté diseñado el malware se puede clasificar en diferentes categorías, algunas de las cuales tienen mayor auge que otras a día de hoy. En la Figura 5.1 se puede observar una serie de datos del crecimiento y descenso de diferentes tipos de malware durante los últimos años.

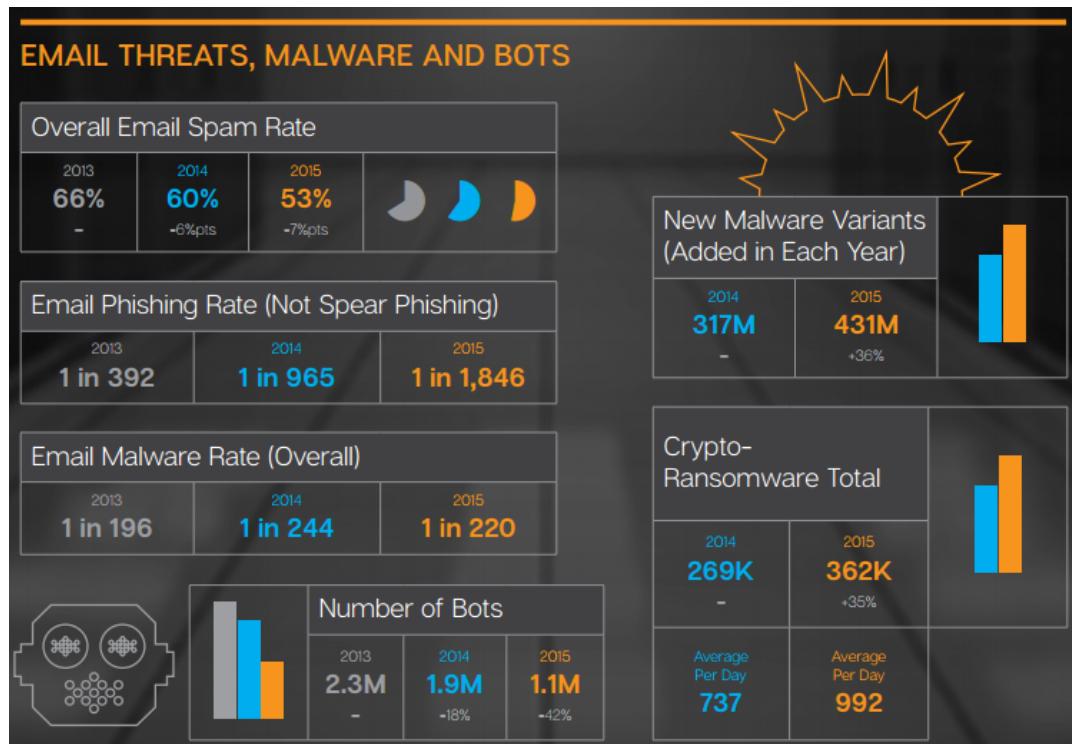


Figura 5.1.: Infografía sobre el malware en 2016 [17]

5.1.1. Gusanos

Una de las primeras categorías de malware que surge es la de los gusanos. Los gusanos (también conocidos como *worms* por su término en inglés) son programas autorreplicantes que, en vez de infectar archivos concretos, se instalan directamente en los sistemas y buscan, mediante diferentes vulnerabilidades en los sistemas o técnicas como la ingeniería social, replicarse en la medida en la que les sea posible.

Fueron uno de los primeros tipos de malware y a su vez una de las causas principales del desarrollo de la seguridad informática más defensiva. Un ejemplo es el famoso gusano *ILOVEYOU* [18], el cual infectó en el año 2000 a decenas de millones de ordenadores con Windows. Otros también famosos son *Code red*, *Sasser* o *Conficker*. Son uno de los tipos de malware que mayor daño económico han causado, causando daños estimados en miles de millones de euros solamente teniendo en cuenta los casos mencionados.

Hoy en día, aunque siguen considerándose una amenaza, han quedado relegados a un segundo plano, por una parte, por la capacidad de los sistemas de firewall o sistemas antivirus de detectarlos y neutralizarlos y, por otra parte, por el surgimiento de otros tipos de malware, como los que se mencionan en los siguientes puntos.

5.1.2. Troyanos

Otro tipo de malware famoso son los troyanos. El nombre de troyano proviene del caballo de Troya, una gran estructura de madera que usó el pueblo griego en la guerra de Troya para introducir a sus soldados en la ciudad, que estaba completamente fortificada. De la misma manera, el malware denominado troyano se camufla, es decir, se oculta como un programa legítimo imitando el comportamiento de dicho programa, cuando en realidad son una forma para dar acceso a un atacante a los recursos de dicho sistema. Uno de los troyanos más famosos fue *Zeus* [18], que llegó a infectar a más de un millón de ordenadores. También existe otros como, por ejemplo, *CryptoLocker*, que es una mezcla entre troyano y ransomware, tipo de malware que se explica a continuación.

5.1.3. Ransomware

Según McAfee Labs [13] (actualmente parte de Intel) el ransomware es una de las mayores amenazas a día de hoy. El ransomware es un tipo de malware informático que busca beneficio económico en base a la extorsión hacia los usuarios. Para ello esta clase de malware cifra los archivos de los usuarios para que sean inaccesibles para ellos, de tal manera que solo mediante el pago de cierta cantidad económica puedan recuperar dichos archivos.

Existen varias familias de ransomware, entre los que destacan *CryptoWall 3*, *CTB-Locker* o *CryptoLocker*. El concepto de familias proviene de que en su mayoría se tratan del mismo malware con prácticamente el mismo funcionamiento, pero con ligeras variaciones en su código fuente que les permiten eludir las posibles medidas de seguridad. En la Figura 5.2 se puede observar su crecimiento durante estos últimos años.

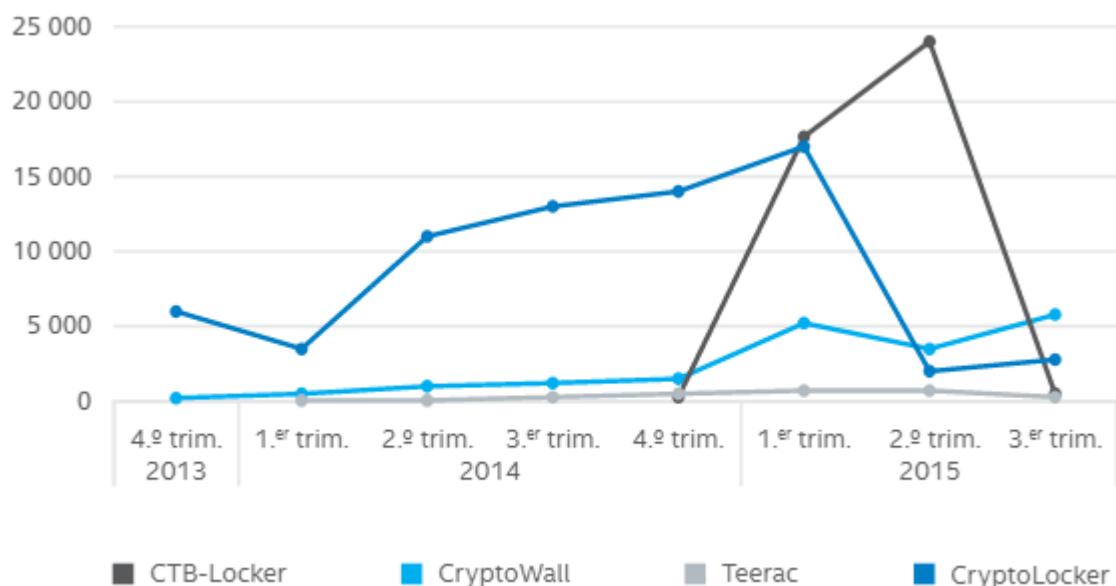


Figura 5.2.: Nuevas muestras de familias de ransomware prominentes [13]

Podemos encontrar incluso ejemplos relativamente recientes como puede ser *WannaCry*, ransomware que afectó hace no mucho a decenas de países inutilizando sistemas de diversa índole, desde la Intranet de Telefónica hasta varios hospitales en Reino Unido [19] [20].

5.1.4. Rootkits

Los rootkits son una clase de malware que tiene como objetivo principal escalar privilegios en el sistema, también conocido como conseguir acceso de superusuario o *root*. No hace falta mencionar que con dichos privilegios el daño que puede causar en un sistema es inmensurable, pudiendo dejarlo completamente inutilizado o eliminar absolutamente toda su información.

5.1.5. RATs

Los *Remote Administration Tools* también conocidos como RAT, buscan ofrecer a los posibles atacantes una puerta trasera para acceder a los sistemas para así poder introducir otros tipos de malware específicos.

5.1.6. Spyware

Un tipo de malware concreto bastante popular desde la década del 2000 es el denominado spyware, cuyo objetivo es obtener información de las acciones de un sistema, infectando la

máquina destino de manera inadvertida para el usuario, el cual desconoce que su actividad y/o sus datos están siendo monitorizados.

5.1.7. Keyloggers

El *Keystroke Logger*, mejor conocido como Keylogger, es otro tipo de malware usado actualmente debido a su efectividad cuyo objetivo consiste en obtener la información de las pulsaciones que se realizan en un teclado de un sistema concreto. Este mecanismo tiene como principal objetivo extraer información, especialmente contraseñas, de un usuario para después poder acceder a los servicios que usa dicho usuario. Se puede encontrar Keyloggers tanto a nivel físico, en forma de pequeños dispositivos¹ USB con una memoria interna², como a nivel de software, estos últimos especialmente sencillos de implementar con solo unas pocas líneas de código [21].

5.2. Dispositivos móviles

El malware afecta a los sistemas y es un riesgo para la seguridad de estos, pero no es el único. Además hay que tener en cuenta que en los últimos años el mundo de las tecnologías de la información ha sufrido cambios drásticos. Lejos queda el modelo clásico de ordenadores personales y servidores estándar que conforman Internet. Han entrado a la palestra nuevas áreas que se han ido desarrollando durante los últimos años. A medida que estas áreas han ido destacando, el campo de la seguridad informática ha ido avanzando para satisfacer las necesidades de seguridad de estas áreas y adaptarse a ellas mediante nuevas técnicas y medidas de seguridad.

Una de las áreas que ha destacado estos últimos años es el área de la seguridad de dispositivos móviles. Datos como que el consumo de Internet a día de hoy es mayor en este tipo de dispositivos que en ordenadores convencionales resultan sorprendentes. En el caso concreto de España, del 78,7 % de la población que se conecta regularmente a Internet, el 88,3 % de los usuarios lo hace a través de un smartphone [22]. Teniendo eso en cuenta, es lógico que la seguridad estos dispositivos sea una prioridad y a su vez uno de los puntos de enfoque por parte de todo tipo de atacantes.

Los smartphones tienen una gran cantidad de usos diferentes. Aparte de ser la principal herramienta de comunicación, también son dispositivos mediante los que se procesa una gran cantidad de información, desde información personal como fotos o mensajes a información relacionada con el ámbito empresarial como documentos o correos. Debido a que la información ya no solo se procesa y transmite desde ordenadores tradicionales, sino que en su mayoría se hace desde dispositivos móviles, resulta necesario definir e implantar técnicas y procedimientos de seguridad en este tipo de dispositivos.

¹<https://www.keelog.com/>

²<https://store.hackaday.com/products/usb-rubber-ducky-deluxe>

5.2.1. Seguridad en smartphones

Aunque en esencia los smartphones se traten de dispositivos basados en Linux, Unix o derivados, este tipo de dispositivos tienen sus particularidades con respecto a los ordenadores personales. Primeramente, estos sistemas no disponen de la arquitectura x86/x86-64 (al menos en su gran mayoría), sino que se basan en procesadores con arquitectura ARM. Debido a esto y al enfoque de uso que tienen, usan diferentes sistemas operativos que distan de los tradicionales. Los sistemas operativos más usados para smartphones son Android e iOS, que se encuentran en un 86,22 % y 12,88 % de los dispositivos vendidos en 2016 [23], respectivamente.

Ambos son sistemas que, al estar por defecto más limitados en uso y configuración, resultan *a priori* más seguros que los sistemas operativos de escritorio. Existen una serie de técnicas que se suelen aplicar independientemente del sistema operativo utilizado, como pueden ser la firma de aplicaciones o el uso de cifrado para ciertos casos. Aun así, este tipo de medidas son prácticamente obligatorias para disponer de sistemas mínimamente seguros. Lo que diferencia a cada uno de los sistemas operativos desde el punto de vista de la seguridad es tanto el enfoque de seguridad que le dan al sistema como las técnicas, medidas, arquitecturas de software o algoritmos, que cambian de un sistema operativo a otro.

5.2.1.1. Seguridad en iOS

En dispositivos con iOS la seguridad se enfoca desde varios puntos. Uno de los puntos en los que iOS destaca es en su tienda de aplicaciones, la *App Store*, que dispone de ciertos filtros y controles de seguridad a la hora de publicar aplicaciones con el objetivo de dificultar en la mayor medida posible que a través de las aplicaciones entre algún tipo de malware.

Por otro lado, uno de los puntos más fuertes de iOS es el relacionado con el cifrado. Debido a la ventaja que otorga el hecho de que la misma compañía elabore tanto el hardware como las diferentes capas de software de sus dispositivos, se pueden integrar medidas de seguridad en el propio hardware que funcionen de manera conjunta a su software. Una de las más destacables es la incorporación del coprocesador llamado *Secure Enclave* [24]. Este coprocesador (disponible en el Apple S2, y en el Apple A7 y posteriores) se encarga de todas las operaciones criptográficas para la gestión de claves de cifrado de datos y mantiene la integridad de la protección de datos, incluso si el kernel del sistema se ha visto comprometido. Usa memoria cifrada e incluye un generador aleatorio de números por hardware. La comunicación entre este coprocesador y el procesador principal está completamente aislada del resto del sistema.

Esto tiene dos ventajas. La primera, que el cifrado por hardware, al realizarse de manera transparente a cualquier capa de software, resulta difícilmente evitable. Por otra parte, a nivel de rendimiento, resulta mucho más eficiente que cualquier tipo de cifrado por software.

Además, todos los dispositivos iOS tienen un sistema de cifrado mediante hardware y AES-256 que se sitúa entre la memoria principal y el almacenamiento flash [24], de tal

manera que los datos almacenados en los dispositivos se mantienen cifrados sin que se pierda rendimiento en ello. Para cada sistema de cifrado se crea identificadores únicos tanto para cada usuario (UID) como para cada grupo distinto de dispositivos (GID), ambos integrados en el propio hardware. Cada UID es único y no es conocido ni por Apple ni por ninguna aplicación ni usuario. Cada GID es común a una familia de dispositivos y se usa para distinguirlos entre sí. Esto hace que los datos que se almacenan sean altamente difíciles de descifrar.

Las operaciones de cifrado y descifrado concretas siguen el esquema que se muestra en la Figura 5.3 , haciendo uso también de una contraseña propia elegida por el usuario, la que se muestra en el diagrama con el nombre de *Passcode Key*.

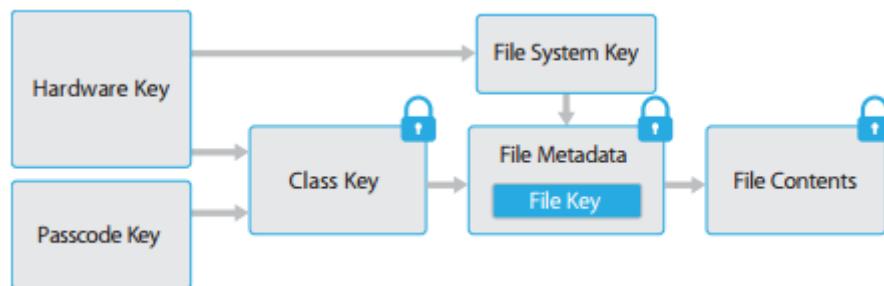


Figura 5.3.: Esquema de cifrado de un archivo en iOS

5.2.1.2. Seguridad en Android

El modelo de seguridad en Android se diferencia del modelo de iOS en varios aspectos. Más que en un fuerte cifrado a todos los niveles, el cual también usa pero en menor medida, el enfoque se basa en hacer lo más segura posible cada aplicación. Esto es lógico teniendo en cuenta que, por un lado, la tienda de aplicaciones de Android, la *Play Store*³, es menos estricta que la de iOS y que, por otro lado, Google (propietaria de Android) no fabrica los dispositivos que integran dicho sistema. Por ello, la arquitectura de software del sistema resulta más jerárquica, y es fundamental para las diversas medidas de seguridad que se implementan. En la Figura 5.4 se muestran las diferentes capas de dicha arquitectura.

³<https://play.google.com/store/apps>



Figura 5.4.: Las diferentes capas que componen la arquitectura de Android

La esencia de la seguridad en Android reside en controlar y limitar la interacción entre las diferentes capas para evitar fallos de seguridad. Para ello se hace uso de diferentes técnicas [25] [26] como:

- El *sandboxing*, donde el código de cada aplicación se ejecuta de manera aislada al resto de aplicaciones y, además, los datos de cada aplicación se mantienen separados.
- Un framework de aplicaciones que implementa ciertas técnicas comunes de seguridad como, por ejemplo, métodos criptográficos, sistemas de permisos o una comunicación entre procesos (*IPC, Interprocess Communication*) segura.
- Implementar diversas tecnologías como ASLR (*Address Space Layout Randomization*), NX (*No-eXecute*), *ProPolice*, *safe_iop*, *OpenBSD dlmalloc*, *OpenBSD calloc*, y *Linux mmap_min_addr* para evitar errores de *buffer overflow*, para evitar que se ejecuten instrucciones concretas o para mitigar otro tipo de riesgos relacionados con la gestión de la memoria.

- Un sistema de cifrado de archivos mediante AES-256 para proteger los datos almacenados en los dispositivos.
- Un sistema de permisos tanto a nivel de usuario como al nivel de cada aplicación concreta para controlar las acciones de los usuarios y las aplicaciones.

Este enfoque hace que, por una parte, el propio Android mediante su arquitectura y el uso de una VM (Virtual Machine) específicamente creada para el sistema (como se puede observar en la Figura 5.4) ofrezca un repertorio de mecanismos de seguridad que hacen de él un sistema relativamente seguro. Por otra parte, también pone empeño en que los desarrolladores tienen la necesidad de implementar medidas de seguridad en sus aplicaciones para garantizar la seguridad de las mismas. Para ello, el SDK de Android proporciona acceso a un gran número de utilidades como, entre otras, utilidades de cifrado, de manejo de credenciales o de intercambio de mensajes.

5.3. Internet of Things

También junto a la seguridad de los dispositivos móviles, que viene determinada por el auge de dichos dispositivos, el área de la seguridad del Internet of Things (conocida como IoT) es una de las áreas que más impulso está teniendo, debido al mismo motivo. Cada vez hay una mayor cantidad de dispositivos conectados a Internet, que no ha hecho más que crecer exponencialmente debido al IoT. Dispositivos que ya no se tratan exclusivamente de móviles, tablets u ordenadores, sino de otro tipo de dispositivos, como electrodomésticos o automóviles, que han sido dotados de capacidad de computación y conexión a Internet para ampliar sus capacidades y poder recopilar información de ellos. En esencia, el IoT se basa en una serie de dispositivos a los que, mediante el uso de sensores, procesadores y conexión a Internet, se les da la capacidad de recoger y transmitir información a otros dispositivos, además de recibir dicha información para actuar de una manera u otra.

5.3.1. Aplicaciones

Las aplicaciones de lo que se denomina Internet of Things son diversas y variadas. Prácticamente cualquier sistema electrónico que disponga de una serie de sensores mediante los cuales obtenga información puede ver multiplicadas sus capacidades añadiéndole una conexión a la red. De esta manera se le dota de capacidad para ya no solo transmitir información sino incluso poder interactuar de forma más compleja con otra serie de sistemas [27] [28].

5.3.1.1. Smart homes

Mediante tecnologías IoT se puede dotar a los diferentes elementos domóticos de un hogar o edificio de capacidades para reaccionar de manera automática a diferentes cambios que se puedan dar. Desde controlar la luz o el agua hasta controlar la temperatura o las persianas,

todo de tal manera que los subsistemas sean capaces de comunicarse y actuar de manera conjunta en función de la información que reciben.

5.3.1.2. Smart Cities

El término Smart Cities se refiere al sistema ciberfísico que conforma una red avanzada de comunicación para optimizar el uso de los sistemas físicos de una ciudad. Puede optimizar servicios como por ejemplo el tendido eléctrico, mediante el análisis de consumo de toda la red eléctrica, o la red de carreteras, mediante sistemas avanzados de control de tráfico (del cual los vehículos autónomos se pueden beneficiar). Esos son solo unos breves ejemplos de lo que un sistema de este tipo puede llegar a optimizar dentro de los procesos que se dan en una ciudad.

5.3.1.3. Monitorización medioambiental

El IoT también puede servir para monitorizar cambios en el medio ambiente e integrar todo esa adquisición de datos para que funcione de manera conjunta con otros sistemas. Una serie de sistemas de tiempo real, a los que se añade la capacidad de comunicarse e interactuar entre sí, pueden ser una plataforma sólida mediante la cual detectar y monitorizar anomalías que afecten a la vida humana, animal y vegetal. Además, pueden ser una herramienta fundamental para obtener información que permita mitigar los efectos de cualquier tipo de contaminación medioambiental.

5.3.1.4. Sanidad

La sanidad también es uno de esos sectores que se puede beneficiar de los avances en el IoT. Por ejemplo, la capacidad para monitorizar diferentes parámetros (como la temperatura corporal, la presión sanguínea o la actividad cardíaca) de manera constante puede ser una herramienta fundamental a la hora de prevenir y tratar diversas enfermedades. Además del nivel patológico, se pueden usar diferentes dispositivos para tomar información que permita mejorar el estilo de vida y la salud de las personas. Dispositivos como los wearables son un buen ejemplo de ello, entre los que se encuentran las famosas pulseras cuantificadoras.

5.3.1.5. Smart Business

El término Smart Business hace referencia a todas esas aplicaciones del Internet of Things que se usan para mejorar tanto la logística interna como los productos y servicios que ofrece una empresa o negocio. La gestión de inventario en tiempo real o la automatización del envío de información sobre productos para ofrecer un mejor servicio post venta son dos ejemplos en los que enfatizan cada vez más empresas.

5.3.1.6. Seguridad y vigilancia

Aunque pueda ser polémico, el Internet of Things también se puede aplicar en el área de la seguridad y vigilancia. Los sistemas interconectados de cámaras o los sensores para detectar químicos nocivos son dos ejemplos. El primero de ellos no se respeta la privacidad del usuario, mientras que el segundo sí. Debido a eso, aplicaciones como la primera son objeto de numerosas críticas por parte de diversas organizaciones.

5.3.2. Seguridad en Internet of Things

Uno de los principales problemas del Internet of Things reside en que ya no solo se trata de el peligro de que la información se vea comprometida o revelada sino que, al tratarse de otra serie de dispositivos, los peligros son mucho mayores [29][30]. Esto es algo de lo que el mundo de la seguridad informática ya se ha percatado. Estimando que el número de dispositivos conectados a Internet pasará de 6,4 mil millones en 2016 a 21 mil millones en 2020 [31] debido al auge del IoT, resulta fundamental indagar en ello.

En la actualidad existen gran cantidad de casos en los que diferentes dispositivos salen al mercado con esas capacidades para conectarse a Internet pero que carecen de prácticamente ningún tipo de medida de seguridad. Se han dado casos como el de Miele [32] donde a causa de la falta de seguridad en un modelo de sus lavavajillas, se puede llegar a acceder a la red local privada a donde este conectado de manera sencilla. Otro caso es el de los automóviles Tesla [33], donde más de una vez se han descubierto vulnerabilidades que permiten obtener el control total de sus vehículos y conducirlos remotamente a placer del atacante. Esto es gravísimo, pudiendo llegar a poner en riesgo la vida de personas. Por ello, los servicios de la seguridad informática son mas necesarios si cabe para este campo.

5.4. Cloud Computing

El Instituto Nacional de Estándares y Tecnologías de Estados Unidos, conocido como NIST (*National Institute of Standards and Technologies*) define el Cloud Computing como:

"Un modelo para hacer posible el acceso a red adecuado y bajo demanda a un conjunto de recursos de computación configurables y compartidos (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios...) cuyo aprovisionamiento y liberación puede realizarse con rapidez y con un mínimo esfuerzo de gestión e interacción por parte del proveedor de dicha nube". [34]

En esencia, podemos entender como Cloud Computing a una extensión de las aplicaciones de Internet, que eran principalmente aplicaciones web y transmisión de datos, a nuevas posibilidades, entre las cuales se incluyen plataformas completas y software más complejo, ofrecidas como un modelo de servicio configurable y escalable.

Las estructuras cloud se pueden clasificar en tres categorías: públicas, comunitarias y privadas. En las primeras el acceso a sus servicios es abierto, mientras que en las segundas está acotado a un determinado entorno, normalmente una empresa. En la última, aunque no se tiene acceso público a sus servicios, estos se comparten entre varias entidades.

5.4.1. Tipos de servicios

El Cloud Computing puede ofrecer una serie de servicios a sus usuarios, y en función de qué servicio se ofrezca, se clasifica en una de las tres categorías diferentes existentes que se mencionan a continuación.

5.4.1.1. Software as a Service

Software as a Service consiste en un despliegue de software en el cual las aplicaciones y los recursos computacionales se han diseñado para ser ofrecidos como servicios de funcionamiento bajo demanda. De esta forma se reducen los costes tanto de software como hardware, así como los gastos de mantenimiento y operación. En esta categoría las consideraciones sobre seguridad son responsabilidad del proveedor del servicio, estando limitada la configuración que puede realizar el suscriptor.

5.4.1.2. Platform as a Service

Platform as a Service es el servicio donde se ofrece una plataforma en la cual el suscriptor del servicio puede desplegar su propio software. Con ello se reducen los costes y la complejidad de la compra, el mantenimiento, el almacenamiento y el control del hardware y el software que componen la plataforma. El suscriptor del servicio tiene un control total sobre las aplicaciones y un control parcial de la configuración del entorno ya que la instalación de los entornos dependerá de la infraestructura que el proveedor del servicio haya desplegado. La seguridad se comparte entre el proveedor del servicio y el suscriptor ya que el suscriptor aporta su propio software.

5.4.1.3. Infrastructure as a Service

Infrastructure as a Service es un modelo en el cual la infraestructura (servidores, software y equipamiento de red) es gestionada por el proveedor como un servicio bajo demanda, en el cual se pueden crear entornos para desarrollar, ejecutar o probar aplicaciones. El fin principal de este modelo es evitar la compra de recursos por parte de los suscriptores, ya que el proveedor ofrece estos recursos como objetos virtuales accesibles a través de un interfaz de servicio. El suscriptor mantiene generalmente la capacidad de decisión del sistema operativo y del entorno que instala. Por lo tanto, la gestión de la seguridad corre principalmente a cargo del suscriptor.

5.4.2. Seguridad en Cloud Computing

Debido al auge del Cloud Computing surge la CSA (*Cloud Security Alliance*). La CSA se define como una organización internacional sin ánimo de lucro que tiene como objetivo promover el uso de una serie de buenas prácticas para garantizar la seguridad en la nube. Entre otras actividades, la CSA elabora un informe periódico con las mayores amenazas de seguridad de sistemas de Cloud Computing [35]. Estas amenazas se actualizan regularmente buscando el consenso de los expertos. A continuación, se resumen las amenazas descritas en este informe. En el informe de 2016, la CSA marca como amenazas más graves, ordenadas por orden de importancia, los siguientes 12 puntos:

1. Brechas de Datos
2. Manejo inseguro de Credenciales de Acceso
3. Interfaces y APIs inseguros
4. Vulnerabilidades de Sistemas y Aplicaciones
5. Secuestro de Cuentas
6. Amenazas Internas
7. Advanced Persistent Threats (APTs)
8. Pérdida de Datos
9. Diligencia insuficiente
10. Abuso y mal uso del Cloud Computing
11. Denegación de Servicio
12. Problemas derivados de las tecnologías compartidas

Con el auge del Cloud Computing, este tipo de amenazas son cada vez más graves. Desde servicios de almacenamiento de datos hasta plataformas de despliegue de aplicaciones, cada vez más empresas y usuarios delegan almacenamiento, infraestructura, o software en servicios de Cloud Computing, por lo que, en primer lugar, ser conscientes de la ventajas, inconvenientes y, sobre todo, riesgos que acarrea el uso de estos servicios resulta fundamental para después, en segundo lugar, ser capaces de securizarlos en la mayor medida posible y hacer un uso seguro de estos servicios.

6

Pentesting

"Si piensas que la tecnología puede solucionar tus problemas de seguridad, está claro que ni entiendes los problemas ni entiendes la tecnología"

— Bruce Schneier

Dentro de la seguridad informática un test de intrusión o *pentesting*, como se conoce en inglés, tiene como objetivo evaluar los diferentes niveles de seguridad de una red de sistemas informáticos, mediante la simulación en un entorno controlado de un ataque por parte de un usuario malicioso, conocido comúnmente como hacker [36]. El propósito de una prueba de penetración es determinar la viabilidad de un ataque y la cantidad de impacto que puede causar este mismo.

6.1. Objetivos

A día de hoy el *pentesting* es una de las técnicas más utilizadas para garantizar tanto la seguridad de la información como la propia seguridad de los sistemas de empresas u organizaciones. Es uno de los pilares básicos en las auditorías de seguridad informática, en las que empresas contratan a expertos en seguridad para evaluar la fortaleza de sus sistemas informáticos.

