Universitat Oberta de Catalunya

uoc.edu

Creació d'una aplicació mòbil per a l'anàlisi de vulnerabilitats d'un servidor

Martí Saladelafont Díaz
Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions
Anàlisi de dades

Cristina Pérez Solà
Joan Caparrós Ramírez

01/06/2021

FITXA DEL TREBALL FINAL

Títol del treball:	*Creació d'una aplicació mòbil per a l'anàlisi de vulnerabilitats d'un servidor*
Nom de l'autor:	*Martí Saladelafont Díaz*
Nom del consultor/a:	*Joan Caparrós Ramírez*
Nom del PRA:	*Cristina Pérez Solà*
Data de lliurament:	*06/2021*
Titulació o programa:	*Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions*
Àrea del Treball Final:	*Anàlisi de dades*
Idioma del treball:	*Català*
Paraules clau	*Aplicació mòbil, Escaneig de ports, Escaneig de xarxa, TCP, UDP, ICMP*

Resum del Treball:

En un món súper connectat a través d'Internet com el nostre cada cop és més estès l'ús de servidors per a emmagatzemar qualsevol tipus de dades. Atès aquest ús, protegir la seguretat dels servidors davant de qualsevol amenaça és una tasca necessària per a la seva correcta utilització. La detecció de vulnerabilitats dels servidors és un punt clau de cara a la protecció de la seguretat. Paral·lelament, l'ús de dispositius mòbils o tauletes també està en augment dins de la societat. Aquests dispositius ens permeten un ús mòbil que no tenen els dispositius fixes.

Aquests dos punts ens porten a la necessitat de poder utilitzar els dispositius mòbils per a poder detectar les vulnerabilitats dels servidors que utilitzem a la xarxa. En aquest treball es planteja i es duu a terme la realització d'una aplicació per a dispositius mòbils que permet la detecció de vulnerabilitats en un servidor.

Aquesta aplicació ens permet realitzar una cerca de dispositius disponibles dins d'una xarxa especificada per l'usuari, així com també realitzar una cerca de ports oberts d'un dispositiu establert per l'usuari. Amb aquestes cerques es pot establir quins serveis s'estan utilitzant a la màquina i poder determinar si és vulnerable o no.

Abstract:

In a super-connected world through Internet like ours, the use of servers to store any kind of data is increasingly popular. According to this use, protect the security related to these servers in front any kind of threat is a necessary task.

The detection of the vulnerabilities of the servers is a key point in order to protect their security. In parallel, the usage of the mobile devices is increasingly popular among the society. These devices allow a mobile usage in front of the fixed devices.
These two points of view lead us to the necessity of use the mobile devices to detect the vulnerabilities of the servers that we use in the network. According to this, on this paper is present and carry out an application for mobile devices that allows the detection of server's vulnerabilities.
This application allows a user to perform a search of accessible devices inside a network specified for the user, and also to perform a search of the open ports of a device. With these two searches, the user could know which services are running in the device and determine if the device is vulnerable or not.

Índex

Llista de figures

1. Introducció

1.1 Context i justificació del Treball

En un món súper connectat a través d'Internet com és el nostre, on cada dia es genera un gran volum de dades, l'ús de servidors connectats a Internet per tal d'emmagatzemar aquestes dades (el que es coneix popularment com *el núvol*) està en clar creixement. Fotografies, documents o arxius són algunes de les coses que es guarden en aquests servidors, però també dades més sensibles com poden ser contrasenyes, dades bancàries o informació sensible de les empreses, així com també l'oferta de diferents serveis com ara l'ús d'aplicacions.

Degut a la gran popularitat en la utilització d'aquesta tecnologia, els servidors s'han convertit en un dels grans objectius de la ciber-delinqüència. A resultes d'això, protegir la seguretat dels servidors s'ha convertit en una prioritat per a les empreses del sector que ofereixen aquests serveis.

Davant d'aquest escenari, l'anàlisi de les vulnerabilitats que poden afectar als servidors és un punt clau per a la protecció d'aquests servidors. La llista de vulnerabilitats és força àmplia: *cross-site scripting*, *SQL Injection* o *command injection* en són alguns exemples. Al mercat existeixen diferents eines automàtiques que permeten l'anàlisi d'aquestes vulnerabilitats. Malgrat això, la majora d'elles estan enfocades en el seu ús a través de plataformes web o en sistemes operatius d'ordinadors convencionals (com ara Windows, MacOS o Linux).

Atesa la gran utilització de dispositius mòbils o tauletes dins de la societat, i al fet que no existeixen significatives aplicacions d'anàlisi de vulnerabilitats per a aquests dispositius, el present treball es centra en la realització d'una aplicació per a plataformes mòbils, amb l'objectiu d'analitzar i detectar les possibles vulnerabilitats que pot tenir un servidor, per poder així mitigar-les.

1.2 Objectius del Treball

L'objectiu principal del treball és realitzar una aplicació per a dispositius mòbils que permeti la detecció i anàlisis de les vulnerabilitats que es poden trobar en un servidor.

Per a poder complir amb l'objectiu marcat, s'han declarat els següents sub-objectius que permetran atendre millor les necessitats del desenvolupament del treball:

- Investigar i entendre bé les diferents plataformes mòbils existents per a trobar la més adequada per a realitzar el treball.

- Decidir quines són les vulnerabilitats més adients per a buscar en un servidor.

- Implementar un o més atacs o escanejos que permetin l'anàlisi de cada una de les vulnerabilitats escollides.

- Desenvolupar una aplicació *user-friendly* que sigui de fàcil ús i que utilitzi els atacs implementats.

1.3 Enfocament i mètode seguit

Per a poder arribar als objectius esmentats a l'apartat anterior s'ha decidit seguir l'estratègia de desenvolupar un producte final nou, que tingui una interfície de fàcil ús per a qualsevol usuari, i que permeti la realització de l'anàlisi de vulnerabilitats així com la realització d'un informe d'aquestes vulnerabilitats.

Tanmateix, es realitzarà una investigació profunda respecte a la possible reutilització dels diferents atacs que permeten avaluar les vulnerabilitats, fent una cerca d'atacs *open-source* i analitzant la millor manera d'acoblar-los a l'aplicació. S'ha decidit de realitzar aquesta reutilització per agilitzar els terminis de finalització del projecte i per poder afegir el màxim nombre d'anàlisis per tal que l'aplicació sigui el més completa possible.

Per tal de complir amb els terminis de les entregues, així com també per a poder desenvolupar l'aplicació de manera més àgil, s'ha decidit seguir la metodologia SCRUM, donat que ens dona la capacitat de respondre més ràpidament als possibles problemes que poden sortir durant el projecte.

Per a poder seguir amb la metodologia escollida durant el projecte, es farà ús de l'eina *Jira*, ja que ens permet administrar les diferents tasques en que consta el projecte a través d'un panell gràfic. Dins mateix de l'eina es definiran els següents estats que podran tenir les diferents tasques:

- TO DO: és l'estat inicial de les tasques abans de que s'hagin començat a desenvolupar.

- IN PROGRESS: és l'estat que tindran les tasques durant tot el seu desenvolupament.

- IN REVIEW: és l'estat que tindran les tasques un cop s'hagin acabat de desenvolupar i abans de que es puguin considerar acabades.

- TO MODIFY: és l'estat que prendran les tasques que han estat revisades i necessitin una modificació.

- IN MODIFICATION: és l'estat que tindran les tasques que s'estiguin modificant.

- DONE: és l'últim estat que tindran les tasques un cop hagin estat finalitzades i revisades.

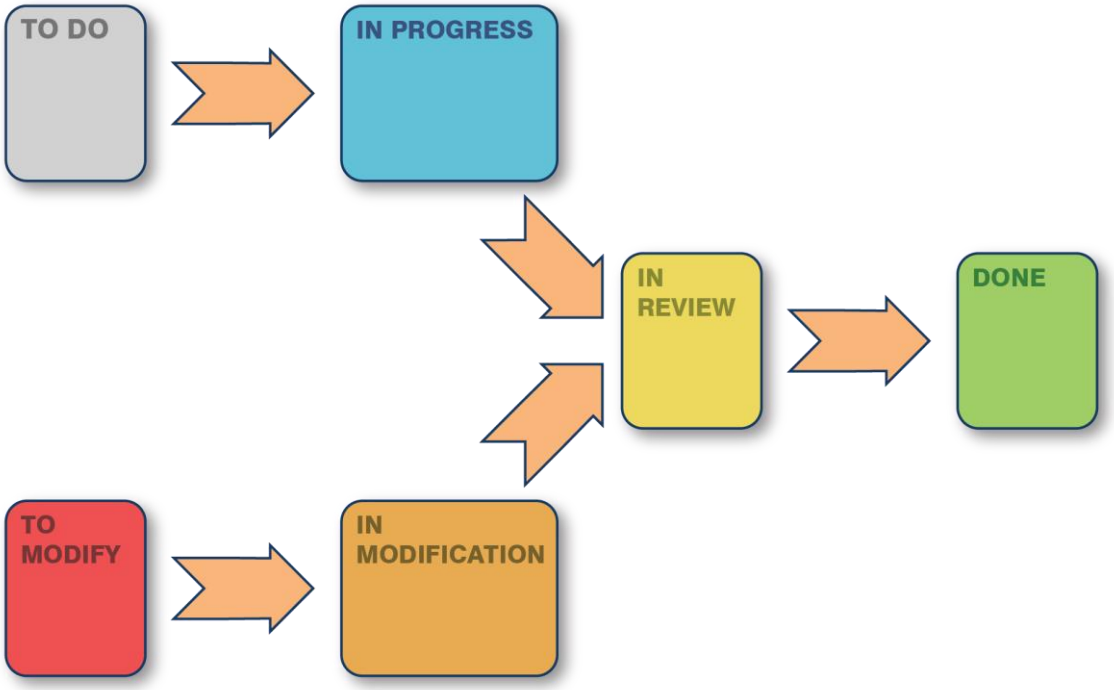

Figura 1. Estats que tindran les tasques

La principal característica de la metodologia SCRUM són els anomenats *sprints*, que és el període de temps donat a una tasca per a ser realitzada. Per a poder realitzar el projecte, es realitzaran uns *sprints* adaptats a cada tasca, de com a màxim una setmana de duració.

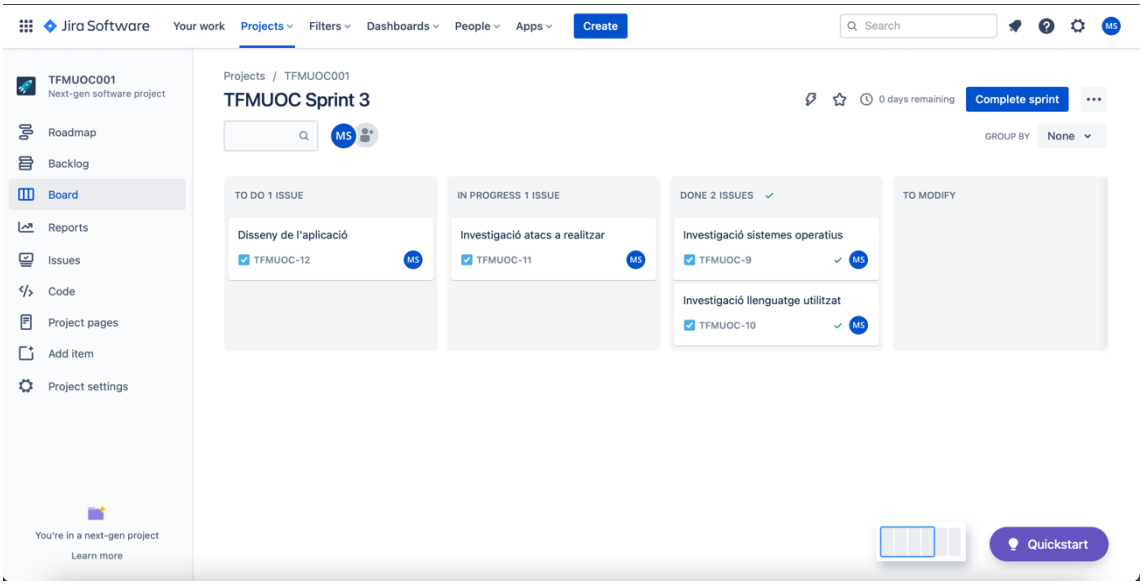

Figura 2. Exemple del *dashboard* de l'aplicació *Jira*

1.4 Planificació del Treball

Tenint en compte els objectius mencionats anteriorment, així com amb la metodologia a seguir, la realització del projecte es realitzarà en les següents fases:

1.4.1 Fase d'investigació

Al llarg d'aquesta fase es concretarà el context sobre el qual estarà plantejat el treball. Es buscarà informació respecte les diferents arquitectures dels dispositius mòbils, així com informació respecte als possibles llenguatges de programació més adients per a la realització del projecte. Un cop triats aquests dos punts, es dissenyarà l'estructura de l'aplicació. Seguidament es realitzarà una investigació sobre les vulnerabilitats que poden afectar als servidors i els possibles atacs que es realitzaran per a poder analitzar la seguretat.

Les principals tasques a realitzar durant aquesta fase seran les següents:

- Definició del problema a resoldre, objectius, metodologia i planificació del projecte
- Anàlisi de l'estat de l'art del sector
- Anàlisi de les arquitectures de dispositius mòbils:
    - Sistema Android
    - Sistema iOS
- Anàlisi dels llenguatges de programació:
    - Sistema Android
    - Sistema iOS
- Anàlisi de les vulnerabilitats i atacs als servidors
- Disseny de l'aplicació

Atès la falta de coneixement que té l'alumne respecte la creació i desenvolupament d'aplicacions mòbils, aquesta fase serà una fase molt important de cara al projecte, ja que en ella es farà una recerca a fons de totes les possibilitats que hi ha respecte a aquest desenvolupament.

1.4.2 Fase d'implementació

En aquesta fase es realitzarà la implementació del *back-end* de l'aplicació, el *front-end* de l'aplicació. També s'implementaran els diferents atacs a realitzar, que s'acoblaran a l'estructura de l'aplicació.

Atesa la tipologia del projecte, es necessitarà d'un servidor sobre el qual poder realitzar els atacs. Així doncs, dins d'aquesta fase es realitzarà la implementació de l'entorn essencial per a poder executar l'aplicació correctament.

Les principals tasques a realitzar durant aquesta fase són les següents:

- Implementació del *back-end* de l'aplicació, utilitzant el llenguatge de programació decidit prèviament

- Implementació del *front-end* de l'aplicació
- Implementació dels atacs a realitzar
- Adequació de l'entorn de proves:
    - Instal·lació d'un servidor per poder realitzar les proves funcionals de l'aplicació

Aquesta fase és la fase més crítica del procés. En funció de la tipologia d'aplicació escollida, conjuntament amb el llenguatge escollit, pot fer que les tasques d'aquesta fase es puguin allargar més del previst a la planificació, donat que l'alumne no té un gran coneixement respecte al desenvolupament d'aplicacions mòbils (veure secció 1.8 Anàlisi de riscs).

1.4.3 Fase d'optimització

Finalment, en aquesta fase es farà les proves necessàries per a assegurar que l'aplicació funciona correctament, fent els ajustos corresponents a la implementació realitzada a la fase anterior.

Les principals tasques a realitzar durant aquesta fase són les següents:

- Proves funcionals del *back-end*
- Proves funcionals del *front-end*
- Proves funcionals dels atacs que s'han implementat
- Redacció de la memòria final

Degut al fet que al llarg del projecte pugui haver-hi canvis en els diferents apartats que el componen, l'ús de la metodologia SCRUM per a la realització del projecte ens permetrà que al llarg de les diferents fases es puguin tornar a realitzar tasques de les fases anteriors.

1.5 Planificació temporal

A continuació es mostra el llistat de tasques que es realitzaran dins de cada fase al llarg del projecte; el dia d'inici i final de cada una de les tasques, fases i projecte; així com també la quantitat de dies de durada. En total es destinaran aproximadament unes 250 hores al projecte al llarg d'aproximadament 90 dies.

Nom	Inici	Finalització	Durada
Treball Final Master	17/2/21	18/6/21	88
Fase d'investigació	17/2/21	30/3/21	30
Definició problema a resoldre	17/2/21	19/2/21	3
Definir objectius	19/2/21	22/2/21	2
Definir metodologia	22/2/21	23/2/21	2
Planificació del treball	23/2/21	26/2/21	4
Anàlisi estat de l'art	26/2/21	2/3/21	3
Entrega 1	2/3/21	2/3/21	1
Investigació arquitectures	2/3/21	8/3/21	5
Investigació codi font	8/3/21	9/3/21	2
Investigació vulnerabilitats	9/3/21	15/3/21	5
Disseny aplicació	15/3/21	30/3/21	12
Entrega 2	30/3/21	30/3/21	1
Fase d'implementació	30/3/21	6/5/21	28
Implementació back-end	30/3/21	8/4/21	8
Implementació atacs	8/4/21	19/4/21	8
Implementació entorn	19/4/21	19/4/21	1
Implementació front-end	19/4/21	6/5/21	14
Entrega 3	27/4/21	27/4/21	1
Fase d'optimització	6/5/21	1/6/21	19
Proves funcionals	6/5/21	1/6/21	19
Entrega Final	1/6/21	1/6/21	1
Realització del vídeo final	1/6/21	8/6/21	6
Entrega vídeo final	8/6/21	8/6/21	1
Defensa del TFM	14/6/21	18/6/21	5

Figura 3. Tasques en què es divideix el projecte

A continuació es mostra el diagrama de Gantt del projecte, on es mostren les tasques que s'han definit prèviament (com es pot veure a la Figura 3) i la seva previsió al calendari.

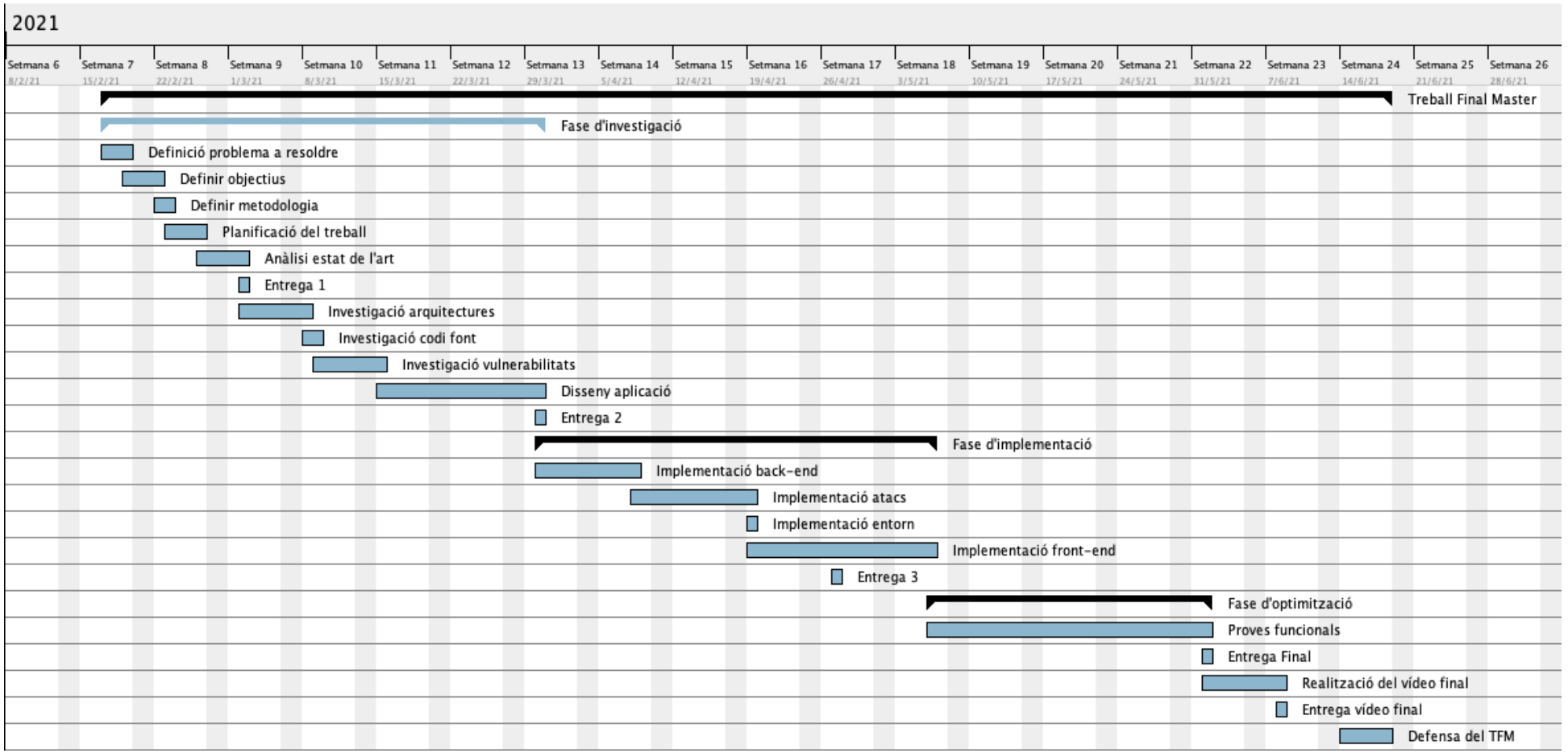

Figura 4. Diagrama temporal del projecte

1.6 Revisió de l'estat de l'art

Durant la primera *fase d'investigació* del projecte s'ha realitzat una cerca de les aplicacions actuals que hi ha al mercat respecte a l'anàlisi i detecció de vulnerabilitats contra servidors.

Atès que la seguretat en aquests sistemes és un puntal bàsic per assegurar un servei correcte als usuaris, existeixen molts tipus d'aplicacions o eines que permeten aquest anàlisi. Aquestes es poden dividir en els següents tipus:

- Aplicacions d'anàlisi de vulnerabilitats basades en el núvol (*cloud-based vulnerability scanners* en anglès). S'utilitzen per a buscar vulnerabilitats de sistemes basats en el núvol.

- Aplicacions d'anàlisi de vulnerabilitats basades en *hosts* (*host-based vulnerability scanners* en anglès). Són aplicacions que s'utilitzen per a analitzar únicament un sol *host* o dispositiu, com ara un ordinador individual o un *switch*.

- Aplicacions d'anàlisi de vulnerabilitats basades en xarxa (*network-based vulnerability scanners* en anglès). Són aplicacions que analitzen, bàsicament, els ports oberts de dispositius que estan connectats a la xarxa, i analitzen els diferents serveis que ofereixen aquests ports.