El *pentesting* no es un área con unas técnicas de actuación concretas. El *pentesting* usa una grandísima cantidad de técnicas, a las cuales se van añadiendo técnicas que van surgiendo, para determinar la seguridad de un sistema informático y, en particular, conocer las debilidades que tiene el sistema informático.

Haciendo un símil con otras áreas completamente diferentes, un *pentesting* podría equivaler a una serie de pruebas prácticas que se realizan, por ejemplo, en controles de calidad para todo tipo de productos, en los cuales se ve que la mejor forma de probar el correcto

funcionamiento de dichos productos consiste en someterlos a una serie de pruebas. Dichas pruebas están basadas en los entornos reales en los que se utilizan esos productos.

Por ejemplo: la mejor manera de probar la efectividad de un chaleco antibalas es realizando disparos sobre él para ver hasta qué punto los materiales como el Kevlar o de los tejidos que han sido compuestos para dicho chaleco son adecuados para parar una bala. De la misma manera, la mejor forma de probar si un sistema informático o una red compuesta de varios sistemas es segura, o mejor dicho, hasta qué punto es segura, es elaborar ciertas pruebas e intentar explotar vulnerabilidades que puedan existir en estos sistemas. Todo ello con el objetivo final de sacar conclusiones y, en base a esas conclusiones, mejorar y fortificar ese sistema.

Aunque, de la misma manera que jamás se harían pruebas con el chaleco antibalas comprometiendo la integridad de ninguna persona física, tampoco se elabora un pentesting comprometiendo de manera real ningún sistema. El pentesting, al igual que las pruebas del chaleco, se elaboran ambas en un entorno controlado y limitado.

Teniendo en cuenta que la seguridad de los sistemas informáticos es vital para la continuidad del negocio y el correcto desarrollo de las actividades de una empresa u organización, y siendo el pentesting una de las mejores herramientas para garantizar dicha seguridad, resulta lógico que este sea uno de los métodos más usados por todo tipo de empresas u organizaciones.

6.2. Partes

Dentro de un pentesting se diferencian una serie de etapas, cada una de las cuales tiene objetivos particulares y concretos. Aunque algunas de estas partes puedan tener sentido de manera independiente, en su conjunto permiten un completo análisis de un sistema informático del cual sacar conclusiones que permitan preservar la seguridad.

A continuación, se enumeran y definen las fases de un test de intrusión o pentesting, que ya se encuentran estandarizadas en el PTES (*Penetration Testing Execution Standard*) [37]:



Figura 6.1.: Logo de Penetration Testing Execution Standard

1. **Reglas del juego, alcance y términos del test de intrusión:** en esta fase se establece

una serie de protocolos entre el realizador del pentesting (normalmente una auditoría de seguridad informática) y el auditado (normalmente una empresa u organización). En esta fase se establecen los objetivos a los que llegar y los límites a los que tendrán que adherirse los auditores.

2. **Recolección de información:** fase en la que se obtiene información del sistema para posteriormente analizarla.
3. **Análisis de las vulnerabilidades:** fase en la que mediante la información obtenida, se pasa a determinar las vulnerabilidades del sistema.
4. **Explotación de las vulnerabilidades:** después de determinar esas vulnerabilidades, se pasa a intentar explotarlas para visualizar su gravedad y el daño real que pueden causar.
5. **Postexplotación del sistema:** tras haber logrado explotar una vulnerabilidad, se intenta minar lo más posible el sistema, intentando lograr un efecto en cadena para ver, hasta qué punto, se puede dañar un sistema mediante una vulnerabilidad concreta.
6. **Generación de informes:** finalmente, se condensa todo el proceso elaborado y las conclusiones a las que se ha llegado en informes de tal manera que, a través de esos informes, se pueda actuar para corregir las vulnerabilidades y fortalecer el sistema.

De todas las fases enumeradas, a continuación se profundizará en las más importantes.

6.3. Recogida de información

"La información es poder". Esa frase, atribuida a Francis Bacon (aunque se desconoce si realmente llegó a pronunciarla en algún momento), no podría ser más cierta actualmente. La informática no es más que la ciencia que trata la información y la seguridad informática es campo que tiene como objetivo protegerla. Por ello, a la hora de realizar un pentesting, resulta esencial obtener la mayor cantidad de información posible. El éxito de muchos de los ataques e intrusiones que sufren empresas y organizaciones se debe en gran parte a la gran cantidad de información que directa e indirectamente un atacante es capaz de obtener sobre sus sistemas [38].

Por ello, la fase de recogida de información o *Information Gathering* resulta fundamental a la hora de realizar un test de intrusión. A mayor cantidad de información obtiene un atacante de un sistema mayor probabilidad de éxito tendrá al atacar.

Dependiendo desde qué punto se realiza, la recogida de información se puede separar en dos categorías: *External Footprinting* e *Internal Footprinting*. La primera hace referencia a obtener información desde fuera del sistema y la segunda a obtener información dentro del sistema.

6.3.1. Internal Footprinting

El Internal Footprinting engloba toda la recogida de información que se realiza una vez se tiene acceso, parcial o completo, al sistema o la red de la que se desea obtener la información. Este tipo de recogida de información no se realiza en la segunda fase de un pentesting, es decir, al comienzo del pentesting, sino que se realiza en la fase de post explotación del sistema, la quinta fase de un test de intrusión. Es lógico que, sin todavía haber explotado ninguna vulnerabilidad, y por ende todavía no se haya accedido al sistema, no se pueda recoger información dentro de él.

6.3.2. External Footprinting

El External Footprinting engloba toda la recogida información, que al contrario que en el Internal Footprinting, se realiza desde fuera del sistema. Este tipo de recogida información sí que se realiza en la segunda fase de un test de intrusión, y de hecho es el primer paso esencial a realizar. A través de diversas técnicas se busca obtener la mayor cantidad de información posible de una red o un sistema para, posteriormente, analizarla y encontrar vulnerabilidades.

Las técnicas de recogida de información englobadas dentro del External Footprinting se pueden dividir a su vez en dos subcategorías, en función del grado de agresividad de las mismas.

6.3.2.1. Active Footprinting

Por un lado, está el descubrimiento activo, denominado Active Footprinting, que destaca por interactuar directamente con la infraestructura del sistema objetivo mediante consultas al DNS, análisis de las cabeceras HTTP, enumeración de puertos y sus servicios, etcétera [36]. Seguidamente se explicarán brevemente en qué consisten algunas de estas técnicas, sin la intención de entrar en las herramientas de software concretas.

Escaneos DNS Una de las formas más comunes a la hora de obtener información consiste en obtener información de los servidores DNS. DNS (Domain Name Services) es un protocolo que permite la conversión entre direcciones de red numéricas, como son las IPs, y direcciones FQDN (Full Qualified Domain Name), que son direcciones del estilo miweb.es. De esta manera, DNS provee una capa de abstracción para hacer más sencillo las conexiones por parte de los usuarios y administradores a otros servicios.

La transformación de un nombre de dominio a una dirección IP se conoce como resolución DNS, y su proceso inverso (de IP a nombre de dominio) se conoce como resolución inversa. Este tipo de operaciones se da en los servidores DNS, de los cuales, mediante diferentes técnicas se puede extraer información relevante, hasta tal punto que se puede llegar a determinar ciertos aspectos de la topología de la red en la que se encuentra el sistema al que estamos intentando acceder.

Fingerprinting El Fingerprinting consiste en obtener información del propio sistema al que se intenta acceder. Datos como el sistema operativo y su versión o las aplicaciones que usa son esenciales a la hora de buscar vulnerabilidades que se puedan explotar en el sistema para lograr acceso a este, aunque no se limita solo a este tipo de datos. Obtener el servidor web que usa determinado dominio o el CMS (Content Management System) que usa dicho servidor web es fundamental a la hora de determinar vulnerabilidades concretas para dicho software. Existen herramientas tanto para obtener esta información como para relacionarla con bases de datos, previamente elaboradas, en las que se encuentran gran cantidad de vulnerabilidades que ya han sido detectadas.

SMTP Otra de las áreas mediante las que se puede obtener información es la relacionada con el protocolo SMTP. Con ello se puede obtener desde información de los dominios de correo electrónico usados hasta direcciones de correo electrónicas concretas, lo que permite centrarse en vulnerabilidades para dichos dominios. Esto deriva en que se pueda lograr suplantar la identidad de cierto usuario.

6.3.2.2. Passive Footprinting

Por otro lado se encuentra el descubrimiento pasivo, lógicamente denominado Passive Footprinting, que recurre a la consulta de la información previamente indexada por motores de búsqueda, registros públicos, foros, etcétera, por lo que no interactúa directamente con el sistema a penetrar [36].

Whois Whois es un protocolo TCP que permite obtener datos sobre el propietario de un nombre de dominio o una dirección IP. Entre los datos que se pueden obtener, se encuentran el correo electrónico, el nombre completo, la ciudad, el código postal o el número de teléfono del propietario. Estos datos son de fácil acceso, existiendo incluso herramientas web que te permiten obtener dicha información mediante el mencionado protocolo.

Hacking con buscadores Los buscadores como Google o Bing son utilizados por la gran mayoría para encontrar sitios web. Lo que no todo el mundo conoce es que se tratan de una poderosísima herramienta para obtener información adicional sobre uno o varios sitios web. La gran mayoría de buscadores disponen de parámetros avanzados de búsqueda que permiten buscar palabras concretas en las URLs, buscar por una determinada extensión o buscar dentro un sitio web concreto. De esta forma, si se sabe qué buscar, se puede obtener información sobre qué vulnerabilidades se pueden explotar en cierta página web.

A parte de los buscadores tradicionales, también existen una serie de buscadores especializados en la búsqueda de dispositivos, enfocados al IoT, en los que se pueden buscar desde webcams hasta electrodomésticos, con la condición de que estén conectados a Internet. Estos buscadores, como Shodan¹, permite también filtrar búsquedas y realizar búsquedas

¹<https://www.shodan.io/>

avanzadas que permitan obtener información sobre otros sistemas que no sean directamente servidores web.

Social network engineering La ingeniería social, aunque no dispone de herramientas concretas, son técnicas para obtener información a partir de las redes sociales. En las redes sociales, muchas veces de manera inconsciente o sin comprender las consecuencias que puede acarrear, se comparte una gran cantidad de información personal que, bien analizada, puede ser determinante para elaborar un ataque concreto.

6.4. Análisis de vulnerabilidades

Una vez obtenida toda la información posible sobre el sistema, se pasa a la siguiente fase definida en el PTES, que consiste en analizar qué vulnerabilidades concretas se pueden explotar. La información recogida se usa para obtener como resultado final un listado de vulnerabilidades que se pueden llegar a explotar en el sistema. Esta fase de análisis pasa por tres períodos.

6.4.1. Pruebas

En el primer periodo se realiza una serie de pruebas basándose en la información que disponemos del sistema, que previamente hemos obtenido. La información que nos proporcionen estas pruebas resulta importante ya que, cuanto mayor número de pruebas se realicen mediante diversas herramientas y técnicas, mejores resultados se obtendrán. Estas pruebas se engloban en pruebas pasivas o activas.

6.4.1.1. Activas

Las pruebas activas requieren interactuar directamente con el componente a auditar. Tienen una estrecha relación con el Active Footprinting, ya que se basan en la información obtenida de esa manera. Dentro de éstas se incluyen las siguientes categorías:

- **Automatizadas:** mediante diversas herramientas de software se interactúa con el sistema, enviando peticiones, escaneando los servicios, etc.
- **Conexión manual:** para evitar falsos positivos que puedan dar las pruebas automatizadas, se llevan a cabo pruebas manuales, que realizan los mismos pasos que las automáticas pero sin el uso de dichas herramientas de análisis de vulnerabilidades.
- **Ofuscadas:** con el objetivo de evitar la detección o el bloqueo por parte de sistemas IDS (Intrusión Detection System), IPS (Intrusion Prevention System) o WAF (Web Application Firewall), se realizan pruebas que difieren en su comportamiento a las

pruebas tradicionales, alargando tiempos de espera entre peticiones, modificando ciertos aspectos de dichas peticiones o alternando entre objetivos.

6.4.1.2. Pasivas

La pruebas pasivas consisten en analizar la información obtenida mediante Passive Footprinting. Analizar dicha información puede permitirnos suponer la existencia de cierta vulnerabilidad en el sistema. A diferencia de las pruebas activas, estas contienen un mayor componente subjetivo y abstracto.

6.4.2. Validación

Una vez elaboradas una serie de pruebas necesitamos correlacionar la información extraída de ellas. En este paso, el objetivo es enmarcar la vulnerabilidad encontrada en un apartado técnico, clasificándola en una serie de categorías e identificándola de una manera concreta. Esta clasificación se puede realizar a varios niveles. El más concreto consiste en identificarlas mediante, valga la redundancia, identificadores concretos como el CVE² (*Common Vulnerabilities and Exposures*). También se puede hacer a un nivel más global, mediante las categorías marcadas en diversas normas, como pueden ser el *NIST SP 800-53*³ o la *Guía OWASP*⁴.

6.4.3. Investigación

Tras la identificación de una vulnerabilidad, y después de haberla categorizado correctamente, es necesario evaluar su gravedad. Para ello se procede a realizar una investigación sobre dicha vulnerabilidad. Esta investigación puede ser privada, elaborando pruebas a nivel interno, como ataques de fuerza bruta o configurar réplicas del entorno mediante el uso de máquinas virtuales (VM) para emular el entorno real y hacer pruebas con ello.

6.5. Explotación de vulnerabilidades: Ataques de penetración

En las dos secciones anteriores se ha explicado la recogida y el análisis de la información, respectivamente. La sucesión de esas dos fases tienen como objetivo final explotar una serie de vulnerabilidades. En ello reside la esencia del pentesting, penetrar en un sistema mediante la explotación de dichas vulnerabilidades, realizando una serie de ataques a estos. Dependiendo del tipo de ataque, la vulnerabilidad que se explota o el objetivo del ataque, se

²<https://cve.mitre.org/>

³<https://nvd.nist.gov/800-53>

⁴<https://www.owasp.org/>

pueden clasificar en prácticamente una infinidad de categorías. En los siguientes puntos se explican algunas de ellas, que son consideradas de mayor relevancia.

6.5.1. Ataques de contraseñas

Las contraseñas son un mecanismo de sobra conocido, que se remonta a mucho antes de la invención de la informática e incluso son anteriores a la propia formalización de la lógica que cimenta todo el desarrollo de la informática y la electrónica digital. Si desde la antigüedad se llevan usando las contraseñas para limitar el acceso solo a ciertas personas a ciertos sitios, en la actualidad se usan para limitar el acceso por parte de cierto usuario a cierto sistema o servicio informático. A día de hoy son, aun con el auge de métodos biométricos, la herramienta de control de acceso más usada.

Partiendo de esa base, los ataques de contraseñas son cualquier técnica o mecanismo orientados a descifrar o romper las contraseñas usadas para proteger esos sistemas. Cabe destacar que todo mecanismo de autenticación por contraseñas se basa en la comparación de *hashes* y no en la comparación de contraseñas, dando autorización solo cuando el hash de la contraseña introducida coincide con el almacenado previamente, que ha sido generado con anterioridad mediante la contraseña elegida para la protección de ese sistema.

Un *hash* no es más que una cadena alfanumérica de longitud fija, la cual se genera en base a una *función hash*. La esencia de esas funciones radica en que generar un hash es sencillo y poco costoso, pero obtener la cadena original (es decir, la contraseña) mediante el hash es extremadamente complicado, aunque dicha complejidad depende del algoritmo usado. Ejemplos de dichos algoritmos son SHA-1 o MD5.

6.5.1.1. Fuerza bruta

Los ataques de contraseñas se pueden elaborar de varias maneras. El método más básico es el de fuerza bruta, que como su nombre indica, consiste en probar de manera secuencial todas las combinaciones posibles de contraseñas. Esto puede ser especialmente costoso a nivel computacional. Para mostrar esto, en las Tablas 6.1 y 6.2 se pueden observar la cantidad de contraseñas posibles para diferentes casos, además del tiempo máximo requerido para romper una contraseña de cada uno de los casos.

Contraseña	Posibilidades	
Numérica de 4 caracteres	10^4	10000
Numérica de 6 caracteres	10^6	1000000
Numérica de 8 caracteres	10^8	100000000
Numérica de 4 a 8 caracteres	$\sum_{n=4}^8 10^n$	111110000
Alfanumérica de 4 caracteres	27^4	531441
Alfanumérica de 6 caracteres	27^6	387420489
Alfanumérica de 8 caracteres	27^8	282429536481
Alfanumérica entre 4 y 8 caracteres	$\sum_{n=4}^8 27^n$	293292190521
Alfanumérica entre 8 y 16 caracteres	$\sum_{n=8}^{16} 27^n$	82834383195202897568241

Tabla 6.1.: Número de diferentes combinaciones de contraseñas posibles para diferentes conjuntos

Contraseña	Tiempo CPU ⁵	Tiempo GPU ⁶
Numérica de 4 caracteres	10 segundos	1,43 segundos
Numérica de 6 caracteres	16,66 minutos	2.381 minutos
Numérica de 8 caracteres	27,78 horas	3.97 horas
Numérica de 4 a 8 caracteres	30,86 horas	4,40 horas
Alfanumérica de 4 caracteres	8,85 minutos	75,92 segundos
Alfanumérica de 6 caracteres	4,48 días	15.37 horas
Alfanumérica de 8 caracteres	107,46 meses	15,36 meses
Alfanumérica entre 4 y 8 caracteres	803540 años	114791 años
Alfanumérica entre 8 y 16 caracteres	2.62 billones de años ⁷	375237,29 millones de años

Tabla 6.2.: Coste computacional de diferentes ataques de fuerza bruta para diferentes conjuntos

Se puede observar que, al menos mediante fuerza bruta y con un único ordenador estándar los ataques de fuerza bruta son prácticamente inútiles cuando la contraseña resulta ser mínimamente larga o compleja. Incluso teniendo en cuenta que, mediante supercomputadores y optimizaciones, se pueden reducir tiempos, siguen siendo soluciones poco eficientes.

6.5.1.2. Por diccionario

Como mejora con respecto a los ataques de fuerza bruta, existen los denominados ataques de diccionario. En seguridad informática se entiende como diccionario una lista de palabras

⁵Teniendo en cuenta que se pueden probar una media de 1000 contraseñas por segundo mediante un núcleo de una CPU potente [39]

⁶Teniendo en cuenta que se pueden probar una media de 7000 contraseñas por segundo mediante una GPU Nvidia GTX 1080 [40]

⁷1 Billón de años = 1.000.000 de millones de años

cualquiera, tengan sentido o no. Los ataques por diccionario simplemente son ataques de fuerza bruta pero probando solo las opciones de un diccionario. Estos diccionarios se pueden obtener de diversas fuentes, o se pueden generar en base a palabras que guarden relación con el objetivo a atacar. De esta manera solo se prueban una serie combinaciones de caracteres que tienen una mayor posibilidad de aparecer en la contraseña, reduciendo drásticamente el tiempo que dura el ataque.

6.5.2. Exploits

El concepto de *exploit* es sumamente sencillo. Un exploit no es más que una pequeña aplicación escrita con el objetivo de aprovecharse de una vulnerabilidad conocida en un software. La palabra proviene del inglés, que significa literalmente aprovechar o explotar. Los exploits son la parte más importante a la hora de explotar vulnerabilidades.

Ligado íntimamente al concepto de exploit existe el concepto de *payload*. Un payload consiste en una parte de código que el exploit se encarga de ejecutar en la máquina que tiene como víctima con el objetivo de realizar algún tipo de acción maliciosa. Acciones como implementar una shell, añadir un usuario al sistema, borrar ciertos archivos o prácticamente lo que al atacante se le pueda ocurrir.

Según como sea el payload se pueden clasificar en tres tipos:

1. Los *singles*: simplemente son código que se encarga de ejecutar una tarea concreta.
2. Los *stagers*: son payloads encargados de crear la conexión entre el atacante y la víctima. Normalmente este tipo de payloads preparan el terreno para los payloads de tipo *staged*.
3. Los *staged*: son payloads con funcionalidades más complejas, que son introducidos por los stagers en las máquinas a atacar.

Sin querer profundizar demasiado en un campo al que bien se le podría dedicar libros enteros, cabe mencionar que existen una gran cantidad de exploits ya programados. Es más, existen grandes bases de datos de exploits listos para ser usados por parte de hackers y técnicos en ciberseguridad. Además, también existen gran cantidad herramientas que permiten descargar, actualizar, gestionar, e inyectar dichos exploits.

6.5.3. Ataques a redes

Los ataques a redes son un tipo de ataques que tienen como objetivo penetrar en una red informática para poder obtener información de su tráfico interno. Una red informática puede tener diferentes topologías y dependiendo de ellas las características del ataque cambiarán drásticamente. El escenario varía de atacar una red PAN (Personal Area Network) a atacar una red LAN (Local Area Network) o WAN (Wide Area Network). También dependiendo de si la red es inalámbrica o no el ataque se podrá elaborar de una u otra manera.

Los ataques se suelen enfocar a las diferentes capas del modelo de red y, en concreto,

a los protocolos específicos que lo componen. El modelo de red extendido actualmente es el modelo **TCP/IP**, que toma su nombre de los dos protocolos más importantes de éste. Los diferentes protocolos usados en cada capa del modelo TCP/IP se pueden observar en la Figura 6.2.

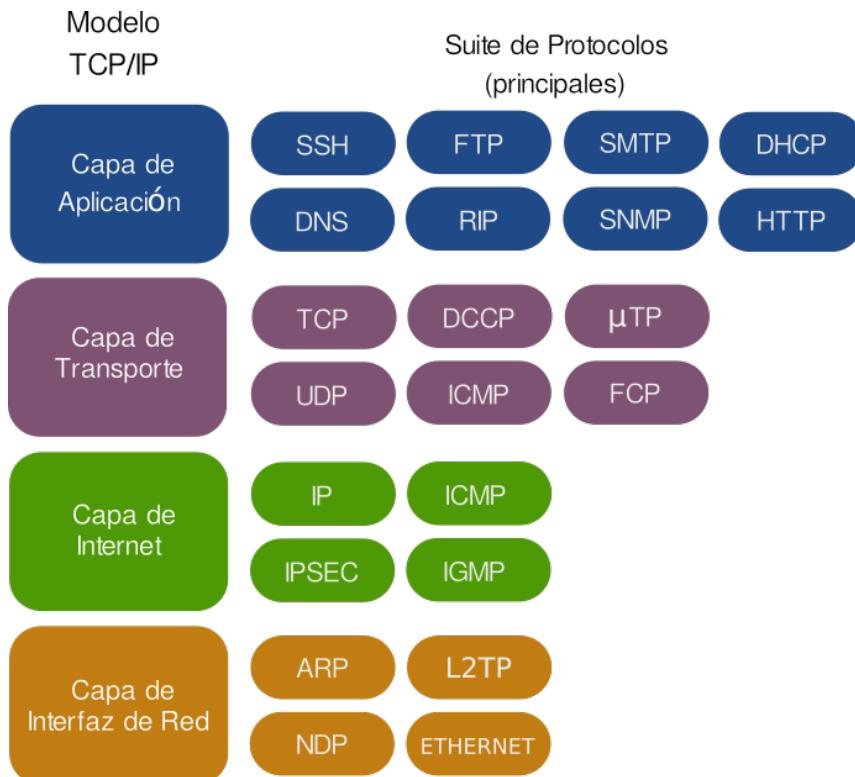


Figura 6.2.: Principales protocolos usados en las cuatro capas de TCP/IP

Existen, independientemente del modelo, diferentes términos que destacan en los ataques a redes, como son el *sniffing*, el *spoofing* o el *hijacking*.

6.5.3.1. Sniffing

El *sniffing* es una técnica que consiste en capturar los paquetes o tramas que pasan por cierta interfaz [41]. Por defecto un sistema rechaza el tráfico que no está dirigido a él. En cambio, si un sistema actúa en modo promiscuo, aceptará todos los paquetes que le lleguen por la red. El sniffing es la técnica que obtiene todos esos paquetes. Para hacerlo, se hacen uso de herramientas llamadas *sniffers*, que permiten capturar estos paquetes y analizar su información capa por capa.

6.5.3.2. Spoofing

Spoofing es un concepto que se basa en suplantar la identidad [41]. Existen numerosas técnicas de suplantación como por ejemplo *MAC Spoofing*, donde se suplanta la dirección física de un dispositivo. También existen otras como *ARP Spoofing*, *DNS Spoofing* o *Web Spoofing*. Consisten básicamente en falsear cierta información para hacernos pasar por otro usuario, otro nodo u otra aplicación en la red.

6.5.3.3. Hijacking

Se conoce como *hijacking* a toda técnica en la que se obtiene el control de un elemento de una red, de tal manera que se pueda modificar la información que pasa por dicho elemento [41]. Existen, al igual que con el spoofing, numerosas técnicas que hacen uso del concepto, como por ejemplo *Session Hijacking* o *Browser Hijacking*. De este tipo de técnicas surgen los ataques MITM (*Man In The Middle*), que como su nombre indica, consiste en interferir en la comunicación entre dos nodos tanto para suplantar la identidad del usuario como para recibir información que se supone que solo debería recibir el usuario.

7

Conclusiones

A lo largo de los capítulos anteriores se ha ido desarrollando un profundo análisis sobre el campo de la seguridad informática, explicando una gran cantidad de conceptos. Primero se han definido una serie de conceptos básicos sobre seguridad informática, intrínsecos a cualquier especialidad o aplicación, como son los servicios de la seguridad de la información.

Por otra parte, se ha elaborado un estado del arte de las diferentes aplicaciones de la seguridad informática, comenzando desde el surgimiento de la necesidad de proteger la información y de hacer que las transmisiones de datos fuesen seguras. Se hace un énfasis en el malware y en los tipos que existen, mostrando de que manera somos vulnerables a ataques y en qué medida nuestra información está desprotegida.

Además, se desmonta el concepto tan arraigado en las mentes de los usuarios de que la seguridad informática simplemente consiste en proteger ordenadores personales de malware. Por una parte el malware no es la única amenaza existente, las intrusiones a redes o los ataques de ingeniería social son otras formas mediante las que se puede acceder a nuestra información. También hay que tener en cuenta que no solo los ordenadores personales son el único target, los avances en diferentes campos, destacando los smartphones, el Internet of Things y el Cloud Computing, hace que estos sean actualmente vértices de enfoque importantes dentro de la seguridad informática.

Tras ese análisis sobre el mundo de la seguridad informática, conocer los diferentes sistemas y sus medidas de seguridad, mencionar diferentes tecnologías y enumerar y describir diferentes tipos de vulnerabilidades se puede obtener una idea clara sobre la seguridad informática. Aun así, esta idea no estará completa si no se conoce cómo actuar ante estas amenazas, como un hacker actúa para conseguir información de un sistema. Por ello se ha hecho especial hincapié en el área del pentesting.

El pentesting es una técnica que, mediante la imitación del comportamiento de un atacante, permite detectar y explotar vulnerabilidades. Dista de un atacante malicioso, o *Black Hat Hacker*, en que el pentester usa el hacking ético y una serie de procedimientos definidos y controlados con el objetivo de detectar vulnerabilidades para, posteriormente, poder

corregirlas, haciendo los sistemas más seguros. Esta técnica, que requiere un profundo conocimiento de los vectores de ataque y vulnerabilidades, además de sólidas nociones sobre redes y sistemas, permite comprender mejor ya no solo como defenderse sino también como atacar, dando una visión más global del conjunto.

7.1. La seguridad informática y los usuarios

Ante esto se plantea un problema serio con respecto a la seguridad de la información. El estado del arte desarrollado contiene un elaborado informe que alberga una considerable cantidad de nociones sobre seguridad informática, pero no es algo que un neófito en la materia pueda comprender. La seguridad informática es un campo técnico que requiere de una gran cantidad de conocimientos para poder aplicar sus medidas. Por lo tanto, ¿la seguridad de la información de los usuarios solo puede mantenerse mediante el trabajo de los profesionales de la seguridad? ¿Los usuarios no pueden, de manera proactiva, controlar sus sistemas haciendo que estos sean más seguros?

Aquí es donde entra en juego la programación. ¿Cómo? Elaborando software que pueda usar cualquier usuario, implementando funcionalidad que permita hacer lo que un hacker realiza en su trabajo, pero de una manera clara y sencilla para el usuario, alejado de aspectos técnicos. Precisamente en ello radica la clave para hacer que los usuarios sean un elemento activo en pro de la seguridad de la información.

Pongamos un ejemplo práctico. Si uno se pone en la piel de un pentester, y quiere hacer un ataque de penetración en una red concreta, buscará vulnerabilidades para poder acceder. También puede querer encontrar maneras para defenderse o detectar elementos en una red. El pentester, un hacker con profundos conocimientos del área, sabe que una de las mejores herramientas de escaneo de redes es Nmap¹. Mediante un par de comandos en su terminal obtendrá la información que necesita. De esa manera tan simple podrá proseguir con su trabajo.

Ahora pongámonos en la piel de un usuario común que simplemente quiere evitar intrusos en su red local. Dicha tarea, que sería trivial para un hacker o alguien con conocimientos de redes, resulta bastante complicada para un usuario normal. Dicho usuario no conoce ni Nmap ni otras herramientas, no sabe ni siquiera acceder a su router para configurarlo, mucho menos tiene conocimientos sobre redes o sistemas. Entonces, ¿cómo podría saber si tiene intrusos en su red? ¿La única opción que tendría sería recurrir a alguien con conocimientos sobre el tema?

¹<https://nmap.org/>

7.2. Herramientas de seguridad informática para usuarios

En ese tipo de casos es donde entra a colación el software, y como este puede ayudar a los usuarios a llevar a cabo este tipo de tareas. Si queremos ofrecer una solución para el usuario, en base a lo aprendido en puntos anteriores podemos obtener varias conclusiones.

Lo primero es que, debido al auge de los smartphones, nuestro software debería programarse para uno de estos sistemas. Si debe ir para uno de estos sistemas, y teniendo que elegir uno, lo más lógico sería elegir Android, por las cuotas de mercado mencionadas anteriormente.

Por otra parte, nuestra aplicación se va a basar en la recogida de información, proceso al que en un pentesting llamábamos *Information Gathering*, por lo que conocer como un pentester recoge la información es fundamental para desarrollar dicha aplicación. Después se puede entrar en que herramientas concretas se pueden usar. Nmap parece a primera vista la opción más viable dentro de ese campo. También hay que tener en cuenta herramientas de desarrollo concretas o herramientas secundarias que también puedan ser útiles.

En la siguiente parte de esta memoria se desarrollará una aplicación con lo mencionado en los párrafos anteriores (y algunas herramientas más). Se explicarán en detalles las tecnologías y herramientas usadas junto a todo el proceso de desarrollo de la aplicación, tanto de programación como de diseño de la interfaz o de la experiencia de usuario, teniendo como objetivo final implementar una prueba de concepto que permita vislumbrar cómo llevar herramientas de seguridad informática a usuarios comunes.

III

Fase 2: Desarrollo de la aplicación

8

Introducción

"La mejor forma de predecir el futuro es implementarlo"

— David Heinemeier Hansson

Como se ha comentado a lo largo de este informe, y tras el posterior estudio sobre el área de la seguridad informática, el objetivo de este TFG radica en desarrollar una aplicación con diversas herramientas de escaneo de redes y recogida de información, que puedan ser útiles para auditores y expertos en seguridad informática. A su vez, se tiene como objetivo acercar este tipo de herramientas a un público más general, usando una interfaz gráfica sencilla, intuitiva y agradable junto con diferentes elementos que mejoren la experiencia de usuario.

Durante esta parte del informe se documentará todo el proceso de desarrollo de la aplicación, empezando por enumerar las herramientas que se usarán durante dicho proceso.

También se hará hincapié en la GUI (Graphical User interface) generada, de la misma manera que en principios de UX (User eXperience), elementos que quedarán reflejados en este informe. Tras ello se documentará el proceso de implementación, donde se hará énfasis en como se desarrollan los diferentes elementos que, a nivel técnico, requieren especial atención.

Para el final de esta parte se obtendrá una aplicación funcional que satisfaga los Requisitos Funcionales establecidos al inicio.

9

Tecnologías y herramientas

"Java es lo más penoso que le ha ocurrido a la informática desde MS-DOS"

— Alan Kay

Durante el próximo capítulo se hablará de las herramientas que se usarán para diseñar, desarrollar, documentar y gestionar el desarrollo de la aplicación mencionada. Desde lenguajes de programación, frameworks, SDKs e IDEs hasta herramientas de control de versiones y herramientas secundarias que, si bien no sirven directamente para desarrollar software, sirven como apoyo para otras partes del proceso de desarrollo de la aplicación.

9.1. Kali Linux

Kali Linux¹ es un sistema operativo basado en Debian que tiene como propósito ofrecer una gran cantidad de herramientas relacionadas con el mundo de la seguridad informática. Es sucesor directo de la conocida distribución BackTrack Linux. Abarca todo tipo de áreas como el *Data Gathering*, *Spoofing* de diferentes tipos, herramientas forenses o ataques a redes inalámbricas, por mencionar solo algunos ejemplos. Es una de las distribuciones más usadas por pentesters y expertos en seguridad informática, en la que se recogen cientos de herramientas concretas, algunas de las cuales son consideradas herramientas de trabajo fundamentales para cualquier hacker, como por ejemplo *Metasploit Framework*, *Johnny The Ripper* o *Nmap*. Esta última será la herramienta que usaremos para dotar a nuestra aplicación de capacidad para obtener información.

Se usará Kali para realizar diversas pruebas con Nmap, previo paso a la integración en la aplicación, para posteriormente poder programar la aplicación que extraerá información de dicha herramienta.

¹<https://www.kali.org/>

9.2. Nmap

Nmap² es un software de escaneo de redes muy usado en auditorías de seguridad. Permite obtener una gran cantidad de información de diferentes nodos o redes, como los servicios que ofrecen, información sobre el sistema operativo o el CMS y sus versiones, qué tipo de filtrado de paquetes realizan, etc.

Nmap es la herramienta de su categoría más usada por varias razones. La primera es que es software libre, con todas las ventajas que eso conlleva: es gratuito, está bien documentado y tiene una comunidad detrás que le da soporte y va mejorándolo y añadiéndole funcionalidades constantemente. Lo segundo es que posee una gran cantidad de herramientas adicionales, teniendo hasta una GUI (*Zenmap*) y un lenguaje propio (*Nmap Scripting Engine, NSE*) que permite crear scripts específicos que Nmap es capaz de ejecutar independientemente del sistema operativo en el que se esté ejecutando.

Se usará Nmap como base para realizar las diferentes operaciones de escaneo de redes. Más adelante se explicará el proceso de integración en la aplicación y cómo se extrae la información que Nmap nos proporciona para usarla en nuestra aplicación.

9.3. Android

Android es un sistema operativo creado en 2005 y que actualmente posee Google. Aun así, el sistema operativo mantiene lo que se denomina el AOSP (*Android Open Source Project*), el cual garantiza que Android siga siendo software libre.

Esto hace que desarrollar para esta plataforma sea completamente gratuito. Si a eso se añade la extensa documentación que existe para desarrollar para esta plataforma, hace que sea una plataforma hacia la cual cada vez más desarrolladores se lanzan a, valga la redundancia, desarrollar aplicaciones en ella.

La elección de este sistema operativo es el paso más lógico teniendo en cuenta que, como se ha mencionado anteriormente, es el sistema operativo con mayor numero de usuarios y además es el que mayores facilidades ofrece para desarrollar.

9.3.1. Android SDK

Se puede desarrollar para Android mediante diversas herramientas, pero la más común de todas es hacer directamente uso del Android SDK (*Software Development Kit*). El SDK nos proporciona todas las herramientas que necesitamos para desarrollar, compilar y depurar aplicaciones para Android.

Existen alternativas que nos abstraen de las capas más bajas de la programación, como puede ser el uso de frameworks multiplataforma que a su vez hacen uso de tecnologías web,

²<https://nmap.org/>

como *Phonegap* o *Ionic*. Este tipo de frameworks convierten en triviales una gran cantidad de tareas que requerirían un mayor tiempo de programación, además de que, mediante el uso de tecnologías web como HTML, CSS y JavaScript, permiten un desarrollo multiplataforma y elaborar interfaces gráficas en una menor cantidad de tiempo.

Por otra parte, se podría optar por soluciones para desarrollar en Android a más bajo nivel, como es el Android NDK (*Native Development Kit*), que permite compilar código C++ para Android, con el aumento de rendimiento que eso conlleva al evitar que nuestro código pase por la MV Dalvik, o permitiendo el uso de librerías complejas que tengamos previamente desarrolladas en dicho lenguaje.

No se va a optar por ninguna de las soluciones mencionadas en los dos párrafos anteriores por varios motivos. Por una parte, si bien es cierto que dichos frameworks simplifican, *a priori*, el desarrollo de ciertas aplicaciones, estas aplicaciones suelen encontrarse limitadas en funcionalidad y, para lo que se pretende desarrollar en este proyecto concreto, no servirían debido a la necesidad del uso de librerías externas complejas.

Por otra parte, se evita el uso del NDK ya que, al no tener la necesidad de aplicar cambios en las librerías externas y además dichas librerías, que usaremos compiladas, ya están optimizadas para obtener el mejor rendimiento posible, carece de sentido el uso del NDK, con lo que se simplifica el proceso de desarrollo.

9.3.2. Kotlin

Una vez optado por programar directamente con el SDK de Android y habiendo dejado a un lado opciones como el uso del NDK o diversos frameworks que proveen de abstracciones, el siguiente paso lógico consiste en la elección de un lenguaje de programación para programar la aplicación. La elección de dicho lenguaje obedece a varios factores. En primer lugar, no se puede desarrollar cualquier tipo de aplicación con cualquier lenguaje. Por ejemplo, a nadie se le ocurriría usar C++ para desarrollar el frontend de una aplicación web, de la misma manera que pocos se atreverían a usar Lisp para el desarrollo de videojuegos. Por otro lado, las características específicas de un lenguaje u otro son determinantes a la hora de elegir entre ellos. A eso se le añade que resulta mala práctica usar un criterio personal a la hora de elegir un lenguaje específico.

Teniendo eso en cuenta, la primera opción para desarrollar aplicaciones con el SDK de Android sería Java, ya que es el lenguaje en el que está programado dicho kit. Por otra parte Java es un lenguaje con un largo recorrido y actualmente, según el índice TIOBE, es el lenguaje de programación más usado [42], ocupando el primer puesto en su ranking. Tiene una gran comunidad a su alrededor y gran parte de el software que usamos actualmente está desarrollado con él.

Sin embargo, se ha optado por elegir Kotlin como lenguaje de programación. Kotlin³ es un lenguaje de programación, (publicado bajo la licencia Apache) que se ejecuta en la

³<https://kotlinlang.org/>

JVM y cuyo desarrollo comenzó y es supervisado por JetBrains, compañía que desarrolla IDEs y plugins para desarrollar aplicaciones. Tiene como principal objetivo convertirse en un sustituto viable de Java, corrigiendo los defectos que este último tiene y aportando nuevos elementos a su sintaxis que permitan facilitar el desarrollo y obtener un código más limpio y claro.

Se ha optado por el uso de Kotlin por varios motivos. El primero es que aporta numerosas ventajas con respecto a Java que, como se ha mencionado, permiten elaborar un código más limpio y fácil de entender. Por otra parte es completamente interoperable con Java, por lo que se complementa a la perfección con partes de código escritas en Java. A todo esto se le añade que recientemente Android ha adoptado a Kotlin como lenguaje oficial [43], por lo que su uso no dará problemas en Android. Eso además hará que se encuentre documentación oficial sobre el uso de Kotlin en dicho sistema, lo que simplificará el desarrollo de la aplicación.

9.3.2.1. Ventajas de Kotlin

A nivel de programación Kotlin representa una mejora notable con respecto a Java, tanto a nivel de código, añadiendo nueva sintaxis, como la funcionalidad adicional que aporta la librería estándar del propio lenguaje. Aunque la lista de mejoras, añadidos y aspectos que se han decidido eliminar con respecto a Java es excesivamente larga como para mostrarla completamente, a continuación se enseñan algunos aspectos del lenguaje que permiten vislumbrar las ventajas que ofrece frente a Java [44].

Control de referencias nulas Kotlin corrige uno de los principales problemas del desarrollo de aplicaciones en Java, que son los *Null Pointer Exceptions*. Kotlin es mucho más seguro a ese nivel ya que, por defecto en Kotlin, una variable no puede ser nula, y si queremos que esa variable tenga la posibilidad de ser nula debemos usar un operador específico (el operador `? .`).

```

1  var a: String = "abc"
2  a = null // Error de compilacion
3
4  var b: String? = "abc"
5  b = null // Ok

```

De la misma manera, se puede especificar en la declaración de una función si esta puede devolver o no un valor nulo. Kotlin no permite asignar un valor que independientemente del tipo que sea, pueda ser nulo a una variable que no pueda serlo.

```

1  fun foo(node: Node): String? {
2      if(node.canString()) {
3          return node.toString()
4      }

```

```

5     else {
6         return null // Esto funciona
7     }
8 }
9
10 fun foo2(node: Node): String {
11     return null // Error de compilacion
12 }
```

De ahí que, si queremos usar una función que pueda devolver un valor nulo o obtener un atributo de una instancia que pueda ser nulo tendremos tres opciones:

1. Comprobar si es nulo antes de usar el operador: Kotlin detecta automáticamente si se ha realizado una comprobación de este tipo, y una vez hecha nos permitirá realizar este tipo de operaciones.

```

1 var b: String? = "abc"
2 if (var != null) {
3     var c: String = b // Esto si se permite, ya que ahora sabemos
4         ↪ que no es null
5 }
6 var d: String = b // Esto da error, puede ser null
```

2. Usar el operador seguro `?.`: en este caso, si el valor existe o la función no devuelve un valor nulo todo funcionará correctamente, si no devolverá un valor nulo.

```

1 var b: String? = "abc"
2 var len: Int? = b?.length // Si b no es null devuelve la longitud
3     ↪ (3)
4 var len2: Int = b?.length // Esto da error, puede ser null
```

3. Usar el operador `!!.`: Este operador es mucho más peligroso. En caso de tratar con valores nulos genera directamente una *Null Pointer Exception*. La ventaja es que permite que las variables sean siempre no nulas.

```

1 var b: String? = "abc"
2 var len: Int? = b!!.length // Si b no es null devuelve la longitud,
3     ↪ si no salta un NPE
4 var len2: Int = b!!.length // Esto NO da error, porque en caso de
5     ↪ ser null saltará un NPE
```

Atributos de una clase Kotlin dispone de mecanismos para generar automáticamente funciones *getter* y *setter*, de tal manera que se ahorra ese engorroso código que se genera

en muchas clases de aplicaciones programadas en Java. Por otra parte se puede modificar estas funciones de manera muy sencilla para que cumplan diversos objetivos, usando menos código que en Java.

```

1  class Lista {
2
3      // ...
4
5      var elementos
6      get() {
7          return elementos.filterNotNull()
8      }
9      set(value) {
10         if(value != null)
11             elementos.add(value)
12     }
13     val isEmpty: Boolean
14     get() = elementos.size == 0
15
16     // ...
17
18 }
```

Singletons Se puede hacer uso del patrón Singleton usando la palabra `object` en vez de `class` a la hora de crear la clase en cuestión, de tal manera que Kotlin se encarga de que solo haya una instancia de esa clase.

```

1  class PDF {
2      // ...
3  }
4
5  object PDFManager {
6      // ...
7  }
8
9  var pdf1 = PDF() // Ok
10 PDFManager.AddPDF(pdf1) // Se accede a sus funciones y variables como si
11    ↳ fuera una variable normal
12 var manager = PDFManager() // Error de compilacion. No se pueden crear
13    ↳ instancias de un singleton, porque Kotlin ya crea la unica instancia
14    ↳ posible
```

Data Classes Kotlin tiene un tipo de clases especial para cuando se desea una clase que solo almacene información y carezca de funcionalidad, denominado Data Classes, que permite ciertas ventajas a la hora de realizar comparaciones o copiar datos, y reduce sensiblemente la cantidad de código de la clase.

```
1 data class User(val name: String, val age: Int)
2 data class User(val name: String = "", val age: Int = 0)
```

Expresiones de rango Kotlin, al igual que otros lenguajes como Python, permite expresiones de rango, que nos permiten iterar sobre rangos de números de forma muy sencilla: además permite determinar la cantidad de aumento o disminución en cada iteración, como se puede observar en los siguientes ejemplos.

```
1 for (i in 1..4 step 2)
2     print(i) // imprime "13"
3 for (i in 4 downTo 1 step 2)
4     print(i) // imprime "42"
```

Aunque existen aun más, con las ventajas mostradas anteriormente queda claramente reflejada la ventaja de programar aplicaciones en Kotlin con respecto a hacerlo en Java.

9.3.3. Android Studio

Android Studio es un IDE que posibilita desarrollar para Android de manera mucho más cómoda. Permite hacer uso del SDK y, asimismo, nos provee de una serie de herramientas, como herramientas de diseño de interfaces gráficas, herramientas de depuración o dispositivos Android virtuales, además de otras ventajas que suele conllevar el uso de un IDE como el autocompletado de código, la automatización de la compilación y el desarrollo o herramientas de análisis de rendimiento. Se trata de un fork del conocido IDE *IntelliJ IDEA*, de la compañía JetBrains, al que se le han añadido las herramientas necesarias para programar en Android.

Es el IDE que se usará para desarrollar la aplicación porque, por una parte, es el que recomienda Android ya que está especialmente diseñado con el propósito de desarrollar aplicaciones para su plataforma y, por otra parte, tiene soporte para Kotlin.

9.4. Otras herramientas

Una vez mencionadas herramientas que permiten desarrollar la aplicación se pasa a mencionar herramientas que, aunque no influyen directamente en la programación de la aplicación, merecen especial mención porque influyen de manera indirecta y se usan para el

desarrollo del proyecto.

9.4.1. Git

Git⁴ es un Sistema de Control de Versiones o VCS (*Version Control Systems*) distribuido y de código libre desarrollado por Linus Torvalds en 2005 con el objetivo de llevar un control sobre el desarrollo del Kernel Linux. Es a día de hoy uno de los VCS más usados, tiene diferentes implementaciones para una gran cantidad de sistemas y dispone de una amplia documentación.

El uso de este VCS tiene como objetivo poder llevar un control sobre el desarrollo y el avance de nuestro código, pudiendo revertir cambios, crear diferentes ramas en el desarrollo para desarrollar diferentes funcionalidades por separado o actuar como sistema de backup mediante el uso de un repositorio remoto que puede estar alojado en servicios de alojamiento de repositorios como GitHub⁵ o similares.

9.4.2. LATEX

LaTeX⁶ es un sistema de composición de textos que permite crear todo tipo de documentos. Suele ser utilizado para escribir de artículos académicos, tesis y libros técnicos debido a que los documentos generados con él son de alta calidad, llegando al nivel de una editorial profesional, con la ventaja de que es completamente libre.

Se usará LaTeX, junto una serie de librerías específicas, para el desarrollo de este informe por la calidad de los documentos que genera y por ser prácticamente un estándar dentro de la comunidad científica y académica.

⁴<https://git-scm.com/>

⁵<https://github.com/>

⁶<http://www.latex-project.org/>

10

Desarrollo de la aplicación

"Los programas deben ser escritos para que los lean las personas, y sólo incidentalmente, para que lo ejecuten las máquinas"

— Abelson and Sussman

Tras haber explicado en el capítulo anterior las diversas herramientas a usar para el desarrollo de la aplicación, en este capítulo se explicaran aspectos sobre el diseño de la aplicación y las diferentes decisiones tomadas a la hora de realizar dicho diseño.

El diseño de la aplicación es simple y minimista. El objetivo del proyecto consiste en hacer una aplicación lo más fácil de usar y con la mayor claridad posible de cara al usuario. Por ello, se ha optado por un enfoque que reduzca el numero de elementos al mínimo, además de que se adapta perfectamente a los estándares de diseño de Material Design¹, que son los recomendados a la hora de diseñar aplicaciones para Android.

10.1. Interfaz gráfica

Para la interfaz gráfica se ha optado por hacer un diseño inicial en papel para después, en base a ese diseño y esa distribución de ventanas, elaborar en Android Studio toda la interfaz gráfica. Se puede observar un diseño primitivo en la siguiente imagen.

La aplicación esta dividida en varias vistas. Debido a que el objetivo principal de la aplicación es que sea lo mas simple posible, se ha diseñado buscando minimizar el numero de ventanas de la aplicación.

A la hora de crear las vistas en Android se usan dos conceptos. *Activity* y *Fragment*. En Android, una *Activity* representa una parte de la UI de una aplicación, y tiene su propio

¹<https://material.io/design/>

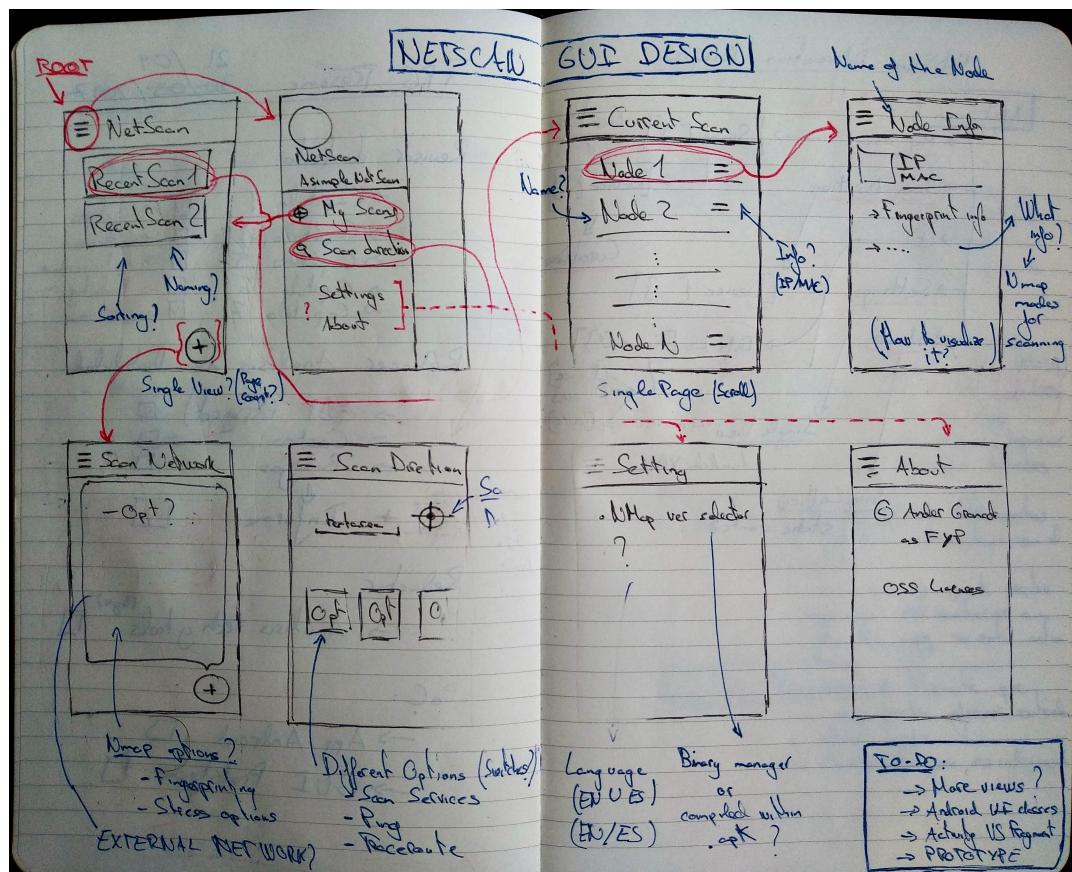


Figura 10.1.: Mockup inicial de la GUI de la aplicación

ciclo de vida y su propia jerarquía de elementos. Una aplicación Android dispone de tantas Activity como el usuario quiera implementar, entendiendo que cada Activity servirá para mostrar cierta información e interactúa con el usuario de una manera concreta. En términos genéricos, equivale a una vista concreta de una aplicación. Los aspectos más técnicos se abordarán durante la explicación de la implementación.

El Fragment, aunque también representa una parte de la UI de la aplicación, depende de una Activity, ya que debe acoplarse a una. A diferencia del Activity, se pueden mostrar varios Fragments a la vez. Hay que tener en cuenta que un Fragment, aun teniendo su propio ciclo de vida (más simplificado que el del Activity) depende del ciclo de vida del Activity al que este acoplado.

En Android, a la hora de diseñar aplicaciones, se usan Fragments para partes reutilizables y comunes de una UI y se usan las Activities para controlar el ciclo de vida de la aplicación. En una aplicación de Android abierta siempre se encuentra activa una Activity, que muestra algún tipo de interfaz.

La aplicación dispone de una serie Activities y Fragments implementados, cuya lista se

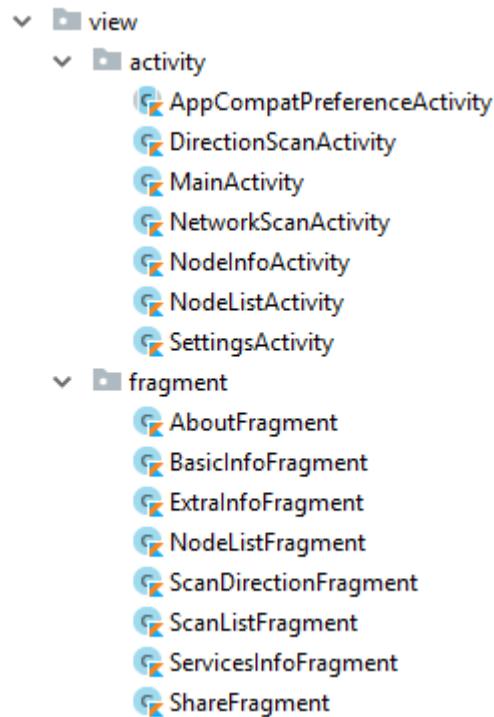


Figura 10.2.: Lista de clases con las Activities y Fragments implementados en la aplicación

muestra en la Figura 10.2.

Cómo se puede observar en la imagen existen una serie de Activities y Fragments que sirven para implementar las diferentes ventanas basándose en ese primer y primitivo diseño de la interfaz. Se observa que también existen más elementos que los correspondientes a las vistas inicialmente diseñadas. A medida que se ha ido desarrollando la aplicación, y en función de las necesidades de la funcionalidad, se han ido añadiendo más elementos. Este diseño de Activities y Fragments corresponde al diseño final de la aplicación.

10.2. Navegación

A la hora de programar una interfaz gráfica, una de las cosas que debemos tener en cuenta es el flujo de interacción entre las diferentes vistas. Esto consiste en entender que vistas interactúan con otras, es decir, el proceso que sigue el usuario para llegar a una u otra ventana concreta. A esto se conoce como *flujo* de la aplicación. El flujo de la aplicación es bastante sencillo como y se muestra en la Figura 10.3

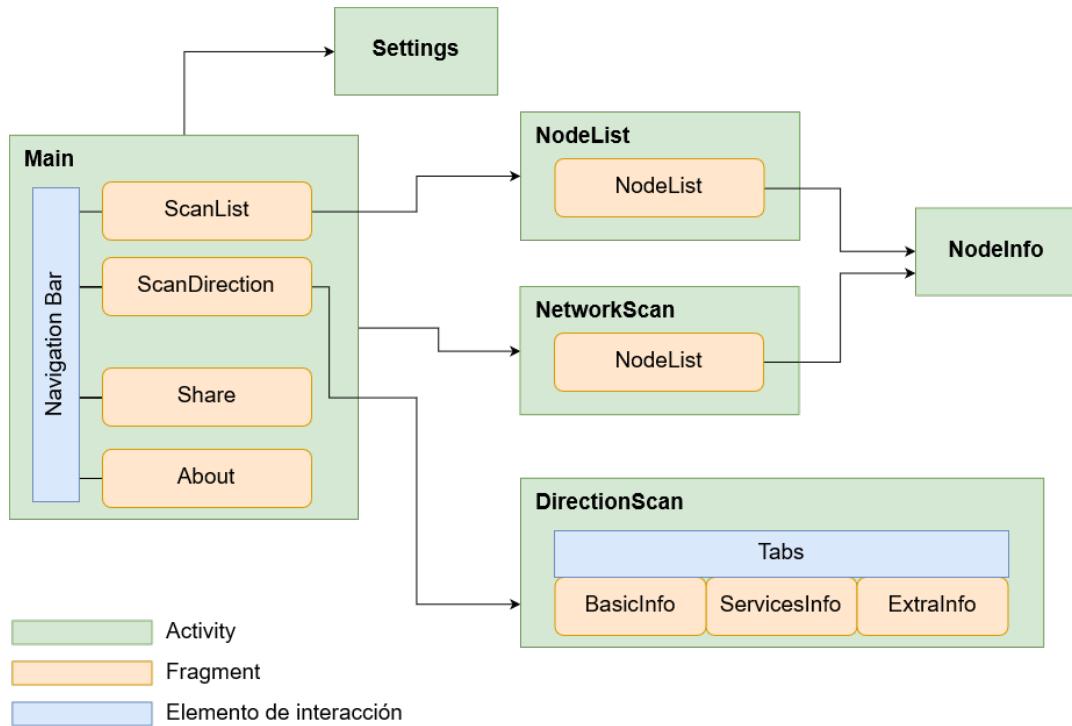


Figura 10.3.: Diagrama de flujo entre los diferentes elementos de la interfaz

El diagrama muestra claramente cómo interactúan los diferentes Activities y Fragments. Como se puede apreciar, la aplicación parte de un punto, de una Activity concreta, llamada **MainActivity**. Esta Activity es la que arrancará al iniciarse la aplicación. Dentro de ella, tenemos cuatro posibles Fragments a mostrar. Cuál se mostrará dependerá de la opción seleccionada en una barra de navegación lateral, o *Navbar*. Por defecto está seleccionada la opción **ScanListFragment**. Accediendo a esa barra de navegación cambiamos entre un menú u otro, implementado en un Fragment u otro. Se puede ver el diseño de la barra de navegación en la Figura 10.4

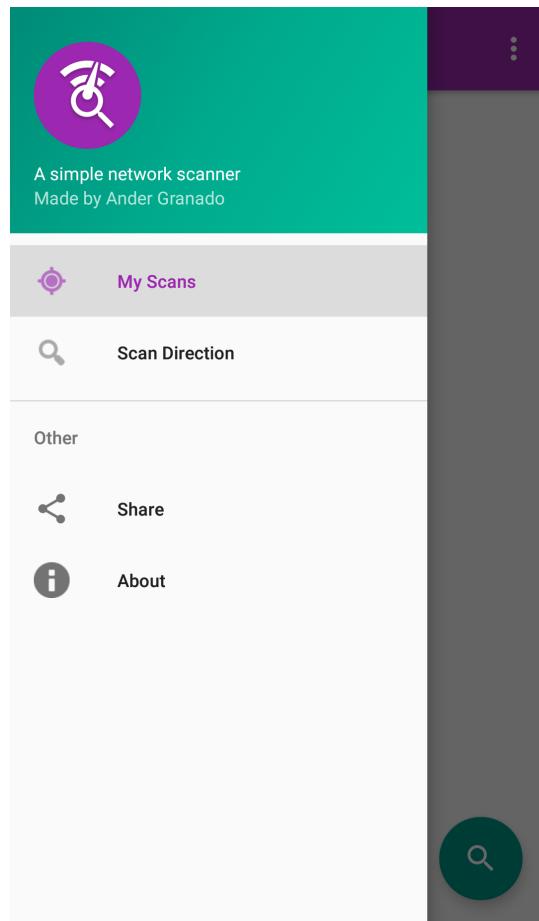


Figura 10.4.: Navbar principal de la aplicación

Se puede elegir entre una de las cuatro opciones y la opción seleccionada se mostrará marcada.