- Aplicacions d'anàlisi de vulnerabilitats basades en bases de dades (*database-based vulnerability scanners* en anglès). Són usades per analitzar sistemes de bases de dades.

Com a part de les principals aplicacions, podem trobar les següents:

- *Nessus*: Creada l'any 1998 com un escàner *open-source*, des de l'any 2005 pertany a l'empresa Tenable, Inc. De manera *normal*, Nessus realitza un escaneig dels ports del sistema a avaluar per comprovar quins estan oberts i quin servei s'hi ofereix, per després realitzar diversos *exploits* a aquests ports.

  Addicionalment, Nessus permet la utilització de diferents *pluguins*, que realitzen altres diferents vulnerabilitats extra a part de les avaluades per defecte. A més, es generen informes de cada una de les vulnerabilitats analitzades.

  Està disponible per plataformes Windows, MacOS i Linux.

- *OpenVAS*: Creada l'any 2005 com a derivada de l'aplicació Nessus quan aquesta va passar a ser de llicència, és una aplicació *open-source* que permet l'escaneig dels ports d'una màquina, generat un informe de les vulnerabilitats trobades.

- *Nmap*: Creada l'any 1998, és una aplicació *open-source* que permet analitzar els ports oberts d'una màquina per obtenir-ne informació.

  No té entorn gràfic, ja que s'executa a través de la línia de comandes. Està disponible per plataformes Windows, MacOS i Linux.

1.7 Recursos i pressupost

Per a poder dur a terme el projecte de la manera en que s'ha descrit anteriorment, més enllà dels recursos en temps descrits a la secció *1.4 Planificació del treball*, també s'haurà d'utilitzar els següents recursos materials:

- Dispositiu mòbil per a realitzar proves
  - Serà el dispositiu utilitzat per a instal·lar l'aplicació i realitzar les proves funcionals.
  - Cost aproximat: 100 €
- Servidor de proves
  - Serà el dispositiu sobre el qual es realitzaran els atacs. Es valoraran dues opcions: utilitzar servidors de prova allotjats a Internet o el desenvolupament d'un servidor propi allotjat a la xarxa privada.
  - Cost: variable en funció de l'opció escollida

Més enllà dels recursos més específics que es necessiten per a la realització del projecte, també s'haurà de tenir en compte els següents recursos necessaris per a la completa realització del projecte:

- Ordinador personal
  - És l'ordinador en què es realitzarà el desenvolupament del projecte i la redacció de la memòria final.
  - Cost aproximat: 500 €
- Accés a Internet
  - Cost: 50 € al mes.
- Eina *Android Studio*
  - És l'eina que s'utilitzarà per al desenvolupament del projecte. És una eina gratuïta desenvolupada per JetBrains dissenyada especialment per a la realització d'aplicacions Android natives. La versió escollida és la 4.1.3.
  - Cost: gratuït
- Eina *Jira*
  - És l'eina que s'utilitzarà per a fer seguiment del projecte i les tasques a realitzar. Té diferents plans de subscripció en funció del nombre de persones de l'equip o de la quantitats de projectes que ha de gestionar. Malgrat això, s'ha decidit d'utilitzar la versió gratuïta donat que ens permet la gestió d'un projecte amb poques persones involucrades.
  - Cost: gratuït
- Eina *Microsoft Word*
  - És el programa que s'utilitzarà per a la redacció de la memòria final.
  - Cost: 8 € al mes.

A continuació es mostra una taula de resum dels costos que tindrà el projecte:

Element	Cost
Dispositiu mòbil	100€
Servidor de proves	-
Ordinador personal	500€
Accés a Internet	50€/mes
Android Studio	Gratuït
Jira	Gratuït
Microsoft Word	8€/mes
Cost total aproximat	840€

Figura 5. Taula de costos

1.8 Anàlisi de riscs

En aquesta secció s'identificaran els diferents riscs que poden sorgir durant les diferents fases de desenvolupament del projecte. Per a poder evitar que els següents riscs puguin comprometre el desenvolupament del projecte, s'ha presentat un conjunt de possibles solucions a cada un dels riscs.

R1. Objectius ambiciosos

Un dels riscs que s'ha pogut identificar és que s'hagi plantejat un projecte massa ambiciós, que no pugui ser desenvolupat en la seva totalitat durant la realització d'aquest TFM. El problema principal és que l'alumne, degut a la compaginació dels estudis amb la seva vida laboral, no pugui dedicar el temps estimat a cada una de les tasques identificades, fet que comporti una desviació en el temps de realització del projecte que faci que no es pugui acabar del tot en el temps establert per al TFM. Per altra banda, els potencials problemes que puguin sorgir durant la fase d'implementació del projecte podrien, també, endarrerir també el temps de realització del mateix.

Solució: la solució establerta per tal de mitigar aquest risc és que es documentarà els possibles problemes sorgits per tal de deixar-ne constància a la memòria. També es donarà l'opció de modificar els objectius del treball per tal d'adaptar-los a la nova realitat temporal si fos escaient.

R2. Problemes en la implementació de l'aplicació

Degut a la falta de coneixement o experiència de l'alumne a l'hora de realitzar el desenvolupament d'aplicacions mòbils, es pot donar el cas que el temps estimat per a les tasques d'implementació del projecte s'incrementi, fet que comporti una desviació en el temps establert per a la realització del projecte.

Solució: la solució establerta per tal de mitigar aquest risc és que l'alumne intentarà realitzar l'aplicació de la manera més àgil i fàcil possible, realitzant-la en un llenguatge d'aprenentatge senzill, i que tingui suport a Internet.

R3. Problemes amb l'entorn

Un dels riscs identificats és els possibles problemes que puguin sortir a les eines d'entorn utilitzades per a la realització del projecte, ja sigui en l'entorn de desenvolupament com en els servidors a avaluar.

Solució: l'alumne realitzarà els canvis necessaris de les eines de desenvolupament per tal que el projecte pugui ser realitzat, documentant i justificant cada un dels canvis a la memòria final.

1.9 Requisits legals

Segons l'estipulat al *Reglament (UE) 2016/679 del Parlament Europeu i del Consell*, conegut com *Reglament General de Protecció de Dades* (RGPD), que va entrar en vigor el 25 de maig de 2016 i és d'aplicació des del 25 de maig de 2018, per a la realització d'aquest projecte s'haurà de tenir en compte una sèrie de requisits legals relacionats amb la protecció de dades personals que s'hauran obtingut a través de l'aplicació creada.

2. Investigació

En aquesta secció es mostrarà la investigació que s'ha dut a terme per tal de decidir quin tipus d'aplicació es realitzarà durant el projecte, analitzant quin és el sistema operatiu i el llenguatge de programació més propicis per a la realització de l'aplicació.

Addicionalment, també es mostrarà la investigació realitzada en relació als atacs o funcionalitats que s'implementaran.

2.1 Sistemes operatius

Un sistema operatiu és un sistema a baix nivell que proveeix serveis a les aplicacions que s'executen sobre d'ell. Aquests sistemes operatius es diferencien dels sistemes operatius d'altres tipus de dispositius (com els ordinadors personals) pel fet que estan dissenyats per incorporar les característiques d'usabilitat dels dispositius mòbils (com pot ser la pantalla tàctil, per exemple), formats multimèdia per a mòbils, conjuntament amb un ús més elaborat de xarxes *wireless* o també l'ús de les targetes SIM per a telefonia.

Al mercat existeixen molts tipus diferents de sistema operatiu per a dispositius mòbils. Degut a l'ús generalitzat i cada vegada més gran d'aquest tipus de dispositius, cada any augmenta el nombre de sistemes operatius disponibles, malgrat que el sistema operatiu *Android* segueix sent el líder indiscutible del mercat.

En la següent taula podem veure la presència al mercat dels diferents sistemes operatius per a dispositius mòbils que existeixen, a data de febrer 2021. En aquest anàlisi podem veure que el sistema operatiu *Android*, propietat de Google, és el que té més presència mundialment, seguit pel sistema operatiu d'Apple *iOS*.

Sistema Operatiu	% al mercat mundial
Android (Google Inc.)	71,9%
iOS (Apple Inc.)	27,33%
Tizen (Samsung Electronics)	0,39%
KaiOS	0,14%
Altres	0,24%

Figura 6. Percentatge de mercat dels sistemes operatius

A la següent figura podem veure l'evolució del mercat de sistemes operatius des de febrer de 2015 fins a febrer de 2021.

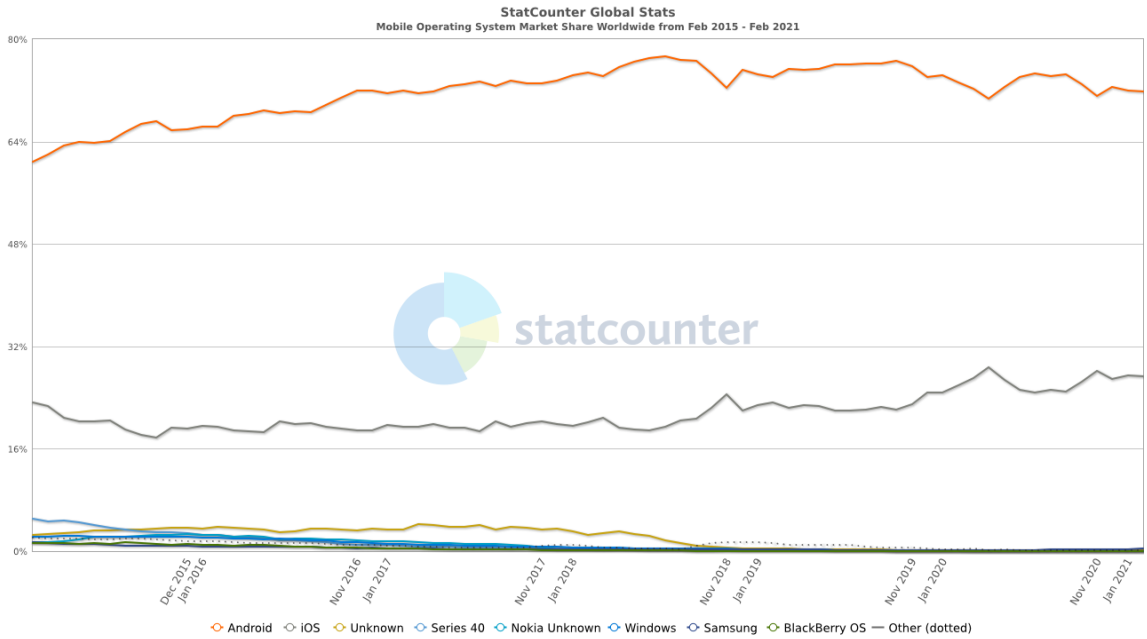

Figura 7. Evolució del mercat de sistemes operatius mòbils (2015-2021) [6]

Seguidament s'analitzen els dos principals sistemes operatius, de manera introductòria.

2.1.1 Sistema operatiu Android

Creat l'any 2007 per l'empresa Android Inc., filial de Google, *Android* és un sistema operatiu de codi obert, basat en Linux i dissenyat principalment per a ser utilitzat en dispositius tàctils com telèfons intel·ligents o tauletes.

La base de la plataforma *Android* és el *kernel* de Linux, seguit d'una capa d'abstracció de hardware que permet connectar les capacitats del hardware del dispositiu (com pot ser el sistema de Bluetooth o la càmera de vídeo) amb la capa *Java API Framework*, que és la base a on es carregaran les aplicacions del dispositiu.

Les funcionalitats del sistema operatiu estan disponibles a través de la capa *Java API Framework*, que és un conjunt d'APIs escrites en llenguatge Java.

2.1.2 Sistema operatiu iOS

iOS és el sistema operatiu creat per Apple Inc. el 2007 per als seus mòbils iPhone. És un sistema operatiu propietari, malgrat que alguns dels seus components són de codi obert.

El sistema operatiu està basat en el *kernel* del sistema operatiu MacOS per a ordinadors personals, anomenat *Darwin*, basat en un *core* Unix.

El llenguatge de programació principal per a aplicacions del sistema operatiu *iOS* és Objective C.

2.2 Aplicacions mòbils

Segons la Viquipèdia, una aplicació mòbil és *"una aplicació informàtica dissenyada per a ser utilitzada en tauletes tàctils, telèfons intel·ligents i altres dispositius mòbils"* [10]. Existeixen tres tipus diferents d'aplicacions mòbils:

- Aplicacions natives
- Aplicacions web
- Aplicacions mixtes

2.2.1 Aplicacions natives

Les aplicacions natives són aquelles aplicacions que requereixen que el seu disseny i implementació sigui específic d'un sistema operatiu en concret. Degut a això, una aplicació nativa només es podrà executar al sistema operatiu per a el que ha estat dissenyada, fent impossible la portabilitat de l'aplicació entre diferents dispositius mòbils amb diferent sistema operatiu.

Atès que les aplicacions natives estan especialment dissenyades per a un sistema operatiu concret, aquestes són molt més ràpides que altres tipus d'aplicacions, sent més confiables a nivell de rendiment. A més aquest tipus d'aplicacions permet l'aplicació de les diferents funcionalitats que té el dispositiu (com pot ser Bluetooth, GPS, sensors, etc.) de manera més àgil.

En quant a la tecnologia utilitzada per a desenvolupar aquest tipus d'aplicacions, dependrà del sistema operatiu escollit. Algun exemple pot ser Java, Python o C++.

2.2.2 Aplicacions web

Les aplicacions web són aquelles aplicacions del qual la característica principal és el fet que es poden utilitzar en qualsevol plataforma, independent de quin sigui el sistema operatiu del dispositiu. Això fa que moltes d'elles no puguin fer ús de les diferents funcionalitats dels dispositius, ja que aquest tipus d'aplicacions s'executen externament, de manera que generalment no és necessari cap descàrrega o instal·lació de programari al mateix dispositiu.

El principal problema d'aquest tipus d'aplicacions és el fet que necessiten estar *online* per a poder executar-se. En el moment en què l'usuari no té accés a Internet, aquestes aplicacions no es poden utilitzar. A més, són dependents del navegador utilitzat, de manera que hi poden haver funcionalitats de l'aplicació activades o desactivades en funció del navegador escollit.

En quant a la tecnologia emprada, bàsicament són aplicacions dissenyades en llenguatge HTML, CSS, JavaScript o altres llenguatges web.

2.2.3 Aplicacions mixtes o híbrides

Aquest tipus d'aplicacions tenen característiques dels dos tipus d'aplicacions vists anteriorment. Principalment són aplicacions que tenen un disseny adaptat a entorns web que fa que l'aplicació pugui ser executada en diferents sistemes operatius, però que també tenen una part nativa que permet l'ús de les diferents funcionalitats dels dispositius, així com un ús *offline* de l'aplicació.

La tecnologia que fan servir aquests tipus d'aplicacions és una barreja entre les tecnologies que utilitzen les aplicacions natives i les que utilitzen les aplicacions web. Com a exemple podríem trobar Ionic, Objective C o React Native.

2.3 Solució proposada per al desenvolupament de l'aplicació

Per tal de poder destriar entre les diferents propostes presentades a l'hora d'escollir quin tipus d'aplicació realitzar i sobre quin sistema operatiu hauria de funcionar, s'ha tingut en compte, principalment, els recursos disponibles per a poder implementar cada una de les característiques presentades a les seccions anteriors. En segon terme, també s'ha tingut en compte la capacitat i els coneixements de l'alumne respecte al desenvolupament d'aplicacions.

Inicialment s'ha escollit quin sistema operatiu seria el més òptim per a desenvolupar l'aplicació.

Respecte als entorns de desenvolupament, tant *Android* com *iOS* proporcionen un programari de suport que permet el desenvolupament d'aplicacions de forma més àgil i ràpida. *Android Studio*, el programari de desenvolupament realitzat per Google, és multi plataforma, així que pot executar-se en la majora de sistemes operatius. *Xcode*, el programari d'Apple, per contra només es pot executar en una màquina MacOS. Malgrat això, l'ordinador que disposa l'alumne per a la realització del treball és un ordinador MacBook. Per tant, l'entorn de desenvolupament no seria un impediment per a la tria del sistema operatiu.

Després de l'anàlisi sobre el mercat de sistemes operatius, però, es pot considerar que el més convenient seria realitzar una aplicació per al sistema operatiu *Android*, degut a que és el sistema operatiu més utilitzat del mercat i, per tant, podrà ser utilitzada per un major nombre de dispositius. Un altre factor que també podria influir en la decisió de realitzar l'aplicació en base a *Android* és el fet que, donat que és el sistema operatiu més estès, hi ha més suport tècnic a l'hora de poder desenvolupar l'aplicació respecte *iOS*, i això fa que es redueixin els riscs a que surtin problemes durant la implementació de l'aplicació.

El fet que *Android* sigui el sistema operatiu més estès al mercat mundial, i conseqüentment sigui un sistema operatiu que té un gran nombre de suport a l'hora de desenvolupar-ne aplicacions fa que sigui el llenguatge més adequat per a realitzar l'aplicació d'aquest projecte. Així doncs, s'ha decidit finalment centrar-nos en el sistema operatiu *Android* per a la realització d'aquest projecte.

Pel que fa a la tipologia d'aplicació, s'ha considerat cada un dels tipus explicats a la secció anterior: aplicacions de tipus web, aplicacions natives, i aplicacions mixtes o híbrides.

Des de bon principi s'ha descartat l'opció de fer una aplicació exclusivament web, ja que s'ha considerat necessari que es pugui utilitzar l'aplicació de manera *offline*, atès que no sempre s'utilitzarà l'aplicació a través d'Internet.

Pel què fa a les aplicacions natives, es considera que aquestes són les més òptimes per tal d'aprofitar al màxim les capacitats del dispositiu. A més, a la xarxa es pot trobar multitud de suport a l'hora de realitzar aquest tipus d'aplicacions, fet que ens pot ajudar a l'hora d'implementar la nostra aplicació.

Respecte a les aplicacions mixtes o híbrides, en què tenen un disseny adaptat a un entorn web però que també es permet l'ús de l'aplicació de manera *offline*, s'ha considerat que podrien ser una tipologia útil a l'hora de realitzar el projecte, degut a la fàcil adaptació que podria tenir el front-end de l'aplicació al sistema operatiu, atès que majoritàriament es realitzara en un llenguatge web.

Després de veure cada una de les tipologies d'aplicació, inicialment l'alumne ha considerat de realitzar una aplicació de tipus mixta o híbrida, atès que l'alumne tenia coneixement previ de llenguatge de programació d'entorns web, com ara HTML o JavaScript. El llenguatge que s'ha considerat ha estat *React Native*. Malgrat el desconeixement de l'alumne respecte aquest llenguatge, s'ha considerat interessant perquè és un llenguatge que utilitza JavaScript, i també atès que té un extens suport tècnic degut a que és un llenguatge desenvolupat per Facebook.

Tot i això, després de fer una recerca per a analitzar com podria ser el disseny final de l'aplicació utilitzant aquest tipus d'aplicació, i concretament aquest llenguatge, s'ha considerat que no és un tipus d'aplicació prou útil per a la realització d'aquest projecte. Això es deu al fet que no s'ha trobat a la xarxa cap informació respecte a com poder implementar els atacs o funcionalitats que es volen realitzar. Malgrat que s'ha trobat alguna llibreria de suport que podria ajudar a la implementació dels atacs, aquestes llibreries no es podrien carregar satisfactòriament sense l'ajuda de la implementació d'un servidor addicional que en permetés el funcionament, fet que comprometia el correcte desenvolupament del projecte degut a la falta d'experiència de l'alumne en aquest context, que podia comportar a un increment de l'esforç i un increment inassolible del temps destinat al projecte.

Per tant, finalment s'ha decidit realitzar una aplicació nativa per al sistema operatiu *Android*, que ens permeti la utilització de llibreries que ens ajudin a la realització i implementació dels atacs que s'ha decidit realitzar.

Atesa a la solució proposada per al desenvolupament del projecte (sistema operatiu *Android* i una aplicació nativa), s'ha realitzat un anàlisi de quin llenguatge de programació seria el més òptim per a la realització

del projecte. El sistema operatiu *Android* només accepta dos llenguatges diferents: Java i Kotlin.

Finalment s'ha decidit de desenvolupar el projecte utilitzant el llenguatge de programació Kotlin. Malgrat que l'alumne sí que coneix el llenguatge Java mentre que té desconeixement respecte Kotlin, s'ha considerat interessant d'utilitzar Kotlin per poder aprendre la utilització d'un nou llenguatge, ja que també té una extensa comunitat de suport que permetrà a l'alumne resoldre els possibles dubtes que puguin sorgir durant la implementació de l'aplicació.

2.4 Atacs en consideració

En aquest punt s'ha investigat sobre quins atacs podrà realitzar l'aplicació. Atès que l'objectiu del projecte és realitzar una aplicació que analitzi les vulnerabilitats d'un servidor, s'ha considerat principalment dos aspectes a l'hora d'implementar aquesta part:

- En primer lloc, la realització d'un escaneig de xarxa. Aquest escaneig ens permetrà veure quins són els dispositius que estan a la mateixa xarxa que el servidor a analitzar.

- En segon lloc, la realització d'un escaneig de ports del servidor a analitzar. Això ens permetrà conèixer com és la màquina i quins serveis ofereix. Aquesta informació serà valuosa per a poder conèixer quines vulnerabilitats són, o poden ser, les que afecten al servidor, i poder després així mitigar-les.

Tenint en compte aquests aspectes, en la realització d'aquest treball s'ha tingut en compte el següent:

2.4.1 Escaneig de xarxa

Un escaneig de xarxa és un dels passos principals de qualsevol aplicació que vulgui realitzar un reconeixement de la xarxa. En aquest cas, la intenció d'aquest tipus d'escaneig és descobrir quins són els dispositius que componen una determinada xarxa, per tal de poder analitzar-los més a fons durant la realització de l'escaneig de ports.

Com veurem a la següent secció referent a l'escaneig de ports, aquest pot ser un procés força costós en temps, degut a la gran quantitat de ports que poden estar oberts en una màquina. Aquest fet fa que intentar descobrir i analitzar tots els ports de les màquines de tot el rang de direccions IP possibles dins d'una xarxa sigui una tasca molt lenta, i en molts casos innecessària.

Amb la finalitat de poder reduir aquest cost, s'ha considerat realitzar un escaneig de xarxa per tal de poder descobrir quines màquines formen part de la mateixa xarxa i així poder analitzar únicament aquestes.