10.2.1. Escaneos de red

Como se ha mencionado, dentro de esas cuatro opciones, la que se muestra por defecto es **ScanListFragment**. Esta opción mostrará los diferentes escaneos realizados en una lista con una interfaz clara y simple, como se aprecia en la Figura 10.5.

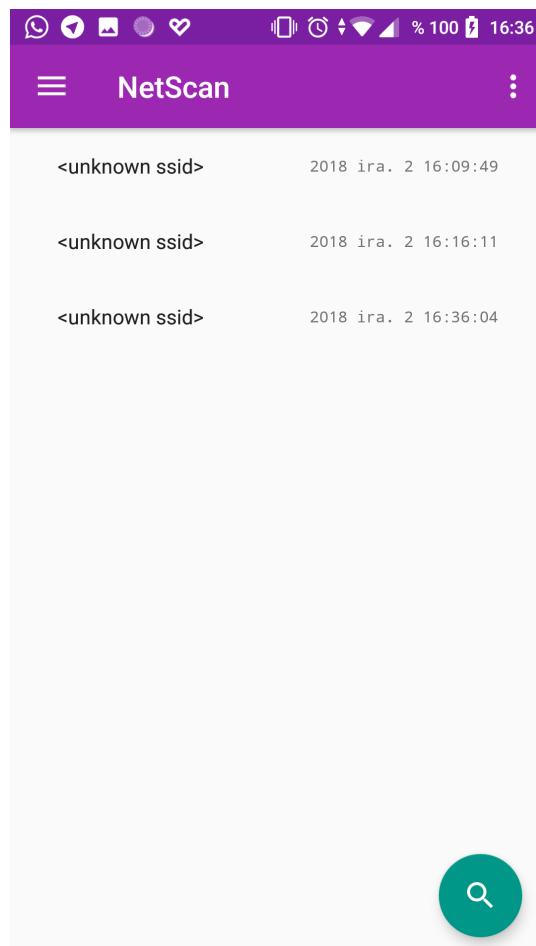


Figura 10.5.: Diseño de ScanListFragment para mostrar los diferentes escaneos realizados

Dentro de esta interfaz también tendremos un elemento básico importante dentro de la aplicación, qué es el botón que encontramos en la parte inferior derecha. Este botón, o mejor dicho, *Floating Action Button*, es un elemento que funciona de la misma manera que lo haría un botón corriente, pero con la particularidad de que se encuentra flotando por encima de la interfaz, lo que quiere decir que su posición es siempre fija, independientemente de si vamos descendiendo por la lista de escaneo realizados.

Pulsando este botón lograremos ejecutar la funcionalidad principal de la aplicación, que consiste en escanear una red. Tal y como muestra el diagrama, para escanear una red se abre otra Activity, llamada **NetworkScanActivity**.

Por lo tanto desde aquí tendremos dos opciones. La primera, pulsar sobre un escaneo ya realizado, lo que nos abrirá un Activity, llamada **NodeListActivity**, que se encarga de mostrarnos los nodos para ese escaneo en concreto. La segunda consiste en pulsar el FAB, lo que iniciará el escaneo de la red y abrirá otra Activity llamada **NetworkScanActivity**.

En el propio diagrama se observa que, aunque cada una de esas dos opciones abren una Activity diferente, ambas van a representar su interfaz gráfica mediante el mismo Fragment, el **NodeListFragment**. Aquí es donde se nota el potencial de los Fragments, que nos permiten reutilizar implementación de la parte gráfica en diferentes puntos de la aplicación, como se muestra en la Figura 10.6.

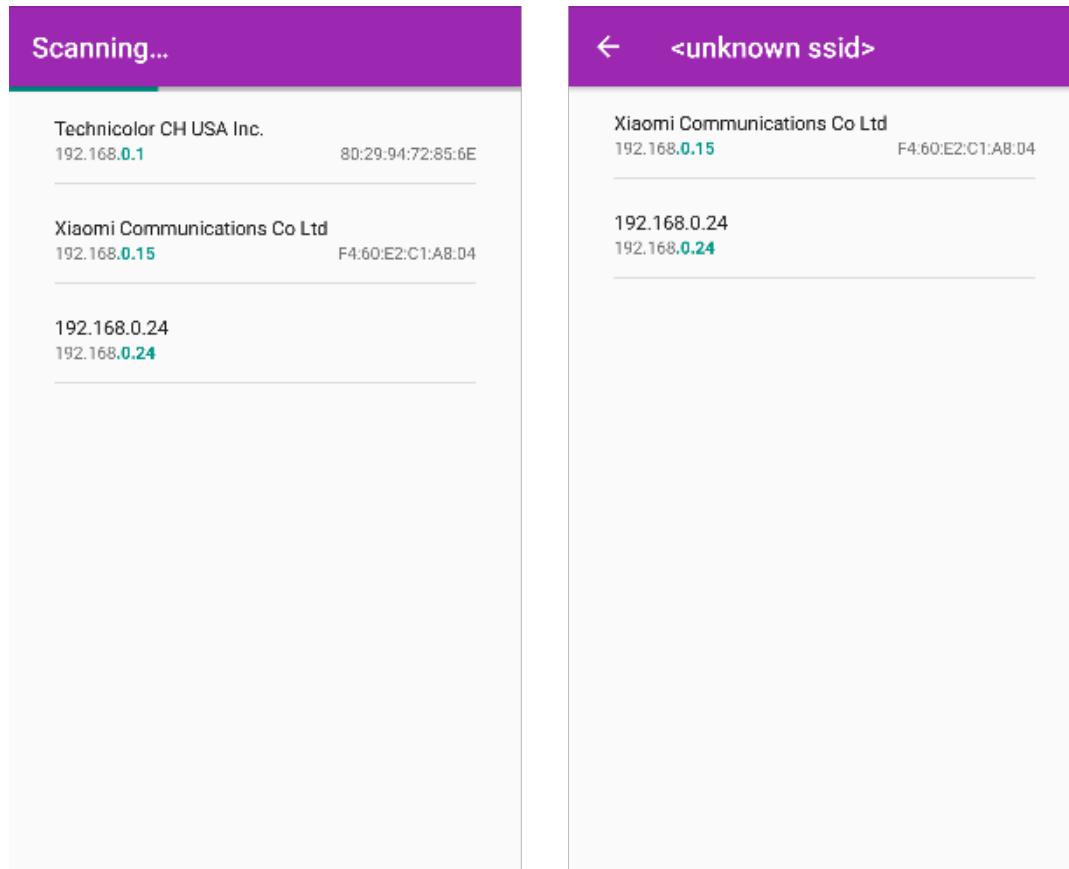


Figura 10.6.: Diferencia en NodeListFragment para un escaneo guardado y uno en curso

Tenemos que tener en cuenta que, en caso de hacer un escaneo de una red, aparte de realizar el escaneo, queremos mostrar los nodos en una lista los nodos que vayamos obteniendo. En caso de querer visualizar un escaneo ya hecho queremos mostrar también una lista de nodos. La funcionalidad para mostrar una lista de nodos se implementa un único Fragment, de tal manera que, aunque las dos operaciones a realizar sean diferentes, comparten la forma de visualizar los datos. Por ello, la parte correspondiente a cada caso particular se implementa dentro de sus respectivos Activities, mientras que la visualización de la lista de nodos se implementa directamente en el Fragment.

Después (en ambas opciones), en caso de seleccionar un nodo concreto, se mostrará información sobre ese nodo, independientemente de que sea un escaneo ya guardado o el escaneo

que acabamos de realizar. Para lograr esto, simplemente se enlaza a otra Activity llamada **NodeInfoActivity**, mostrada a continuación.

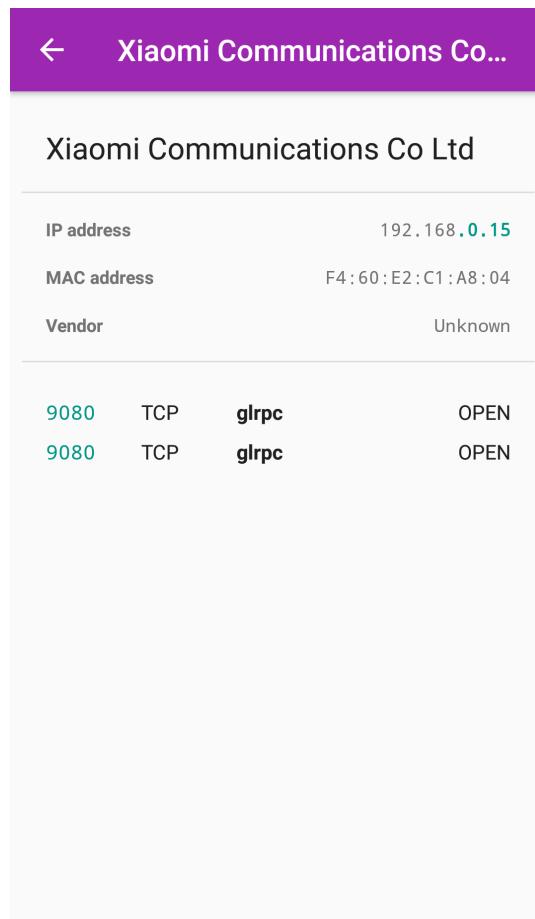


Figura 10.7.: Diseño del NodeInfoActivity

En esta Activity se muestra la información obtenida. Información como la dirección IP, la dirección de hardware, el nombre del host, el fabricante o una tabla que muestra los diferentes servicios que tiene accesibles, con sus respectivos puertos, protocolos e información asociada.

10.2.2. Escaneos de host

De esas opciones, la otra que también resulta de interés es la que podremos usar para analizar una dirección concreta, que se muestra en Figura 10.8 y está implementada en el Fragment **ScanDirectionFragment**.

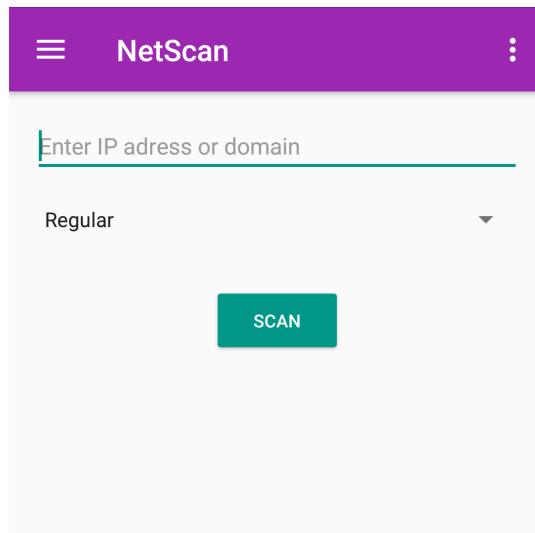


Figura 10.8.: Diseño del ScanDirectionFragment

Una vez accedemos a él, vemos que su interfaz también es muy sencilla. Está compuesta solamente de tres elementos. El primero es un campo de texto donde podremos introducir la dirección IP o el dominio del nodo host que queramos analizar. El segundo nos permite elegir el modo de escaneo. Como se puede observar, tenemos cinco modos diferentes que influyen en la cantidad de información que podremos obtener. Obviamente, cuanto más exhaustivo sea el modo de escaneo, más tiempo va a tardar la aplicación en analizar dicho nodo.

Una vez terminado el escaneo del host se abrirá otra Activity, llamada DirectionScanActivity, que nos muestra la información del host escaneado, como se puede ver en la Figura 10.9 y en la Figura 10.10. En este caso la información se muestra de manera diferente a la información de un nodo de una red. Tenemos otra vista, esta vez separada en diferentes pestañas. Estas tres pestañas contienen partes de la información obtenida.

La primera de ellas, implementada en el **BasicInfoFragment**, simplemente muestra información con la dirección IP y el nombre del dominio. La segunda, llamada **ServicesInfoFragment**, muestra la información de los diferentes servicios abiertos en dicho host. La tercera y última, llamada **ExtraInfoFragment**, muestra información extra, como por ejemplo el tiempo de escaneo o el número de host detectados.

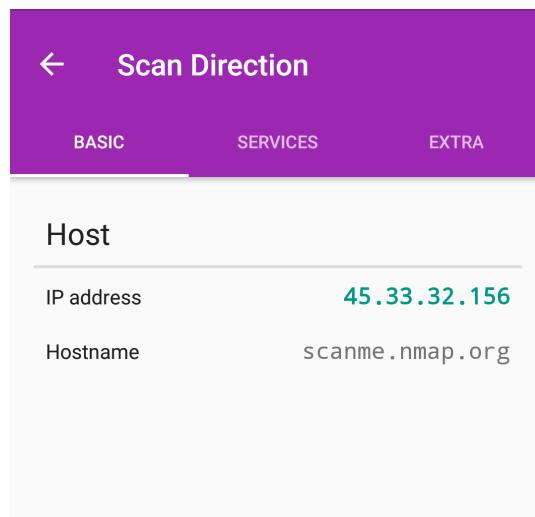


Figura 10.9.: Diseño del DirectionScanActivity y su pestaña con BasicInfoFragment

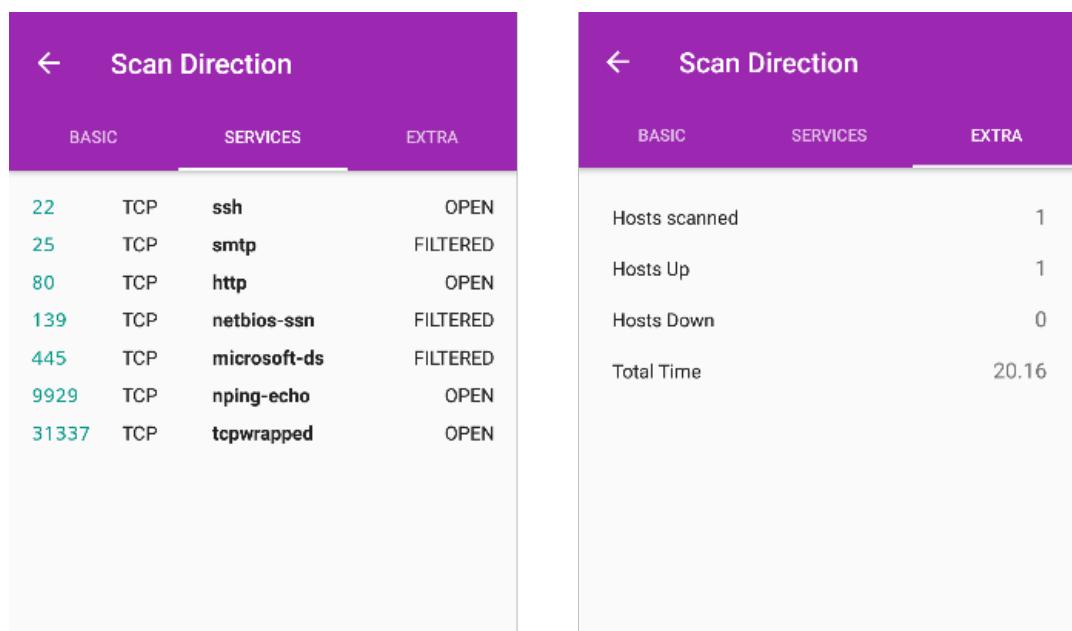


Figura 10.10.: Diseño del resto de pestañas del DirectionScanActivity

Cabe destacar que, por diversos motivos, se ha diferenciado entre la visualización de la información de un nodo de una red y la información de un host concreto que queramos analizar. El primero de ellos es que podemos tener diferentes objetivos a la hora de examinar los nodos que tenemos en nuestra red inalámbrica o examinar la información de un solo dominio. Al haber implementado la interfaz gráfica de manera diferente, podremos en un futuro ampliar la funcionalidad del escaneo del host para que muestre otros aspectos más

complejos, sin con ello afectar a la información que se muestra de un nodo.

10.2.3. Ajustes

Por último se muestra un Activity especial para gestionar los ajustes de la aplicación, llamado **SettingsActivity**. Este se puede acceder desde el menú de tres puntos de la barra superior. Como se ve en la Figura 10.11, se abrirá un menú contextual que permitirá acceder a los ajustes.

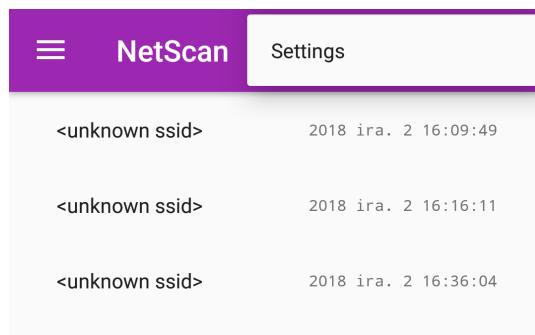


Figura 10.11.: Menú contextual para acceder a los ajustes

Tras abrirlo, se nos abrirá los ajustes, que contendrán varias categorías (Figura 10.12). Dentro de esas categorías, se muestran diversos ajustes (Figura 10.13) que permitirán modificar el comportamiento de la aplicación. Ajustes que cambiarán valores que influirán en ciertas partes del desarrollo

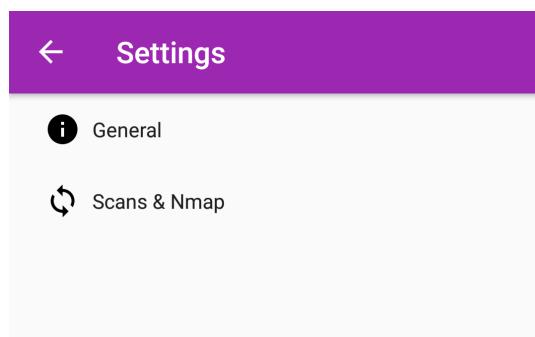


Figura 10.12.: Diferentes categorías dentro de los ajustes

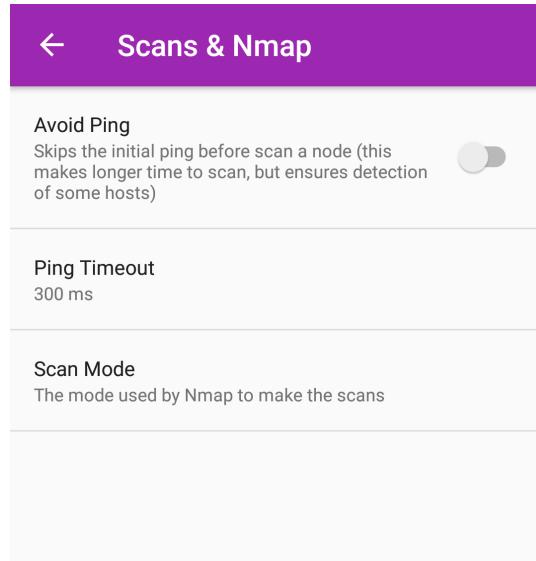


Figura 10.13.: Ajustes de la sección sobre los escaneos y Nmap

10.3. Aspectos gráficos de Material Design

Hasta ahora se han ido analizando individualmente cada Activity y Fragment y mostrando sus relaciones entre ellos. Además, se han ido mostrando imágenes del aspecto de cada uno de ellos. En dichas imágenes se han podido apreciar ciertos aspectos de la interfaz gráfica, aspectos en los que merece la pena indagar para explicar cómo y porqué se han implantado de esa manera.

10.3.1. Color

El primero de ellos es el color: en las imágenes mostradas se ha podido observar que en la interfaz existe, aparte del blanco, dos colores principales. Estos colores no son una elección arbitraria y además guardan relación entre ellos.

La elección de colores dentro de Material Design está acotada una cierta paleta [45], que permite que los colores mostrados sean lo más agradables a la vista posible. A la hora de desarrollar una aplicación se han de elegir como mínimo dos colores. Los que se denominaría como color primario y color secundario, respectivamente. Para estos dos colores no podremos elegir cualquier color, sino una serie de valores concretos dentro de una paleta.

Dentro de esa paleta existe una manera concreta de especificar la claridad u oscuridad de un color en concreto. Se utilizan valores fijos que van desde el 900 hasta el 50, como se muestra para un color arbitrario en la Figura 10.14.

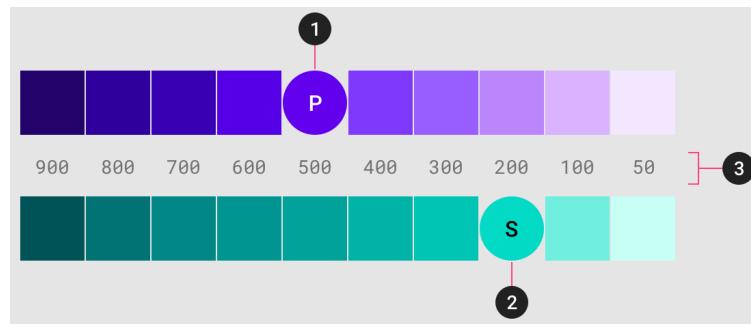


Figura 10.14.: Ejemplo de distinción de colores primario y secundario en Material Design

A medida que el número aumenta, el color se vuelve más oscuro. Cada color concreto de Material Design tiene toda esta gama de rangos disponibles para elegir. Lo que Material Design recomienda es elegir un color principal con una tonalidad 500 y un color secundario con una tonalidad de entre 500 a 200. Además estos dos colores no pueden ser cualquiera, sino que lo recomendable sería que de alguna manera fueran complementarios.

En base a todo lo mencionado se han elegido los colores *Purple* y *Teal* tan característicos de la aplicación (ver Figura 10.15). Además también se ha elegido un color de énfasis para el color principal, que se utiliza en algunos puntos concretos de la aplicación.

Purple 50	#F3E5F5	Teal 50	#EOF2F1
100	#E1BEE7	100	#B2DFDB
200	#CE93D8	200	#80CBC4
300	#BA68C8	300	#4DB6AC
400	#AB47BC	400	#26A69A
500	#9C27B0	500	#009688
600	#8E24AA	600	#00897B
700	#7B1FA2	700	#00796B
800	#6A1B9A	800	#00695C
900	#4A148C	900	#004D40

Figura 10.15.: Colores finales elegidos para la aplicación

Los códigos hexadecimales de los colores están definidos en `colors.xml`, qué es el archivo que se utiliza en Android para definir los colores y tenerlos accesibles desde cualquier

parte de nuestra aplicación.

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#9C27B0</color>
    <color name="colorPrimaryDark">#7B1FA2</color>
    <color name="colorPrimaryLight">#BA68C8</color>
    <color name="colorAccent">#009688</color>
    <color name="colorBackground">#FAFAFA</color>
    <color name="colorUnfocused">#E0E0E0</color>
</resources>

```

Figura 10.16.: Definición de los colores en el fichero colors.xml del proyecto en Android Studio

Por último en lo que al color se refiere, destacar que, en caso de haber usado el fondo claro (en este caso blanco) el texto mostrado se dibuja, o bien en negro, o en tonalidades de gris oscuro. Y al contrario, en caso de haber un color oscuro (en este caso el color principal) los elementos de texto se escriben en blanco. Esto permite que los diferentes textos mostrados en la aplicación sean claramente legibles.

10.3.2. Layout

Otro de los elementos importantes a la hora de diseñar la aplicación ha sido la distribución de elementos en ella, o lo que se conoce en inglés como *layout*. Es muy importante distribuir estos elementos correctamente por varios motivos.

El primero de ellos es la sensación que puede dar al usuario en la interfaz. Si tenemos los elementos agrupados en un espacio muy concreto, con poca separación o márgenes, daremos una sensación de sobrecarga y al usuario le resultará más difícil comprender la aplicación. Si los tenemos desordenados dificultará también su uso. Es necesario colocarlos ordenados y especiados correctamente en pro de ofrecer al usuario la información de la manera más clara posible.

Por otra parte, el uso de márgenes concretos estandarizados y programados de tal manera que se adapten a cualquier tipo de pantalla, hará que nuestra aplicación se vea bien independientemente de la pantalla del dispositivo que la esté ejecutando.

Durante toda la aplicación se han utilizado márgenes concretos y dinámicos. Para especificar en Android distancias, como puede ser por ejemplo la distancia de un margen, se usan una serie de medidas llamadas *dp*. Estas medidas lo que hacen es modificarse en función del tamaño de la interfaz especificado en el dispositivo. Es decir, van a aumentar o a decrecer en función de que se activen o no ciertos parámetros de accesibilidad en el sistema, o se cambien cosas como el tamaño de textos.

Utilizando estas medidas, y no medidas absolutas como pueden ser los píxeles, haremos que independientemente de esas características, activadas o desactivadas en un dispositivo u otro, nuestra aplicación se adapte sin ningún tipo de problema.

Además se han utilizado solamente ciertos valores estándar dentro de las interfaces gráficas en Android. En este caso, los únicos valores que se han utilizado, tanto a la hora de definir márgenes como a la hora de definir el tamaño de un elemento, son distancias en *dp* que sean múltiplos de 8, como pueden ser: 8dp, 16dp, 32dp, 48dp. Esto da una sensación de consistencia y unidad a los márgenes.

Por otra parte en lo que a interfaz adaptativa o diseño responsive se refiere, se ha hecho uso de ciertos tipos de layout concretos que implementan las Activities de Android. A la hora de añadir elementos en un Activity de Android, estos deben ser añadidos a un layout. Un layout está compuesto de uno o más elementos, como pueden ser campos de texto, botones o también otros layouts. De esta manera se construyen interfaces gráficas que tienen diferentes apartados o secciones, dentro de las cuales tienen agrupaciones de elementos diferentes. Según el tipo de layout que se utilice, este ordenará los elementos de una manera u otra.

En concreto, se ha hecho uso del *ConstraintLayout* y del *LinearLayout*.

10.3.2.1. ConstraintLayout

El ConstraintLayout es un tipo de layout que permite definir una jerarquía concreta a la hora de distribuir los elementos. Es decir, cada elemento estará relacionado y tendrá unos márgenes concretos en función de otros elementos. De esta manera podemos, por ejemplo, poner que un elemento esté separado a una distancia concreta por debajo de otro elemento. Por lo tanto si el segundo elemento se mueve debido a un cambio, la interfaz este primero actuará en consecuencia.

Android Studio tiene una herramienta que permite definir de manera sencilla mediante un editor de este tipo de distribución de elemento. A fin de cuentas, el layout de un Activity no es más que un fichero XML que podemos modificar a mano, pero mediante esta herramienta que Android Studio provee, podremos editar dicho archivo XML pero interactuando de manera gráfica con la interfaz.

Para mostrar un ejemplo de como se ha implementado este layout, se va a usar el ejemplo del **NodeInfoActivity**. Los diferentes elementos que componen dicha Activity se muestran en la Figura 10.17.

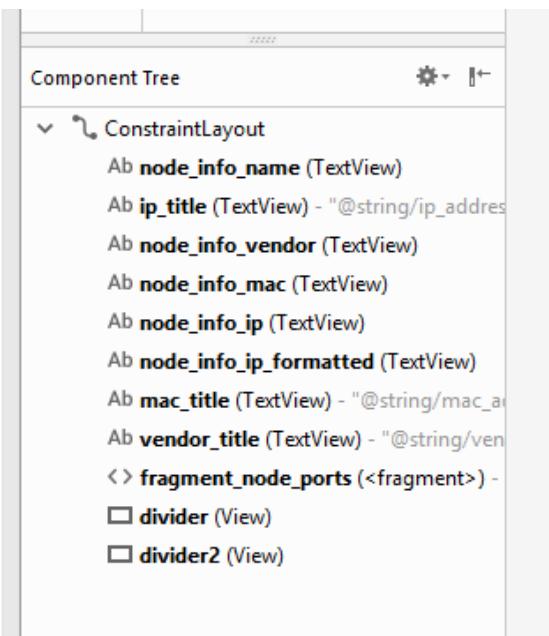


Figura 10.17.: Elementos del ConstraintLayout del NodeInfoActivity

En dicha imagen se muestra una sección del editor de interfaces de Android Studio. En esa sección es donde podremos observar los diferentes elementos y la jerarquía de dichos elementos. Como se muestra, tenemos un layout de tipo ConstraintLayout y, dentro de él, diferentes elementos. Tenemos campos de texto para mostrar diferentes tipos de información, otros campos de texto que simplemente muestran nombres e incluso elementos más complejos, como barras de división o directamente otro Fragment entero, incrustado dentro de la Activity, que se encuentra al mismo nivel que el resto de elementos.

Anteriormente se ha mostrado una imagen sobre cómo se muestra la información de un nodo usando el **NodeInfoActivity**. Mientras que dicha imagen corresponde a la aplicación en ejecución, si entramos en el diseñador de interfaces gráficas de Android Studio, la imagen que obtendremos es diferente, como podemos observar en la Figura 10.18. En este caso tendremos dos vistas diferentes, la vista de diseño y la vista de blueprint.

La vista de diseño, que es la de la izquierda, nos permite hacernos una idea de cómo se van a visualizar los elementos en nuestra aplicación (aunque debemos tener en cuenta que no es del todo exacta, ya que muchos de los elementos se cargarán o modificarán mediante código y, en este caso, lo único que estamos haciendo es mostrar los elementos del XML correspondiente).

La segunda vista, la de blueprint, nos permite trabajar sin distracciones sobre ciertas propiedades de los elementos, como pueden ser los márgenes, los tamaños o qué elementos están enlazados con otros.

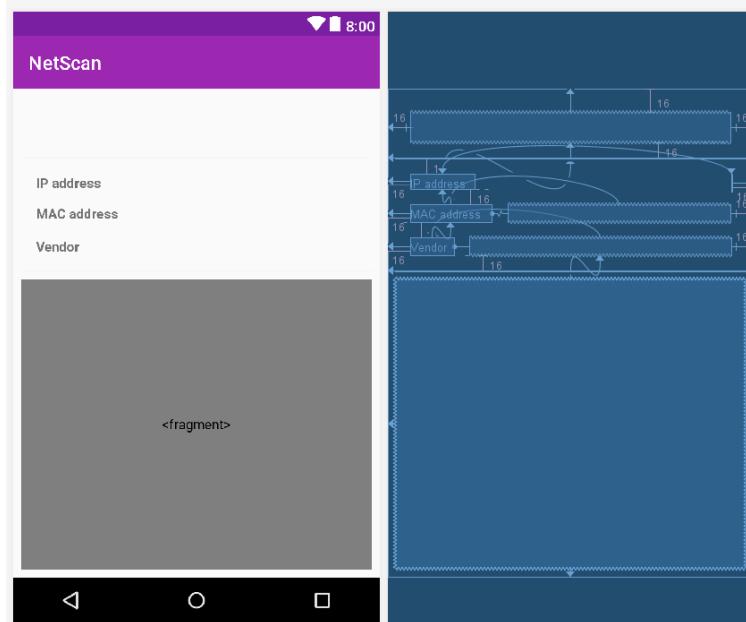


Figura 10.18.: Vistas de diseño y de blueprint del editor de interfaces gráficas de Android Studio

En este caso vamos a hacer énfasis en el modo blueprint de parte de esa interfaz, que nos permitirá examinar el tamaño que ocupan los elementos y la relación entre ellos, sin tener las distracciones de los propios elementos.

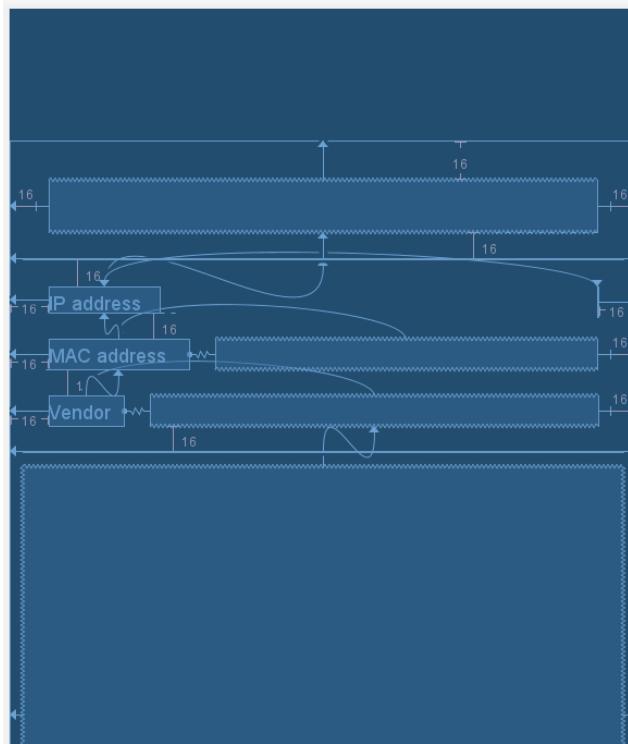


Figura 10.19.: Layout y márgenes en un blueprint del NodeInfoActivity

El blueprint de la Figura 10.19 muestra una serie de flechas que básicamente indican qué elementos están relacionados con que otros. Podemos distinguir un primer elemento que tiene un margen de 16dp con respecto al borde superior. En base a ese elemento se van uniendo el resto de elementos, de tal manera que en caso de ampliarse algún tipo de elemento el resto de elementos se moverán en consonancia, ya que están enlazados unos con otros. A su vez, cada uno de los diferentes elementos tiene márgenes tanto a la izquierda como a la derecha. Estos márgenes, todos de 16dp, permiten que, independientemente del tamaño de la pantalla, tengamos los elementos perfectamente distribuidos.

Mediante el uso de márgenes con medidas concretas y enlazando los elementos con las opciones que nos permite el ConstraintLayout, nos queda una interfaz clara y completamente adaptativa, pudiéndose visualizar perfectamente independientemente de la pantalla.

10.3.2.2. LinearLayout

El segundo layout utilizado en la aplicación es el LinearLayout. Este layout, como bien su nombre indica, nos permitirá distribuir elementos de manera lineal. Esto lo podemos hacer tanto en vertical como en horizontal. En este caso, se ha utilizado en formato vertical y principalmente para mostrar listas de escaneos y de nodos, en los Fragments **ScanListFragment** y **NodeListFragment**.

La gran ventaja del uso de este layout reside en que al añadir elementos al layout se irán posicionando automáticamente en vertical o en horizontal, formando una lista, por lo que podremos añadir elementos de manera dinámica, es decir, mediante código sin preocuparnos por especificar nada con respecto a dónde se van a mostrar.

10.3.3. Tipografía

En lo que a la tipografía se refiere, se ha aplicado principios similares a otros aspectos. La tipografía utilizada es la que se usa por defecto en las aplicaciones Android, denominada *Roboto*. Esta tipografía es clara, legible y se integra la perfección con el sistema Android, ya que es la fuente que este último utiliza.

A la hora de usar diferentes tamaños o establecer diferentes opciones para dicha fuente, solo se han realizado ligeras modificaciones. En los aspectos en los que se quiere destacar algún elemento, como puede ser por ejemplo la dirección IP en alguna de las vistas, se ha hecho uso, por una parte, de la tipografía en negrita y, por otra parte, del uso del color secundario para darle mayor visibilidad.

También se ha hecho uso de la tipografía en formato monoespaciado cuando se ha querido representar valores que siempre van a tener la misma longitud. Es por todos sabido que, por ejemplo, una dirección de hardware siempre tiene la misma longitud, y representarlo con la tipografía monoespaciada permite que siempre tenga el mismo tamaño el bloque de texto a mostrar. Se puede apreciar estos cambios en la Figura 10.20

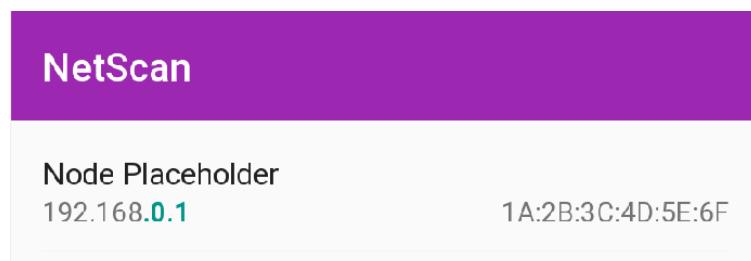


Figura 10.20.: Diferentes opciones aplicadas a la tipografía original

Por último con respecto a la tipografía, los tamaños utilizados siguen unos principios similares a los utilizados para los márgenes. En este caso el sistema que se utiliza de medida no es el utilizado para las distancias, márgenes, que es el *dp*, sino que se trata del *sp*. Este sistema para dimensionar las tipografías que permite es que, si en el propio sistema cambiamos la tipografía para que sea de un tamaño superior e inferior, esta se adapte en función de dicho ajuste.

Durante toda la aplicación se ha mantenido en la mayor parte del texto los tamaños estándar (de 16sp), exceptuando algunas pequeñas ocasiones en las que se ha mostrado elementos con un tamaño mayor, siempre utilizando medidas definidas dentro del estándar de Material Design.

10.3.4. Iconografía

En una aplicación, la iconografía, junto a la tipografía y otros elementos de diseño, definen el estilo de la aplicación. Aunque es un área que abarca muchísimos campos del diseño y puede ser todo lo compleja que se quiera, para esta aplicación en concreto simplemente se han tenido en cuenta dos aspectos básicos.

El primero de ellos son los diferentes iconos usados para mostrar diferentes opciones, como pueden ser los usados en la barra de navegación o en los ajustes. Los iconos utilizados son los iconos estándar que se usan en Android. Esto, junto a los otros elementos, da una sensación de unidad mayor, ya que se asemejan estilo a los iconos del sistema de Android.

Por otro lado, está el ícono de la propia aplicación. El ícono de la aplicación es simple a más no poder. Está compuesto únicamente de la conjunción de dos de los iconos estándar de Android y el color principal. Estos dos iconos son el ícono de la lupa, que represent una búsqueda de algún tipo, y el ícono de la búsqueda de redes, que representa la búsqueda de una red. La conjunción de ambas representa el objetivo principal de la aplicación. Unido al color principal usado, logramos un ícono sumamente sencillo, pero que encaja a la perfección con la aplicación. El ícono se puede ver en la Figura 10.21.



Figura 10.21.: Ícono de la aplicación en formato redondeado

10.4. Conclusiones

El desarrollo de interfaces gráficas en Android y la aplicación de los principios de Material Design es un campo inmensamente amplio. Según qué aplicación se esté desarrollando, y de lo grande que sea, podemos encontrar perfiles concretos desarrolladores que invierten su trabajo únicamente en este aspecto, el de la interfaz gráfica y el de la experiencia de usuario. Es un campo que abarca muchas más áreas de las que se han cubierto en esta sección y el abarcar todos esos campos ya sería elemento suficiente para constituir un proyecto completamente independiente.

Lo que se ha querido mostrar en esta sección es que, aunque la aplicación desarrollada sea sumamente sencilla a nivel gráfico, se han seguido aplicando principios para mantener la consistencia han dicho interfaz. En estos casos, principios de Material Design, que logran una interfaz más clara, ordenada y atractiva a la vista, además de adaptarse a los diseños de interfaces de otras aplicaciones de Google y del propio sistema de Android.

De esta manera, en caso de que en un posible trabajo futuro se quiere ampliar la funcionalidad de la aplicación, simplemente debemos continuar siguiendo estos principios de diseño,

para tener una aplicación unificada, fácil de usar y lo más atractiva posible.

11

Implementación de la aplicación

"Talk is cheap. Show me the code"

— Linus Torvalds

Dentro del desarrollo de la aplicación, la parte que se puede considerar como la más importante es la parte de la implementación. Tomando en base todo el diseño, tanto a nivel conceptual, gráfico o de software, se ha implementado la aplicación, siendo esta uno de los dos objetivos del proyecto.

En este capítulo se explican los diferentes aspectos del proceso de desarrollo, tanto como se ha implementado la persistencia de datos, los escaneos que realiza la aplicación y todo tipo de consideraciones que se han tenido en cuenta durante la implementación y que merece la pena mencionar.

Este capítulo no pretende ser ni una descripción de cada elemento que forma parte del código, ni una documentación formal como puede ser la documentación de una API. Por razones de espacio y claridad, no se incluye todo el código desarrollado, tanto porque entorpecería las explicaciones de como se ha ido desarrollando y como se ha ido implementado la aplicación como porque haría este informe del proyecto excesivamente grande.

11.1. Estructura del proyecto

El código del proyecto se puede dividir en dos grandes bloques. Por una parte disponemos de los ficheros que contienen código en Kotlin y por otra parte los ficheros XML que definen, tanto layouts, elementos que se pueden dibujar o variables asociadas a las diferentes cadenas de caracteres (de ahora en adelante *strings*) o a variables numéricas como colores o dimensiones.

El propio entorno de Android Studio permite diferencias ambos tipos de archivos, dispo-

niendo del código en la carpeta *java* (el nombre puede llevar a confusión, pero es el nombre por defecto en la creación de proyecto, aunque contenga únicamente código en Kotlin) y de los diferentes archivos XML en la carpeta *res*. Esta última a su vez se divide en diferentes carpetas para organizar los archivos XML en función de su utilidad.

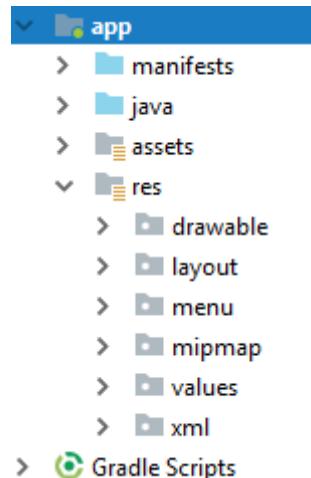


Figura 11.1.: Estructura básica de archivos del proyecto

Otras carpetas importantes son las carpetas de *manifests* y la carpeta de *assets*. En la primera es donde se guardan los diferentes *Manifest* de la aplicación. Un *Manifest*¹ en Android es un fichero XML donde se definen una serie de variables importantes a nivel global con respecto a la aplicación. Entre otras funciones, es donde se define el nombre de la aplicación, su ícono y sus diferentes componentes, que pueden ser Activities, Services, BroadcastReceivers o ContentProviders.

Por otra parte mencionar la existencia de diferentes scripts de Gradle, que es el sistema de automatización de construcción de aplicaciones más usado en Android para compilar el código y generar la aplicación ejecutable, entre otras cosas.

11.1.1. Estructura de paquetes de código

Dentro de la carpeta *java*, debido a la gran cantidad de ficheros de código Kotlin, existe una estructura de paquetes que divide los ficheros de código en diferentes categorías. Cada uno de los ficheros de código existentes implementa una única clase, interfaz o tipo enumerado. Esto es así para mantener cada elemento claramente visible y al mismo nivel y a su vez evitar archivos excesivamente grandes que compliquen entender el código. La estructura de paquetes se muestra en la Figura 11.2.

¹<https://developer.android.com/guide/topics/manifest/manifest-intro>

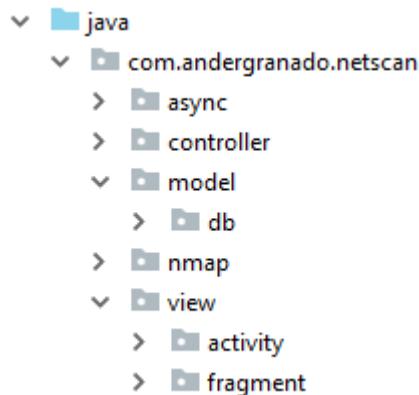


Figura 11.2.: Estructura de paquetes de código del proyecto

En los paquetes *model*, *controller* y *view* se implementan las diferentes partes de las que está formada una aplicación que sigue el patrón MVC (Model View Controller). Es un patrón largamente usado a la hora de diseñar aplicaciones con interfaz gráfica, que consiste básicamente en dividir los datos (Model), las diferentes vistas de la aplicación (View) y la parte que actúa como puente entre los datos y las vistas (Controller) en entidades separadas. Esto permite que el código generado sea escalable, legible y elimine dependencias innecesarias.

En el caso de Android, las diferentes vistas se implementan en diversas Activities y Fragments. El modelo es básicamente una serie de clases (algunas de ellas Data Classes de Kotlin) que definen los diferentes datos que existen, tanto para la aplicación como datos concretos que son persistentes en la aplicación (separados en el paquete *db*) y los controladores están implementados usando RecyclerViews de Android, cuyo funcionamiento se explicará más adelante.

Por último, mencionar los paquetes *async* y *nmap*. El paquete *async* contiene toda aquella funcionalidad que se ejecuta de manera paralela o asíncrona, como diferentes AsyncTasks o clases que se basan en la ejecución de Threads o hilos. El paquete *nmap* contiene todo el código necesario para implementar Nmap en la aplicación, tanto para instalarlo, ejecutarlo como para interpretar los datos que genera. Todo el código que hace uso de Nmap se encuentra en este paquete, dejando el resto de la aplicación completamente aislada de las particularidades de Nmap.

11.2. Integración de Nmap

Integrar soluciones de terceros en una aplicación o un sistema que está en desarrollo puede ser desde algo simple, más bien mecánico, a todo un quebradero de cabeza, en función del tipo de software o funcionalidad que queramos añadir desde fuera.

En concreto, a la hora de integrar Nmap en la aplicación hay que tener en cuenta una serie de factores. Integrar Nmap en una aplicación difiere completamente de integrar, por

ejemplo, una librería con una API definida. Integrar una librería en una aplicación consiste simplemente en una serie de pasos para configurar esa librería que, una vez realizados, nos permiten mediante una interactuar mediante una API con esa librería y aprovechar toda la funcionalidad que nos provee.

En cambio, con Nmap es radicalmente diferente. Debido a que Nmap no es una librería, sino un software complejo que nos permite analizar redes de ordenadores, no podemos integrarlo como si de una librería se tratase. Por lo tanto, debemos buscar otros métodos diferentes para integrarlo.

Nmap es una solución de código libre que lleva siendo desarrollada por la comunidad que ha generado alrededor durante más de 10 años. Cabría pensar que una posibilidad para integrar la funcionalidad de Nmap en la aplicación es obtener su código fuente (que al ser software libre es público y está disponible para su uso) e integrarlo directamente en el proyecto.

El principal problema de esta posibilidad es que Nmap es un software muy complejo. El núcleo de Nmap está desarrollado en C, pero también diversas partes están desarrolladas en lenguajes como Lua, C++ o Python, como se puede observar en la FIGURA de su repositorio². Esto impide que podamos introducir todo ese código en lenguajes no soportados por Android en el proyecto, haciendo que Nmap no se pueda integrar directamente a nivel de código.

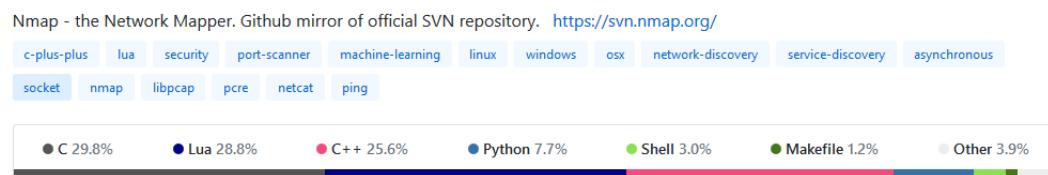


Figura 11.3.: Distribución de lenguajes en el repositorio de Nmap

Es cierto que se puede, mediante el uso del NDK de Android, introducir código tanto en C como en C++ en nuestra aplicación. Pero el funcionamiento de Nmap es tan complejo que intentar integrar solamente esa parte del código y esperar que funcione eliminando el resto de módulos sería una tarea tan titánica como inútil.

Por ello la última opción, y la única realmente viable, para integrar Nmap en nuestra aplicación es interactuar directamente con archivos binarios de Nmap. Es decir, incluir la aplicación compilada dentro del proyecto, ejecutarla mediante código y obtener e interpretar la información que devuelva.

²<https://github.com/nmap/nmap>

11.2.1. Instalación

Esto a la vez genera una serie de cuestiones. Si queremos introducir un binario dentro de nuestra aplicación y controlar su ejecución en un dispositivo lo primero que tenemos que tener en cuenta es que la arquitectura para la que ha sido compilada el binario coincide con la arquitectura del dispositivo.

Nmap es una aplicación que se puede ejecutar tanto en sistemas Windows y OS X como en sistemas basados en Linux. Aunque android es un sistema Linux, Android no se suele ejecutar en las plataformas en las que se suele ejecutar normalmente Linux, que son x86 y x64, arquitecturas Intel. La mayor parte de dispositivos Android son dispositivos con procesadores ARM BUSCAR REFERENCIA, por lo tanto no podemos ejecutar una versión estándar de Nmap para Linux.

Para solucionar esto se puede optar por dos opciones. La primera sería compilar Nmap para arquitecturas especialmente para Android en ARM. Compilar un proyecto de la envergadura de Nmap es una tarea engorrosa y difícil de realizar. La segunda sería buscar binarios ya compilados especialmente para Android. Por suerte existen binarios de Nmap para dispositivos Android.

La propia web de Nmap³ nos ofrece binarios de Nmap específicos para Android, la desventaja es que no se trata de binarios oficiales, aunque avalados por Nmap, sino de binarios generados por un usuario concreto⁴. Otra desventaja de estos binarios es que no están completamente actualizados. La versión más reciente de los binarios de Nmap para Android es la 7.31 (25 de octubre de 2016), unas versiones más atrasadas desde la última, la 7.70⁵ (20 de marzo de 2018) que a día de hoy es la última versión de Nmap.

Aunque la última versión disponible de Nmap para Android no está tan actualizada como la última versión de Nmap, hay que tener en cuenta que Nmap es un software con un desarrollo avanzado y que, aunque vaya añadiendo funcionalidad, lleva durante años siendo un software robusto y estable. Por esto utilizar Nmap para Android no debería ser un gran problema.

Nmap para Android viene compilado o para una serie de arquitecturas concretas, como se puede observar en la FIGURA. Aunque tenemos un rango amplio de arquitecturas para elegir, debemos tener en cuenta que nuestro público objetivo es un público general. La mayor parte de móviles Android utilizan procesadores ARM. Aun así, y para garantizar la compatibilidad con todos los dispositivos, se han integrado binarios de Nmap para Android para arquitecturas ARM, x86 y MIPS, con sus equivalentes de 64 bits. Con esto cubriremos todos los dispositivos móviles posibles.

³<https://secwiki.org/w/Nmap/Android>

⁴<https://github.com/kost/nmap-android>

⁵<http://seclists.org/nmap-announce/2018/0>

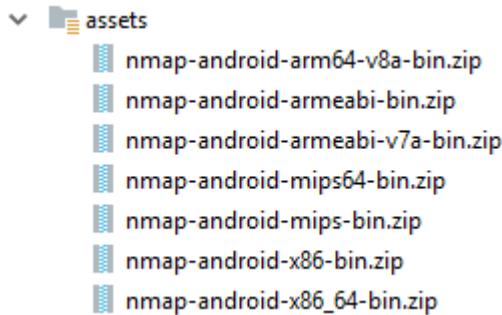


Figura 11.4.: Los diferentes binarios para cada arquitectura usados en la aplicación

La forma de introducir estos binarios en el dispositivo es sencilla y está completamente automatizada. Por una parte se almacena cada uno de los diferentes binarios en diferentes ficheros comprimidos en formato ZIP. Cada uno de estos ficheros ZIP se encuentra metido en la carpeta *assets* del proyecto, como se muestra en la Figura 11.4. Esta carpeta sirve para guardar todo tipo de recursos que queramos cargar en nuestra aplicación. Recursos como pueden ser ser archivos de texto, imágenes, archivos de audio, y un largo etcétera. Aunque en principio no esté diseñada para guardar binarios y programas completos, en nuestro caso servirá para guardar los ficheros de Nmap y después poder desplegarlos en el dispositivo.

Desplegarlos en el dispositivo es relativamente sencillo. Para ello solo debemos tener en cuenta la arquitectura el dispositivo en el que se está ejecutando la aplicación y descomprimir el archivo con los binarios correspondientes.

La localización donde guardemos los binarios de Nmap es fundamental, ya que según donde se guarden tendremos permiso para ejecutarlos o no. En este caso se guardan en lo que denominamos Internal Storage⁶ de Android. Dentro de la estructura de archivos de Android existe una carpeta llamada *data*, que a su vez tiene otra carpeta del mismo nombre. Dentro de esa última carpeta existe una carpeta por cada aplicación instalada en el sistema. Cada una contiene una aplicación y sus archivos asociados. Esta carpeta sólo es accesible para la propia aplicación, que tiene permisos de lectura, escritura y ejecución dentro de ella, lo que la convierte en el lugar idóneo para depositar los binarios.

Toda este volcado de binarios de Nmap para el dispositivo se gestiona desde una única clase en la aplicación llamada *NmapInstaller*. Esa clase, que implementa el patrón Singleton mediante el uso de la palabra clave *object* de Kotlin, contiene todo el código necesario para volcar los binarios de Nmap en el dispositivo.

Listing 1: Función que instala Nmap en el dispositivo

```
fun install(activity: Activity, force: Boolean = false): File {
    val context = activity.applicationContext
    nmapPath = context.filesDir.path
```

⁶<https://developer.android.com/training/data-storage/files>

```

nmapBinPath = "$nmapPath/nmap/bin/nmap"

if (!nmapDirExists() || !installed || force) {
    val assetManager = activity.assets
    val ins = assetManager.open(filePrefix + Build.SUPPORTED_ABIS[0] +
        fileSuffix)
    val zin = ZipInputStream(ins)
    try {
        var entry: ZipEntry = zin.nextEntry
        do {
            if (entry.isDirectory) {
                val myDir = File("$nmapPath/${entry.name}")

                if (!myDir.isDirectory)
                    if (!myDir.mkdirs())
                        throw IOException("Cannot create the directory")

            } else {
                val buffer = ByteArray(2048)
                val outStream = FileOutputStream("$nmapPath/${entry.name}")
                val bufferOut = BufferedOutputStream(outStream, buffer.size)

                var size = zin.read(buffer, 0, buffer.size)
                while (size != -1) {
                    bufferOut.write(buffer, 0, size)
                    size = zin.read(buffer, 0, buffer.size)
                }

                bufferOut.flush()
                bufferOut.close()
            }
            entry = zin.nextEntry
        } while (entry != null)
    } catch (e: Exception) {
        Log.e("Nmap unzipping...", e.message)
    }
    zin.close()
    nseDbUpdate()
}

val nmapExec = File(nmapBinPath)
nmapExec.setExecutable(true)

installed = true

return nmapExec
}

```

Como se puede observar en el código, dentro de la función `install()` se realiza ese volcado pudiendo tener opciones para reescribir los archivos en caso de que se quiere escribir. Estos archivos se cargan a través del Asset Manager y mediante las clases de Java para interactuar con ficheros comprimidos se extraen en los ficheros internos de la aplicación.

A su vez también se actualiza la base de datos de script NSE de Nmap, en caso de que se quiera después ejecutar alguno de esos script.

11.2.2. Ejecución

Una vez tenemos disponibles todos los binarios de Nmap y archivos necesarios para ejecutar Nmap introducidos e instalados en el dispositivo, el siguiente paso necesario es implementar la funcionalidad que nos permita ejecutarlos.

Todo el código que permite ejecutar en el mapa se encuentra en otra clase llamada `NmapRunner`. Esta clase tiene diversos métodos que nos permiten interactuar con los binarios y en última instancia ejecutar escaneos.

Lo que tenemos que hacer para ejecutar un escaneo es ejecutar el binario de Nmap con los parámetros correspondientes. Para ejecutar en el mar basta simplemente con lanzar un proceso que llame a la ejecución del binario. La forma de ejecutar un binario es muy sencilla, y consiste en crear un proceso para una shell de Linux y en el llamar al ejecutable de Nmap. Para poder interactuar con ese proceso necesitamos configurar la entrada y la salida de ese proceso para poder enviarle información y recibirla. Eso se realizan la función `startProcess()`, que se muestra a continuación.

Listing 2: Función para arrancar el proceso con la shell que ejecutará Nmap

```
private fun startProcess() {
    if (scanProcess == null || processInputReader == null || processOutputStream
        == null) {
        // Creates a new process that runs a shell
        val processBuilder = ProcessBuilder("sh")
        processBuilder.redirectErrorStream(true)
        scanProcess = processBuilder.start()

        // Creates a couple of streams to redirect the IO
        processOutputStream = DataOutputStream(scanProcess?.outputStream)
        processInputReader =
            BufferedReader(InputStreamReader(scanProcess?.inputStream))
    }
}
```

Después de esto debemos construir un comando de Nmap. Este comando es básicamente lo que haríamos si estuviéramos ejecutando Nmap en la terminal de nuestro ordenador. En nuestro caso solo queremos realizar una serie de escaneos concretos a nodos puntuales, por lo tanto construir este comando es tan sencillo como concatenar unos pocos strings. La función que realiza esto se muestra a continuación.

Listing 3: Función que crea el comando a ejecutar de Nmap

```
private fun commandBuilder(hosts: List<String>, outputFile: File): String {
    var hostsString = ""
    hosts.forEach { hostsString += "$it " }
    val args = when (scanType) {
        ScanType.REGULAR -> ""
        ScanType.PING -> "-sn"
        ScanType.QUICK -> "-T4"
        ScanType.FULL -> "-A --no-stylesheet"
    }
    ...
}
```

```

    }
    return "${NmapInstaller.nmapBinPath} $args $hostsString -oX
           ${outputFile.path}\n"
}

```

Una vez realizado esto queda pasar ese comando al proceso y ejecutar Nmap. A partir de aquí podríamos obtener la información del escaneo de dos maneras. La primera sería leer la salida que nos da el proceso en texto plano y extraer la información de. La segunda, mejor alternativa, consiste en ejecutar Nmap de tal manera que la información del escaneo se guarde en un fichero XML.

Las ventajas de utilizar un fichero XML para cobrar esa información son obvias. En un fichero XML dispondremos de la información estructurada que podremos leer utilizando la librería para trabajar con archivos XML que provee el SDK de Android. Por lo tanto, y como se ha podido observar, se añade el parámetro `-oX` cuándo se crea el comando a ejecutar, qué es el que indica que la salida se almacene en un fichero XML.

Una vez realizado el escaneo y habiéndose generado ese fichero XML, solo queda obtener la información de ese fichero y deshacernos de todos los recursos ya innecesarios, como el propio fichero XML una vez leído, o el propio proceso de la shell, que consume recursos no necesarios.

Listing 4: Función con todo el proceso de ejecución de un escaneo en Nmap

```

fun runScan(hosts: List<String>): NmapScan? {
    if (!NmapInstaller.installed)
        throw Exception("Nmap is not installed in the device")

    startProcess()
    var outputFile = setupOutputFile()

    processOutputStream?.writeBytes(commandBuilder(hosts, outputFile))
    processOutputStream?.writeBytes("exit\n")
    processOutputStream?.flush()

    // Gets all the output from the process
    var pstdout: String? = processInputReader?.readLine()
    val wholeOutput = mutableListOf<String>()
    while (pstdout != null) {
        pstdout += "\n"
        wholeOutput.add(pstdout)
        pstdout = processInputReader?.readLine()
    }

    scanProcess?.waitFor()
    outputFile = File(outputFile.path)

    val scan = parser.parse(outputFile.inputStream())
    outputFile.delete()
    endProcess()
}

```

```

    return scan
}

```

11.2.3. Leer datos

La tercera y última parte del proceso para integrar Nmap es leer los datos que han sido generados. Como se ha mencionado dichos datos se encuentran en un fichero XML. Por lo tanto, necesitaremos una clase que obtenga la información de dicho fichero. A este tipo de clases se le suele denominar *parser* y básicamente van leyendo las diferentes etiquetas de los ficheros XML y obteniendo su información. A continuación se muestra ciertas partes del código para hacerse la idea de cómo es leer de manera jerarquizada y estructura de un fichero XML. No se muestra toda la implementación, ya que el tamaño de la implementación de un parser de XML es directamente proporcional al tamaño de la jerarquía de los elementos que contiene.