Procediment

El primer pas necessari en qualsevol escaneig de xarxa és obtenir la direcció d'aquesta. En aquest cas, però, com que no només volem

obtenir la direcció d'una sola màquina, s'haurà d'introduir un rang de direccions a analitzar.

Per a fer això, l'usuari haurà d'especificar una direcció IP i la màscara de xarxa, tal i com s'explica a continuació.

Adreces IP i màscares de xarxa

Una adreça IP és, segons la definició a la Viquipèdia, "*el número que identifica inequívocament un dispositiu lògic connectat a la xarxa*" [11].

Les adreces IP es representen com un número binari de 32 bits o de 128 bits, en funció del protocol que s'esculli. Les adreces de 32 bits es corresponen al protocol *IPv4*, mentre que les adreces de 128 bits es corresponen al protocol *IPv6*.

Per a la realització d'aquest treball s'ha decidit implementar únicament el protocol *IPv4*, atès que és el més estès. Així doncs, tant l'escaneig de xarxa com l'escaneig de ports es realitzaran a màquines identificades amb una IP amb aquest protocol.

Com s'ha dit anteriorment, el protocol *IPv4* utilitza adreces binaries de 32 bits. Per tenir una major llegibilitat, aquests bits es divideixen en quatre grups de 8 bits cada un, que es representen com a valors decimals separats per un punt.

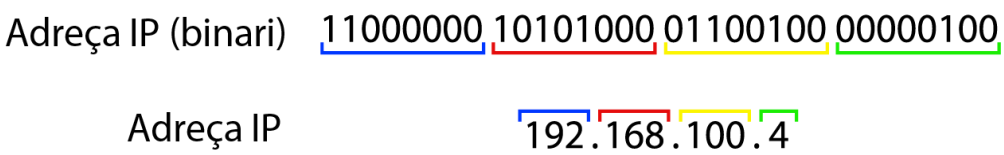

Figura 8. Representació binària i decimal d'una adreça IP amb protocol IPv4

Cada grup de 8 bits (que s'anomena *octet*) pot tenir un valor màxim de 255, ja que és la representació decimal del nombre binari 11111111. Així doncs, el nombre d'IP possibles és molt gran, atès que el rang d'adreces va des de l'adreça 0.0.0.0 fins a la 255.255.255.255.

Per a poder delimitar un rang de direccions IP s'utilitza el que s'anomena *màscara de xarxa*. Una màscara de xarxa és, segons la definició de la Viquipèdia, "*una combinació de bits que serveix per delimitar l'àmbit d'una xarxa*" [12]. Amb aquesta màscara podem identificar quina és la part de l'adreça que es correspon a l'identificador de la xarxa i quina part es correspon a cada un dels dispositius que componen aquesta xarxa.

Les màscares de xarxa es representen de la mateixa manera que les adreces IP, és a dir, de forma binària o decimal. D'aquesta manera, com a exemple, la màscara de xarxa 11111111000000000000000000000000 es pot representar com 255.0.0.0. D'altra banda, per facilitar la identificació de la màscara, aquesta també es pot representar com el nombre en decimal del número de 1 en què es correspon la màscara. Prosseguint amb l'exemple, la màscara 255.0.0.0 es representa com a 8. Aquest número decimal se sol afegir al final de l'adreça IP, separat per una barra /, per tal de poder saber quants dispositius formen part (o poden formar part) de la xarxa.

Per a poder determinar la quantitat de dispositius que componen la xarxa, s'ha de realitzar una operació AND entre el número binari de l'adreça IP de la xarxa i el número binari de la màscara de xarxa. D'aquesta manera podrem identificar quin és l'identificador de la xarxa, que s'anomena *subnet*, i el nombre de dispositius que la componen.

Com a exemple de la identificació del número de dispositius que componen una xarxa, tenim l'adreça 192.168.100.1/24. Tal i com s'especifica, l'adreça IP és 192.168.100.1, mentre que la màscara de xarxa és /24 (que es pot representar com 255.255.255.0). Després de realitzar l'operació AND, obtenim l'adreça de la subnet, que es correspon amb 192.168.100.0. D'aquesta manera podem obtenir el rang d'adreces IP que tindran els dispositius (o *hosts*) de la xarxa, que serà des de l'adreça 192.168.100.1 fins a l'adreça 192.168.100.254, atès que l'adreça 192.168.100.255 s'utilitza com a adreça *broadcast*. Així doncs, el nombre de dispositius d'aquesta xarxa serà de 254.

Existeixen fins a 32 màscares de xarxa diferents, una per cada bit en què es compon l'adreça. Tot i això, les màscares més comunes són les següents:

Màscara	Binari	Número de hosts
/8	255.0.0.0	16777214
/16	255.255.0.0	65534
/24	255.255.255.0	254

Figura 9. Màscares de xarxa més utilitzades

De cara a aquest projecte, s'utilitzaran únicament aquestes tres màscares de xarxa per tal de poder realitzar l'escaneig. S'ha escollit aquestes tres degut a que són les més comunes a l'hora de definir i implementar les xarxes.

Després d'obtenir el número de dispositius que haurem de buscar i les seves adreces, el següent pas de l'escaneig de xarxa és realitzar una connexió a cada un dels dispositius per poder comprovar si el dispositiu està en funcionament o no. Per a fer això es realitzarà un test ICMP, conegut popularment com *ping*.

Test ICMP

Aquest test es correspon en l'enviament de paquets ICMP (*Internet Control Message Protocol* en anglès) i esperar una resposta d'aquests. A través d'aquest tipus de test es pot determinar si un dispositiu està en funcionament dins d'una xarxa.

La composició del paquet ICMP que s'envia a la màquina analitzada es mostra a la Figura 10.

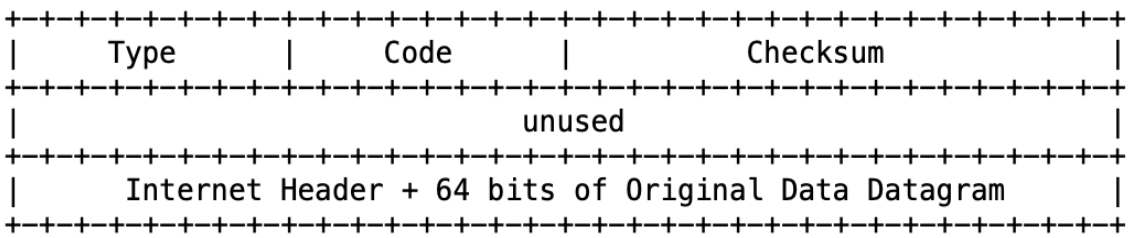

Figura 10. Estructura del paquet ICMP

Aquest paquet ICMP s'encapsula dins del paquet IP que s'envia a la màquina analitzada. En el missatge que s'envia s'hi especifiquen els valors següents:

- El tipus (*type* a la Figura 10) ha de ser 8.
- El codi (*code* a la Figura 10) ha de ser 0.

La màquina analitzada respon al missatge ICMP amb un altre missatge ICMP de resposta. El format és el mateix que es mostra a la Figura 10, però en aquest cas el tipus haurà de ser 0.

En el cas que la màquina a la qual s'envia el paquet ICMP no respongui es considera que aquesta no està en funcionament. En cas contrari, si la màquina respon amb un altre missatge ICMP es considera que està en funcionament.

2.4.2 Escaneig de ports

Un escaneig de ports és, segons la definició a la Viquipèdia, "*l'acció d'analitzar, a través d'un programa, l'estat dels ports oberts d'una màquina connectada a la mateixa xarxa*" [14].

Un escàner s'utilitza per detectar quin port està obert, tancat, o protegit per un tallafocs, així com quins serveis s'estan oferint per a cada un dels ports de la màquina. Amb aquesta informació, es pot detectar quines vulnerabilitats té la màquina per així poder mitigar-les.

Existeixen moltes aplicacions que et permeten realitzar aquest tipus d'escaneigs, com ara *nmap* o *Masscan*.

Atès que per analitzar les vulnerabilitats d'un servidor és necessari conèixer a fons quin tipus de màquina és i quins serveis té, s'ha considerat que realitzar un escaneig de ports és un element clau en aquest anàlisi.

Procediment

Per a poder realitzar l'escaneig de ports, el primer que es necessitarà serà la IP del servidor que es voldrà analitzar. Aquesta IP podrà ser obtinguda a través de l'escaneig de xarxa realitzat prèviament, o introduint-la manualment.

El següent pas serà decidir el rang de ports a analitzar. Respecte això, s'ha decidit enfocar aquest punt des de dos formes diferents: realitzar un

escaneig de la totalitat dels ports disponibles pel dispositiu, i realitzar un escaneig a partir d'un llistat de ports.

En quant al primer punt, es realitzarà un escaneig sencer de tots els ports possibles del dispositiu, és a dir des del port 1 al port 65536. Atesa la gran quantitat de ports a analitzar, i donat que la majora de ports no són utilitzats generalment pels dispositius, s'ha decidit d'implementar el segon punt, que consisteix en poder introduir un llistat de ports perquè únicament siguin aquests els analitzats. Per defecte, els ports que estaran en aquest llistat seran els 20 ports més utilitzats segons la base de dades de l'aplicació *nmap*. Aquests ports són els següents:

Port	Servei
21	ftp
22	ssh
23	telnet
25	smtp
53	domain
80	http
110	pop3
111	rpcbind
135	msrpc
139	netbios-ssn
143	imap
443	https
445	microsoft-ds
993	imaps
995	pop3s
1723	pptp
3306	mysql
3389	ms-wbt-server
5900	vnc
8080	http-proxy

Figura 11. Ports més utilitzats segons la base de dades de nmap

Un cop s'ha decidit quina serà la màquina a analitzar a través de la seva IP, i quins seran els ports que s'analitzaran, es decideix quin és el protocol de comunicació cap a aquesta màquina que s'analitzarà. En el cas d'un escàner de ports, existeixen principalment dos tipus d'escaneig, cada un en funció d'un protocol: TCP i UDP.

Escaneig amb el protocol TCP

És el tipus d'escaneig de ports més comú. Es basa en el protocol TCP (*Transmission Control Protocol* en anglès), que permet crear una connexió entre una màquina client i una màquina servidor.

Per poder establir la connexió, normalment el servidor obre un *socket* en un determinat port TCP i es queda en espera de les peticions del client. Per altra banda, el client inicia la connexió enviant un paquet SYN cap al servidor. Aquest és el primer dels tres passos en que consta la negociació per establir la connexió. El servidor, un cop rep el paquet SYN, contesta amb un paquet SYN/ACK sempre que el port en qüestió

estigui obert. Finalment, el client respon amb un paquet ACK, completant els tres passos (SYN, SYN/ACK i ACK), i per poder així començar l'enviament de les dades.

La Figura 12 mostra un esquema de l'establiment de connexió amb el protocol TCP.

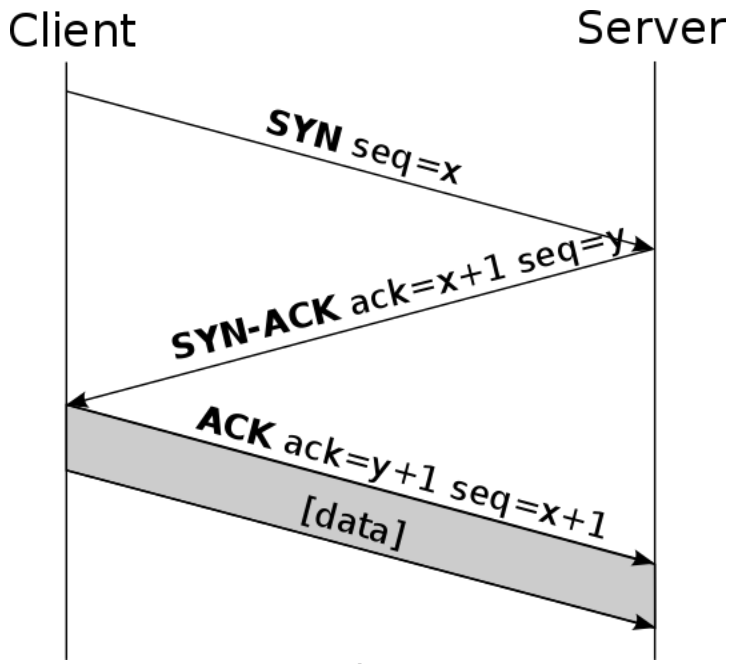

Figura 12. Establiment de connexió del protocol TCP (Font: Viquipèdia)

Respecte als escaneigs utilitzant aquest protocol, es considera que un port està obert quan la connexió s'ha pogut establir correctament. D'altra banda, si aquesta no es pot establir, es considera que el port està tancat. Així doncs, des de l'aplicació es realitzarà una connexió a un port del servidor a analitzar, i es comprovarà el resultat d'aquesta connexió.

Els escaneigs que realitzen l'establiment de la connexió de forma completa (els tres passos SYN, SYN/ACK i ACK) s'anomenen *escaneigs complets* (o *full scan* en anglès). Malgrat això, aquest tipus d'escaneigs solen ser més costosos en temps, atès que s'ha de realitzar completament els tres passos, fet que comporta també que quedi més rastre a la xarxa.

Per mitigar això, i poder realitzar escaneigs més ràpidament, existeix una manera diferent a realitzar l'escaneig, anomenada *escaneigs intermitjos* (o *half scan* en anglès), que consisteix en enviar, per part del client, un paquet SYN i esperar la resposta SYN/ACK del servidor. Com que el servidor contesta amb la resposta SYN/ACK, el client entén que la connexió amb aquest port està oberta. Per tant coneix l'estat del port, que és bàsicament el que es busca a l'escaneig. Així doncs, abandona la connexió, deixant-la incompleta.

Escaneig amb el protocol UDP

Aquest tipus d'escaneig es basa en el protocol UDP (*User Datagram Protocol* en anglès). Aquest protocol també permet crear una

comunicació entre una màquina client i una màquina servidor a través d'un port en concret. A diferència del protocol TCP, l'intercanvi de paquets no té un sistema de confirmació d'entrega o recepció del paquet enviat.

El protocol permet l'enviament de paquets (en aquest cas, anomenats *datagrames*) a través d'una xarxa sense que s'hagi obert cap connexió entre les dues màquines, degut a que la capçalera dels datagrames conté la suficient informació perquè el paquet arribi a la destinació.

La Figura 13 mostra l'estructura de la capçalera d'un datagrama UDP. Aquesta capçalera es compon del port d'origen, el port de destí, la mida de les en *bytes* del datagrama complet i un *checksum* utilitzat com a camp de control. Seguit d'aquests camps trobem el camp de dades amb les dades del datagrama.

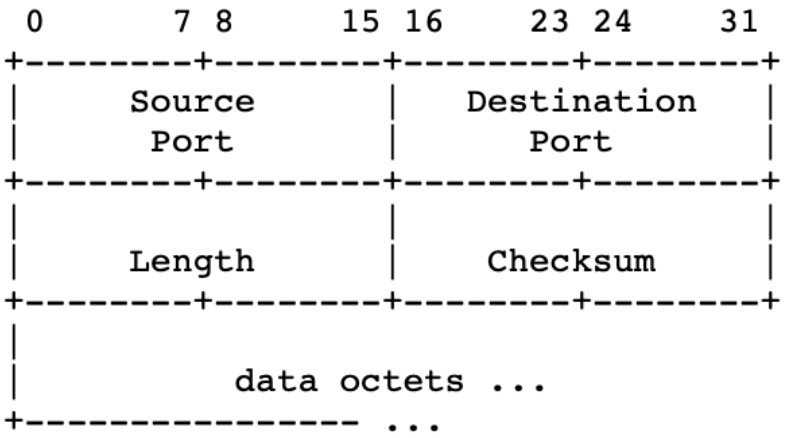

Figura 13. Estructura de la capçalera d'un datagrama UDP [16]

Atès que aquest protocol no té un sistema de comprovació de si el paquet ha arribat o no correctament, la comprovació de si un port està obert o no durant l'escaneig és substancialment diferent a la del protocol TCP.

De cara a aquesta comprovació, s'ha decidit utilitzar el mateix sistema d'escaneig que utilitza l'aplicació *nmap*. En aquest cas, des de l'aplicació s'enviarà un paquet UDP a un port del servidor a analitzar. Seguidament s'analitzarà la resposta a aquest paquet, i es determinarà si el port està obert o tancat en funció dels següents comportaments:

- Si la resposta és un error del tipus *"ICMP port unreachable*" (error ICMP de tipus 3 amb codi 3) es considerarà que el port està *tancat*.
- Si la resposta és un error del tipus "*ICMP port unreachable*" amb codi d'error diferent a 3 es considerarà que el port està *filtrat*.
- Si la resposta és l'esperada, és a dir que respon al paquet enviat, es considerarà que el port està *obert*.
- Si no hi ha resposta es considerarà que el port està *obert/filtrat*.

Detecció dels serveis que s'executen en un port

Després de detectar si un port està obert o no, i de saber amb quin protocol ens hi hem de comunicar, és interessant de saber quin és el servei que està corrent en aquell port.

Com podem veure a la Figura 11, a cada port s'hi executa un servei diferent. Tot i això, no sempre serà el mateix servei que s'executarà pel mateix port, atès que un administrador pot canviar el port d'escolta d'un servei.

Els principals escàners de ports que hi ha al mercat realitzen una sèrie de proves a cada un dels ports oberts que troba l'escàner, per tal de determinar quin és el servei que hi corre. Per exemple, l'aplicació *nmap* té una base de dades (anomenada *nmap-service-probes*) que s'encarrega de realitzar peticions de determinats serveis als ports oberts que ha trobat. En funció dels resultats obtinguts en aquestes proves, l'aplicació és capaç d'obtenir el servei que s'està utilitzant en el port en concret.

A part d'aquest sistema de detecció de serveis, també existeixen bases de dades que permeten una detecció del servei més ràpidament. Aquestes bases de dades solen contenir una relació entre el número de port, el servei més comú que s'hi executa i el protocol utilitzat. Tot i això, aquestes bases de dades no són fiables completament, perquè només proporcionen el servei més comú que s'utilitza en un determinat port, però, com s'ha comentat anteriorment, pot ser que la màquina analitzada no estigui executant el mateix servei per el mateix port.

De cara al desenvolupament del nostre projecte, s'ha considerat la realització de proves per determinar el servei que s'executa en un port obert. Tot i això, s'ha arribat a la conclusió que la implementació d'aquest tipus de proves podria comportar un increment molt gran en el temps de dedicació del projecte, degut a la falta d'experiència de l'alumne en la implementació d'aplicacions mòbils.

Degut a això, s'ha decidit que l'aplicació utilitzi una base de dades que permeti la detecció del servei, encara que aquesta detecció no sigui completament fiable. De cara a l'aplicació, s'ha utilitzat la base de dades que proporciona l'organització *Iana*, ja que és una base de dades que s'actualitza regularment.

3. Disseny

En aquest apartat s'explicarà el disseny que s'ha realitzat per a l'aplicació.

Tal i com podem veure a la secció 1.2 Objectius del Treball, un dels objectius que tenim per a la realització del treball és realitzar una aplicació *user-friendly*, que sigui de fàcil ús per a qualsevol usuari. Seguint aquesta premissa, s'ha realitzat una aplicació senzilla i intuïtiva, que permeti a l'usuari realitzar els atacs que s'han dissenyat sense dificultats.

Per poder explicar el disseny que s'ha realitzat, es mostraran el diagrama de flux de l'aplicació i el disseny el prototipat de l'aplicació.

3.1 Diagrama de flux

El diagrama de flux de l'aplicació ens permet obtenir, de manera molt esquemàtica, una visió global de totes les interaccions que tindrà l'aplicació durant tot el seu funcionament. Es mostra, de manera gràfica, quines són les accions que podrà realitzar un usuari de l'aplicació.

En aquest diagrama es distingeixen quatre grans grups, diferenciats en colors, que es corresponen a les diferents parts en què es compon l'aplicació.

Per una banda tenim la pàgina d'inici i la pàgina principal (diferenciats a l'esquema en color verd). La pàgina d'inici és una pàgina inicial que es mostra a l'obrir l'aplicació, que serveix, tal i com veurem més endavant, per fer una presentació de l'aplicació. Automàticament es passarà a la pàgina principal, que és la part central de l'aplicació i des d'on l'usuari es podrà anar movent cap a les diferents funcionalitats que presenta.

Des de la pàgina principal de l'aplicació podem accedir a dos menús de l'aplicació que ens permeten configurar-la i navegar més fàcilment per a les diferents funcionalitats:

- Des de la pàgina principal es pot obrir el menú de configuració (ressaltat en vermell). Aquest menú és el que ens permet configurar de manera general tota l'aplicació. Quan aquest menú es tanca, es retorna a la pàgina principal.

- També des de la pàgina principal es pot obrir el menú lateral (ressaltat en groc). Aquest menú ens permet accedir a totes les funcionalitats de l'aplicació, com ara les funcionalitats implementades i el menú de configuració. Quan aquest menú es tanca, es retorna a la pàgina principal.

Com es pot veure a la Figura 14, des de la pàgina principal com des del menú lateral es pot accedir als menús de les funcionalitats de l'aplicació: escaneig de ports (ressaltat en color lila al diagrama) i escaneig de xarxa (ressaltat en color blau).

Cada una d'aquestes funcionalitats té quatre pàgines diferents dins de l'aplicació. Aquestes pàgines es corresponen al següent:

- Inicialment s'accedeix a un menú, que actua com a pàgina principal de la funcionalitat. Des d'aquest menú es pot accedir a les altres tres pàgines: menú de configuració i pàgina on s'executa l'atac.

- El menú de configuració ens permet configurar els diferents paràmetres del qual es compon l'atac. Quan aquest menú es tanca es retorna a la pàgina principal de la funcionalitat.

- Finalment, hi ha la pàgina de realització de l'atac, que és on s'executa la funcionalitat. Quan finalitza, es retorna a la pàgina principal de la funcionalitat.