```

@Throws(XmlPullParserException::class, IOException::class)
private fun readNmapRun(parser: XmlPullParser): NmapScan {
    var info: NmapScanInfo? = null
    val hosts = mutableListOf<NmapHost>()
    var stats: NmapRunStats? = null

    parser.require(XmlPullParser.START_TAG, namespace, "nmaprun")
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.eventType != XmlPullParser.START_TAG)
            continue

        when (parser.name) {
            "scaninfo" -> info = readScanInfo(parser)
            "host" -> hosts.add(readHost(parser))
            "runstats" -> stats = readRunStats(parser)
            else -> skip(parser)
        }
    }
    return NmapScan(info, hosts, stats)
}

@Throws(IOException::class, XmlPullParserException::class)
private fun readScanInfo(parser: XmlPullParser): NmapScanInfo {
    parser.require(XmlPullParser.START_TAG, namespace, "scaninfo")
    val numServices = parser.getAttributeValue(null, "numservices").toInt()
    val protocol = when (parser.getAttributeValue(null, "protocol")) {
        "ip" -> Protocol.IP
        "tcp" -> Protocol.TCP
        "udp" -> Protocol.UDP
        "sctp" -> Protocol.SCTP
        else -> Protocol.TCP // The default scan uses only TCP, and the XML attribute is required in the
        // nmap.dtd
    }
    val services = parser.getAttributeValue(null, "services")
    parser.next()

    return NmapScanInfo(numServices, protocol, servicesStringToList(services))
}

@Throws(IOException::class, XmlPullParserException::class)
private fun readHost(parser: XmlPullParser): NmapHost {
    var status: HostStatus? = null
    var address: Address? = null
    var hostNames = mutableListOf<HostName>()
    var ports = mutableListOf<NmapPort>()

    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.eventType != XmlPullParser.START_TAG)
            continue

        when (parser.name) {
            "status" -> status = readStatus(parser)
            "address" -> address = readAddress(parser)
            "hostnames" -> hostNames = readHostNames(parser)
        }
    }
    return NmapHost(status, address, hostNames, ports)
}

```

```

        "ports" -> ports = readPorts(parser)
    }
}
if (status != null && address != null)
    return NmapHost(status, address, hostNames, ports)
else
    throw XmlPullParserException("Can't read status or address")
}

```

Listing 5: Extracto de la implementación de un parser de un XML de Nmap

Almacenar la información en XML la almacena jerarquizada en cierta manera la manera en la que Nmap guarda información estructurada en un fichero XML es como la que se muestra en el 6:

Listing 6: Fichero XML con la información de un escaneo estándar de Nmap

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nmaprun>
<?xml-stylesheet href="file:///usr/bin/../share/nmap/nmap.xsl" type="text/xsl"?>
<!-- Nmap 7.60 scan initiated Fri Feb 23 10:32:25 2018 as: nmap -oA 1-regular google.com -->
<nmaprun args="nmap -oA 1-regular google.com" scanners="nmap" start="1519399945" startstr="Fri Feb 23
→ 10:32:25 2018" version="7.60" xmloutputversion="1.04">
<scaninfo numservices="1000" protocol="tcp"
→ services="1,3-4,6-7,9,13,17,19-26,30,32-33,37,42-43,49,53,70,79-85,88-90,99-100,106, ...."
→ type="syn"/>
<verbose level="0"/>
<debugging level="0"/>
<host endtime="1519399960" starttime="1519399945">
    <status reason="reset" reason_ttl="255" state="up"/>
    <address addr="216.58.201.142" addrtype="ipv4"/>
    <hostnames>
        <hostname name="google.com" type="user"/>
        <hostname name="mad06s25-in-f142.1e100.net" type="PTR"/>
    </hostnames>
    <ports>
        <extraports count="990" state="filtered">
            <extrareasons count="990" reason="no-responses"/>
        </extraports>
        <port portid="21" protocol="tcp">
            <state reason="syn-ack" reason_ttl="64" state="open"/>
            <service conf="3" method="table" name="ftp"/>
        </port>
        <port portid="25" protocol="tcp">
            <state reason="syn-ack" reason_ttl="64" state="open"/>
            <service conf="3" method="table" name="smtp"/>
        </port>
        <port portid="80" protocol="tcp">
            <state reason="syn-ack" reason_ttl="64" state="open"/>
            <service conf="3" method="table" name="http"/>
        </port>
        <port portid="110" protocol="tcp">
            <state reason="syn-ack" reason_ttl="64" state="open"/>
            <service conf="3" method="table" name="pop3"/>
        </port>
        <port portid="443" protocol="tcp">
            <state reason="syn-ack" reason_ttl="64" state="open"/>
            <service conf="3" method="table" name="https"/>
        </port>
        <port portid="2000" protocol="tcp">
            <state reason="syn-ack" reason_ttl="64" state="open"/>
            <service conf="3" method="table" name="cisco-sccp"/>
        </port>
    </ports>
    <times rttvar="46949" srtt="31610" to="219406"/>
</host>
<runstats>
    <finished elapsed="15.38" exit="success" summary="Nmap done at Fri Feb 23 10:32:40 2018; 1 IP
→ address (1 host up) scanned in 15.38 seconds" time="1519399960" timestr="Fri Feb 23 10:32:40
→ 2018"/>
    <hosts down="0" total="1" up="1"/>
</runstats>
</nmaprun>

```

Sin entrar en detalles sobre el funcionamiento cada una etiqueta se puede observar que se genera una etiqueta *host* por cada uno de los nodos encontrados en el escaneo. Un solo escaneo puede servir tanto para único host como para un rango completo de redes CIDR, por lo que habrá tantas etiquetas host como nodos se hayan detectado.

Dentro de la información generada para cada host, aunque depende del tipo de escaneo realizado, se pueden observar etiquetas que aparecerán para la gran mayoría de casos. Por ejemplo *status*, que nos indica el estado del host, *address*, que nos indica la dirección IP del host, *hostname*, que es el nombre que recibe dicho host, o la etiqueta *ports*, que contiene una lista de todos los puertos abiertos o filtrados que se han encontrado para cada nodo, con información sobre configuraciones, métodos usados para escanearlos o nombres de dichos puertos.

Por último una vez leído ese fichero a medida que se va leyendo ese fichero se va guardando la información en diferentes clases, creadas específicamente para guardar los datos obtenidos de Nmap antes de ser procesados. Dichas clases se encuentran dentro del paquete *model*. En concreto, todas las clases que se usan para datos obtenidos directamente de Nmap llevan el prefijo Nmap, para diferenciarlas de clases del modelo relacionadas con la estructura de la información en la base de datos. En los Códigos 7, 8 y 9 se muestran unos ejemplos.

Listing 7: Data Classes para la información de un scan en Nmap

```
package com.andergranado.netscan.model

import java.io.Serializable

data class NmapScan(val scanInfo: NmapScanInfo?,
                    val hosts: List<NmapHost>,
                    val runStats: NmapRunStats?) : Serializable
```

Listing 8: Data Classes para la información de un host en Nmap

```
package com.andergranado.netscan.model

import java.io.Serializable

data class NmapHost(val status: HostStatus,
                    val address: Address,
                    val hostNames: List<HostName>,
                    val ports: List<NmapPort>) : Serializable

data class HostStatus(val state: HostStates,
                      val reason: String) : Serializable

enum class HostStates { UP, DOWN, UNKNOWN, SKIPPED }

data class Address(val address: String,
                  val addressType: AddressType) : Serializable

enum class AddressType { IPV4, IPV6, MAC }
```

```
data class HostName(val name: String,  
                    val type: HostType) : Serializable  
  
enum class HostType { USER, PTR }
```

Listing 9: Data Classes para la información de un puerto en Nmap

```
package com.andergranado.netscan.model  
  
import java.io.Serializable  
  
data class NmapPort(val id: Int,  
                     val type: Protocol,  
                     val service: String,  
                     val state: PortState) : Serializable  
  
enum class Protocol { IP, TCP, UDP, SCTP }  
  
data class PortState(val state: StateType,  
                     val reason: String) : Serializable  
  
enum class StateType { OPEN, FILTERED, UNFILTERED, CLOSED, OPEN_FILTERED,  
→ CLOSED_FILTERED, UNKNOWN }
```

11.3. Persistencia de datos

A día de hoy prácticamente cualquier aplicación, independientemente de si es una aplicación móvil o una aplicación para cualquier otro sistema, requiere almacenar datos de manera persistente. Ya sea desde simples valores para guardar configuraciones o ajustes de usuario a todo un conjunto de datos que se genera, gestiona y define el funcionamiento de dicha aplicación.

En el caso de nuestra aplicación, que tiene como objetivo principal analizar redes informáticas y obtener información, necesitaremos guardar la información de cada uno de esos análisis para que el usuario la tenga siempre accesible en todo momento.

La persistencia de datos una aplicación se puede implementar de diversas maneras, y la elección de una manera u otra obedece a varios factores. Hay que tener en cuenta, entre otras cosas, el volumen de datos que se vaya a generar, la frecuencia con la que se van a modificar esos datos o si esos datos van a ser transferidos en algún momento.

A grosso modo se pueden dividir las formas de implementar persistencia de datos en dos principales tipos: el uso de ficheros o el uso de bases de datos.

El uso de ficheros es el modo más simple de almacenar datos. Consiste en guardar los datos escribiendo en ficheros y en obtener datos leyendo de ellos. Esos ficheros pueden contener los datos en sí mismos o junto a algún tipo de estructura, siendo esta la más sencilla de mantener. Como ejemplos de datos estructurados tenemos una serie de formatos como XML, CSV o JSON.

Por otra parte, tenemos el uso de bases de datos. A medida que los datos crecen en complejidad y en número, se vuelve cada vez más complicado el acceso lectura y escritura dichos datos. El uso de bases de datos permite almacenar una gran cantidad de datos relacionados entre ellos, de los cuales podemos obtener información concreta y modificar la información más concreta mediante lenguajes como SQL.

A la hora de elegir entre una de las dos para nuestra aplicación debemos tener en cuenta que el uso de bases de datos permite mayor escalabilidad y nos permite gestionar mejor los datos que son generados. Aun añadiendo otra capa más, como es sistema de gestión de bases de datos, que a priori resulta más costoso computacionalmente respecto al uso de ficheros, a la larga va a ser la mejor opción para guardar los datos de nuestra aplicación.

Una vez elegido que vamos a implementar la persistencia de datos usando bases de datos el siguiente paso a realizar es elegir qué sistema de gestión de bases de datos vamos a utilizar. En este caso la elección resulta bien sencilla. El propio SDK de Android implementa funcionalidad para poder trabajar con bases de datos SQLite. SQLite es un sistema de gestión de bases de datos relacional basado en SQL que es conocido por ser ligero y eficiente.

11.3.1. Diseño de la base de datos

El diseño de nuestra base de datos resultó sumamente simple. Únicamente queremos guardar la información de una serie de escaneos. Sabemos que un escaneo al final está compuesto de la información de una serie de nodos y, a su vez, de cierta información secundaria, como la fecha que se ha hecho o el nombre de la red. Cada nodo tiene su propia información, desde el fabricante del dispositivo a la dirección IP o la lista de puertos que tiene abiertos, entre otros. El modelo que se plantea para esta base de datos es sumamente sencillo y se muestra en la Figura 11.5.

Cómo podemos observar solo contamos únicamente con cuatro tablas (se han omitido las tablas generadas automáticamente por Android). Una para cada escaneo, otra para las diferentes estadísticas sobre cada escaneo, otra para la información de cada nodo y una última para la información de un puerto de un nodo concreto. Este caso se ha optado por almacenar esta información, pero resultaría fácilmente escalable simplemente añadiendo más campos en nuestra base de datos.

11.3.2. Implementación de la base de datos

Tras tener la base de datos diseñada debemos implementar la nuestra aplicación. Implementar una base de datos en Android se puede hacer de dos maneras. La primera, la manera tradicional, es utilizar la API estándar de Android para interactuar con bases de datos SQLite. Esta manera implica que debemos introducir nuestras sentencias SQL y ejecutarlas nosotros mediante las diferentes funciones y métodos que nos ofrece la API.

Name	Type
Tables (7)	
Node	
id	INTEGER
name	TEXT
ip	TEXT
mac	TEXT
vendor	TEXT
timeElapsed	REAL
scanId	INTEGER
Port	
id	INTEGER
nodeId	INTEGER
protocol	TEXT
service	TEXT
state	TEXT
reason	TEXT
Scan	
id	INTEGER
date	INTEGER
name	TEXT
ScanStats	
scanId	INTEGER
scannedHosts	INTEGER
hostsUp	INTEGER
hostsDown	INTEGER
scanTime	REAL

Figura 11.5.: Arquitectura de Room e interrelación entre sus diferentes componentes

Una alternativa a esta API estándar sería el uso de la librería *Room*⁷. Esta librería es una de las librerías que forman parte de lo que se denomina Android Architecture Components. Los Architecture Components son diferentes librerías pensadas para solucionar problemas que surgen a la hora de desarrollar aplicaciones en Android y simplificar en gran medida el desarrollo de ciertos tipos de funcionalidad.

Room está pensado para abstraernos de toda la lógica de SQL y permitirnos interactuar con la base de datos de una manera más limpia y natural. Al abstraer toda esa capa de composición de sentencias SQL para realizar nuestras consultas y modificaciones de la base de datos, el código a mantener resulta muchísimo más sencillo y elegante.

Para lograr esto se basa en una arquitectura compuesta de tres bloques o componentes principales. Dicha arquitectura se puede observar en la FIGURA.

⁷<https://developer.android.com/topic/libraries/architecture/room>

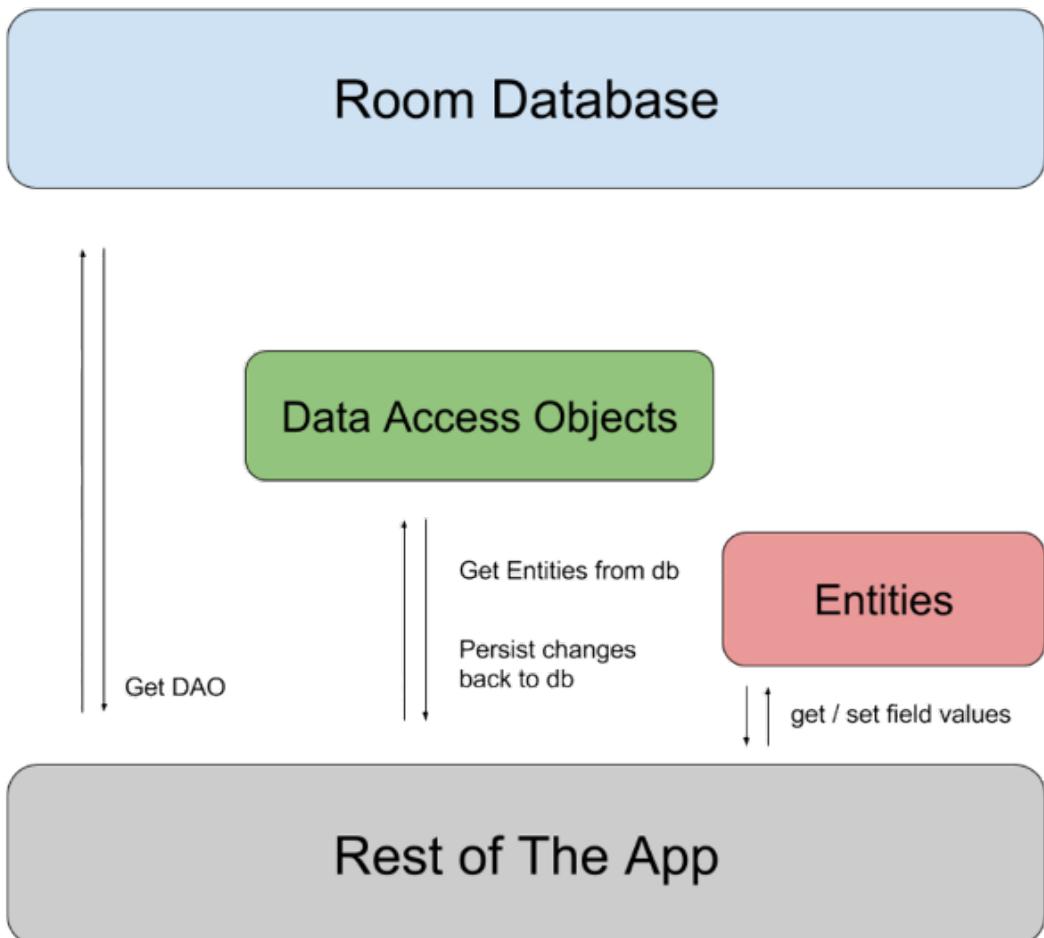


Figura 11.6.: Arquitectura de Room e interrelación entre sus diferentes componentes

El primero es el componente *Database*. Este componente es el que nos permite realizar tareas con la base de datos a nivel global. Desde definir las diferentes tablas que pueda haber a obtener la instancia de la base de datos para poder trabajar con ella.

Listing 10: Implementación de la clase AppDatabase

```

/**
 * A [RoomDatabase] subclass to interact with the database and their DAOs.
 */
@Database(entities = [Scan::class, Node::class, Port::class, ScanStats::class],
        version = 7)
@TypeConverters(Converters::class)
abstract class AppDatabase : RoomDatabase() {

    abstract fun scanDao(): ScanDao

    abstract fun nodeDao(): NodeDao
}
  
```

```

abstract fun portDao(): PortDao

abstract fun scanStatsDao(): ScanStatsDao

companion object {
    const val DATABASE_NAME: String = "NetScanDB"

    private var instance: AppDatabase? = null

    fun getInstance(context: Context): AppDatabase {
        if (instance == null) {
            synchronized(AppDatabase::class) {
                instance = Room.databaseBuilder(context,
                    AppDatabase::class.java, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build()
            }
        }
        return instance!!
    }

    fun destroyInstance() {
        instance = null
    }
}
}

```

En el código 10 se muestra la implementación de ese componente en nuestra aplicación. Aún no habiendo visto nunca cómo funciona Room se pueden incluir ciertas cosas examinando el código. La primera es que se define mediante anotaciones ciertas clases. Esas clases son las que definen cada una de las tablas que vamos a tener en nuestra base de datos. En Room, las tablas son conocidas como *Entities*, que es otro de los tres componentes de Room. También se puede observar que se añaden una serie de funciones que contienen la nomenclatura *DAO*. Esas funciones son las que nos van a permitir obtener los diferentes DAOs que puedan existir en Room, siendo este tipo de componente el tercero y último.

Como se ha mencionado, las Entities corresponden a cada una de las tablas que existe en la base de datos. Para reflejar esto en una clase de tipo Entity, de la misma manera que se hacía para el componente Database, se utilizan anotaciones.

Listing 11: Implementación de diferentes clases Entity

```

/**
 * A single node in a network scan.
 */
@Entity
open class Scan(val name: String) {

    @PrimaryKey(autoGenerate = true)
    var id: Int = 0

    var date: Date = Date()
}

```

```

@Entity(foreignKeys = [ForeignKey(entity = Scan::class, parentColumns = ["id"],
    childColumns = ["scanId"])])
open class ScanStats(@PrimaryKey
    var scanId: Int,
    var scannedHosts: Int,
    var hostsUp: Int,
    var hostsDown: Int,
    var scanTime: Float)

/**
 * A single node in a network scan.
 */
@Entity
open class Node(val name: String,
    val ip: String,
    val mac: String,
    val vendor: String,
    val timeElapsed: Float,
    val scanId: Int) {

    @PrimaryKey(autoGenerate = true)
    var id: Int = 0
}

/**
 * A single port of a concrete node.
 */
@Entity(primaryKeys = ["id", "nodeId"])
open class Port(val id: Int,
    val nodeId: Int,
    val protocol: Protocol,
    val service: String,
    val state: StateType,
    val reason: String)

```

Cómo se puede observar en el ejemplo que se ha puesto en el CÓDIGO, se utilizan anotaciones para definir una entidad o para definir aspectos como la clave primaria de una tabla de la base de datos. Cada uno de los atributos de esa clase equivaldría a un campo de la tabla. Los tipos de datos de la tabla equivalen a los tipos de datos que se han utilizado en los atributos de la clase. Para tipos de atributos complejos Room implementa un componente llamado *Converters* que nos permite implementar funciones que convierten tipos de datos complejos, como imágenes, datos personalizados o clases complejas, en tipos de datos almacenables en una base de datos, de tal manera que se ejecuten automáticamente a la hora de interactuar con la base de datos.

Por último, el tercer componente que se ha mencionado son los *DAO* (Data Access Object). Estos componentes son interfaces que definen una serie de funciones. Esas funciones son las que se utilizarán para hacer consultas o modificaciones en la base de datos. El uso de sentencias SQL como *SELECT*, *INSERT*, *UPDATE* o *DELETE* queda encapsulado mediante el uso de estas funciones, que tanto reciben como devuelven, dependiendo el caso, tipos de datos manipulables a nivel de código. Cada interfaz DAO permite interactuar con una o mas tablas, siendo lo mas recomendable implementar una para cada tabla definida.

Listing 12: Ejemplo de una interfaz DAO para interactuar con la base de datos

```

/**
 * DAO interface with all the DB management for a [Scan].
 */
@Dao
interface ScanDao {

    @get:Query("SELECT * FROM scan")
    val all: List<Scan>

    @Query("SELECT max(id) FROM scan")
    fun lastInsertedId(): Int

    @Query("SELECT * FROM scan WHERE id = :id")
    fun getScan(id: Int): Scan

    @Query("SELECT * FROM scan WHERE name = :name")
    fun getScan(name: String): Scan

    @Query("SELECT * FROM node WHERE scanId = :id")
    fun getNodeList(id: Int): List<Node>

    @Update
    fun updateScan(vararg scans: Scan)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertScan(vararg scans: Scan)

    @Delete
    fun deleteScan(vararg scans: Scan)
}

```

Como se puede apreciar en el código 12, mediante anotaciones es donde se especifican las sentencias SQL a las que equivale la ejecución de esas funciones.

Por lo tanto para ejecutar algún tipo de operación en la base datos simplemente hay que combinar estos tres elementos, para disponer de herramientas para definir el modelo de la base de datos e interactuar con la instancia de la base de datos. Por poner unos ejemplos, en los códigos 13 y 14 se muestran partes concretas de la aplicación en las que se realiza una consulta y una inserción en la base de datos, respectivamente.

Listing 13: Ejemplo de obtención de información de la base de datos Room

```

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                           savedInstanceState: Bundle?): View? {
    val view = inflater.inflate(R.layout.fragment_node_list, container, false)

    // Set the adapter
    if (view is RecyclerView) {
        val db: AppDatabase = AppDatabase.getInstance(view.context)
        val nodes = db.nodeDao().getNodesFromScan(scanId)
        nodeRecyclerViewAdapter =
            MyNodeRecyclerViewAdapter(nodes.toMutableList(), listener)
        view.adapter = nodeRecyclerViewAdapter
    }
}

```

```

    }
    return view
}

```

Listing 14: Ejemplo de inserción de información en la base de datos Room

```

override fun onProgressUpdate(vararg values: Node?) {
    if (emptyScan) {
        db.scanDao().insertScan(Scan(scanName))
        scanId = db.scanDao().lastInsertedId()
        emptyScan = false
    }
}

```

Se ha podido ir percibiendo, en ningún momento se ha implementado código que realmente ejecute sentencias SQL. Aquí reside la magia de Room. Room abstrae completamente la programación de ese tipo de código siendo la propia librería quién se encarga de generar ese código a la hora de compilar la aplicación.

Las ventajas de utilizar Room con respecto a la API estándar de SQLite son obvias. Tanto es así que la propia documentación ”recomienda encarecidamente usar Room en vez de SQLite”[46]. En primer lugar, abstraemos completamente la generación de código donde concatenamos, generamos y ejecutamos sentencias SQL. Por otra parte, el código que gestiona la base de datos resulta mucho más claro y conciso limitándose a consistir en un par de llamadas a funciones. Por último, la creación de las tablas de la base de datos en sí misma se hace también de manera automática. Incluso tenemos herramientas para poder generar diferentes versiones de la propia base datos a medida que la aplicación vaya escalando, guardándose históricas de versiones de la base de datos para en caso de revertir cambios volver a la estructura de una de las bases de datos anteriores.

11.4. Obtención de información de medios externos

Gracias al uso de Nmap podemos obtener información de los diferentes nodos de las redes a escanear. Aún así, a causa de varios factores, necesitamos adquirir partes concretas de información desde medios externos.

11.4.1. Obtención del fabricante de un nodo de la red

Uno de los puntos fuertes de nuestra aplicación es el que al escanear los nodos de la red, podamos identificar el fabricante del dispositivo escaneado. El problema es que Nmap únicamente es capaz de hacer esto si es capaz de conocer la dirección de hardware (también conocida como dirección MAC o dirección física) del dispositivo que está escaneando. Debido a que, por el propio funcionamiento de Nmap, solo puedo obtener la dirección de hardware

si se tiene privilegios de superusuario, esa información no podemos obtenerla directamente mediante el uso Nmap.

Para ello, y como alternativa, se ha programado la obtención de la dirección de hardware de manera manual. La obtención de esta dirección de hardware se realiza analizando la información de las tablas ARP. ARP es un protocolo de comunicaciones responsable de encontrar la dirección de hardware que corresponde a una determinada dirección IP. las tablas ARP son las que contienen la información que relacionan esas direcciones de hardware con las direcciones IP. Mediante un sencillo código, implementado en una función, podemos leer este fichero, que no deja de ser un fichero de texto plano, y obtener dicha MAC de manera sencilla, y juntarla al resto de la información del nodo para finalmente insertar dicha información en nuestra base de datos. Se puede observar la implementación en el Código 15.

Listing 15: Código que obtiene la dirección física en base a una dirección IP

```
protected fun getMacAddress(ip: String): String? {
    val reader = BufferedReader(InputStreamReader(FileInputStream(ARP_TABLE),
        "UTF-8"))
    reader.use {
        var line = it.readLine()

        while (line != null) {
            val arpLine = line.split("\s+".toRegex()).dropLastWhile {
                it.isEmpty() }

            val arpIp = arpLine[0]
            val flag = arpLine[2]
            val macAddress = arpLine[3]

            if (arpIp == ip)
                if (flag != ARP_INCOMPLETE && macAddress != ARP_INACTIVE)
                    return macAddress.toUpperCase()

            line = it.readLine()
        }
    }
    return null
}
```

Tras obtener la dirección MAC, necesitamos, en base a ella, obtener el fabricante del dispositivo. Esto es posible debido a que el reparto de direcciones MAC está regulado y a la hora de establecer una interfaz de red en un dispositivo, que va a tener una dirección de hardware única, el fabricante necesita previamente un espacio de direcciones MAC para sus dispositivos.

Cabe recordar que una dirección MAC está formada por 48 bits y se suele representar con cada uno de sus 6 bytes en hexadecimal, separados entre ellos por dos puntos. Dentro de esta dirección de hardware, los tres primeros bytes, es decir la primera mitad de la dirección de hardware, corresponde a un único fabricante, que tiene permiso para distribuir dispositivos con direcciones de hardware que comiencen por esos 3 bytes. Al estar reguladas estas direcciones de hardware, solo debemos buscar en algún medio externo esa información. Dicha

información la provee directamente ICANN (Internet Corporation for Assigned Names and Numbers), que es una entidad que se encarga de repartir direcciones IP y MAC, entre otras cosas.

En nuestro caso concreto utilizamos la información de un fichero de texto⁸ contenido en las bases de datos de Wireshark Foundation, conocida empresa por el desarrollo del programa Wireshark (qué se utiliza para sniffing de paquetes de una red). Este archivo lo bajamos directamente de Internet y en base a él obtenemos información del fabricante. El archivo tiene el formato que se muestra en el fichero 16.