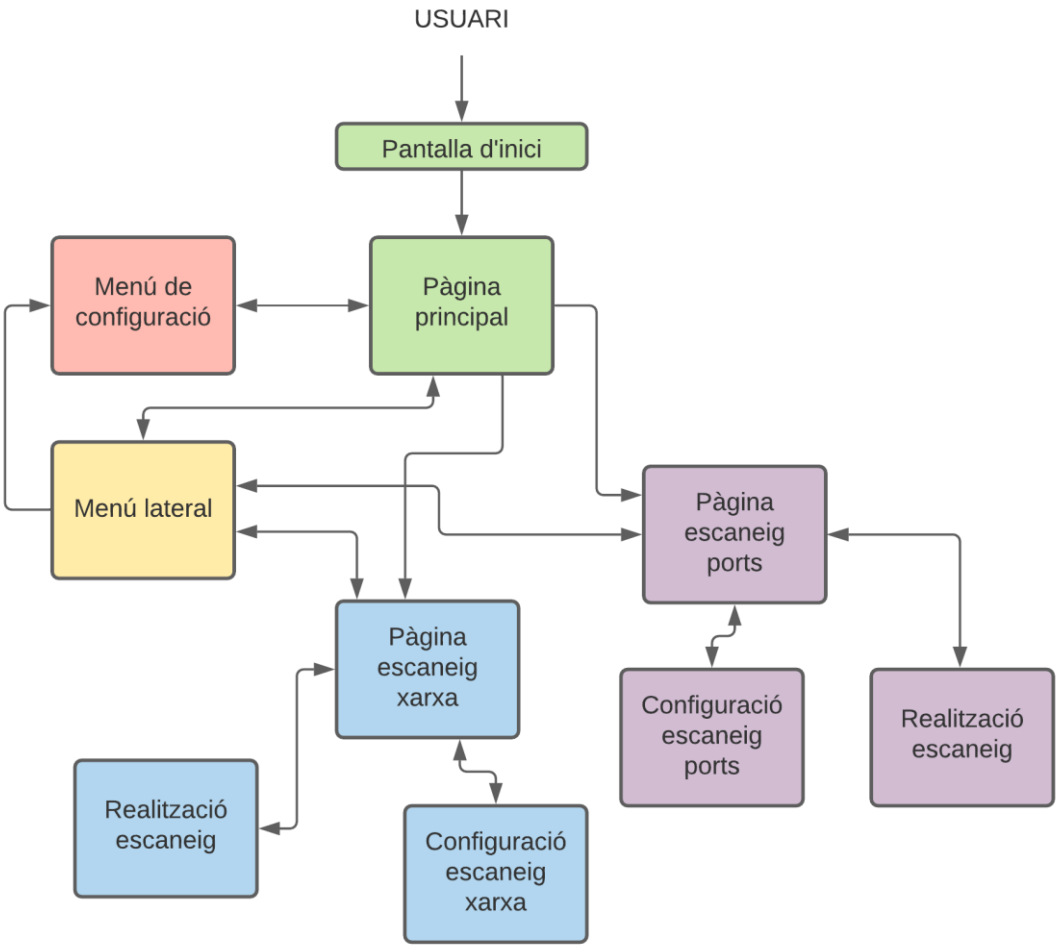

Figura 14. Diagrama de flux de l'aplicació

3.2 Prototipat

Després de dissenyar el flux de l'aplicació s'ha procedit a realitzar-ne el prototipat.

El prototipat d'una aplicació ens permet previsualitzar quin serà el resultat visual de l'aplicació sense necessitat de començar a programar-la. Això ens permet optimitzar el disseny, l'aparença i la usabilitat de l'aplicació, modificant-ne el disseny perquè s'adapti als nostres objectius.

Per a la realització d'aquest prototipat s'ha realitzat uns prototips de baixa qualitat, anomenats *mockup*, de cada una de les pàgines i accions que es poden realitzar a l'aplicació, que ens permeten veure de manera esquemàtica el comportament de l'aplicació.

3.2.1 Pantalla d'inici i pàgina principal

A l'iniciar l'aplicació al telèfon mòbil, s'obrirà una pantalla d'inici, que és una presentació de l'aplicació. Automàticament aquesta pantalla canviarà a la pantalla principal de l'aplicació.

La pantalla principal de l'aplicació ens mostrarà una llista amb les diferents funcionalitats de l'aplicació, que es podran clicar per a poder accedir a elles. A més, la pantalla principal tindrà, a la barra superior, els dos botons d'accés a els menús lateral i de configuració.

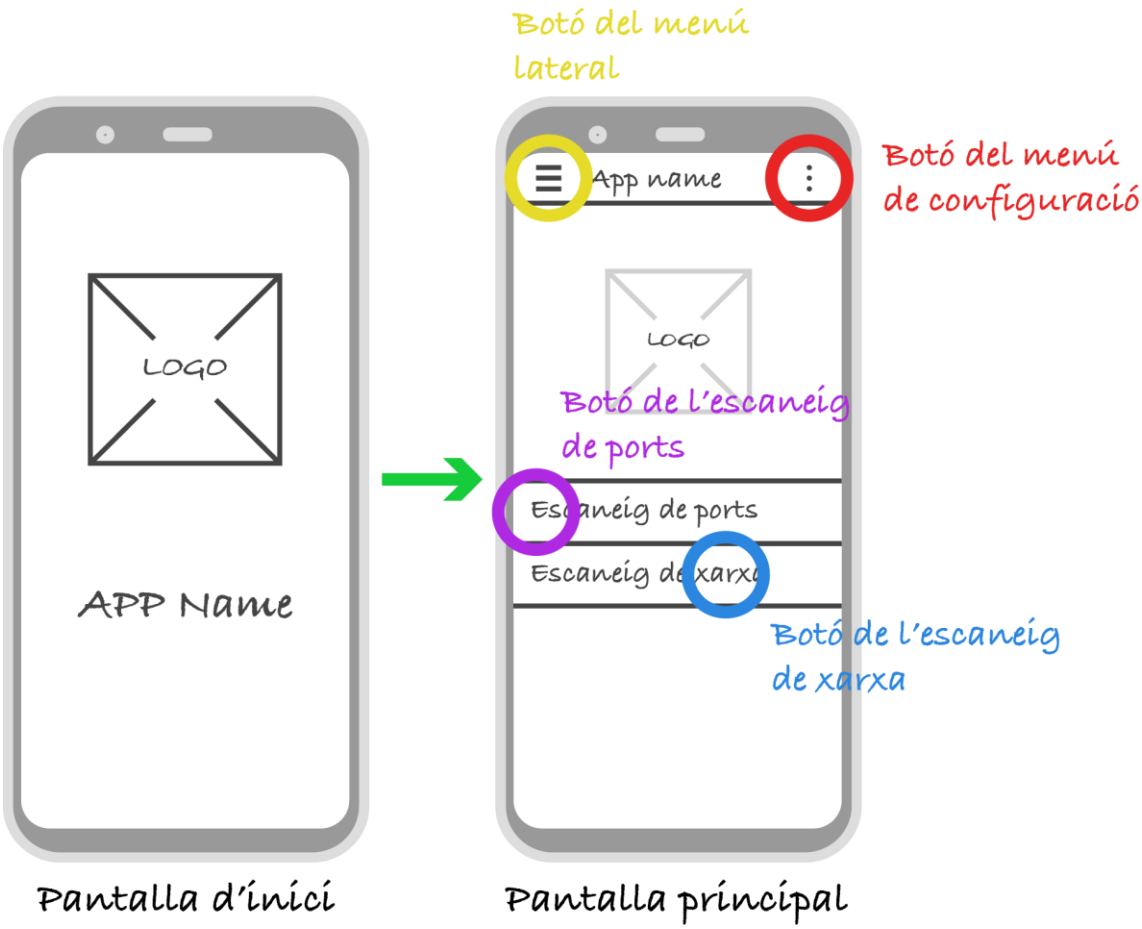

Figura 15. Prototip de la pantalla d'inici i la pantalla principal

3.2.2 Menú de configuració

Des de la pantalla principal de l'aplicació com des del menú lateral es podrà accedir al menú de configuració de l'aplicació.

En aquest menú es podrà configurar els paràmetres generals de l'aplicació, com ara l'idioma, o les configuracions de les diferents funcionalitats de l'aplicació (configuració de l'escaneig de ports i de l'escaneig de xarxa).

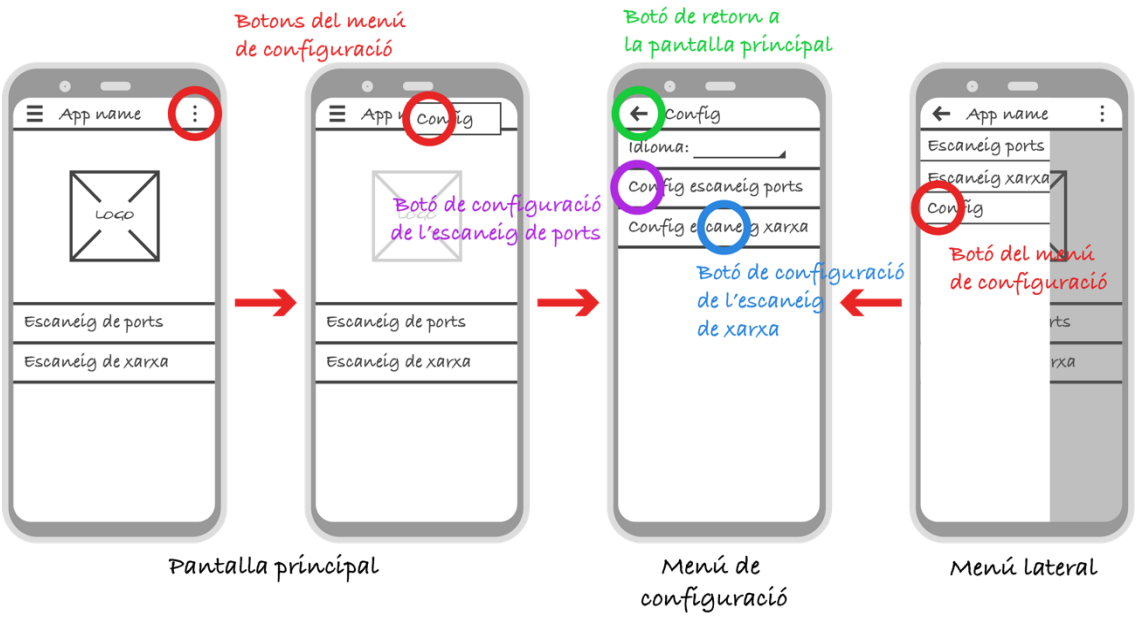

Figura 16. Prototip del menú de configuració

3.2.3 Menú lateral

El menú lateral és un menú addicional que conté el mateix contingut que la pantalla principal, i per tant llista totes les funcionalitats que té l'aplicació.

A aquest menú s'hi pot accedir tant des de la pantalla principal com des de les pantalles generals de cada una de les funcionalitats de l'aplicació (a la Figura 17 es mostra com s'hi pot accedir des de la pantalla principal).

Des d'ell es pot accedir a la pàgina principal dels dos atacs (escaneig de ports i escaneig de xarxa), així com també al menú de configuració general de l'aplicació.

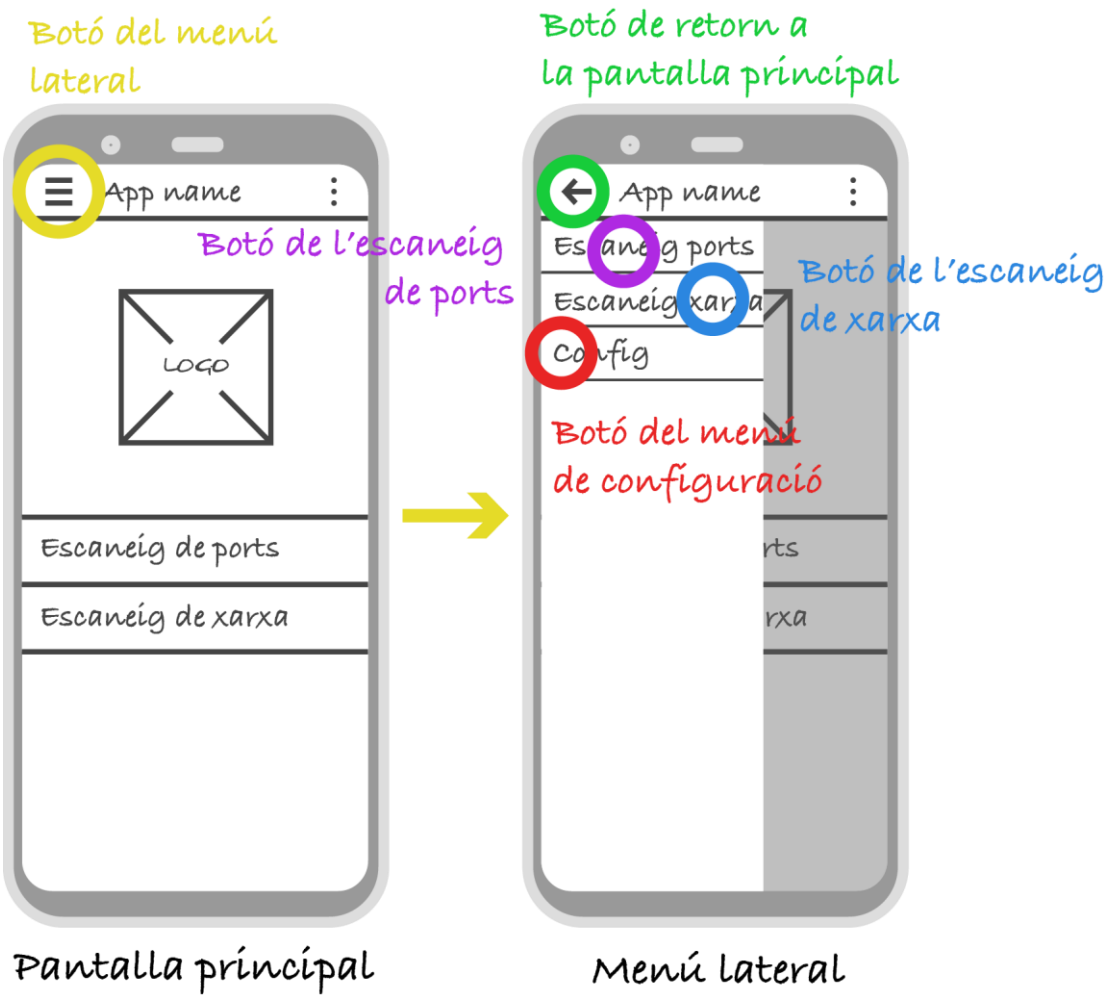

Figura 17. Prototip del menú lateral

3.2.4 Escaneig de ports

La part que es correspon a la funcionalitat de l'escaneig de ports es compondrà de tres pàgines diferents. Aquestes pàgines seran les següents:

- Una pàgina principal que farà de menú de la funcionalitat. És des d'aquesta pàgina que es podrà accedir a les diferents accions de la funcionalitat. A aquesta pantalla s'hi pot accedir tant des de la pantalla principal de l'aplicació com des del menú lateral.

  En aquesta pàgina es mostrarà una llista amb la configuració actual, on s'hi podrà veure les IP d'inici i final de l'escaneig, així com el rang de ports escollits, etc. Aquests valors no es podran modificar en aquesta pantalla, donat que només es podran modificar al menú de configuració de l'atac.

  També es mostraran els botons del menú lateral i del menú de configuració, que estaran situats a la barra superior de la pantalla. Per últim, es mostrarà un botó d'inici d'un nou escaneig.

Figura 18. Prototip de la pantalla d'escaneig de ports

- Una pàgina que serà el menú de configuració de l'atac. En aquest menú es podran modificar els paràmetres de configuració de l'atac.

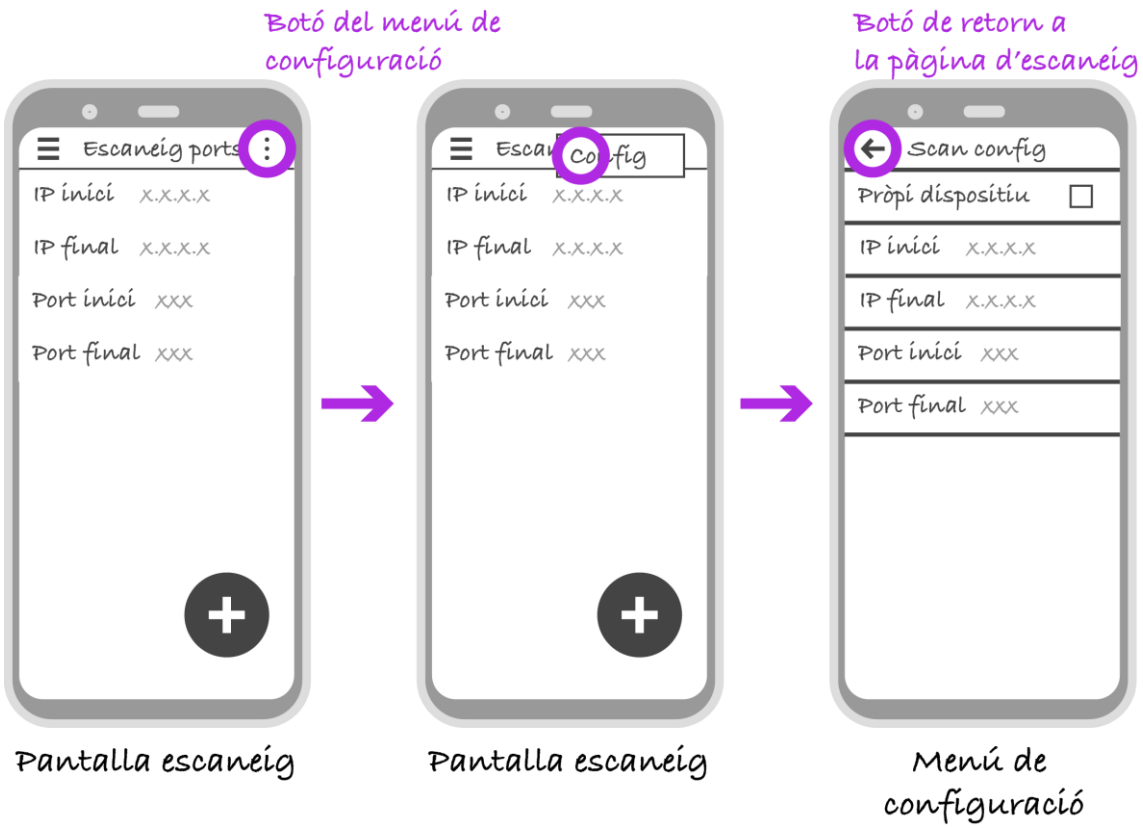

Figura 19. Prototip de la configuració de l'escaneig de ports

- Per últim, una pàgina que serà la pàgina on es mostrarà l'escaneig en curs.
  Aquesta pàgina mostrarà una llista dels ports analitzats per l'escaneig, indicant quin és l'estat en què es troba.

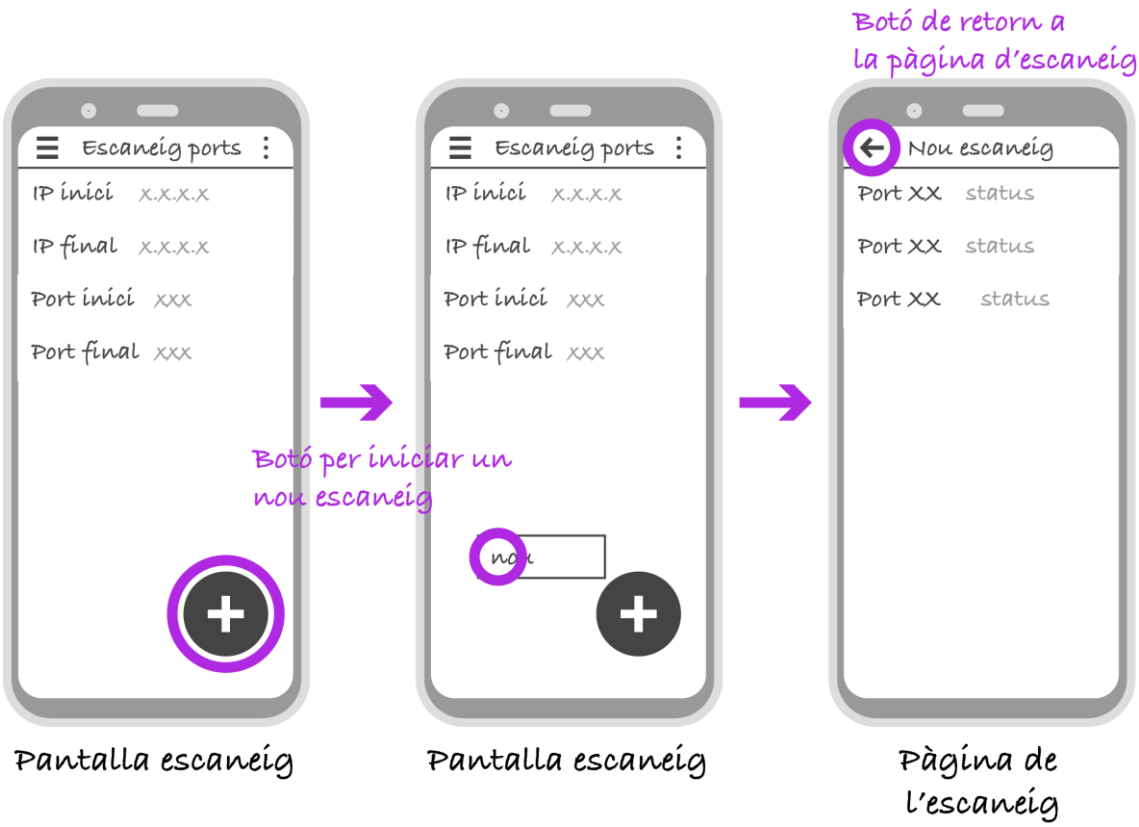

Figura 20. Prototip de la pàgina de nou escaneig de ports

3.2.5 Escaneig de xarxa

El disseny de la part que es correspon a l'escaneig de xarxa és molt semblant al que hem vist a la secció 3.2.4 Escaneig de ports en referència a l'escaneig de ports, ja que es correspondrà en tres pàgines diferents, cada una per a implementar les diferents funcionalitats. Aquestes tres pàgines són les següents:

- Una pàgina principal que farà de menú de la funcionalitat. És des d'aquesta pàgina que es podrà accedir a les diferents accions de la funcionalitat. A aquesta pantalla s'hi pot accedir tant des de la pantalla principal de l'aplicació com des del menú lateral.

  En aquesta pàgina es mostrarà una llista on s'hi podrà veure l'adreça IP que es vol escanejar així com la màscara de xarxa.

  També es mostraran els botons del menú lateral i del menú de configuració, que estaran situats a la barra superior de la pantalla. Per últim, es mostrarà un botó d'inici d'un nou escaneig.

Figura 21. Prototip de la pantalla principal de l'escaneig de xarxa

- Una pàgina que serà el menú de configuració de l'atac. En aquest menú es podran modificar els paràmetres de l'adreça IP i la màscara de xarxa.