Listing 16: Extracto del fichero que relaciona direcciones MAC on fabricantes

```
// ...
10:DA:43      Netgear
10:DD:B1      Apple     Apple, Inc.
10:DD:F4      MaxwayEl  Maxway Electronics CO., LTD
10:DE:E4      Automati  automationNEXT GmbH
10:DF:8B      Shenzhen  Shenzhen CareDear Communication Technology Co., Ltd
10:DF:FC      Siemens   Siemens AG
10:E2:D5      QiHardwa  Qi Hardware Inc.
10:E3:C7      SeohwaTe  Seohwa Telecom
10:E4:AF      AprLlc    Apr, Llc
10:E6:8F      Kwangsun  Kwangsung Electronics Korea Co., Ltd.
10:E6:AE      SourceTe  Source Technologies, LLC
10:E7:C6      HewlettP Hewlett Packard
10:E8:78      Nokia
10:E8:EE      Phasespa  PhaseSpace
10:EA:59      CiscoSpv  Cisco SPVTG
10:EE:D9      CanogaPe  Canoga Perkins Corporation
10:F0:05      IntelCor   Intel Corporate
10:F1:63      Tnk       Tnk Co., Ltd
10:F1:F2      LgElectr  LG Electronics (Mobile Communications)
10:F3:11      Cisco     Cisco Systems, Inc
// ...
```

Para gestionar la descarga la lectura y cotejar la dirección MAC con esa información obtenida mediante el archivo se han implementado varias partes. Lo primero es una interfaz genérica conocida como `DownloadableResource`. Esta clase abstracta simplemente define una serie de variables y métodos para implementar clases que sirvan para descargar un recurso de Internet.

Listing 17: Clase abstracta para un recurso descargable

```
abstract class DownloadableResource {

    abstract val urlString: String
    abstract var downloaded: Boolean
        protected set
    abstract protected val downloadThread: Thread
```

⁸https://code.wireshark.org/review/gitweb?p=wireshark.git;a=blob_plain;f=manuf

```
    abstract fun downloadFile()
    abstract fun waitForDownload()
}
```

Cómo se puede observar en el Código 17, se definen funciones para descargar un archivo mediante el uso de un *Thread* o hilo. El descargar el archivo mediante el uso de un hilo, es decir de manera paralela a la ejecución de la aplicación, es necesario ya que por motivos de usabilidad, Android por defecto no permite realizar operaciones de red en el principal de la aplicación (lanzando en caso de hacerlo una excepción de tipo `NetworkOnMainThreadException`). El Thread principal está exclusivamente reservado a la ejecución de operaciones relacionadas con la interfaz. De esta manera, no se bloqueará la ejecución de la aplicación simplemente por estar descargando un archivo. Por otro lado, el crear esta clase abstracta o interfaz permite en un futuro, si es necesario, implementar más recursos de este tipo para obtener información de diferentes fuentes.

En este caso concreto heredado de esta clase abstracta, creando una nueva clase (que implementa el patrón Singleton) llamada `OUIs`. Esta clase contiene todos los métodos para gestionar la descarga de fichero de texto, además funciones para obtener, en base a una dirección MAC, el fabricante del dispositivo analizado. Esto se hace de manera muy sencilla, recorriendo el archivo, leyendo los valores e identificando las direcciones MAC hasta encontrar el resultado, como se puede ver en el Código 18.

Listing 18: Obtención el fabricante en base a la dirección MAC

```
object OUIs: DownloadableResource() {

    override val urlString: String =
        "https://code.wireshark.org/review/gitweb?p=wireshark.git;a=blob_plain;f=manuf"

    private data class Vendor(val mac: String, val vendorShort: String, val
        vendorFull: String)
    private val vendorList: MutableList<Vendor> = mutableListOf()

    override var downloaded = false

    override val downloadThread = Thread(Runnable /* ... */)

    override fun downloadFile() /* ... */

    override fun waitForDownload() /* ... */

    fun checkVendorFromMac(mac: String): String {
        if (mac == "")
            return ""

        val macPrefix = mac.substring(0, 8)
        for (vendor in vendorList)
            if (vendor.mac == macPrefix)
                return vendor.vendorFull

        return ""
    }
}
```

También de manera análoga a la obtención del fabricante de un dispositivo se ha obtenido información sobre diferentes puertos.

La información que da sobre un puerto Nmap resulta de utilidad, pero no permite entender para qué sirve ese puerto. Se puede llegar a conocer el funcionamiento de puertos comunes, como el puerto 80, utilizado para HTTP, o el puerto 22, utilizado para SSH. Pero en caso de tratarse de otros puertos menos usuales algo de información textual resulta de gran ayuda. Mediante Nmap sólo podemos obtener el nombre que tiene ese puerto (a parte de información sobre su estado, etc.). Para obtener información textual podemos usar otra fuente de información, que casualmente también se ha obtenido de ICANN.

Obtener información de este archivo es ligeramente más complicado, ya que no se trata simplemente de un fichero de texto sino de un fichero XML, con una estructura información más compleja. De todas maneras la descarga del archivo y la gestión de activos se realiza de la misma manera, ya que hemos implementado una clase abstracta que define cómo tenemos que hacerlo. Por otro lado, para la lectura del archivo XML se utiliza otro parser de XML, mucho más sencillo que el que se utiliza para leer la información de Nmap, pero que está implementado de la misma manera.

Todo esto se ha implementado en otra clase (también Singleton) para poder, mediante métodos concretos (como se hacía a la hora de obtener el fabricante), indicarle un puerto y obtener más información textual sobre dicho puerto.

Listing 19: Obtención información del objetivo de un puerto concreto

```
// Añadir el código en este fragmento
```

En este caso se han implementado estos dos sistemas para obtener información desde medios externos, pero debido a la manera en la que están implementados, mediante esa interfaz abstracta, implementar más métodos de obtención de datos de terceras partes resulta bastante sencillo.

Por último mencionar que estos elementos contienen relativamente una gran cantidad de información, pero al ser información en formato texto plano o XML, descargarlos no resulta costoso. Además, por conveniencia, se ha optado por lanzar las descargas de estos archivos en el momento en el que arranca la aplicación. De esta manera los archivos estarán descargados cuando queramos utilizarlos, de una manera completamente transparente al usuario, sin que tenga que hacer absolutamente nada.

Se ha optado por utilizar este modelo (descargar en cada inicio de la aplicación toda la información de medios externos), y no guardarla de manera persistente en nuestra aplicación. Esto es debido principalmente a que descargarla no es una tarea demasiado costosa y podemos tener siempre la información actualizada. Esto requiere que siempre que lancemos la aplicación se tenga conexión a Internet, pero como vamos a requerir de dicha conexión para aprovechar cualquiera de las funcionalidades, esa conexión, de una manera u otra, siempre va a ser necesaria.

11.5. Hilos, Tareas y paralelización

Un aspecto también fundamental para entender el comportamiento de la aplicación es todo lo relacionado con la ejecución de hilos, tareas o procesos. Si bien esta aplicación en no se trata de una aplicación que esté basada en multithreading o programación paralela, se hace uso de diferentes técnicas relacionadas en ciertos puntos de la implementación. En los próximos párrafos se indagará en dichos aspectos.

Por una lado (y esto es un aspecto que ya se ha mencionado previamente), la ejecución de Nmap es completamente independiente a la ejecución de la aplicación. Android, como cualquier sistema operativo basado en Linux, tiene como elementos fundamentales de ejecución los procesos. Esto quiere decir que, en el sistema, tendremos todo tipo de procesos corriendo. Esos procesos pueden corresponder a aplicaciones o ser procesos del propio sistema operativo. En este caso la ejecución del binario de Nmap se lanza en un proceso independiente, uniéndose al resto de procesos que se están ejecutando en ese momento en el dispositivo.

El proceso de Nmap no consiste únicamente en la ejecución del binario. El proceso en sí que se lanza consiste en la ejecución de una *shell*, dentro de la cual se lanzarán los comandos que permitan ejecutar Nmap. Por lo tanto, no es que tengamos un proceso de ejecución, sino que tenemos una shell asociada que tendrá configuradas tanto la entrada como la salida para que se comuniquen con nuestra aplicación. Comunicar la entrada y la salida de un proceso es bastante sencillo, y se realiza de manera similar a como se haría con un archivo, como se muestra en el Código 20.

Listing 20: Creación del proceso, la entrada y salida para Nmap

```
private fun startProcess() {
    if (scanProcess == null || processInputReader == null || processOutputStream
        == null) {
        // Creates a new process that runs a shell
        val processBuilder = ProcessBuilder("sh")
        processBuilder.redirectErrorStream(true)
        scanProcess = processBuilder.start()

        // Creates a couple of streams to redirect the IO
        processOutputStream = DataOutputStream(scanProcess?.outputStream)
        processInputReader =
            BufferedReader(InputStreamReader(scanProcess?.inputStream))
    }
}
```

En este caso concreto lo más importante es la entrada, porque es la forma en la que lanzaremos los comandos de Nmap, y por ende, ejecutar la aplicación. La salida, en este caso, no resulta importante ya que lo que hacemos al ejecutar Nmap es indicar que los resultados se guarden en un fichero XML, por lo tanto aunque tengamos abierta la salida de ese proceso y redirigida a nuestra aplicación, en principio no se hará uso de ella, únicamente para tareas puntuales de depuración.

Otro de los aspectos, que también se ha mencionado ya en otras secciones, es el relacio-

nado con los hilos o Threads de ejecución. Teniendo en cuenta que no se puede ejecutar operaciones de red en el hilo principal de ejecución de la aplicación (al menos por defecto), se han implementado ciertas partes mediante el uso de Threads. Principalmente lo que se ha implementado mediante Threads son las descargas de los diferentes archivos que se utilizan para obtener información de medios externos. Se ha programado mediante Threads de Android (que viene a ser Threads de Java) y no mediante otros mecanismos de ejecución asíncrona de código debido a la simplicidad de la implementación de un Thread y también porque las tareas a realizar no tienen complejidad a nivel de programación. Consisten en descargar y leer archivos de Internet, lo que, como se puede comprobar en el Código 21, se puede hacer con unas pocas líneas de código.

Listing 21: Ejecución de la descarga y lectura de un archivo en un Thread

```
override val downloadThread = Thread(Runnable {
    val url = URL(urlString)
    val urlConnection = url.openConnection()
    val bufferedReader =
        BufferedReader(InputStreamReader(urlConnection.getInputStream()))
    val urlDataText = bufferedReader.readLines()
    val commentsRemoved = urlDataText.filter { it.isNotEmpty() && it[0] != '#' }
    for (line in commentsRemoved) {
        val data: List<String> = line.split('\t')
        if (data.size == 3) {
            vendorList.add(Vendor(data[0], data[1], data[2]))
        }
    }
    bufferedReader.close()
    downloaded = true
})
```

El tercer y último bloque dentro de toda la parte de ejecución asíncrona y paralelización de código es el usado a la hora de actualizar la interfaz gráfica. Android, al igual que muchísimos sistemas, utiliza el patrón de diseño MVC. Esto tiene una serie de ventajas, permitiendo separar la presentación o vista de la aplicación de la lógica o modelo. Con ello se logra separar la vista del resto, permitiendo poder modificarla en todo momento de manera independiente.

La vista se comunica con el modelo a la hora de obtener datos mediante el uso de controladores. La forma más sencilla de cargar nuestra vista con los datos que contiene el modelo es obtener los datos del modelo interactuando con el controlador cuando se realiza la creación de la vista.

Esto en ciertas partes de la aplicación es viable, como por ejemplo en la parte en la que mostramos la lista de escaneos realizados o la parte en la que visualizamos los diferentes nodos de un escaneo concreto. En ambos casos se da la situación en la que lo que hacemos es, a la hora de cargar el Activity o el Fragment, Usar el controlador para comunicarnos con el modelo y obtener los datos para mostrarlos desde un principio.

Pero hay casos en los que esto no es posible. El caso más importante a tener en cuenta es cuando se está realizando el escaneo de una red. Al realizar el escaneo de una red vamos

a tener a Nmap ejecutándose y iremos obteniendo la información que vaya sacando. Para implementar esto se pueden hacer varias aproximaciones.

La primera sería lanzar la ejecución de Nmap, realizar el escaneo completo de la red (esperando mientras se realiza) y obtener la información que ha generado. El principal inconveniente de esta aproximación consiste en que no tendríamos ningún tipo de salida hasta que no terminase el escaneo completo. Esto quiere decir que no se visualizará ningún cambio en la Activity hasta finalizar. Claramente es negativo, ya que dejaríamos al usuario sin información durante un tiempo de espera muy grande.

Otra aproximación, mejor que la primera, sería ir obteniendo partes de la información e ir mostrándola en pantalla. Para realizar esto necesitamos lanzar una tarea que se ejecute de manera asíncrona y vaya realizando los escaneos por partes. Perdemos esa sencillez que nos da el hecho de escanear toda la red mediante una única ejecución de Nmap, pero ganamos la flexibilidad de poder ir obteniendo la información poco a poco.

Para ello se hará uso de un *AsyncTask*⁹ de Android. Un *AsyncTask* es un método para ejecutar una tarea de manera asíncrona en Android. Para implementar una tarea asíncrona, debemos heredar de esta clase e implementar las diferentes funciones que pide implementar. Un ejemplo vacío de la clase se muestra en el Código 22.

Listing 22: Ejemplo de la estructura de una *AsyncTask*

```
class MyAsyncTask() : AsyncTask<Params, Progress, Result>() {
    override fun onPreExecute() { /* ... */ }

    override fun doInBackground(vararg params: Param) { /* ... */ }

    override fun onProgressUpdate(progress: Progress) { /* ... */ }

    override fun onPostExecute(result: Result) { /* ... */ }
}
```

Las funciones que requiere implementar son cuatro, y se enumeran a continuación:

1. `onPreExecute()`: Se invoca antes de ejecutarse. Normalmente se usa para inicializar valores o preparar lo necesario para la ejecución.
2. `doInBackground(Params...)`: Se invoca ya en un Thread aparte y es donde se realiza el contenido de la tarea. Dentro de ella, se llama a la función `publishProgress(Progress...)` en caso de que se quiera indicar algún tipo de progreso.
3. `onProgressUpdate(Progress...)`: Se ejecuta cuando se ha recibido algún tipo de progreso. Normalmente se usa para actualizar la interfaz gráfica e indicar al usuario que se han realizado progresos.
4. `onPostExecute(Result)`: Se ejecuta una vez la tarea ha terminado, y permite interactuar con el resultado

⁹<https://developer.android.com/reference/android/os/AsyncTask>

La implementación de una AsyncTask permite que podamos usar el tipo de datos que queramos tanto para Params, Progress o Result, o ninguno en caso de no necesitar algunos de ellos.

Para el caso concreto de la aplicación, se han implementado varias tareas asíncronas. La primera de ellas, NetworkScan (mostrada en parte en el Código 23, sirve como base para definir diferentes tipos de escaneos, y en ella se implementa funcionalidad común para cualquier tipo de escaneo. Esta clase, como se muestra en CÓDIGO es abstracta y no puede ser instanciada.

Listing 23: Implementación de métodos de una AsyncTask en la clase NetworkScan

```
abstract class NetworkScan(protected val db: AppDatabase, private val
    wifiManager: WifiManager)
    : AsyncTask<Unit, Node, Unit>() {

    // .....

    override fun onPreExecute() {
        val ip = NmapRunner.intToIp(wifiManager.connectionInfo.ipAddress)
        val netmask = NmapRunner.intToIp(wifiManager.dhcpInfo.netmask)

        scanName = wifiManager.connectionInfo.ssid.trim('\'')
        addresses = SubnetUtils(ip, netmask).info.allAddresses

        scanStartTimestamp = System.nanoTime()
    }

    override fun onProgressUpdate(vararg values: Node?) {
        if (emptyScan) {
            db.scanDao().insertScan(Scan(scanName))
            scanId = db.scanDao().lastInsertedId()
            emptyScan = false
        }

        for (scan in values)
            if (scan != null)
                hostsUp++
    }

    override fun onPostExecute(result: Unit?) {
        val scanTimeInSeconds = ((System.nanoTime() - scanStartTimestamp!!) /
            Math.pow(10.0, 9.0)).toFloat()
        db.scanStatsDao().insertScanStats(ScanStats(scanId, addresses.size,
            hostsUp, addresses.size - hostsUp, scanTimeInSeconds))
    }

    // .....
}
```

Basandonos en NetworkScan, la implementación más sencilla de un escaneo de una red está programada en la clase SequentialNetworkScan. En esa clase, como se puede observar en el Código 24, ya se implementan las llamadas a Nmap y el guardado en la base de datos. El nombre de la clase viene dado porque realiza el escaneo de manera secuencial,

es decir, uno por uno.

Listing 24: Implementación de parte de la tarea asíncrona SequentialNetworkScan

```
abstract class SequentialNetworkScan(db: AppDatabase, wifiManager: WifiManager) : NetworkScan(db,
    ↪ wifiManager) {

    override fun doInBackground(vararg __nothing: Unit) {
        val nmapRunner = NmapRunner(ScanType.REGULAR)
        for (address in addresses) {
            val inetAddress = InetAddress.getByName(address)
            val reachable = inetAddress.isReachable(pingTimeout)

            if (reachable) {
                val singleHostScan = nmapRunner.runScan(listOf(address))

                if (!isCancelled) nmapRunner.scanProcess?.waitFor()

                if (singleHostScan != null
                    && singleHostScan.hosts.isNotEmpty()
                    && singleHostScan.hosts[0].status.state == HostStates.UP) {

                    val ip = singleHostScan.hosts[0].address.address
                    val mac: String = getMacAddress(ip) ?: ""
                    val name =
                        if (singleHostScan.hosts[0].hostNames.isNotEmpty())
                            singleHostScan.hosts[0].hostNames[0].name
                        else {
                            if (!OUIs.downloaded)
                                OUIs.waitForDownload()

                            if (OUIs.checkVendorFromMac(mac) == "")
                                ip
                            else
                                OUIs.checkVendorFromMac(mac)
                        }
                    val vendor: String =
                        if (!OUIs.downloaded)
                            OUIs.checkVendorFromMac(mac)
                        else
                            ""

                    val timeElapsed: Float =
                        if (singleHostScan.runStats != null)
                            singleHostScan.runStats.timeElapsed
                        else
                            -1.0f
                    val scanId = db.scanDao().lastInsertedId()

                    currentNode = Node(name, ip, mac, vendor, timeElapsed, scanId)
                    db.nodeDao().insertNode(currentNode as Node)

                    for (nmapPort in singleHostScan.hosts[0].ports) {
                        val port = Port(nmapPort.id,
                            db.nodeDao().lastInsertedId(),
                            nmapPort.type,
                            nmapPort.service,
                            nmapPort.state.state,
                            nmapPort.state.reason)
                        db.portDao().insertPort(port)
                    }

                    publishProgress(currentNode)

                } else {
                    currentNode = null
                }
            }
            triedHosts++
            updateUi()
        }
    }
}
```

También se puede observar que esta clase, al igual que la clase base, también es abstracta, ya que consiste en una implementación abstracta, únicamente con la lógica del escaneo de Nmap. Para poder realizar un escaneo, debemos implementar una subclase de ésta dentro de

la vista (ya sea Activity o Fragment) en la que queramos mostrar los datos. De esta manera, las particularidades de la vista y de cómo se cargan los datos en ella quedan relegados a código dentro de la vista, manteniendo el uso del patrón MVC. La implementación interna en una Activity quedaría como se muestra en el 25, y, como se puede apreciar, solo contiene código relacionado con la interfaz de usuario.

Listing 25: Implementación de un NetworkScan dentro de un Activity

```

class NetworkScanActivity : AppCompatActivity(),
    NodeListFragment.OnListFragmentInteractionListener {

    // .....

    override fun onResume() {
        super.onResume()

        if (wifiManager.wifiState == WifiManager.WIFI_STATE_ENABLED
            && wifiManager.connectionInfo.suplicantState ==
            ↪ SuplicantState.COMPLETED) {
            if (networkScanTask.status == AsyncTask.Status.PENDING)
                networkScanTask.execute()
        }

        // .....
    }

    // .....

    inner class NetworkScan : SequentialNetworkScan(db, wifiManager) {

        override fun onPreExecute() {
            super.onPreExecute()

            network_scan_progress_bar.max = addresses.size
            network_scan_progress_bar.progress = 0
            network_scan_progress_bar.visibility = View.VISIBLE
            setTitle(R.string.scanning)
        }

        override fun onProgressUpdate(vararg values: Node?) {
            super.onProgressUpdate(*values)

            if (currentNode is Node)
                nodeListFragment.addNode(currentNode as Node)
        }

        override fun onPostExecute(result: Unit?) {
            super.onPostExecute(result)

            network_scan_progress_bar.visibility = View.GONE
            setTitle(R.string.scanned)
            ended = true
        }

        override fun updateUi() {
            network_scan_progress_bar.progress = triedHosts
        }
    }
}

```

Para lanzar una tarea asíncrona simplemente debemos, una vez construido el objeto, llamar al un método llamado `Execute()`. Ejecución que solo se podrá realizar una vez para cada instancia, debido a las características internas de un `AsyncTask`, para evitar que una tarea ya realizada se vuelva a realizar.

En conclusión, se ha desarrollado un modelo que permite, por una parte, separar la lógica de la interfaz gráfica y, por otra parte, disponer de una jerarquía extensible que permite realizar diferentes tipos de escaneos. Una jerarquía de clases que se pueden extender para realizar escaneos mediante otros métodos, como pueden ser ejecuciones en paralelo u otro tipo de implementaciones.

11.6. Interfaz gráfica

Ya se ha comentado en el capítulo anterior aspectos como las diferentes partes de la interfaz gráfica, la navegación entre ellas o los diferentes aspectos de Material Design implementados en el diseño de la interfaz.

En esta parte no se pretende volver a repetir conceptos ya mencionados dicho capítulo, sino explicar cómo se han implementado conceptos sobre la interfaz gráfica, principalmente cómo interactúan las diferentes Activities con el sistema para obtener la información o cómo se realizan las transiciones entre unas y otras. También explicar cómo está implementado el patrón de diseño MVC, que Android permite integrar de una manera relativamente sencilla.

11.6.1. Ciclo de vida de Activities y Fragments

Ya se ha mencionado anteriormente lo que es un Activity lo que es un Fragment. A esas definiciones dadas cabe añadir que cada uno de ellos tiene un ciclo de vida concreto. Es decir, hay una serie de funciones, que se pueden implementar, que se ejecutan en cierto orden. Existen funciones que se ejecutan antes de mostrar el Activity o Fragment, otras al pausar ejecución del elemento u otras a la hora detener la ejecución, por poner unos ejemplos.

Ambos ciclos de vida son similares, pero tienen particularidades concretas. Resulta necesario entender ese ciclo de vida, para entender cómo se han implementado ciertos aspectos de la aplicación.

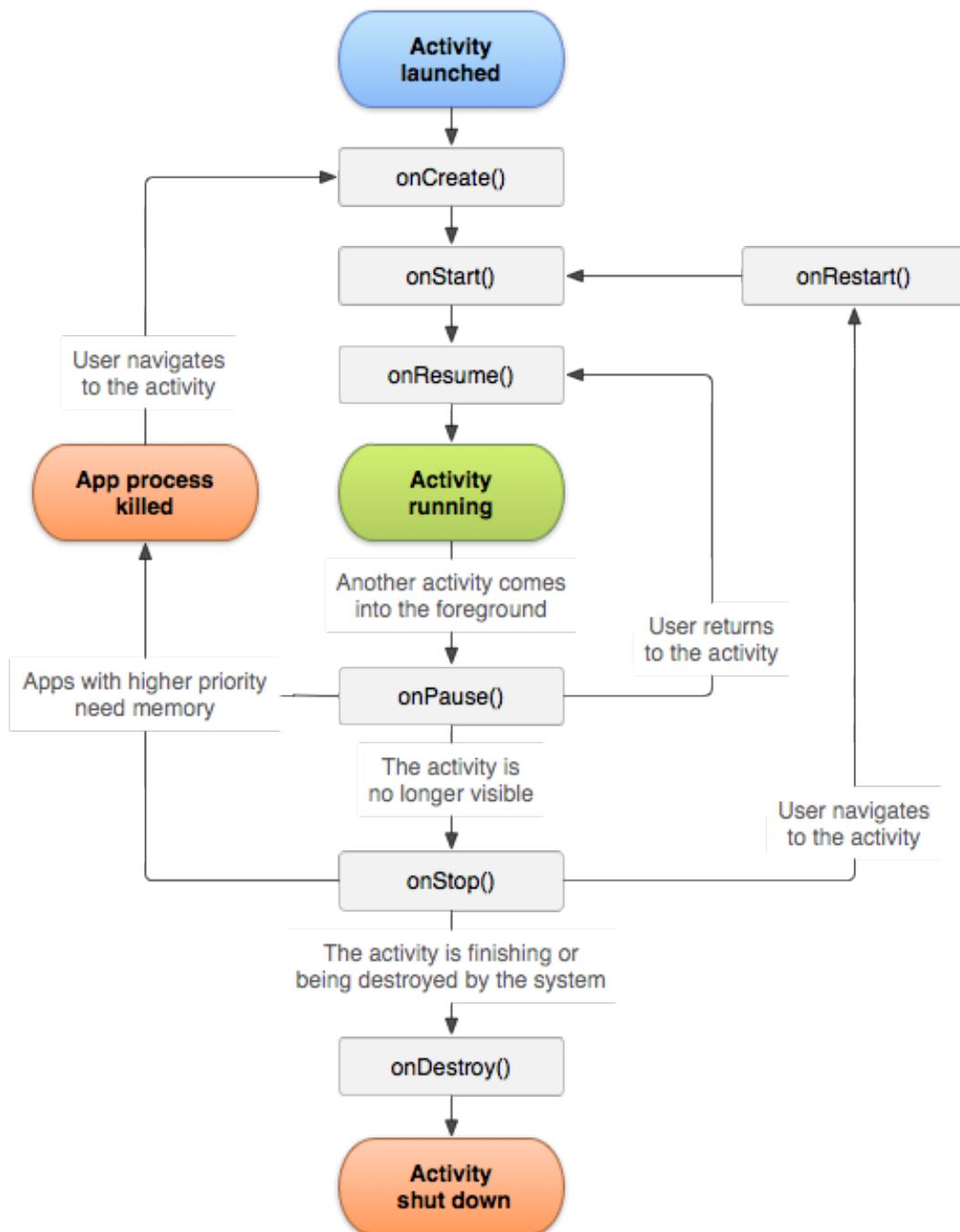


Figura 11.7.: Ciclo de vida de un Activity de Android

Dentro del ciclo de vida de un Activity (ver Figura 11.7), las funciones más importantes son `onCreate()`, `onResume()` y `onPause()`. En la primera es donde se inicializan una

serie de elementos para utilizar en nuestra Activity. En la segunda es donde se inicializan los elementos puntuales que no están disponibles todavía en la función `onCreate()`, como ciertos elementos de la vista. La última de las tres nos permitirá guardar ciertos valores cuando, por ciertos motivos, la ejecución del Activity se pause. La ejecución se puede pausar por diversos motivos, como que el usuario haya salido de la aplicación, haya surgido un evento que haya lanzado otra Activity por encima, etc.

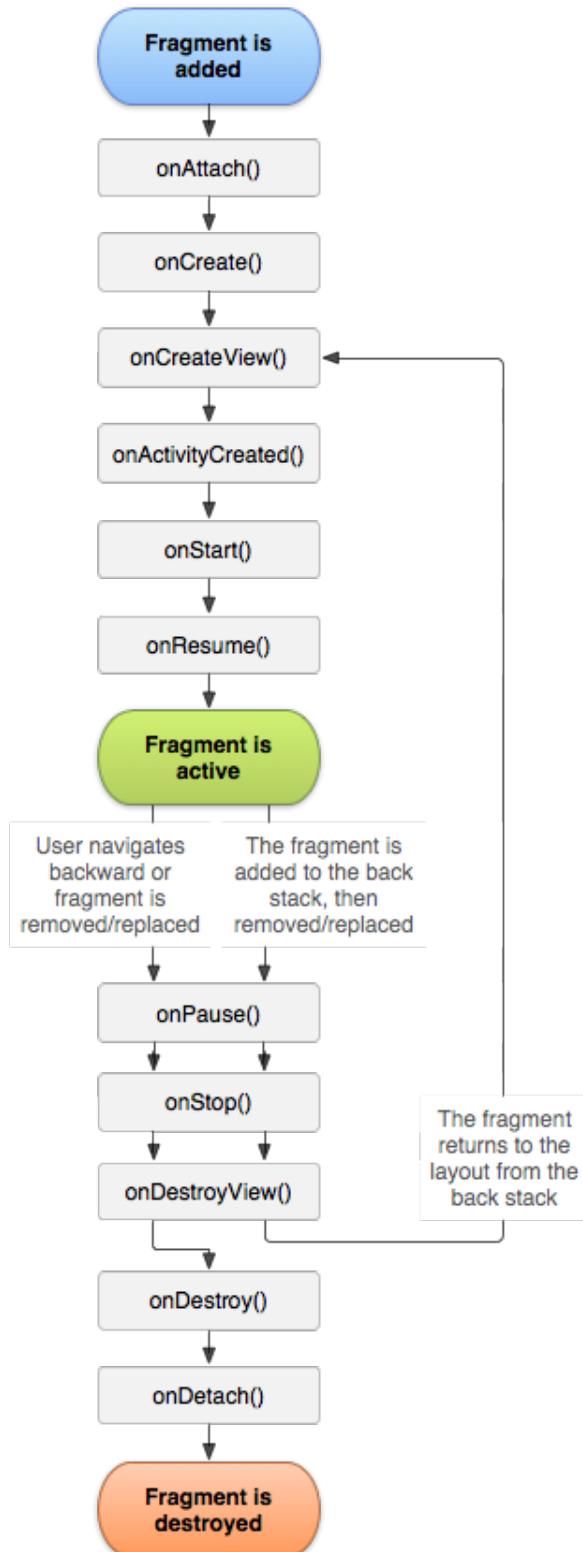


Figura 11.8.: Ciclo de vida de un Fragment de Android

El ciclo de vida de un Fragment (Figura 11.8) es parecido al ciclo de vida de una Activity, pero con ciertas particularidades. La principal, y la que más nos interesa, es la diferenciación entre `onCreate()` y `onCreateView()`. Una de las diferencias que existe, de manera similar a como ocurre con los Activities, en la primera función no tenemos acceso a elementos de la interfaz gráfica, mientras que en la segunda función sí. Por otro lado, debemos tener en cuenta que cuando se crea el Activity, se llama la función `onCreate()`, pero no se llama la función `onCreateView()` hasta que se muestra dicho Fragment. De la misma manera, cuando dejamos de mostrar un Fragment y volvemos a mostrarlo no se vuelve a llamar a la función `onCreate()`, pero sí a `onCreateView()`.

11.6.2. Navegación

Cómo se ha comentado, la aplicación arranca desde el Activity llamado `MainActivity`. Dentro de la clase del Activity, se encuentra una serie funcionalidades. En concreto, para poder cambiar entre los diferentes Fragments, lo que se hace es incrementar la función `onNavigationItemSelected()`. Esta función que recibe el ítem que se ha pulsado, y en función de ese ítem sustituimos un Fragment por otro, cómo se puede observar en el Código 26.

Listing 26: Selección de un Fragment u otro en función del valor seleccionado en el Navbar

```
override fun onNavigationItemSelected(item: MenuItem): Boolean {
    val transaction = supportFragmentManager.beginTransaction()
    netscan_fab.visibility = View.GONE
    when (item.itemId) {
        R.id.nav_my_scans -> {
            transaction.replace(R.id.content_main, scanListFragment).commit()
            netscan_fab.visibility = View.VISIBLE
        }
        R.id.nav_scan_direction -> {
            transaction.replace(R.id.content_main,
                scanDirectionFragment).commit()
        }
        R.id.nav_share -> {
            transaction.replace(R.id.content_main, shareFragment).commit()
        }
        R.id.nav_about -> {
            transaction.replace(R.id.content_main, aboutFragment).commit()
        }
    }
    drawer_layout.closeDrawer(GravityCompat.START)
    item.isChecked = true
    return true
}
```

Para lograr esto, se guarda una referencia cada uno de los Fragments dentro de la propia Activity, referencia que se inicializa al comienzo de la ejecución, cómo se puede observar en el Código 27.

Listing 27: Referencias a cada uno de los Fragments definidas dentro del MainActivity

```

class MainActivity : AppCompatActivity(),
    NavigationView.OnNavigationItemSelectedListener,
    ShareFragment.OnFragmentInteractionListener,
    ScanListFragment.OnListFragmentInteractionListener,
    ScanDirectionFragment.OnFragmentInteractionListener,
    AboutFragment.OnFragmentInteractionListener {

    private var scanListFragment = ScanListFragment()
    private val scanDirectionFragment = ScanDirectionFragment()
    private val shareFragment = ShareFragment()
    private val aboutFragment = AboutFragment()

    // .....
}

```

Algunos de estos Fragments permiten lanzar otras Activities. Para ello se detecta la pulsación de un evento concreto para lanzar una Activity u otra. En el Código 28 se muestra cómo se implementa el evento que detecta la pulsación del FAB y abre el Activity que permite realizar el escaneo de una red.

Listing 28: Evento para lanzar una nueva Activity para escanear una red

```

netscan_fab.setOnClickListener {
    startActivity(Intent(this, NetworkScanActivity::class.java))
}

```

Lanzar un Activity es tan sencillo como invocar la función `startActivity()`. Esta función recibirá un objeto de tipo `Intent`. Este tipo de objetos son los que definen tanto qué Activity se va a lanzar, como los diversos parámetros con los que se va a lanzar dicha Activity. De esta manera, antes de ejecutarla podremos pasar de ciertos parámetros para que a la hora de cargar el Activity se cargue de una u otra manera en función de esos parámetros.

Según de qué Activity se trate, éste necesitará o no ciertos parámetros. Por poner un ejemplo, en el caso de un `NodeListActivity`, se mostrará una lista de nodos de un escaneo concreto. Por lo tanto, a la hora de ejecutar el Activity, necesitaremos recibir de qué escaneo se trata para obtener esos datos concretos.

Listing 29: Creación del NodeListActivity y recepción de parámetros

```

class NodeListActivity : AppCompatActivity(),
    NodeListFragment.OnListFragmentInteractionListener {

    // .....

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_node_list)
    }
}

```

```

        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        if (intent.extras != null) {
            scanId = intent.extras.getInt("scan_id")
            scanName = intent.extras.getString("scan_name")

            getPreferences(Context.MODE_PRIVATE).edit().putInt("scan_id",
                → scanId).apply()
            getPreferences(Context.MODE_PRIVATE).edit().putString("scan_name",
                → scanName).apply()
        }

        title = scanName
        nodeListFragment = NodeListFragment.newInstance(scanId, scanName)
        supportFragmentManager.beginTransaction().add(R.id.content_node_list,
            → nodeListFragment).commit()
    }

    //.....
}

```

Como se puede ver en el Código 29, cuando se crea el Activity se evalúan los parámetros recibidos por el Intent, de tal manera que podamos cargar el Fragment que nos permite mostrar la lista de nodos pasándole como parámetro el ID y el nombre del escaneo en particular.

El Fragment recibirá la información y, en función de ella, cargará a unos datos. En el Código 30 se puede apreciar que existen dos funciones diferentes, la función `onCreate()` y la función `onCreateView()`. La primera se llama antes que la segunda, siguiendo el flujo de ejecución de un Fragment. En la primera recibimos los argumentos y en la segunda obtenemos los datos que queremos montar.

Listing 30: Creación del NodeListFragment y uso del RecyclerView

```

class NodeListFragment : Fragment() {

    //.....

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        if (arguments != null) {
            scanId = (arguments as Bundle).getInt(ARG_SCAN_ID)
            scanName = (arguments as Bundle).getString(ARG_SCAN_NAME)
        }
    }

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?): View? {
        val view = inflater.inflate(R.layout.fragment_node_list, container,
            → false)

        // Set the adapter
        if (view is RecyclerView) {
            val db: AppDatabase = AppDatabase.getInstance(view.context)
            val nodes =
                if (scanId != null)

```

```

        db.nodeDao().getNodesFromScan(scanId!!)
    else
        listOf()
nodeRecyclerViewAdapter =
    MyNodeRecyclerViewAdapter(nodes.toMutableList(), listener)
view.adapter = nodeRecyclerViewAdapter
}

return view
}

//.....
}

```

En dicho código se usa un elemento llamado `RecyclerView`. Este elemento es lo que dentro del patrón MVC denominamos como controlador. En este caso utilizamos el correspondiente a un nodo, llamado `MyNodeRecyclerViewAdapter`. Esto nos permite, mediante los datos que hemos obtenido en las líneas anteriores (usando el `nodeDao()`), cargar el adaptador que vuelca esos datos en elementos de la interfaz. Así, con unas pocas líneas de código, se recogen una serie de datos y se trasladan a la interfaz de una manera sencilla y automática.

Examinando el código de `MyNodeRecyclerViewAdapter` (Código 31), vemos que tenemos variables que almacenan los datos y una serie de funciones que simplemente en las otras dos elementos concretos de la interfaz gráfica, como pueden ser `TextViews` en este caso.

Listing 31: Definición de un `RecyclerView` para usar con una lista de nodos

```

class MyNodeRecyclerViewAdapter(private val values: MutableList<Node>,
                               private val listener:
                                   → OnListFragmentInteractionListener?) :
    RecyclerView.Adapter<MyNodeRecyclerViewAdapter.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder
    ← {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.fragment_node, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.item = values[position]
        holder.nodeNameView.text = values[position].name
        val ipSplitted = values[position].ip.split(".")
        holder.nodeIpView.text = ipSplitted[0] + "." + ipSplitted[1]
        holder.nodeIpFormattedView.text = " ." + ipSplitted[2] + " ." +
            ipSplitted[3]
        holder.nodeMacView.text = values[position].mac

        holder.view.setOnClickListener {
            listener?.onListFragmentInteraction(holder.item as Node)
        }
    }

    override fun getItemCount(): Int {
        return values.size
    }
}

```

```
    }

    fun addItem(node: Node) {
        values.add(node)
        notifyDataSetChanged()
    }

    inner class ViewHolder(val view: View) : RecyclerView.ViewHolder(view) {
        val nodeNameView: TextView = view.findViewById<View>(R.id.node_name) as
        ↵ TextView
        val nodeIpView: TextView = view.findViewById<View>(R.id.node_ip) as
        ↵ TextView
        val nodeIpFormattedView: TextView =
        ↵ view.findViewById<View>(R.id.node_ip_formatted) as TextView
        val nodeMacView: TextView = view.findViewById<View>(R.id.node_mac) as
        ↵ TextView
        var item: Node? = null
    }
}
```

De esta manera queda implementada el patrón MVC. Para resumir, tenemos las diferentes vistas implementadas en Activities o Fragments. Los motivos entre la elección de una u otra ya se ha comentado en el capítulo anterior. Por otra parte de los elementos del controlador están implementados en los que se denominan RecyclerView, que como se ha podido apreciar, son clases que simplemente contienen los datos y los enlazan a ciertos campos del layout o interfaz gráfica de un Activity o Fragment. Por último, el modelo que se usa y ya se ha mencionado, consiste en una serie de clases que interactúan con la base de datos mediante el uso de Room, cómo se ha explicado anteriormente.

IV

Análisis y conclusiones del Trabajo

12

Conclusiones

El objetivo principal de este proyecto, como se ha ido mencionando, es doble. Por una parte se busca analizar el área de la seguridad informática para ver su estado actual, desarrollo, y como cumple los objetivos para los que el campo ha sido desarrollado. Todo ese análisis ha quedado volcado en lo que se ha denominado *Fase 1*. El análisis cubre de manera general el campo de la seguridad y va concretándose hasta una parte en concreto, que es la del Pen-testing, una de las áreas más interesantes dentro de la seguridad informática en general y de la seguridad defensiva en particular.

En base a lo obtenido en dicha fase, el proyecto se ha centrado en ofrecer una solución a un problema más concreto. Solución que ha sido desarrollada en lo que se ha llamado *Fase 2*. El objetivo principal de este proyecto no es la aplicación que ha sido desarrollada, sino todo el análisis del campo, las conclusiones, las pautas, razonamientos y toma de decisiones seguidas para intentar dar solución a un problema concreto.

En concreto, mostrar cómo se puede, tras analizar las problemáticas que pueden surgir en un campo, dotar de soluciones concretas que satisfagan las necesidades de los usuarios. En este área en particular, una de las mayores debilidades que se pueden apreciar es que, de alguna u otra manera, el propio usuario queda relegado a un papel secundario a la hora de garantizar su seguridad.

Está claro qué es un usuario no tiene los conocimientos necesarios como para actuar como un experto en seguridad, pero esto no quiere decir que tenga que jugar un papel puramente pasivo. De la misma manera que tenemos sistemas de apoyo a la decisión que no deciden por nosotros, sino que en base a herramientas informáticas desarrolladas por expertos, nos permiten tomar decisiones en diferentes campos (como puede ser, por ejemplo, el mercado bursátil), la idea con este proyecto es enseñar cómo se puede, mediante soluciones de software, dotar al usuario de información más clara y mejor estructurada. Información con la que pueda ser consciente de la seguridad de sus sistemas y actuar en consecuencia.

La aplicación desarrollada no pretende ser una solución final, sino más bien una prueba de concepto que permita entender los pasos dados durante el desarrollo para poder alcanzar ese

tipo de soluciones finales. Si bien la aplicación en sí forma un conjunto cerrado y funcional y, a su vez, abierto y escalable a nueva funcionalidad. Algo fundamental, ya que de esta manera sirve como punto de partida para extenderse a ámbitos de actuación más concretos. Además, puede ser ampliada con más funcionalidad en pro de solucionar más problemas o dotar de mayor información al usuario.

12.1. Problemas durante el desarrollo

Cabe destacar que todos los objetivos concretos desarrollo o hitos de programación no han sido realmente resueltos. Esto ha sido debido a dos problemas encontrados durante el desarrollo, que han actuado como factores limitantes para poder alcanzar los objetivos

El mayor problema a la hora de desarrollar la aplicación del proyecto ha residido en las limitaciones a la hora de el uso de Nmap. Si bien se ha logrado integrar de una manera clara sencilla y fácil Nmap (solventando el hecho de que no se trata de una librería sino de un software más complejo) y utilizarlo, exprimir toda su funcionalidad no ha sido posible. El principal problema a la hora de usar ciertas características de Nmap es que se requiere privilegios de superusuario. Esto genera un conflicto ineludible entre los dos objetivos de la aplicación. Entre obtener la mayor cantidad de información posible y generar una aplicación que puede utilizar cualquier usuario.

En concreto, a la hora de obtener información más como las versiones de sistemas operativos de diferentes nodos o la dirección MAC del propio dispositivos mediante el uso de Nmap, requerimos de esa elevación de privilegios. La única forma de considerar elevación de privilegios ejecutar la aplicación en un sistema con *root* habilitado.

Obviamente, si tenemos en cuenta que el *target* de la aplicación son usuarios estándar sin conocimientos de sistemas informáticos, de programación o de seguridad, llegamos a la conclusión de que no van a tener sus dispositivos con este *root* habilitado. Esto al final limita el uso de Nmap a la hora de obtener información. Aún así Nmap sigue siendo la única herramienta que nos hubiera permitido obtener esta información de manera sencilla.

El simple hecho de desarrollar funcionalidad para obtener información como versiones de sistemas operativos u otro tipo de información concreta, se escapa completamente del alcance de este Trabajo de Fin de Grado, ya que su dificultad técnica resultan relativamente alta, haciendo del trabajo algo mucho más extenso. Por esto mismo este tipo de herramienta sólo están disponibles en software tan maduro y desarrollado como Nmap.

El segundo problema que ha surgido a la hora de desarrollar la aplicación ha sido el tema del rendimiento. Obtener información de una red requiere cierto tiempo.

Una de las dificultades reside en que necesitamos un tiempo mínimo para analizar cada nodo de esa red. Si a la hora de escanear un nodo usamos un *ping* a una dirección concreta para ver si está activa o no, este ping puede ser exitoso no en función del tiempo que dedicemos. Si se dedican, por poner un ejemplo, 50 milisegundos a ese ping puede que, debido al corto tiempo dedicado, obtengamos un falso negativo. En cambio si dedicamos un tiem-

po mayor, como de 300 milisegundos para ese ping, evitaremos esos falsos negativos, pero haremos con nuestra aplicación tarde más en escanear un host concreto.

Esto, al hablar de escanear un único nodo no parece tener relevancia, ya que estamos hablando del orden de milisegundos. Pero esto hay que sumar que solo se trata del paso inicial lanzar un escaneo. El escaneo completo con Nmap implica lanzar un proceso, ejecutar un binario, obtener una salida y leer esa salida. Aunque Nmap no es una aplicación que requiere un sistema de alto rendimiento, todo el proceso que se realiza para escanear un host requiere su tiempo de computación. Sobre todo hay que tener en cuenta para los casos en los que no estemos examinando uno u dos nodos, sino una gran cantidad de ellos, como es el caso de una red informática.

Una red estándar, como puede ser una red doméstica, normalmente suele tener una máscara de red 255.255.255.0. Obviamente esta máscara de red que indica es que tenemos 254 posibles direcciones para los nodos de esa red. A priori, uno puede asumir que las direcciones que va a repartir el router de esa red a los diferentes nodos estén entre los valores más bajos, pero esto no es algo que esté garantizado. Si queremos ver todos los dispositivos conectados en la red debemos analizar todas las posibles direcciones que puede tener esa red.

Ya solo en ese caso concreto, algo sencillo en lo que no tendremos una gran cantidad de dispositivos, ya tenemos que analizar 254 direcciones. Si antes hablábamos de un pin de 300 milisegundos, a lo cual le podemos sumar otros tantos milisegundos para ejecutar todas las operaciones relacionadas con Nmap, podemos acercarnos al segundo en la ejecución. Un segundo no parece mucho, pero cuando hablamos de 254 veces esa operación entonces hablamos de un tiempo importante.

Esto se complica mucho más para redes complejas como redes de universidades, redes en un entorno laboral o redes públicas grandes. Ya no solo es que las configuraciones concretas de estas redes puede hacer que Nmap no detecte toda la información correctamente, algo difícil controlar porque no podemos conocer el funcionamiento interno la como toda la configuración de una red a simple vista y modificar nuestros comandos para que ejecute de la mejor manera para cada red concreta. Por otra parte probablemente tengamos un número mayor de direcciones posibles. Esto hace que ejecutar un escaneo de esta red sea muchísimo más costoso a nivel de tiempo.

12.2. Trabajo futuro

La aplicación no resulta más que una prueba de concepto, una prueba de concepto de cómo se debe implementar una solución hacia un problema. La gran ventaja de esta prueba de concepto es que es altamente escalable. A la hora de su desarrollo han surgido varios problemas. Uno de los vectores de trabajo futuro puede ser examinar estudiar e implementar soluciones a estos problemas concretos.

Para el problema de Nmap, se pueden buscar soluciones basadas en otras librerías para implementar las soluciones concretas que permitan obtener esos pedazos concretos de infor-

mación. En la aplicación desarrollada ya se ha visto como a la hora de obtener por ejemplo la dirección MAC, se usa la información del protocolo de enrutamiento ARP, al haber sido incapaces de obtener la dirección mediante Nmap.

También se han tenido más información de medios externos, como por ejemplo el fabricante de ese dispositivo, usando esa dirección de hardware y obteniendo en base a ella el fabricante desde una información de confianza, como es la información de la IANA. Algo parecido se ha hecho también, para adquirir información sobre los diferentes puertos y diferentes protocolos. Por ello, otro de los vectores de trabajo puede ser el encontrar fuentes externas que nos permitan ampliar la información mediante parámetros que ya hayamos obtenido nuestra aplicación.

Apéndices

13

Código fuente y documentación del proyecto

En todo momento se ha desarrollado este proyecto para que sirviera como base para cualquier usuario para entender aspectos sobre Seguridad Informática, además de para mostrar cómo dotar de soluciones a problemas concretos. Por otro lado, la aplicación obtenida se ha desarrollado teniendo en cuenta que fuera lo más escalable posible, tanto a nivel de funcionalidad, como en el apartado gráfico. La aplicación pretende una solución a un problema concreto, pero también pretende ser extensible a otros ámbitos, además de dar la capacidad de poder ser mejorada.

Se pretende que la aplicación sea accesible por cualquiera. Debido a esto, la aplicación está disponible como software libre, bajo una licencia abierta, que permite que cualquier usuario pueda tanto analizar el código y entender cómo funciona como disponer del código para, en base a él, desarrollar sus propias soluciones.

Existen muchos motivos para apostar por el software libre. Está claro que todo el desarrollo de la informática sería completamente diferente si todo el software del que disponemos fuera privativo. El software libre ha sido una rama fundamental a la hora de que se desarrollen los sistemas y las tecnologías de la información. A nivel tecnológico en general, y informático en particular, viviríamos en un mundo mucho menos desarrollado tecnológicamente si no fuera por todos esos programadores, desarrolladores, ingenieros, científicos y organizaciones sin ánimo de lucro que desarrollan tanto software libre como conocimiento abierto. Software y conocimiento que permite que la informática sea accesible a todos, sin que esté en manos de unos pocos adinerados.

Tal y como dice Richard Stallman¹, fundador del proyecto GNU y de la Free Software Foundation, *las obras de conocimiento deben ser libres*. Todo lo aprendido durante este proyecto y la aplicación desarrollada puede ser de gran ayuda para usuarios y desarrolladores que simplemente quieran aprender o quieran basarse en este trabajo para sus propios desarrollos. Por ello, tanto este mismo informe como el código íntegro de la aplicación desarrollada serán liberados, una vez presentado este Trabajo de Fin de Grado, con licencias abiertas.

¹<https://www.20minutos.es/noticia/199929/0/Stallman/Premio/softwarelibre/>

En el caso del documento, y como ya se añade en el propio documento, se liberará mediante la GNU Free Documentation License. En el caso del código se liberará mediante la licencia GPLv3.

El código de la aplicación se encuentra disponible en el siguiente enlace:

<https://github.com/ander94lakx/NetScan>

Esta misma memoria se encuentra disponible en el siguiente enlace:

https://github.com/ander94lakx/TFG_Doc

Bibliografía

- [1] Oficina de Seguridad del Internauta. 2017. URL: <https://www.osi.es/es/herramientas-gratuitas>.
- [2] Inc GitHub. 2017. URL: <https://github.com/showcases/security>.
- [3] El Mundo. *Naciones Unidas declara el acceso a Internet como un derecho humano*. Jun. de 2011. URL: <http://www.elmundo.es/elmundo/2011/06/09/navegante/1307619252.html>.
- [4] Microsoft. 2017. URL: https://www.microsoftstore.com/store/mseea/es_ES/pdp/Windows-10-Home/productID.320437800.
- [5] Microsoft. 2017. URL: https://www.microsoftstore.com/store/mseea/es_ES/pdp/Project-Profesional-2016/productID.324452600.
- [6] Critical Tools. 2017. URL: <https://store.criticaltools.com/>.
- [7] Gene Spafford. *Computer Recreations: Of Worms, Viruses and Core War*. Mar. de 1989. URL: <http://spaf.cerias.purdue.edu/quotes.html>.
- [8] Instituto Nacional de Ciberseguridad INCIBE. *Implantación de un SGSI en la empresa*. URL: https://www.incibe.es/extfrontinteco/img/File/intecocert/sgsi/img/Guia_apoyo_SGSI.pdf.
- [9] International Organization for Standardization ISO. *ISO 7498-2:1989. Information processing systems - Open Systems Interconnection - Basic Reference Model*. 2002. URL: <https://www.iso.org/standard/14256.html>.
- [10] International Organization for Standardization ISO. *ISO/IEC 17799:2005. Information technology – Security techniques – Code of practice for information security management*. 2005. URL: <https://www.iso.org/standard/39612.html>.
- [11] ISO. 2013. URL: <http://www.iso27000.es/sgsi.html>.
- [12] Ismael Etxeberria Aguiriano. “Sistemas de Gestión de la Seguridad de Sistemas de Información”. 2014.
- [13] McAfee Labs. *Informe de Predicciones sobre amenazas para 2016*. 2016. URL: <https://mcafee.app.box.com/v/2016predictions>.

- [14] John von Neumann y Arthur Walter Burks. *Theory of self-reproducing automata*. 1966. URL: https://archive.org/details/theoryofselfrepr00vonn_0.
- [15] Xataka. *La historia de Creeper, el primer virus informático jamás programado*. Abr. de 2017. URL: <https://www.xataka.com/historia-tecnologica/la-historia-de-creeper-el-primer-virus-informatico-jamas-programado>.
- [16] Panda Security. *Classic Malware: su historia, su evolución*. URL: <http://www.pandasecurity.com/mexico/homeusers/security-info/classic-malware/>.
- [17] Symantec. *ISTR: Internet Security Threat Report*. Abr. de 2016. URL: <https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf>.
- [18] Hongkiat. *10 Most Destructive Computer Viruses*. URL: <http://www.hongkiat.com/blog/famous-malicious-computer-viruses/>.
- [19] The Telegraph. *What is WannaCry and how does ransomware work?* Mayo de 2017. URL: <http://www.telegraph.co.uk/technology/0/ransomware-does-work/>.
- [20] Business Insider UK. *A massive cyberattack using leaked NSA exploits has hit 99 countries, and it's still spreading*. Mayo de 2017. URL: <http://uk.businessinsider.com/telefonica-and-other-firms-have-been-infected-by-wannacry-malware-2017-5>.
- [21] Ander Granado. *HearItAll. A little Keylogger for Windows developed in C++*. Ene. de 2016. URL: <https://github.com/ander94lakx/HearItAll>.
- [22] El Mundo. *El móvil supera por primera vez al ordenador para acceder a Internet*. 2016. URL: <https://mcafee.app.box.com/v/2016predictions>.
- [23] Gartner. *Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter of 2016*. 2016. URL: <http://www.gartner.com/newsroom/id/3415117>.
- [24] Apple Inc. *iOS Security - iOS 10 -White Paper*. 2017. URL: https://www.apple.com/business/docs/iOS_Security_Guide.pdf.
- [25] Android. *Security Tips*. URL: <https://developer.android.com/training/articles/security-tips.html>.
- [26] VI Jornada de Seguridad y Protección de Datos de Carácter Personal. Vitoria-Gasteiz, Spain: UPV/EHU, 2014. URL: <http://lsi.vc.ehu.es/wdocs/pdd/pdd-2014/2014-Programa.pdf>.
- [27] Daniele Miorandi y col. “Internet of things: Vision, applications and research challenges”. En: *Ad Hoc Networks* 10.7 (2012), págs. 1497-1516. ISSN: 1570-8705. DOI: <https://doi.org/10.1016/j.adhoc.2012.02.016>. URL: <http://www.sciencedirect.com/science/article/pii/S1570870512000674>.

- [28] Jayavardhana Gubbi y col. “Internet of Things (IoT): A vision, architectural elements, and future directions”. En: *Future Generation Computer Systems* 29.7 (2013). Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services; Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond, págs. 1645-1660. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2013.01.010>. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X13000241>.
- [29] Quartz Media. *The easy way your smart coffee machine could get hacked and ruin your life*. Feb. de 2017. URL: <https://qz.com/901823/the-easy-way-your-smart-coffee-machine-could-get-hacked-and-ruin-your-life/>.
- [30] TechRadar. *The internet of things can be hacked – and the risks are growing every day*. Feb. de 2017. URL: <http://www.techradar.com/news/the-internet-of-things-can-be-hacked-and-that-puts-your-life-at-risk>.
- [31] SearchDataCenter. *RSA Conference 2016: IoT se estrellará y arderá si la seguridad no está primero*. 2016. URL: <http://searchdatacenter.techtarget.com/es/cronica/RSA-Conference-2016-IoT-se-estrellara-y-ardera-si-la-seguridad-no-esta-primer>.
- [32] Xataka. *Miele tuvo la gran idea de incluir un servidor web en un lavavajillas, pero no de aplicar la seguridad necesaria*. Mar. de 2017. URL: <https://www.xataka.com/seguridad/de-dispositivo-conectado-a-hackeado-lo-que-puede-ocurrir-al-incluir-un-servidor-web-en-un-lavavajillas>.
- [33] El País. *La seguridad de los coches Tesla en duda... por una aplicación Android*. Nov. de 2016. URL: http://cincodias.elpais.com/cincodias/2016/11/24/motor/1479986182_473873.html.
- [34] Instituto Nacional de Tecnologías de la Comunicación INTECO. *Riesgos y amenazas en Cloud Computing*. Mar. de 2011. URL: https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/cert_inf_riesgos_y_amenazas_en_cloud_computing.pdf.
- [35] Cloud Security Alliance CSA. *The Treacherous 12: Cloud Computing Top Threats in 2016*. Feb. de 2016. URL: https://downloads.cloudsecurityalliance.org/assets/research/top-threats/Treacherous-12_Cloud-Computing_Top-Threats.pdf.
- [36] Pablo González Pérez, Gemán Sánchez Garcés y Jose Miguel Soriano de la Cámara. *Penetrating con Kali*. Juan Ramón Jimenez, 8. 28932 Madrid (España): 0xWORD, 2013. ISBN: 978-84-616-7738-2.
- [37] Penetration Testing Execution Standard. 2017. URL: http://www.pentest-standard.org/index.php/Main_Page.

- [38] Instituto Nacional de Tecnologías de la Comunicación INTECO. *Pentest: Recolección de Información (Data Gathering)*. URL: https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/cert_inf_seguridad_information_gathering.pdf.
- [39] Openwall Community Wiki. *John the Ripper benchmarks*. Mayo de 2016. URL: <http://openwall.info/wiki/john/benchmarks>.
- [40] Digital Trends. *NVIDIA'S GTX 1080 CAN CRACK PASSWORDS AS EASILY AS IT CAN GAME*. Ago. de 2016. URL: <https://www.digitaltrends.com/computing/nvidia-gtx-1080-crack-passwords/>.
- [41] Juan Luís García Rambla. *Ataques en redes de datos IPv4 e IPv6*. 2.^a ed. Juan Ramón Jimenez, 8. 28932 Madrid (España): 0xWORD, 2014. ISBN: 978-84-616-8383-3.
- [42] TIOBE. *Tiobe Index, September 2017*. Sep. de 2017. URL: <https://www.tiobe.com/tiobe-index/>.
- [43] Android Developers. *Kotlin and Android*. Sep. de 2017. URL: <https://developer.android.com/kotlin/index.html>.
- [44] Comparison to Java Programming Language. *JetBrains s.r.o.* Oct. de 2017. URL: <https://kotlinlang.org/docs/reference/comparison-to-java.html>.
- [45] The Color System of Material Design. *Material Design*. Jul. de 2018. URL: <https://material.io/design/color/#tools-for-picking-colors>.
- [46] Save data in a local database using Room. *Android Developers*. Jul. de 2018. URL: <https://developer.android.com/training/data-storage/room/>.