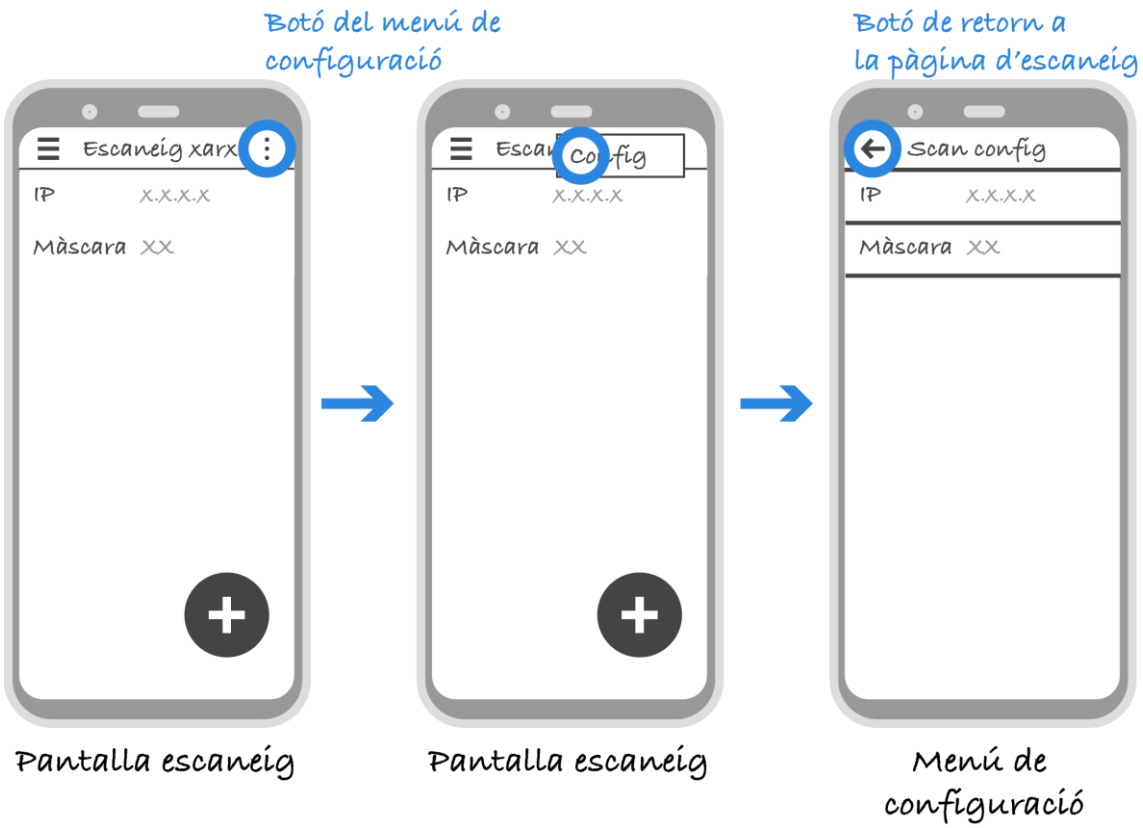

Figura 22. Prototip de la configuració de l'escaneig de xarxa

- Per últim, una pàgina que serà la pàgina on es mostrarà l'atac en curs.

  Aquesta pàgina ens mostrarà una llista amb els resultats de l'atac.

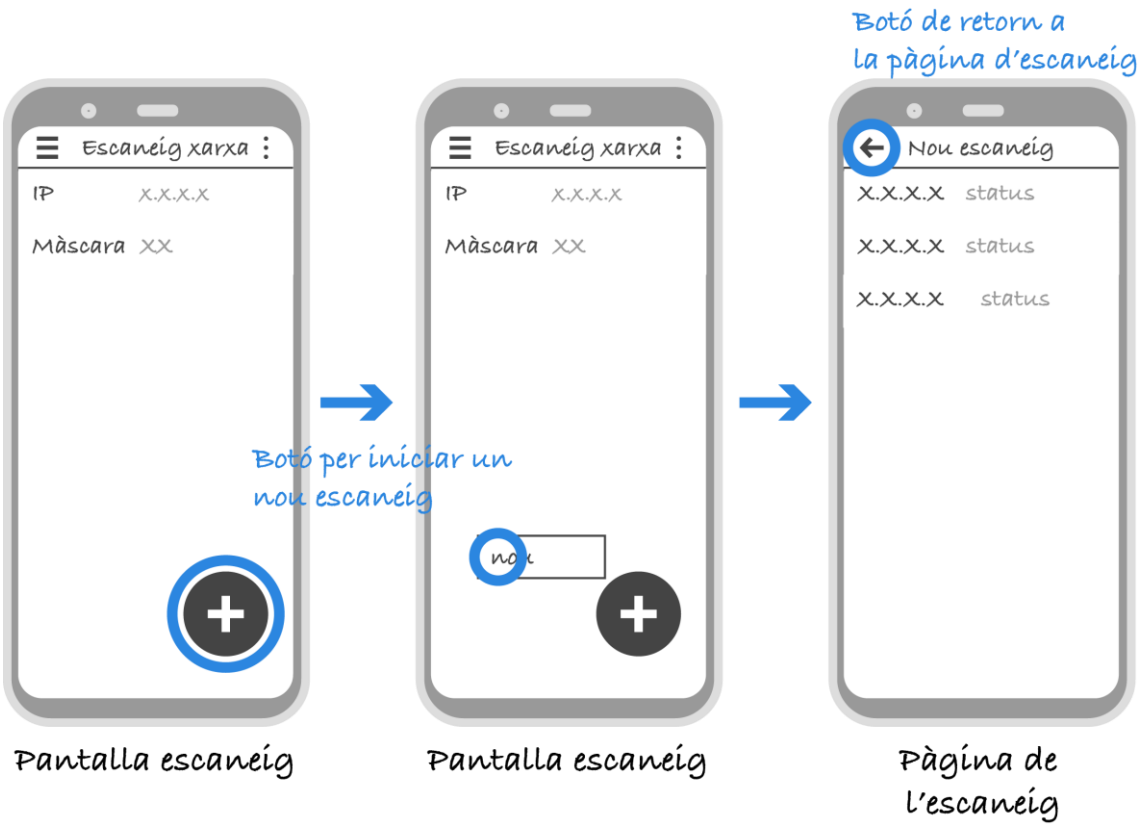

Figura 23. Prototip de la pàgina d'un nou escaneig de xarxa

4. Implementació

4.1 Eines utilitzades

Durant la fase d'implementació de l'aplicació s'han utilitzat una sèrie d'eines i APIs indispensables per a la realització de l'aplicació tal i com ha estat dissenyada.

4.1.1 Android Studio

Per a desenvolupar el projecte s'ha utilitzat l'eina *Android Studio*, atès que com s'ha especificat anteriorment, l'aplicació està basada en el sistema operatiu Android.

Android Studio és l'entorn de desenvolupament integrat (*IDE – Integrated Development Environment*) per al desenvolupament d'aplicacions Android natives, basat en l'eina *IntelliJ IDEA*. *Android Studio* ofereix als desenvolupadors, a part de l'editor dels fitxers de codi, una sèrie de funcions que augmenten la productivitat durant la compilació de les aplicacions.

A més, l'eina proporciona un simulador de l'aplicació que permet visualitzar i interactuar amb l'aplicació programada sense la necessitat d'instal·lar-la en el dispositiu mòbil.

Figura 24. Logotip de l'eina Android Studio

4.1.2 API java.net.InetAddress

Aquesta API és una API utilitzada principalment per a representar adreces IP. Una instància de la classe representa una única adreça.

A l'aplicació, les instàncies d'aquesta classe s'han creat utilitzant el constructor *InetAddress.getByName()*, passant com a paràmetre una representació en format *String* de l'adreça.

Els mètodes d'aquesta classe que s'han utilitzat per al desenvolupament de l'aplicació són els següents:

- Mètode *isReachable()*: Comprova si l'adreça és accessible o no.

4.1.3 API java.net.Socket

Aquesta API implementa el *socket* del client per tal de poder crear una connexió TCP a un port d'una altra màquina. Una instància de la classe representa una connexió entre la màquina client i la màquina servidor.

A l'aplicació, les instàncies es creen amb el constructor *Socket()*, passant per paràmetre l'adreça IP de la màquina a analitzar i el port.

Els mètodes utilitzats d'aquesta classe són els següents:

- Mètode *isConnected()*: retorna l'estat de la connexió entre les dues màquines.

4.1.4 API com.github.doyaaaaaken.kotlincsv.dsl.csvReader

Aquesta API implementa la funcionalitat de lectura i escriptura de fitxers CSV per a Kotlin.

Les instàncies d'aquesta API s'han creat a través del constructor *csvReader().open()*, passant com a paràmetre el fitxer CSV.

4.1.5 API java.net.DatagramSocket

Aquesta API implementa el *socket* necessari per a l'enviament de datagrames d'una màquina client a una màquina servidor. D'aquesta manera es pot establir una connexió UDP. Una instància de la classe representa la connexió entre les màquines.

A l'aplicació, les instàncies d'aquesta classe es creen amb el constructor *DatagramSocket()*.

Els mètodes utilitzats d'aquesta classe són els següents:

- Mètode *setSoTimeout()*: estableix el temps màxim d'espera de resposta des de la màquina servidor.
- Mètode *connect()*: crea una connexió entre les dues màquines.
- Mètode *send()*: Envia un datagrama a la màquina servidor.
- Mètode *receive()*: Rep un datagrama de la màquina servidor.
- Mètode *close()*: Tanca el *socket*.

4.1.6 API java.net.DatagramPacket

Aquesta API s'utilitza per a representar els paquets dels datagrames. Una instància d'aquesta classe representa el datagrama a enviar.

Les instàncies s'han creat utilitzant el constructor *DatagramPacket()*, passant per paràmetre el missatge, la seva mida, l'adreça IP de destí i el port de destí.

4.2 Implementació de l'aplicació

L'aplicació s'ha desenvolupat de forma nativa utilitzant l'eina *Android Studio*, i és compatible amb els dispositius que tinguin una versió Android *4.1 – Jelly Bean* o superior. Això permetrà que aquesta aplicació es pugui utilitzar en el 99.8% dels dispositius Android que hi ha al mercat, segons la informació proporcionada pel mateix *Android Studio*.

4.2.1 Implementació

En aquesta secció s'explicarà com s'ha realitzat la implementació de cada una de les pàgines en què consta l'aplicació.

Pàgina principal

La pàgina principal de l'aplicació es compon de dos passos diferenciats. Per una banda trobem la pantalla d'inici, o *splash screen*. Per l'altra tenim la pantalla principal de l'aplicació.

Pel què fa a la pantalla d'inici, aquesta serà una pantalla amb el logotip de l'aplicació que es mostrarà durant un breu instant com a presentació de l'aplicació. Per a realitzar això s'ha creat una nova activitat amb el nom de *SplashScreenActivity*, que es guarda al fitxer *SplashScreenActivity.kt*. En aquesta activitat s'ha creat la classe ***SplashScreenActivity***, que conté únicament el mètode ***onCreate()***. Aquest mètode crida inicialment a la funció *setContentView()*, que especifica el *layout* de la pantalla (definit al fitxer *activity_splashscreen_layout.xml*). Seguidament s'espera durant el temps definit i executa l'activitat de la pantalla principal de l'aplicació.

Respecte a aquesta pantalla principal, s'ha creat una nova activitat amb el nom *MainPageActivity* (guardada al fitxer *MainPageActivity.kt*). Per a la implementació de l'activitat s'ha creat la classe ***MainPageActivity***, que conté els mètodes següents:

- Mètode ***onCreate()***
- Mètode ***onCreateOptionsMenu()***
- Mètode ***onPostCreate()***
- Mètode ***onOptionsItemSelected()***
- Mètode ***onNavigationItemSelected()***

El mètode principal és el mètode ***onCreate()***. Dins d'aquest mètode es realitzen les següents accions:

- Inicialment es fa una crida a la funció *setContentView()*, que mostrarà el layout de la pantalla. El layout establert per la pantalla principal està definit al fitxer *activity_mainpage_layout.xml*.

- Seguidament es crea una llista amb la classe pròpia d'Android *ListView*, on es mostraran les funcionalitats de l'aplicació. Des d'aquest llistat es pot accedir a les pàgines principals de cada una de les funcionalitats.

  Per a poder mostrar correctament els elements de la llista s'ha creat un adaptador, definit al fitxer *MainPageListAdapter.kt* (i que podem veure a la secció A.1.3 MainPageListAdapter.kt de l'Annex).

- Finalment es creen la barra de navegació de la pàgina, així com el menú lateral, utilitzant les classes *Toolbar* i *NavigationView*, pròpies d'Android.

El codi font de l'activitat el podem veure a la secció A.1.2 MainPageActivity.kt. Per altra banda, el codi font de la pantalla d'inici o *splash screen* el podem veure a la secció A.1.1 SplashScreenActivity.kt. La ens permet veure com es mostra la pantalla principal de l'aplicació.

Figura 25. Representació de la pantalla principal de l'aplicació

Menú configuració

Pel menú de configuració principal s'ha creat una activitat amb el nom de *MainPagePreferencesActivity*. Aquesta activitat s'ha guardat al fitxer *MainPagePreferencesActivty.kt*.

En aquesta activitat s'ha creat la classe ***MainPagePreferencesActivity***, que conté els següents mètodes:

- Mètode ***onCreate()***
- Mètode ***onOptionsItemSelected()***

El principal mètode de la classe és el mètode ***onCreate()***. Dins d'aquest mètode es defineix la implementació del menú. Inicialment es crida la funció *setContentView()*, on s'hi especifica el *layout* de la pantalla, que es correspon al *layout* definit al fitxer *activity_mainpagepreferences_layout.xml*.

Seguidament es crea una llista amb la classe *ListView* pròpia d'Android. En aquesta llista s'hi introdueixen dos elements, que es corresponen cada una de les funcionalitats implementades a l'aplicació: l'escaneig de xarxa i l'escaneig de ports. Pitjant als elements de la llista s'accedeix a la pàgina de configuració de la funcionalitat.

Per a poder introduir els valors i poder-los visualitzar a la pantalla, s'ha utilitzat l'adaptador creat per a la pantalla principal (definit al fitxer *MainPageListAdapter.kt*).

A la secció A.2.1 MainPagePreferencesActivity.kt de l'Annex es pot veure el codi font de la implementació. La Figura 26 mostra com es veu el menú de configuració a l'aplicació.

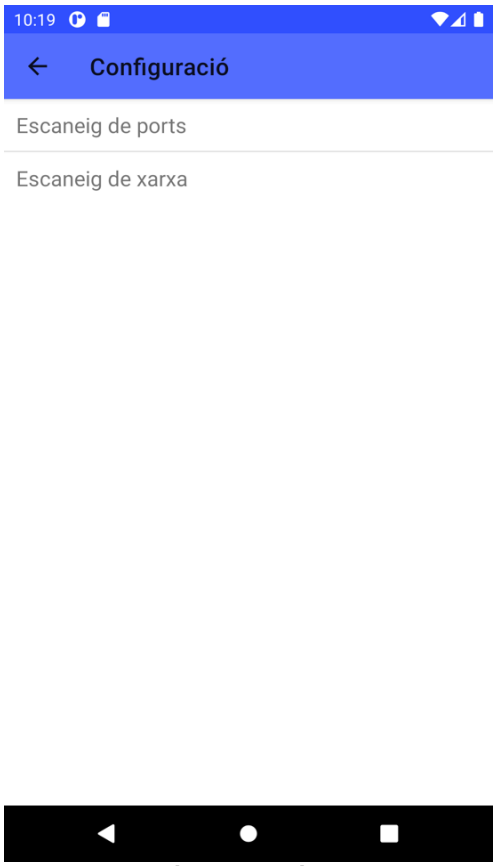

Figura 26. Representació del menú de configuració principal

Menú lateral

Pel què fa a la implementació del menú lateral, aquest s'implementa al mètode principal de les activitats on es mostrarà, que en aquest cas és la pantalla principal, la pantalla principal de l'escaneig de xarxa (*MainNetworkScannerActivity*) i la pantalla principal de l'escaneig de ports (*MainPortScannerActivity*). Per a la implementació s'ha utilitzat la classe *NavigationView* pròpia d'Android.

La Figura 27 mostra com és la representació del menú lateral dins de l'aplicació.

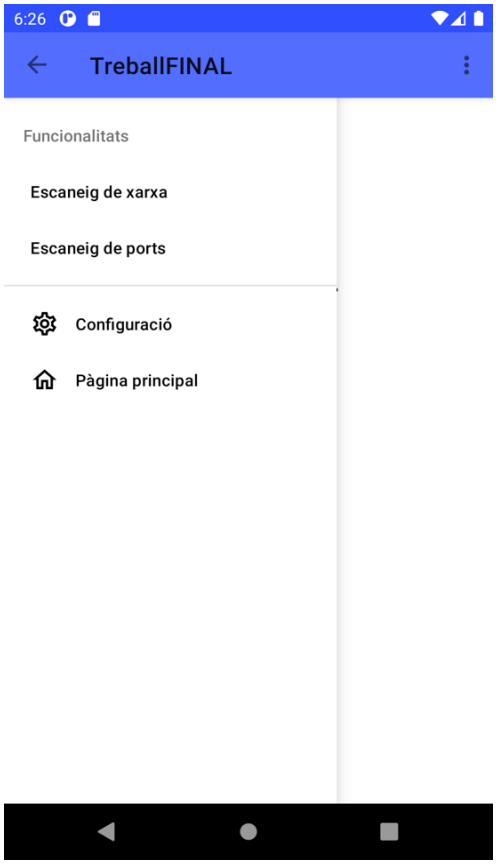

Figura 27. Representació del menú lateral a l'aplicació

Escàner de xarxa

La funcionalitat de l'escàner de xarxa dins de l'aplicació es correspon en tres activitats diferents, que s'explicaran a continuació:

- *MainNetworkScannerActivity*
- *NetworkScannerPreferencesActivity*
- *NetworkScannerActivity*

De cara a la realització de la pàgina d'escaneig de xarxa, s'ha creat una activitat amb el nom de *MainNetworkScannerActivity*, que serà l'activitat base de la funcionalitat. Aquesta activitat s'ha guardat al fitxer *MainNetworkScannerActivity.kt*.

Per a la implementació de l'activitat, s'ha creat una classe amb el nom de ***MainNetworkScannerActivity***, que conté els mètodes necessaris per a l'execució de la funcionalitat. Aquests mètodes són els següents:

- Mètode ***onCreate()***
- Mètode ***onCreateOptionsMenu()***
- Mètode ***onPostCreate()***
- Mètode ***onConfigurationChanged()***
- Mètode ***onOptionsItemSelected()***
- Mètode ***onNavigationItemSelected()***

El mètode principal de la classe és el mètode ***onCreate()***. Aquest mètode s'executarà al iniciar-se l'activitat. Dins d'aquest mètode es realitzen les següents accions:

- En els primers passos del mètode es crida la funció *setContentView()*, on s'hi especifica el *layout* de la pantalla. En aquest cas, es correspon al *layout* definit al fitxer *activity_mainnetworkscanner_layout.xml*.

- Seguidament s'obté els valors de les preferències de l'activitat. Inicialment es carreguen les preferències amb la funció *PreferenceManager.getDefaultSharedPreferences()* i s'assignen a la variable *sharedPreferences*. Seguidament s'obtenen els valors de les preferències de l'escaneig de xarxa cridant a la funció *SharedPreferences.getString()*, passant com a paràmetre de la funció el nom de la preferència.

- A continuació es crea la llista que mostrarà a la pantalla les opcions o valors de preferències que hi haurà al moment de l'execució. Per això s'utilitza la classe *ListView* pròpia d'Android. Per a poder introduir els valors que tindrà la llista, s'ha creat un model específic per aquesta llista. Aquest model està definit al fitxer *MainNetworkScannerListModel.kt* (com podem veure a la secció A.3.2 MainNetworkScannerListModel.kt). Per a la utilització d'aquest nou model, s'ha creat un *adaptar*, necessari per a la visualització de la llista. Aquest adaptador s'ha creat al fitxer *MainNetworkScannerListAdapter.kt* (com podem veure a la secció A.3.3 MainNetworkScannerListAdapter.kt).

- Seguidament s'ha creat la barra de navegació de la pàgina així com el menú lateral. Per a aquests elements, s'ha utilitzat les classes *Toolbar* i *NavigationView* pròpies d'Android.

- Per últim, s'ha creat el botó flotant que permet començar l'execució de la funcionalitat. Per aquest botó s'ha utilitzat la classe *View* pròpia d'Android.

El codi font d'aquesta activitat el podem veure a la secció A.3.1 MainNetworkScannerActivity.kt de l'Annex. La Figura 28 ens permet veure com es mostrarà la pantalla principal de l'escaneig de xarxa a l'aplicació.

Figura 28. Pantalla principal de l'escaneig de xarxa

Pel què fa a la pantalla de preferències de la funcionalitat, s'ha creat una activitat anomenada *NetworkScannerPreferencesActivity*. Aquesta activitat es guarda al fitxer *NetworkScannerPreferencesActivity.kt*.

L'activitat es crida des de la pantalla principal de l'escaneig de xarxa, concretament des de la barra de navegació.

Per a la implementació de l'activitat, s'ha creat una classe amb el nom de ***NetworkScannerPreferencesActivity***, que conté els mètodes necessaris per a l'execució de la funcionalitat. Aquests són els següents:

- Mètode ***onCreate()***
- Mètode ***onOptionsItemSelected()***
- Classe ***SettingsFragment***

El mètode principal de la classe és el mètode ***onCreate()***, que s'executa a l'iniciar-se l'activitat. Dins del mètode es crea, primerament, la barra de navegació, on s'hi especifica el botó de retorn a la pantalla principal. Seguidament es crea el *Fragment* que mostrarà les preferències, utilitzant la classe *FragmentManager* pròpia d'Android. Per a crear el fragment es fa una crida a la classe interna ***SettingsFragment***.

La classe interna ***SettingsFragment*** conté únicament un sol mètode, anomenat ***onCreatePreferences()***, que realitza una crida al fitxer que conté les preferències de l'escaneig de xarxes, anomenat *preferences_mainnetworkscanner.xml*. Les preferències són les següents:

- *Adreça IP*, que es correspon una adreça IP de la xarxa que es vol analitzar.
- *Màscara de xarxa*, que és un selector de la màscara de xarxa que s'utilitzarà a l'escaneig. En aquest cas, com s'ha dit anteriorment, es poden utilitzar les màscares /8, /16 i /24.

El codi font de l'activitat el podem veure a la secció A.3.4 NetworkScannerPreferencesActivity.kt de l'Annex. La Figura 29 ens mostra com es veu el menú de configuració de l'escaneig de xarxa.

Figura 29. Pantalla del menú de configuració de l'escaneig de xarxa

Finalment, l'última pantalla de la funcionalitat de l'escaneig de ports és la pantalla on es realitza aquest escaneig. Per a aquest fi, s'ha creat una activitat amb el nom de *NetworkScannerActivity*. Aquesta activitat s'ha guardat al fitxer *NetworkScannerActivity.kt*.

Per a la implementació de l'activitat, s'ha creat una classe amb el nom de ***NetworkScannerActivity***, que conté els mètodes necessaris per a l'execució de la funcionalitat. Aquests mètodes són els següents:

- Mètode ***onCreate()***
- Mètode ***onOptionsItemSelected()***
- Mètode ***getSubnet()***
- Mètode ***netMask8()***
- Mètode ***netMask16()***
- Mètode ***netMask24()***
- Mètode ***isDevice()***

El mètode principal de l'activitat és el mètode ***onCreate()***, on es realitzen les següents accions:

- Inicialment es crida a la funció *setContentView()* on s'hi especifica el *layout* de la pantalla. El *layout* que s'utilitza és el que està definit al fitxer *activity_networkscanner_layout.xml*.

- Seguidament s'ha creat la barra de navegació de la pàgina. Això s'ha fet utilitzant la classe *ActionBar* pròpia d'Android.

- A continuació s'ha creat una barra de progrés per tal de que es mostri mentre s'està executant l'escaneig, a mode d'espera. Aquesta barra de progrés serà circular, i desapareixerà quan es mostrin els resultats de l'escaneig. Per a crear la barra de progrés, s'ha utilitzat la classe *ProgressBar* pròpia d'Android.

- Per últim, s'ha creat tot el procés que permet l'execució de l'escaneig. Aquest procés el podem dividir en quatre punts diferents:

    - En primer lloc es carreguen els valors de la configuració de l'activitat. Tal i com s'ha explicat en referència a la pantalla principal de l'escaneig de xarxa, s'ha utilitzat la funció *PreferenceManager.getDefaultSharedPreferences()* per obtenir-los.

    - A continuació es crea una *coroutine*, que realitzarà l'escaneig de xarxa en funció de la configuració establerta. En Kotlin, una corutina s'utilitza per a poder executar dos o més processos en paral·lel. En aquest cas, s'ha creat la corutina per tal de poder executar l'escaneig de xarxa sense que es vegi afectat el funcionament principal de l'aplicació. Per a fer això s'ha creat una corutina asíncrona, utilitzant la funció *CoroutineScope(Dispatchers.IO).async{}*. A dins de la corutina es realitza una crida al mètode ***getSubnet()*** per tal d'obtenir la subnet de la xarxa. Seguidament, en funció de la màscara escollida, es realitza una crida als mètodes ***netMask8()***, ***netMask16()*** i ***netMask24()***, que són els que executaran l'escaneig de xarxa.

      Com podem veure a la crida de la corutina, s'indica que aquesta s'executarà al *Dispatchers.IO*. Els *dispatchers* és l'indicador de quin procés de l'aplicació serà el que executi la corutina. En aquest cas el procés (o *dispatcher*) que executarà la corutina on es realitza l'escaneig serà el de *IO* (o *input/output*), atès que Android no permet executar operacions de xarxa al procés principal.

    - Addicional a la corutina anterior, s'ha creat una altra corutina que executarà la barra de progrés mentre duri l'escaneig. Aquesta corutina s'ha creat utilitzant la funció *CoroutineScope(Dispatchers.Main).async{}*, i a dins s'ha definit que la barra de progrés sigui visible a la pantalla.

Com podem veure a la crida de la corutina, aquesta s'executarà al procés principal (*Main*) de l'aplicació. D'aquesta manera la barra de progrés es mostrarà per la pantalla mentre que l'escaneig s'executarà en segon terme.

- Finalment, en paral·lel a les altres dues corutines, s'ha creat una tercera corutina, que ens permet agrupar els resultats de les altres dues corutines i mostrar la llista de dispositius per la pantalla. Aquesta corutina s'ha creat fent una crida a la funció *CoroutineScope(Dispatchers.Main).launch{}*. En aquest cas, la corutina no és asíncrona, sinó que s'executarà en primer lloc, i a continuació es farà una crida a les altres dues corutines utilitzant el mètode *await()*. Amb aquest mètode s'esperarà a que les altres dues corutines acabin la seva execució per continuar amb aquesta.

  Un cop finalitzin les dues corutines, es crea una llista amb la classe *ListView* pròpia d'Android, que és a on es mostraran els resultats de l'escaneig. Per a poder introduir els diferents elements de la llista, s'ha creat un model específic, que està definit al fitxer *NetworkScannerListModel.kt* (com podem veure a la secció A.3.6 NetworkScannerListModel.kt). Per a la utilització d'aquest model i per a la visualització de la llista, s'ha creat un adaptador, definit al fitxer *NetworkScannerListModel.kt* (que podem veure a la secció A.3.7 NetworkScannerListAdapter.kt).

  Un cop es mostrin els resultats de l'escaneig, l'usuari pot escollir un dels dispositius llistats per tal de poder realitzar-li l'escaneig de ports, pitjant el dispositiu a la llista.

Com s'ha comentat anteriorment, els mètodes principals on es realitza l'escaneig de xarxa són ***netMask8()***, ***netMask16()*** i ***netMask24()***. En aquest cas realitzarem una explicació de la implementació del mètode ***netMask24()***, atès que la implementació dels mètodes ***netMask8()*** i ***netMask16()*** és la mateixa, amb la única diferència que el volum de dispositius cercats en cada un d'ells varia en funció de la màscara escollida.

El mètode ***netMask24()*** rep com a paràmetre la subnet de la xarxa. El primer que es realitza es una iteració *for*, que recorre tots els valors entre 1 i 254, que és el número de possibles dispositius de la xarxa amb una màscara /24. Seguidament, i dins de la iteració *for*, es construeix la IP a escanejar, ajuntant el valor de la subnet passat per paràmetre amb el valor de la iteració. Un cop tenim l'adreça IP, es crea una variable *InetAddress* utilitzant la funció *InetAddress.getByName()*, passant-l'hi per paràmetre la IP a escanejar. Un cop creada aquesta variable, utilitzem el mètode *InetAddress.isReachable()* per comprovar si el dispositiu amb la IP que escanegem està disponible o no a la xarxa.

Si el dispositiu està disponible, es realitza una comprovació amb el mètode ***isDevice()***, que comprova si és el mateix dispositiu que el de

l'usuari de l'aplicació, per tal d'indicar si ho és. Finalment, s'afegeix el dispositiu a una llista que serà la que es mostrarà per pantalla al finalitzar l'escaneig.

El codi font de l'activitat el podem veure a la secció A.3.5 NetworkScannerActivity.kt de l'Annex. La Figura 30. Resultat de l'escaneig de xarxa ens mostra com es veu el resultat de l'escaneig de xarxa.

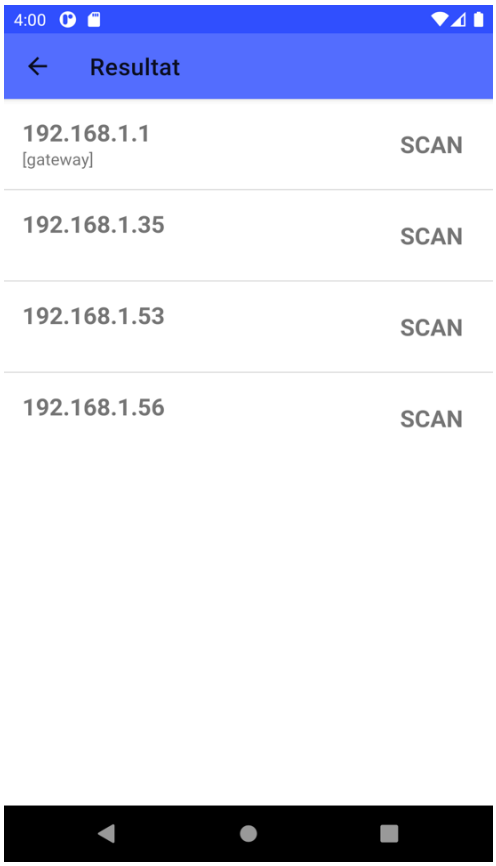

Figura 30. Resultat de l'escaneig de xarxa

Escàner de ports

La funcionalitat de l'escàner de ports dins de l'aplicació es correspon en tres activitats diferents, que s'explicaran a continuació:

- *MainPortScannerActivity*
- *PortScannerPreferencesActivity*
- *PortScannerActivity*

Per a la realització de la pàgina d'escaneig de ports, s'ha creat una activitat amb el nom de *MainPortScannerActivity*. Aquesta és l'activitat base de la funcionalitat, ja que des d'ella es podrà navegar a les altres dues pantalles definides al disseny: el menú de configuració de l'escàner i la pantalla on es mostren els resultats.

L'activitat està guardada en el fitxer *MainPortScannerActivity.kt*. Dins del fitxer s'hi ha creat la classe ***MainPortScannerActivity***, conjuntament amb els mètodes necessaris per a implementar els diferents elements de la pàgina, com ara la barra de navegació o la llista amb la configuració actual de l'escaneig.

Els mètodes necessaris per a la interacció amb l'aplicació i la funcionalitat de cada un dels elements que es mostren a la pàgina són els següents:

- Mètode ***onCreate()***
- Mètode ***onCreateOptionsMenú()***
- Mètode ***onPostCreate()***
- Mètode ***onConfigurationChanged()***
- Mètode ***onOptionsItemSelected()***
- Mètode ***onNavigationItemSelected()***

El mètode principal de la classe és el mètode ***onCreate()***, que s'executarà a l'iniciar la classe. Dins d'aquest mètode es realitzen els següents processos:

- Inicialment es realitza una crida al *layout* de l'activitat, utilitzant la funció *setContentView()*. El *layout* utilitzat es correspon al definit al fitxer *activity_mainportscanner_layout.xml*.

- Seguidament s'obtenen els valors de la configuració actual de l'escaneig (adreça IP, rang de ports a analitzar i el tipus d'escaneig) mitjançant el mètode *SharedPreferences.getString()*.

- Un cop obtinguda la configuració es crea una llista, utilitzant la classe *ListView* pròpia d'Android, que ens servirà per mostrar per pantalla la configuració actual. Per poder introduir les diferents opcions a la llista s'ha creat un model de vista, definit al fitxer *MainPortScannerListModel.kt* (que podem veure a la secció A.4.2 MainPortScannerListModel.kt). A més, per a la visualització de la llista, també s'ha creat un adaptador, definit al fitxer *MainPortScannerListAdapter.kt*, i que podem veure a la secció A.4.3 MainPortScannerListAdapter.kt.

- Seguidament es crea la barra de navegació de la pàgina així com el menú lateral, utilitzant les classes *Toolbar* i *NavigationView* respectivament (les dues són pròpies d'Android).

- Finalment, es crea el botó flotant que ens portarà a la pàgina de realització de l'escaneig, utilitzant la classe *View* pròpia d'Android.

El codi font de l'aplicació el podem veure a l'Annex, a la secció A.4.1 MainPortScannerActivity.kt. La pàgina principal de l'escaneig de ports es veurà a l'aplicació tal i com es mostra a la Figura 31.

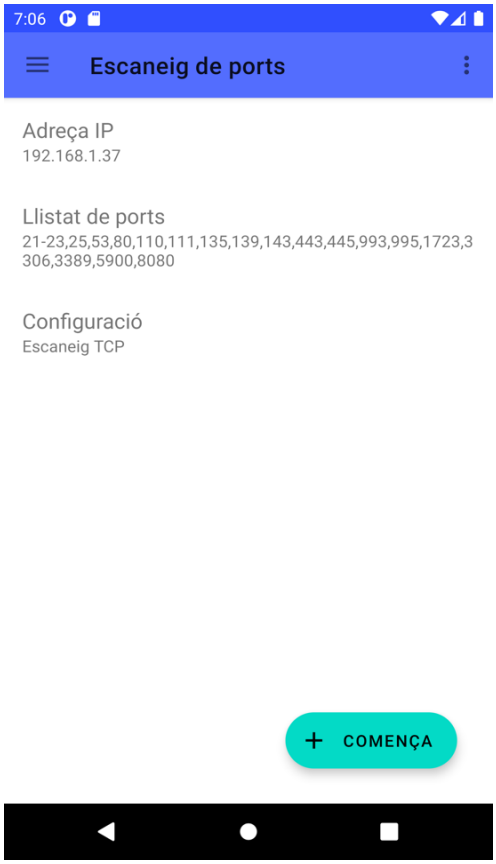

Figura 31. Pantalla principal de l'escaneig de ports en l'aplicació

Respecte al menú de configuració de l'escaneig de ports, s'ha creat una nova activitat anomenada *PortScannerPreferencesActivity*. Aquesta activitat està guardada al fitxer *PortScannerPreferencesActivity.kt*, i és cridada des de la barra de navegació de la pàgina principal de l'escaneig.

Pel què fa al codi del menú de configuració, s'ha creat la classe ***PortScannerPreferencesActivity***, que conté els mètodes necessaris per a la correcta utilització del menú, així com una classe interna anomenada ***SettingsFragment*** que ens permetrà mostrar les opcions del menú.

Els mètodes que implementa la classe són els següents:

- Mètode ***onCreate()***
- Mètode ***onOptionsItemSelected()***
- Classe ***SettingsFragment***

El principal mètode de la classe és el mètode ***onCreate()***. Aquest mètode inicialment crea la barra superior de navegació, per després fer una crida perquè es mostri el *Fragment* que ens proporcionarà el menú de configuració. Aquest *Fragment* està gestionat per la classe ***SettingsFragment***.

Al seu temps, a la classe ***SettingsFragment*** es cridarà, a través del mètode ***onCreatePreferences()***, al fitxer que conté les diferents opcions del menú de configuració, que es correspon al fitxer *preferences_mainportscanner.xml*. Aquestes opcions són les següents:

- *Adreça IP*: es correspon a l'adreça IP de la màquina que es vol analitzar.
- *Llistat de ports*: es correspon al llistat de ports per ser analitzats.
- *Tots els ports*: és un selector que en cas d'estar activat es realitzarà un escaneig de tots els ports de la màquina (de l'1 al 65536). En cas contrari, es realitzarà l'escaneig dels ports introduïts al llistat de ports.
- *Escaneig TCP*: és un selector que s'utilitza per indicar si es realitzarà o no l'escaneig dels ports TCP. Per defecte és l'escaneig que està programat.
- *Escaneig UDP*: és un selector que s'utilitza per indicar si es realitzarà o no l'escaneig dels ports UDP.

El codi font de l'activitat el podem veure a la secció A.4.4 PortScannerPreferencesActivity.kt. El menú de configuració de l'escaneig de ports es mostrarà a l'aplicació tal i com mostra la Figura 32:

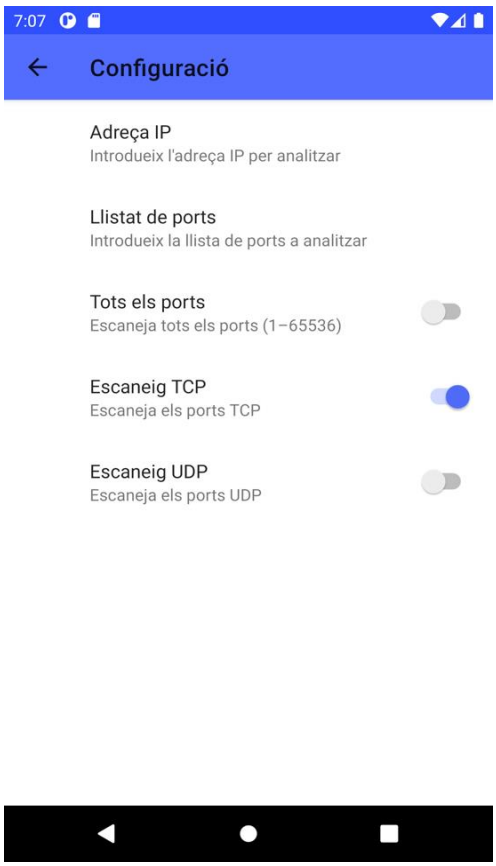

Figura 32. Pantalla del menú de configuració de l'escaneig de ports

Per últim, la pantalla on es realitza l'escaneig de ports. S'ha creat una activitat amb el nom de *PortScannerActivity*. Aquesta activitat s'ha guardat al fitxer *PortScannerActivity.kt*.

Per a la implementació, s'ha creat una classe amb el nom de ***PortScannerActivity***, que conté els mètodes que permeten l'execució de la funcionalitat. Aquests mètodes són els següents:

- Mètode ***onCreate()***
- Mètode ***onOptionsItemSelected()***

- Mètode ***getPortList()***
- Mètode ***scannerTCP()***
- Mètode ***scannerUDP()***

El mètode principal de la classe és el mètode ***onCreate()***. En aquest mètode s'hi executarà el cos principal de la classe, que és el següent:

- Inicialment es crida, a través de la funció *setContentView()*, al *layout* de la classe. Aquest *layout* està definit al fitxer *activity_portscanner_layout.xml*.

- A continuació es crea la barra de navegació de l'activitat, utilitzant la classe pròpia d'Android *ActionBar*.

- Seguidament es crea una llista a partir de la base de dades des d'on es buscarà quin servei s'està utilitzant en els ports oberts de la màquina. Aquesta base de dades està guardada en el fitxer *service-names-port-numbers.csv* i conté una relació entre el número de port, el protocol utilitzat i el nom del servei.

  Per a poder accedir al contingut de la base de dades s'ha fet servir el mètode *csvReader().open()*, on s'hi passa per paràmetre el fitxer CSV de la base de dades. Aquest mètode forma part de la llibreria c*om.github.doyaaaaaken.kotlincsv.dsl.csvReader*, que s'ha utilitzat per a tal fi. Les dades extretes de la base de dades es guarden en dues llistes diferents (una pel protocol TCP i l'altra pel protocol UDP), on s'hi guarden la relació entre número de port i servei.

  Per a poder guardar les dades extretes de la base de dades a la llista, s'ha creat un model per tal de poder tenir les dades més ordenades. Aquest model està definit al fitxer *PortListModel.kt* (com podem veure a la secció A.4.6 PortListModel.kt)

- A continuació s'ha creat la barra de progrés, que es mostrarà mentre s'està executant l'escaneig. Per a crear la barra de progrés s'ha utilitzat la classe *ProgressBar* d'Android.

- Finalment, s'ha creat el procés que implementa l'escaneig de ports. S'ha realitzat el mateix sistema de corutines que l'explicat per a la funcionalitat de l'escaneig de xarxa. S'han realitzat tres corutines diferents, dues de les quals s'executaran en paral·lel (la que realitza l'escaneig i la que mostra per pantalla la barra de progrés), mentre que la tercera serveix per mostrar els resultats finals a la pantalla. El procés de l'escaneig s'explica a continuació:

  - En primer lloc es carreguen els valors de cada un dels paràmetres de configuració. Això es fa utilitzant la funció *PreferenceManager.getDefaultSharedPreferences()*.

  - A continuació es crea la primera corutina, que realitza l'escaneig de ports. Aquesta corutina s'executa al procés de IO de l'aplicació, i es crea amb la funció

CoroutineScope(Dispatchers.IO).async{}. Dins de la corutina es realitza una crida als mètodes ***scannerTCP()*** i ***scannerUDP()*** en funció de la configuració escollida.

- Seguidament es crea la segona corutina, que és la que mostrarà per la pantalla principal la barra de progrés. Aquesta corutina es crea amb la funció *CoroutineScope(Dispatchers.Main).async{}*.

- Per últim, es crea la tercera corutina, que executarà les altres dues i mostrarà els resultats per la pantalla principal, un cop les dues corutines hagin acabat el seu procés. Aquesta tercera corutina s'ha creat utilitzant la funció *CoroutineScope(Dispatchers.Main).launch{}*.

  Un cop finalitzin les dues corutines, es crea una llista amb la classe *ListView*. Per tal de poder introduir cada un dels resultats a la llista, s'ha creat un model específic, que està definit al fitxer *PortScannerListModel.kt*, i que podem veure a la secció A.4.7 PortScannerListModel.kt. També, per poder mostrar els resultats, s'ha hagut de crear un adaptador específic per a la llista. Aquest adaptador està definit al fitxer *PortScannerListAdapter.kt* (i que podem veure a la secció A.4.8 PortScannerListAdapter.kt).

Tal i com s'ha comentat anteriorment, l'escaneig de ports es realitza als mètodes ***scannerTCP()*** i ***scannerUDP()***, que són cridats durant el mètode ***onCreate()***. Aquests dos mètodes s'implementen de la següent manera:

- El mètode ***scannerTCP()*** rep com a paràmetres l'adreça IP, el port a analitzar i la llista de serveis TCP. El primer que es realitza és una connexió entre el dispositiu on s'està executant l'aplicació i la màquina a analitzar. Per a fer això s'ha utilitzat la classe *Socket* de Java, on s'hi passa per paràmetre l'adreça IP de la màquina analitzada i el port. Aquesta classe realitza una connexió TCP al port especificat. Seguidament es comprova si la connexió s'ha establert correctament utilitzant la funció *Socket.isConnected()*.

  En el cas que la connexió s'hagi establert correctament, es considera que el port està obert. En aquest cas, s'obté el possible servei de la llista de serveis TCP i s'introdueix el resultat a la llista de resultats que es mostrarà per pantalla.

- El mètode ***scannerUDP()*** rep els mateixos paràmetres que el mètode anterior, canviant la llista de serveis pels del protocol UDP. El primer que es realitza és la creació del *socket* que permetrà l'enviament de paquets a la màquina a analitzar. Això es fa creant una instància de la classe *DatagramSocket*.

  Seguidament es crea el paquet a enviar, creat una instància de la classe *DatagramPacket()*, i passant-li per paràmetre el missatge, la seva mida, l'adreça IP de destí i el port de destí.

Un cop creat el datagrama, es configura el temps d'espera d'una resposta per part de la màquina de destí, utilitzant el mètode *DatagramSocket.setSoTimeout()*. A continuació es realitza una connexió entre les dues màquines amb la funció *DatagramSocket.connect()*, i s'envia el datagrama amb el mètode *DatagramSocket.send()*.

Per últim, es fa una crida al mètode *DatagramSocket.receive()* per tal d'esperar una resposta, i es tanca la connexió amb el mètode *DatagramSocket.close()*.

En el cas que la resposta al datagrama sigui *ICMP Port Unreachable*, es considerarà que el port està filtrat. Per altra banda, si no es rep cap resposta, es considerarà que està tancat.

Finalment, en funció del resultat (si aquest és obert o filtrat) es buscarà a la llista de serveis quin és el possible servei que s'està executant en aquell port i s'introduirà el resultat a la llista de resultats.

El codi font de l'activitat el podem veure a la secció A.4.5 PortScannerActivity.kt. La ens mostra com es mostra el resultat de l'escaneig a l'aplicació.

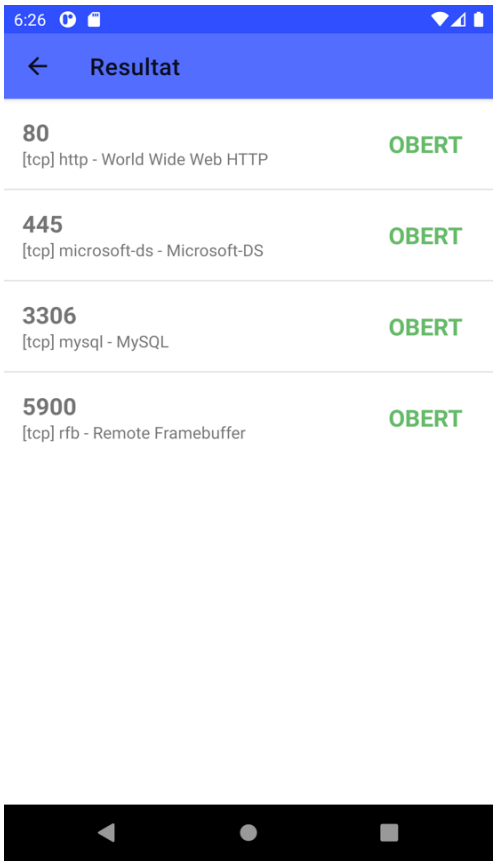

Figura 33. Resultat de l'escaneig de ports

4.2.2 Estructura de carpetes

L'aplicació dissenyada segueix el següent format d'estructura de carpetes.

En primer lloc veurem llistat els documents del codi font, que es troben al directori *app/src/main/java/android/app/tfmuoc001/*:

- *MainNetworkScannerActivity.kt*
- *MainNetworkScannerListAdapter.kt*
- *MainNetworkScannerListModel.kt*
- *MainPageActivity.kt*
- *MainPageListAdapter.kt*
- *MainPagePreferencesActivity.kt*
- *MainPortScannerActivity.kt*
- *MainPortScannerListAdapter.kt*
- *MainPortScannerListModel.kt*
- *NetworkScannerActivity.kt*
- *NetworkScannerListAdapter.kt*
- *NetworkScannerListModel.kt*
- *NetworkScannerPreferencesActivity.kt*
- *PortListModel.kt*
- *PortScannerActivity.kt*
- *PortScannerListAdapter.kt*
- *PortScannerListModel.kt*
- *PortScannerPreferencesActivity.kt*
- *SplashScreenActivity.kt*

En segon lloc veurem els diferents fitxers utilitzats per a poder representar les pantalles de l'aplicació, com ara els layouts, menús o preferències:

- Directori *app/src/main/res/layout/*
    - *actionbar_mainnetworkscanner_layout.xml*
    - *actionbar_mainpage_layout.xml*
    - *actionbar_mainportscanner_layout.xml*
    - *activity_mainnetworkscanner_layout.xml*
    - *activity_mainpage_layout.xml*
    - *activity_mainpagepreferences_layout.xml*
    - *activity_mainportscanner_layout.xml*
    - *activity_networkscanner_layout.xml*
    - *activity_portscanner_layout.xml*
    - *activity_splashscreen_layout.xml*
    - *listview_header_layout.xml*
    - *listview_mainnetworkscanner_layout.xml*
    - *listview_mainpage_layout.xml*
    - *listview_mainportscanner_layout.xml*
    - *listview_networkscanner_layout.xml*
    - *listview_portscanner_layout.xml*
    - *navigationview_layout.xml*
- Directori *app/src/main/res/menu/*:
    - *actionbar_mainnetworkscanner_menu.xml*
    - *actionbar_mainpage_menu.xml*
    - *actionbar_mainportscanner_menu.xml*
    - *navigationview_menu.xml*
- Directori *app/src/main/res/vàlues/*:
    - *arrays.xml*

    - *colors.xml*
    - *dimens.xml*
    - *strings.xml*
    - *styles.xml*
- Directori *app/src/main/res/xml/*:
    - *preferences_mainnetworkscanner.xml*
    - *preferences_mainportscanner.xml*

Respecte a la base de dades dels serveis possibles en cada un dels ports, aquesta es guarda al fitxer *service-names-port-numbers.csv*, que es troba al directori *app/src/main/assets/*.

Per últim, el manifest d'Android, necessari per tal de declarar totes les activitats de l'aplicació, es troba al directori *app/src/main/*.

4.2.3 Codi font

L'annex ens mostra la part del codi font referent als fitxers en llenguatge Kotlin de l'aplicació.

Tot i això, l'aplicació completa realitzada en aquest projecte es pot trobar al repositori de GitHub de l'alumne. L'enllaç del projecte és el següent: https://github.com/msaladelafont/TFMUOC001

5. Conclusions

A la fase d'investigació s'han establert una sèrie d'objectius a realitzar durant el projecte. Aquests objectius s'han pogut assolir amb més o menys mesura, degut a les dificultats que han sorgit durant el desenvolupament del projecte, que han estat relacionades amb els riscos que també s'han analitzat prèviament.

L'objectiu principal del projecte fa referència a la realització d'una aplicació per a dispositius mòbils que permeti la detecció i anàlisi de vulnerabilitats en un servidor. Després de la realització del projecte es pot concloure que aquest objectiu s'ha assolit completament, ja que s'ha pogut realitzar una aplicació mòbil que permet, a través d'un escaneig de ports, detectar quins serveis s'estan utilitzant al servidor, fent possible detectar possibles vulnerabilitats en ells.

Respecte al primer sub-objectiu, s'ha realitzat un anàlisi de les plataformes mòbils per poder escollir millor quina podria ser la més adequada per la realització del treball. Després de l'anàlisi, s'ha considerat realitzar l'aplicació per la plataforma Android, ja que presenta un millor desplegament dins del mercat dels mòbils, i té un entorn de suport tècnic més gran que la resta. També s'ha decidit utilitzar el llenguatge Kotlin degut a que és un llenguatge de fàcil aprenentatge i ús, fet que ha ajudat a l'alumne a iniciar-se en el desenvolupament d'aplicacions. Per tant es considera que aquest sub-objectiu s'ha assolit correctament.

Pel què fa al segon sub-objectiu, inicialment l'alumne s'ha plantejat la realització d'una aplicació que permetés la cerca i detecció de vulnerabilitats i la possible explotació d'alguna d'aquestes vulnerabilitats. Malgrat això, l'alumne ha considerat que aquest punt era massa ambiciós, i podia posar en risc el desenvolupament del projecte, atenent als possibles riscos plantejats prèviament, com ara la falta d'experiència de l'alumne amb el desenvolupament d'aplicacions mòbils així com amb el temps de dedicació al projecte. Així doncs, finalment s'ha considerat realitzar únicament una cerca i possible identificació de vulnerabilitats. Per a això, s'ha realitzat un anàlisi de quins mètodes ens permeten aquesta detecció, centrant l'anàlisi en fer una cerca de dispositius en una xarxa i una cerca de ports oberts d'un dispositiu. Així doncs, es pot considerar que aquest sub-objectiu s'ha assolit.

Respecte al tercer sub-objectiu, s'ha realitzat una implementació de les dues funcionalitats definides durant la fase d'investigació. Com que finalment no s'ha decidit d'implementar cap funcionalitat que permeti explotar les vulnerabilitats, l'únic que s'ha implementat ha estat la cerca i detecció d'aquestes, mitjançant els escanejos de xarxa i ports. Per tant, aquest objectiu s'ha assolit parcialment, ja que per bé que no s'ha implementat cap atac que permeti l'explotació de les vulnerabilitats, sí que s'ha implementat una cerca d'aquestes.

Per últim, respecte al quart sub-objectiu, s'ha desenvolupat una aplicació per a mòbils simple, intuïtiva i de fàcil ús, que permet l'execució de les funcionalitats i que mostra de manera entenedora quins són els resultats obtinguts. En aquest punt es pot considerar que s'ha assolit completament l'objectiu.

Pel què fa a la planificació del treball, l'alumne ha vist que aquest no s'havia planificat del tot correctament. La fase d'investigació s'ha allargat una mica respecte al que estava establert, ja que l'alumne no va poder dedicar el temps que havia planejat destinar al projecte, i degut a la falta d'experiència en desenvolupament, va dedicar més temps del previst a la investigació del tipus d'aplicació escollida. Això ha tingut impacte a la fase d'implementació, ja que s'ha començat més tard del planejat, i ha fet que la última fase, dedicada a la optimització, s'hagués de realitzar simultàniament amb la implementació de l'aplicació.

Respecte a la fase d'implementació, no ha estat necessari l'adequació d'un entorn de proves que permetés l'explotació de les vulnerabilitats tal i com s'ha plantejat a l'inici, ja que, com s'ha comentat anteriorment, únicament s'ha implementat la cerca de dispositius de xarxa i de ports oberts en un dispositiu. En aquest cas, les proves de funcionament de l'aplicació s'han realitzat utilitzant l'ordinador personal de l'alumne com a dispositiu a analitzar.

Respecte a la solució presentada durant la implementació, hi ha dos punts a considerar per millorar en un futur. El primer és respecte a la funcionalitat de l'escaneig de xarxa. Tal i com s'ha plantejat, l'aplicació mòbil té la capacitat d'analitzar els dispositius que estan oberts o en funcionament dins d'una xarxa, donant a escollir a l'usuari quina màscara vol utilitzar per tal de saber quants dispositius analitzar. En aquest punt, s'ha donat la opció d'utilitzar la màscara /8, que permet la detecció de 16777214 dispositius, tal i com s'ha explicat durant la investigació. Com es pot comprovar, el nombre de cerques és molt gran, i pot ser poc útil de realitzar des d'un dispositiu mòbil. Així doncs, seria interessant en un futur poder analitzar millor la utilització o no d'aquesta màscara, i veure si altres màscares diferents poden ser més útils en una aplicació mòbil.

El segon punt és respecte al temps d'espera establert per a realitzar les cerques de dispositius i ports, conegut també com a *timeout*. Durant la fase d'optimització s'ha provat diferents valors per aquest temps d'espera. En funció del valor, però, el resultat de la cerca ha variat, ja que al provar valors baixos de *timeout* el temps d'execució era molt baix, però no detectava correctament els dispositius o ports. Per contra, a l'utilitzar valors alts, la detecció era correcta, però el temps d'execució s'incrementava substancialment. Finalment s'ha utilitzat un valor entremig, que permet un temps d'execució raonable juntament amb una detecció correcta, tot i que no sempre es detectin tots els dispositius o ports. Ateses les circumstàncies, seria interessant en un futur investigar més sobre com realitzar aquestes cerques, i trobar un millor valor del *timeout* que permeti un temps d'execució òptim i la detecció completa de dispositius o ports.

També, com a possibles accions de futur, seria interessant millorar la detecció del servei que s'està utilitzant en un port del servidor, realitzant les operacions necessàries per tal de ser més acurat a l'hora de determinar quin servei s'utilitza i així no dependre únicament de les bases de dades utilitzades en aquest projecte. Finalment, també seria interessant afegir noves funcionalitats que permetin una explotació de les possibles vulnerabilitats identificades, fent una aplicació més completa que l'actual.

Per acabar, com a reflexió final, l'alumne considera que el treball realitzat és conseqüent amb els objectius plantejats, ja que s'ha presentat una aplicació de

fàcil ús per a un usuari, i que ajuda a la detecció de vulnerabilitats en un servidor. Aquest treball ha permès a l'alumne introduir-se al desenvolupament d'aplicacions per a dispositius mòbils, un món que considera molt interessant i amb moltes perspectives de futur.

6. Bibliografia

[1] *Atlassian Jira Software* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.atlassian.com/software/jira

[2] *OWASP Vulnerability Scanning Tools* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://owasp.org/www-community/Vulnerability_Scanning_Tools

[3] *Tenable NESSUS Professional* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.tenable.com/products/nessus/nessus-professional

[4] *OpenVAS* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.openvas.org

[5] *Nmap* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://nmap.org

[6] *statcounter – Mobile Operating System Market Share Worldwide* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://gs.statcounter.com/os-market-share/mobile/worldwide

[7] *Android* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.android.com

[8] *Apple iOS* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.apple.com/ios/

[9] *CleverTap - What Are the Different Types of Mobile Apps? And How Do You Choose?* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://clevertap.com/blog/types-of-mobile-apps/

[10] *Viquipèdia – Aplicació mòbil* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://ca.wikipedia.org/wiki/Aplicació_mòbil

[11] *Viquipèdia – Adreça IP* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://ca.wikipedia.org/wiki/Adreça_IP

[12] *Viquipèdia – Màscara de xarxa* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://ca.wikipedia.org/wiki/Màscara_de_xarxa

[13] *Internet Control Message Protocol* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.ietf.org/rfc/rfc792.txt

[14] *Viquipèdia – Escàner de ports* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://ca.wikipedia.org/wiki/Escàner_de_ports

[15] *Transmission Control Protocol* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.ietf.org/rfc/rfc793.txt

[16] *User Datagram Protocol* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.ietf.org/rfc/rfc768.txt

[17] *Iana* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://www.iana.org

[18] *Service Name and Transport Protocol Port Number Registry* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[19] *Android Studio* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://developer.android.com/studio

[20] *InetAddress Reference* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://developer.android.com/reference/java/net/InetAddress

[21] *Socket Reference* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://developer.android.com/reference/java/net/Socket

[22] *kotlin-csv Reference* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://github.com/doyaaaaaken/kotlin-csv

[23] *DatagramSocket Reference* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://developer.android.com/reference/java/net/DatagramSocket

[24] *DatagramPacket Reference* [en línia] [data de consulta: 28 de maig de 2021]. Disponible a: https://developer.android.com/reference/java/net/DatagramPacket

A. Annex

A.1 Codi font de la pàgina principal

A.1.1 SplashScreenActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.os.Bundle
import android.os.Handler
import androidx.appcompat.app.AppCompatActivity

class SplashScreenActivity : AppCompatActivity() {

    private val SPLASH_TIMEOUT: Long = 600

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

        Handler().postDelayed({
            startActivity(Intent(this, MainPageActivity::class.java))
            finish()
        }, SPLASH_TIMEOUT)
    }
}
```

A.1.2 MainPageActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.widget.ListView
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.core.view.GravityCompat
import androidx.drawerlayout.widget.DrawerLayout
import com.google.android.material.navigation.NavigationView

class MainPageActivity : AppCompatActivity(), NavigationView.OnNavigationItemSelectedListener {

    private lateinit var drawer: DrawerLayout
    private lateinit var toggle: ActionBarDrawerToggle
    private lateinit var navview: NavigationView

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

        // List
        val listview: ListView = findViewById(R.id.mainpage_listview)
        val list = mutableListOf<String>()
        list.add(getString(R.string.portScannerName))
        list.add(getString(R.string.networkScannerName))
        val adapter = MainPageListAdapter(this, R.layout.listview_mainpage_layout, list)
        listview.adapter = adapter
        val header = layoutInflater.inflate(R.layout.listview_header_layout, null)
        listview.addHeaderView(header)

        // Action bar
        val toolbar: Toolbar = findViewById(R.id.mainpage_actionbar)
        toolbar.setTitle(R.string.appName)
        setSupportActionBar(toolbar)
```

```
        // Navigation bar
        drawer = findViewById(R.id.mainpage_drawerlayout)
        toggle = ActionBarDrawerToggle(this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close)
        drawer.addDrawerListener(toggle)
        toggle.syncState()
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setHomeButtonEnabled(true)
        navview = findViewById(R.id.mainpage_navigationview)
        navview.setNavigationItemSelectedListener(this)
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.actionbar_mainpage_menu, menu)
        return super.onCreateOptionsMenu(menu)
    }

    override fun onPostCreate(savedInstanceState: Bundle?) {
        super.onPostCreate(savedInstanceState)
        toggle.syncState()
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.mainpage_settings_button) {
            startActivity(Intent(this, MainPagePreferencesActivity::class.java))
            return true
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onNavigationItemSelected(item: MenuItem): Boolean {
        val id = item.itemId
        val drawer: DrawerLayout = findViewById(R.id.mainpage_drawerlayout)
        if (id == R.id.navigationview_portscanner_button) {
            startActivity(Intent(this, MainPortScannerActivity::class.java))
            return true
        }
        if (id == R.id.navigationview_networkscanner_button) {
            startActivity(Intent(this, MainNetworkScannerActivity::class.java))
            return true
        }
        if (id == R.id.navigationview_home_button) {
            drawer.closeDrawer(GravityCompat.START)
            return true
        }
        if (id == R.id.navigationview_settings_button) {
            startActivity(Intent(this, MainPagePreferencesActivity::class.java))
            return true
        }
        drawer.closeDrawer(GravityCompat.START)
        return true
    }
}
```

A.1.3 MainPageListAdapter.kt

```
package android.app.tfmuoc001

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView

class MainPageListAdapter(private var cntxt: Context, var resource: Int, var items: List<String>) :
ArrayAdapter<String>(cntxt, resource, items) {

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val layoutInflater: LayoutInflater = LayoutInflater.from(cntxt)
```

```
        val view: View = layoutInflater.inflate(resource, null)
        val titleView: TextView = view.findViewById(R.id.mainpage_textview_title)
        titleView.text = items[position]

        return view
    }
}
```

A.2 Codi font del menú de configuració

A.2.1 MainPagePreferencesActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.os.Bundle
import android.view.MenuItem
import android.widget.AdapterView
import android.widget.ListView
import androidx.appcompat.app.AppCompatActivity

class MainPagePreferencesActivity : AppCompatActivity() {

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

        // Action bar
        supportActionBar?.setTitle(R.string.menuSettings)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        // ListView
        val listview: ListView = findViewById(R.id.mainpagepreferences_listview)
        val list = mutableListOf<String>()
        list.add(getString(R.string.portScannerName))
        list.add(getString(R.string.networkScannerName))
        val adapter = MainPageListAdapter(this, R.layout.listview_mainpage_layout, list)
        listview.adapter = adapter
        listview.onItemClickListener = AdapterView.OnItemClickListener { parent, view, position, id ->
            if (position == 0) {
                startActivity(Intent(this, NetworkScannerPreferencesActivity::class.java))
            }
            if (position == 1) {
                startActivity(Intent(this, PortScannerPreferencesActivity::class.java))
            }
        }
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (android.R.id.home == item.itemId) {
            startActivity(Intent(this, MainPageActivity::class.java))
            return true
        }
        return true
    }
}
```

A.3 Codi font de l'escaneig de xarxa

A.3.1 MainNetworkScannerActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.content.res.Configuration
import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.view.View
```

```
import android.widget.ListView
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.core.view.GravityCompat
import androidx.drawerlayout.widget.DrawerLayout
import androidx.preference.PreferenceManager
import com.google.android.material.navigation.NavigationView

class MainNetworkScannerActivity : AppCompatActivity(),
NavigationView.OnNavigationItemSelectedListener {

    private lateinit var drawer: DrawerLayout
    private lateinit var toggle: ActionBarDrawerToggle
    private lateinit var navview: NavigationView
    private lateinit var listview: ListView

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

        // NetworkPreferences
        val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this)
        val ipAddress = sharedPreferences.getString("settings_networkscanner_ipaddress", "-").toString()
        var netMask = sharedPreferences.getString("settings_networkscanner_networkmask", "3").toString()
        if (netMask == "1") {
            netMask = "8"
        }
        if (netMask == "2") {
            netMask = "16"
        }
        if (netMask == "3") {
            netMask = "24"
        }

        // ListView
        listview = findViewById(R.id.mainnetworkscanner_listview)
        val list = mutableListOf<MainNetworkScannerListModel>()
        list.add(MainNetworkScannerListModel(getString(R.string.portScannerSettingsAddressIP),
"$ipAddress/$netMask"))
        listview.adapter = MainNetworkScannerListAdapter(this,
R.layout.listview_mainnetworkscanner_layout, list)

        // ActionBar
        val toolbar: Toolbar = findViewById(R.id.mainnetworkscanner_actionbar)
        toolbar.setTitle(R.string.networkScannerName)
        setSupportActionBar(toolbar)

        // NavigationBar
        drawer = findViewById(R.id.mainnetworkscanner_drawerlayout)
        toggle = ActionBarDrawerToggle(this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close)
        drawer.addDrawerListener(toggle)
        toggle.syncState()
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setHomeButtonEnabled(true)
        navview = findViewById(R.id.mainnetworkscanner_navigationview)
        navview.setNavigationItemSelectedListener(this)

        // FloatingButton
        val fab: View = findViewById(R.id.mainnetworkscanner_floatingactionbutton)
        fab.setOnClickListener {
            startActivity(Intent(this, NetworkScannerActivity::class.java))
        }
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.actionbar_mainnetworkscanner_menu, menu)
        return super.onCreateOptionsMenu(menu)
    }

    override fun onPostCreate(savedInstanceState: Bundle?) {
```

```
        super.onPostCreate(savedInstanceState)
        toggle.syncState()
    }

    override fun onConfigurationChanged(newConfig: Configuration) {
        super.onConfigurationChanged(newConfig)
        toggle.onConfigurationChanged(newConfig)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.mainnetworkscanner_settings_button) {
            startActivity(Intent(this, NetworkScannerPreferencesActivity::class.java))
            return true
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onNavigationItemSelected(item: MenuItem): Boolean {
        val id = item.itemId
        val drawer: DrawerLayout = findViewById(R.id.mainnetworkscanner_drawerlayout)
        if (id == R.id.navigationview_portscanner_button) {
            startActivity(Intent(this, MainPortScannerActivity::class.java))
            return true
        }
        if (id == R.id.navigationview_networkscanner_button) {
            drawer.closeDrawer(GravityCompat.START)
            return true
        }
        if (id == R.id.navigationview_home_button) {
            startActivity(Intent(this, MainPageActivity::class.java))
            return true
        }
        if (id == R.id.navigationview_settings_button) {
            drawer.closeDrawer(GravityCompat.START)
            return true
        }
        drawer.closeDrawer(GravityCompat.START)
        return true
    }
}
```

A.3.2 MainNetworkScannerListModel.kt

```
package android.app.tfmuoc001

class MainNetworkScannerListModel (val title:String, val detail:String)
```

A.3.3 MainNetworkScannerListAdapter.kt

```
package android.app.tfmuoc001

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView

class MainNetworkScannerListAdapter(var cntxt: Context, var resource: Int, var items:
List<MainNetworkScannerListModel>) : ArrayAdapter<MainNetworkScannerListModel>(cntxt, resource,
items) {

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val layoutInflater: LayoutInflater = LayoutInflater.from(cntxt)
        val view: View = layoutInflater.inflate(resource, null)
        val titleView: TextView = view.findViewById(R.id.mainnetworkscanner_textview_title)
        val detailView: TextView = view.findViewById(R.id.mainnetworkscanner_textview_detail)
        val element: MainNetworkScannerListModel = items[position]
        titleView.text = element.title
        detailView.text = element.detail
```

```
        return view
    }

}
```

A.3.4 NetworkScannerPreferencesActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.os.Bundle
import android.view.MenuItem
import androidx.appcompat.app.AppCompatActivity
import androidx.preference.PreferenceFragmentCompat

class NetworkScannerPreferencesActivity : AppCompatActivity() {

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

        supportActionBar?.setTitle(R.string.menuSettings)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        if (supportFragmentManager.findFragmentById(android.R.id.content) == null) {
            supportFragmentManager.beginTransaction().add(android.R.id.content,
SettingsFragment()).commit()
        }
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (android.R.id.home == item.itemId) {
            startActivity(Intent(this, MainNetworkScannerActivity::class.java))
            return true
        }
        return true
    }

    class SettingsFragment : PreferenceFragmentCompat() {
        override fun onCreatePreferences(savedInstanceState: Bundle?, rootkey: String?) {
            setPreferencesFromResource(R.xml.preferences_mainnetworkscanner, rootkey)
        }
    }

}
```

A.3.5 NetworkScannerActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.net.wifi.WifiManager
import android.os.Bundle
import android.text.format.Formatter
import android.view.MenuItem
import android.view.View
import android.view.WindowManager
import android.widget.*
import androidx.appcompat.app.AppCompatActivity
import androidx.preference.PreferenceManager
import kotlinx.coroutines.*
import java.net.InetAddress
import java.util.*
import kotlin.experimental.and


class NetworkScannerActivity : AppCompatActivity() {

    private val listDevices: ArrayList<NetworkScannerListModel> = arrayListOf()
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_networkscanner_layout)
        window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)

        // Action bar
        supportActionBar?.setTitle(R.string.resultsName)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        // Progress bar
        val progressBar: ProgressBar = findViewById(R.id.networkscanner_progressbar);
        val progressBarText: TextView = findViewById(R.id.networkscanner_progressbar_text)
        progressBarText.text = getString(R.string.progressBarName)

        // Network scanner
        val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this)
        val ipAddress = sharedPreferences.getString("settings_networkscanner_ipaddress", "-").toString()
        val netMask = sharedPreferences.getString("settings_networkscanner_networkmask", "3").toString()

        val statusScanner: Deferred<Boolean> = CoroutineScope(Dispatchers.IO).async {
            val subnet = getSubnet(ipAddress, netMask)
            if (netMask == "1") {
                netMask8(subnet)
            }
            if (netMask == "2") {
                netMask16(subnet)
            }
            if (netMask == "3") {
                netMask24(subnet)
            }
            true
        }

        val statusProgressbar: Deferred<Boolean> = CoroutineScope(Dispatchers.Main).async {
            progressBar.visibility = View.VISIBLE
            progressBarText.visibility = View.VISIBLE
            true
        }

        CoroutineScope(Dispatchers.Main).launch {
            statusScanner.await()
            statusProgressbar.await()
            progressBar.visibility = View.GONE
            progressBarText.visibility = View.GONE
            val listview: ListView = findViewById(R.id.networkscanner_listview)
            val adapter = NetworkScannerListAdapter(this@NetworkScannerActivity,
R.layout.listview_networkscanner_layout, listDevices)
            listview.adapter = adapter
            listview.onItemClickListener = AdapterView.OnItemClickListener { parent, view, position, id ->
                val preferenceEditor = sharedPreferences.edit()
                preferenceEditor.putString("settings_portscanner_ipaddress", listDevices[position].ip)
                preferenceEditor.commit()
                startActivity(Intent(this@NetworkScannerActivity, MainPortScannerActivity::class.java))
            }
        }
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (android.R.id.home == item.itemId) {
            startActivity(Intent(this, MainNetworkScannerActivity::class.java))
            return true
        }
        return true
    }

    private fun getSubnet(ip: String, netMask: String): MutableList<String> {
        val ipAddress = InetAddress.getByName(ip).address
        val subnetByte = ByteArray(4)
        if (netMask == "1") {
            val netMask8 = byteArrayOf(0xFF.toByte(), 0x00.toByte(), 0x00.toByte(), 0x00.toByte())
            for (byte in 0..3) {
                subnetByte[byte] = netMask8[byte] and ipAddress[byte]
```

```
        }
    }
    if (netMask == "2") {
        val netMask16 = byteArrayOf(0xFF.toByte(), 0xFF.toByte(), 0x00.toByte(), 0x00.toByte())
        for (byte in 0..3) {
            subnetByte[byte] = netMask16[byte] and ipAddress[byte]
        }
    }
    if (netMask == "3") {
        val netMask24 = byteArrayOf(0xFF.toByte(), 0xFF.toByte(), 0xFF.toByte(), 0x00.toByte())
        for (byte in 0..3) {
            subnetByte[byte] = netMask24[byte] and ipAddress[byte]
        }
    }
    val subnet = mutableListOf<String>()
    for (byte in subnetByte) {
        subnet.add(byte.toUByte().toInt().toString())
    }
    return subnet
}

private fun netMask8(subnet: MutableList<String>) {
    for (netMask8 in 1..254) {
        for (netMask16 in 1..254) {
            for (netMask24 in 1..254) {
                val scanIPstr = subnet[0] + "." + netMask8 + "." + netMask16 + "." + netMask24
                val scanIP = InetAddress.getByName(scanIPstr)
                val status = scanIP.isReachable(50)
                if (status) {
                    var detail = ""
                    if (netMask24 == 1) {
                        detail = "[gateway]"
                    }
                    if (isDevice(scanIPstr)) {
                        detail = "[${getString(R.string.networkScannerThisDevice)}]"
                    }
                    listDevices.add(NetworkScannerListModel(scanIPstr, detail, "scan"))
                }
            }
        }
    }
}

private fun netMask16(subnet: MutableList<String>) {
    for (netMask16 in 1..254) {
        for (netMask24 in 1..254) {
            val scanIPstr = subnet[0] + "." + subnet[1] + "." + netMask16 + "." + netMask24
            val scanIP = InetAddress.getByName(scanIPstr)
            val status = scanIP.isReachable(50)
            if (status) {
                var detail = ""
                if (netMask24 == 1) {
                    detail = "[gateway]"
                }
                if (isDevice(scanIPstr)) {
                    detail = "[${getString(R.string.networkScannerThisDevice)}]"
                }
                listDevices.add(NetworkScannerListModel(scanIPstr, detail, "scan"))
            }
        }
    }
}

private fun netMask24(subnet: MutableList<String>) {
    for (netMask24 in 1..254) {
        val scanIPstr = subnet[0] + "." + subnet[1] + "." + subnet[2] + "." + netMask24
        val scanIP = InetAddress.getByName(scanIPstr)
        val status = scanIP.isReachable(50)
        if (status) {
            var detail = ""
            if (netMask24 == 1) {
                detail = "[gateway]"
```

```
            }
            if (isDevice(scanIPstr)) {
                detail = "[${getString(R.string.networkScannerThisDevice)}]"
            }
            listDevices.add(NetworkScannerListModel(scanIPstr, detail, "scan"))
        }
    }
}

private fun isDevice(ip: String): Boolean {
    val wifiManager: WifiManager = applicationContext.getSystemService(WIFI_SERVICE) as
WifiManager
    val localIP: String = Formatter.formatIpAddress(wifiManager.connectionInfo.ipAddress)
    return localIP == ip
}
}
```

A.3.6 NetworkScannerListModel.kt

```
package android.app.tfmuoc001

class NetworkScannerListModel(val ip: String, val detail: String, val scan: String)
```

A.3.7 NetworkScannerListAdapter.kt

```
package android.app.tfmuoc001

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView

class NetworkScannerListAdapter(var cntxt: Context, var resource: Int, var items:
ArrayList<NetworkScannerListModel>) : ArrayAdapter<NetworkScannerListModel>(cntxt, resource, items)
{

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val layoutInflater: LayoutInflater = LayoutInflater.from(cntxt)
        val view: View = layoutInflater.inflate(resource, null)
        val ipView: TextView = view.findViewById(R.id.networkscanner_textview_ip)
        val detailView: TextView = view.findViewById(R.id.networkscanner_textview_detail)
        val scanView: TextView = view.findViewById(R.id.networkscanner_textview_scan)
        val element: NetworkScannerListModel = items[position]
        ipView.text = element.ip
        detailView.text = element.detail
        scanView.text = element.scan

        notifyDataSetChanged()

        return view
    }

}
```

A.4 Codi font de l'escaneig de ports

A.4.1 MainPortScannerActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.content.res.Configuration
import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.view.View
```

```
import android.widget.ListView
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.core.view.GravityCompat
import androidx.drawerlayout.widget.DrawerLayout
import androidx.preference.PreferenceManager
import com.google.android.material.navigation.NavigationView

class MainPortScannerActivity : AppCompatActivity(), NavigationView.OnNavigationItemSelectedListener {

    private lateinit var toggle: ActionBarDrawerToggle

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

        // Test preferences
        val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this)
        val ipAddress = sharedPreferences.getString("settings_portscanner_ipaddress", "-").toString()
        val portList = sharedPreferences.getString("settings_portlist", "-").toString()
        val allPorts = sharedPreferences.getBoolean("settings_all", false)
        val scanTCP = sharedPreferences.getBoolean("settings_tcp", false)
        val scanUDP = sharedPreferences.getBoolean("settings_udp", false)
        var configText = ""
        if (allPorts) {
            configText += getString(R.string.portScannerSettingsAllPorts).plus(", ")
        }
        if (scanTCP) {
            configText += getString(R.string.portScannerSettingsTCPScan).plus(", ")
        }
        if (scanUDP) {
            configText += getString(R.string.portScannerSettingsUDPScan).plus(", ")
        }
        configText = configText.dropLast(2)

        // List
        val listview: ListView = findViewById(R.id.mainportscanner_listview)
        val list = mutableListOf<MainPortScannerListModel>()
        list.add(MainPortScannerListModel(getString(R.string.portScannerSettingsAddressIP), ipAddress))
        list.add(MainPortScannerListModel(getString(R.string.portScannerSettingsListPorts), portList))
        list.add(MainPortScannerListModel(getString(R.string.menuSettings), configText))
        listview.adapter = MainPortScannerListAdapter(this, R.layout.listview_mainportscanner_layout, list)

        // Action bar
        val toolbar: Toolbar = findViewById(R.id.mainportscanner_actionbar)
        toolbar.setTitle(R.string.portScannerName)
        setSupportActionBar(toolbar)

        // Navigation bar
        val drawer: DrawerLayout = findViewById(R.id.mainportscanner_drawerlayout)
        toggle = ActionBarDrawerToggle(this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close)
        drawer.addDrawerListener(toggle)
        toggle.syncState()
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setHomeButtonEnabled(true)
        val navview: NavigationView = findViewById(R.id.mainportscanner_navigationview)
        navview.setNavigationItemSelectedListener(this)

        // Floating button
        val fab: View = findViewById(R.id.mainportscanner_floatingactionbutton)
        fab.setOnClickListener {
            startActivity(Intent(this, PortScannerActivity::class.java))
        }
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.actionbar_mainportscanner_menu, menu)
        return super.onCreateOptionsMenu(menu)
    }
```

```
    override fun onPostCreate(savedInstanceState: Bundle?) {
        super.onPostCreate(savedInstanceState)
        toggle.syncState()
    }

    override fun onConfigurationChanged(newConfig: Configuration) {
        super.onConfigurationChanged(newConfig)
        toggle.onConfigurationChanged(newConfig)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.mainportscanner_settings_button) {
            startActivity(Intent(this, PortScannerPreferencesActivity::class.java))
            return true
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onNavigationItemSelected(item: MenuItem): Boolean {
        val id = item.itemId
        val drawer: DrawerLayout = findViewById(R.id.mainportscanner_drawerlayout)
        if (id == R.id.navigationview_portscanner_button) {
            drawer.closeDrawer(GravityCompat.START)
            return true
        }
        if (id == R.id.navigationview_networkscanner_button) {
            startActivity(Intent(this, MainNetworkScannerActivity::class.java))
            return true
        }
        if (id == R.id.navigationview_home_button) {
            startActivity(Intent(this, MainPageActivity::class.java))
            return true
        }
        if (id == R.id.navigationview_settings_button) {
            drawer.closeDrawer(GravityCompat.START)
            return true
        }
        drawer.closeDrawer(GravityCompat.START)
        return true
    }
}
```

A.4.2 MainPortScannerListModel.kt

```
package android.app.tfmuoc001

class MainPortScannerListModel(val title: String, val detail: String)
```

A.4.3 MainPortScannerListAdapter.kt

```
package android.app.tfmuoc001

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView

class MainPortScannerListAdapter(var cntxt: Context, var resource: Int, var items:
List<MainPortScannerListModel>) : ArrayAdapter<MainPortScannerListModel>(cntxt, resource, items) {

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val layoutInflater: LayoutInflater = LayoutInflater.from(cntxt)
        val view: View = layoutInflater.inflate(resource, null)
        val titleView: TextView = view.findViewById(R.id.mainportscanner_textview_title)
        val detailView: TextView = view.findViewById(R.id.mainportscanner_textview_detail)
        val element: MainPortScannerListModel = items[position]
        titleView.text = element.title
        detailView.text = element.detail
```

```
        return view
    }

}
```

A.4.4 PortScannerPreferencesActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.os.Bundle
import android.view.MenuItem
import androidx.appcompat.app.AppCompatActivity
import androidx.preference.PreferenceFragmentCompat

class PortScannerPreferencesActivity : AppCompatActivity() {

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

        supportActionBar?.setTitle(R.string.menuSettings)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        if (supportFragmentManager.findFragmentById(android.R.id.content) == null) {
            supportFragmentManager.beginTransaction().add(android.R.id.content,
SettingsFragment()).commit()
        }
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (android.R.id.home == item.itemId) {
            startActivity(Intent(this, MainPortScannerActivity::class.java))
            return true
        }
        return true
    }

    class SettingsFragment : PreferenceFragmentCompat() {
        override fun onCreatePreferences(savedInstanceState: Bundle?, rootkey: String?) {
            setPreferencesFromResource(R.xml.preferences_mainportscanner, rootkey)
        }
    }

}
```

A.4.5 PortScannerActivity.kt

```
package android.app.tfmuoc001

import android.content.Intent
import android.os.Bundle
import android.view.MenuItem
import android.view.View
import android.widget.ListView
import android.widget.ProgressBar
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.preference.PreferenceManager
import com.github.doyaaaaaken.kotlincsv.dsl.csvReader
import kotlinx.coroutines.*
import java.net.*

class PortScannerActivity : AppCompatActivity() {

    private val listPort: ArrayList<PortScannerListModel> = arrayListOf()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_portscanner_layout)
```

```
// Action bar
supportActionBar?.setTitle(R.string.resultsName)
supportActionBar?.setDisplayHomeAsUpEnabled(true)

// Port list database
val portListTCP = mutableListOf<PortListModel>()
val portListUDP = mutableListOf<PortListModel>()
val csvfile = assets.open("service-names-port-numbers.csv")
csvReader().open(csvfile) {
    readAllAsSequence().forEach { row ->
        if (row[2] == "tcp") {
            portListTCP.add(PortListModel(row[1], row[0], row[2], row[3]))
        }
        if (row[2] == "udp") {
            portListUDP.add(PortListModel(row[1], row[0], row[2], row[3]))
        }
    }
}

// Progress bar
val progressBar: ProgressBar = findViewById(R.id.portscanner_progressbar);
val progressBarText: TextView = findViewById(R.id.portscanner_progressbar_text)
progressBarText.text = getString(R.string.progressBarName)

// Port scanner
val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this)
val ipAddress = sharedPreferences.getString("settings_portscanner_ipaddress", "-").toString()
val portList = sharedPreferences.getString("settings_portlist", "-").toString()
val allPorts = sharedPreferences.getBoolean("settings_all", false)
val scanTCP = sharedPreferences.getBoolean("settings_tcp", false)
val scanUDP = sharedPreferences.getBoolean("settings_udp", false)

val statusScanner: Deferred<Boolean> = CoroutineScope(Dispatchers.IO).async {
    if (allPorts) {
        for (port in 1..65536) {
            if (scanTCP) {
                scannerTCP(ipAddress, port, portListTCP)
            }
            if (scanUDP) {
                scannerUDP(ipAddress, port, portListUDP)
            }
        }
    } else {
        val strList = portList.split(",").toTypedArray()
        val list = getPortList(strList)
        list.sort()
        for (port in list) {
            if (scanTCP) {
                scannerTCP(ipAddress, port, portListTCP)
            }
            if (scanUDP) {
                scannerUDP(ipAddress, port, portListUDP)
            }
        }
    }
    true
}

val statusProgressbar: Deferred<Boolean> = CoroutineScope(Dispatchers.Main).async {
    progressBar.visibility = View.VISIBLE
    progressBarText.visibility = View.VISIBLE
    true
}

CoroutineScope(Dispatchers.Main).launch {
    statusScanner.await()
    statusProgressbar.await()
    progressBar.visibility = View.GONE
    progressBarText.visibility = View.GONE
    val sortedPortList = listPort.sortedBy { PortScannerListModel -> PortScannerListModel.port.toInt() }
    val listview: ListView = findViewById(R.id.portscanner_listview)
```

```
            val adapter = PortScannerListAdapter(this@PortScannerActivity,
R.layout.listview_portscanner_layout, sortedPortList)
            listview.adapter = adapter
        }
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (android.R.id.home == item.itemId) {
            startActivity(Intent(this, MainPortScannerActivity::class.java))
            return true
        }
        return true
    }

    private fun getPortList(list: Array<String>): MutableList<Int> {
        val portlist = mutableListOf<Int>()
        for (port in list) {
            if (port.contains("-")) {
                val range = port.split("-")
                for (element in range.elementAt(0).toInt()..range.elementAt(1).toInt()) {
                    portlist.add(element)
                }
            } else {
                portlist.add(port.toInt())
            }
        }
        return portlist
    }

    private fun scannerTCP(ip: String, port: Int, portList: MutableList<PortListModel>) {
        val statusTCP = try {
            val socket = Socket(ip, port)
            socket.isConnected
        } catch (e: Exception) {
            false
        }
        if (statusTCP) {
            val model: PortListModel? = portList.find { it.port == port.toString() }
            val description = model?.description ?: "-"
            val service = model?.service ?: "-"
            val detail = "[tcp] $service - $description"
            listPort.add(PortScannerListModel(port.toString(), detail, "open"))
        }
    }

    private fun scannerUDP(ip: String, port: Int, portList: MutableList<PortListModel>) {
        val statusUDP = try {
            val socketSender = DatagramSocket()
            val message = "hello world".toByteArray()
            val packetSender = DatagramPacket(message, message.size, InetAddress.getByName(ip), port)
            socketSender.soTimeout = 100
            socketSender.connect(InetAddress.getByName(ip), port)
            socketSender.send(packetSender)
            socketSender.receive(packetSender)
            socketSender.close()
            "open"
        } catch (e: PortUnreachableException) {
            "closed"
        } catch (e: SocketTimeoutException) {
            "open|filtered"
        } catch (e: Exception) {
            "other"
        }
        if (statusUDP == "open") {
            val model: PortListModel? = portList.find { it.port == port.toString() }
            val description = model?.description ?: "-"
            val service = model?.service ?: "-"
            val detail = "[udp] $service - $description"
            listPort.add(PortScannerListModel(port.toString(), detail, statusUDP))
        }
        if (statusUDP == "open|filtered") {
            val model: PortListModel? = portList.find { it.port == port.toString() }
```

```
            val description = model?.description ?: "-"
            val service = model?.service ?: "-"
            val detail = "[udp] $service - $description"
            listPort.add(PortScannerListModel(port.toString(), detail, statusUDP))
        }
    }
}
```

A.4.6 PortListModel.kt

```
package android.app.tfmuoc001

class PortListModel(val port: String, val service: String, val protocol: String, val description: String)
```

A.4.7 PortScannerListModel.kt

```
package android.app.tfmuoc001

class PortScannerListModel(val port: String, val detail: String, val status: String)
```

A.4.8 PortScannerListAdapter.kt

```
package android.app.tfmuoc001

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView
import androidx.core.content.ContextCompat

class PortScannerListAdapter(var cntxt: Context, var resource: Int, var items: List<PortScannerListModel>)
: ArrayAdapter<PortScannerListModel>(cntxt, resource, items) {

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val layoutInflater: LayoutInflater = LayoutInflater.from(cntxt)
        val view: View = layoutInflater.inflate(resource, null)
        val portView: TextView = view.findViewById(R.id.scanner_textview_title)
        val detailView: TextView = view.findViewById(R.id.scanner_textview_detail)
        val statusView: TextView = view.findViewById(R.id.scanner_textview_status)
        val element: PortScannerListModel = items[position]
        portView.text = element.port
        detailView.text = element.detail
        if (element.status == "open") {
            statusView.text = cntxt.getString(R.string.portScannerStatusOpen)
            statusView.setTextColor(ContextCompat.getColor(cntxt, R.color.green_400))
        }
        if (element.status == "closed") {
            statusView.text = cntxt.getString(R.string.portScannerStatusClosed)
            statusView.setTextColor(ContextCompat.getColor(cntxt, R.color.red_400))
        }
        if (element.status == "open|filtered") {
            statusView.text = cntxt.getString(R.string.portScannerStatusOpenFiltered)
            statusView.setTextColor(ContextCompat.getColor(cntxt, R.color.orange_400))
        }

        notifyDataSetChanged()

        return view
    }

}
```