Estudi del Tapjacking en Android

Nom: Alejandro García Rodríguez

Consultor: Carlos Ares Angulo

Projecte Final de Carrera- Àrea de Seg. Informàtica

Universitat Oberta de Catalunya (UOC) 2011 -2012

©Aquest treball es distribueix amb llicència **CC** (Creative Commons) Podeu copiar, distribuir, comunicar públicament l'obra i/o fer-ne obres derivades, sempre citant com a font aquesta obra. A més, no podeu utilitzar aquesta obra amb finalitats comercials. Si altereu o transformeu aquesta obra, o en genereu obres derivades, només podeu distribuir l'obra generada amb una llicència idèntica a aquesta.

Índex

1-Resum....8
2-Introducció....8
2.1-Justificació del projecte....8
2.2-Estat de l'art....11
2.3-Productes obtinguts....12
2.4-Planificació del projecte....12
2.4.a-Contingut de les entregues parcials....12
2.4.b-Entregues no presentades....13
2.4.c-Diagrama de Gantt....13
3-Plataforma Android....15
3.1-Història d'Android....15
3.2-Android com a Sistema Operatiu....17
4-Vulnerabilitat....21
4.1-Introducció....21
4.2-Tapjacking....21
4.3-Anàlisi del tapjacking....22
4.4-La solució de Google....24
5-Aplicació ANDROID HUNT 26
5.1-Prova de concepte....26
5.1.a-Arquitectura....27
5.1.b-L'aplicació en funcionament....28
6-Aplicació MIST....31
6.1-Introducció....31
6.2-Diagrama de classes....31
6.2.a-Classe Main....33
6.2.b-Classe FilterManagerTab....34
6.2.c-Classe AddFilterTab....35
6.2.d-Classe ListFilterTab....35
6.2.e-Classe ListBlocked....36
6.2.f-Classe Widget....36
6.2.g-Classe Constants i classe FilterSQL....37
6.2.h-Classe FilterListener....38
6.2.i-Classe pare Filter i descendents....40
7-Valoració final de la solució proposada....41
8-Annex: Instal.lació de les aplicacions....43
8.1-Aplicacions Android (SDK d'Android)....43
8.2-Altres....43
8.2.a-Instal.lar plugin ADT en Eclipse....43
8.2.b-Importar projecte Android en Eclipse....44
8.2.c-Base de dades SQLite en Android....44
9-Bibliografia....47
10-Enllaços d'interès....47
11-Agraïments....48

Figures

Ilustración 1: Evolució d'Android....9
Ilustración 2: Nombre de dispositius per versions....11
Ilustración 3: Diagrama de Gantt....14
Ilustración 4: Arquitectura d'Android....17
Ilustración 5: Permisos en l'Android Market....20
Ilustración 6: Clickjacking....21
Ilustración 7: Esquema Tapjacking....22
Ilustración 8: Element Toast en Android....22
Ilustración 9: Diagrama classes ANDROID HUNT....27
Ilustración 10: Pantalla inicial....28
Ilustración 11: Tapjacking amb SMS....29
Ilustración 12: Tapjacking amb trucada....29
Ilustración 13: Tapjacking en acció....30
Ilustración 14: Diagrama de classes Projecte MIST....32
Ilustración 15: Pantalla "Home"....33
Ilustración 16: Pantalla inicial....34
Ilustración 17: Afegir filtre....35
Ilustración 18: Llistat de filtres....35
Ilustración 19: Accions....36
Ilustración 20: Widget a l'escriptori....37
Ilustración 21: Notificacions en Android....39
Ilustración 22: Selecció d'aplicació....41
Ilustración 23: File Explorer en Eclipse....45
Ilustración 24: Taula Filter en SQLiteBrowser....46
Ilustración 25: Taula List en SQLiteBrowser....46

1- Resum

Aquest projecte s'emmarca dins de l'àrea de seguretat informàtica. L'objectiu d'aquest treball de fi de carrera engloba dos aspectes:

- Per una banda desenvolupar una prova de concepte sobre una vulnerabilitat que afecta a les primeres versions del sistema operatiu **Android** desenvolupat per Google.
- Per l'altre banda, fent servir les mateixes eines que proveu la plataforma Android, desenvolupar una aplicació com a solució d'aquesta vulnerabilitat.

La vulnerabilitat que estudiarem en aquest projecte s'ha batejat com ***tapjaking*** i afecta a les **versions 1.5, 1.6, 2.0/2.1 i 2.2** de la plataforma **Android**.

Tot i no ser una vulnerabilitat actual que encara estigui activa, és una molt bona aproximació a la plataforma **Android** desde una òptica diferent. S'ha partit desde l'anàlisi d'una vulnerabilitat no recent estudiant l'afectació a la plataforma Android. S'ha cobert la fase de detallar en que consisteix aquesta vulnerabilitat i la fase de desenvolupament d'una prova de concepte. Com afegit, s'ha tancat el cercle amb la darrera fase en que s'ha desenvolupat una solució pel software afectat.

2- Introducció

2.1- Justificació del projecte

Avui dia, segons estudis de diverses consultories[1], hi ha de l'ordre de 500 milions dels anomenats *smartphones* o telèfons intel.ligents a tot el món. I aquest nombre no para de créixer. Segons aquest estudi, dintre de 4 anys el nombre passarà del actual 32% fins al 55%, és a dir, més de la meitat dels telèfons mòbils seran *smartphones*. Aquest es un mercat molt atractiu i els principals fabricants de telèfons ho saben. *HTC, Samsung, Motorola o LG* presenten models contínuament, que abarquen desde els anomenats "*Low cost*" fins a models més exclusius.

La clau d'aquest augment no es deu a millors aparells o més atractius. La clau es troba a l'interior, al sistema operatiu que gestiona el telèfon. Actualment hi ha diversos actors, com pot ser *Apple* amb el seu *iPhone* gestionat pel sistema operatiu *iOS*, el recent Windows Phone 7 o el futur *Tizen* avalat per *Intel* i la *Linux Foundation*. Hem deixat per al final el que segurament es avui dia l'actor principal (sense desmerèixer *Apple* amb *iOs*), amb prop d'un milió d'activacions diàries. Parlem d'Android, el sistema operatiu per a *smartphones* de Google. Cal dir que aquest sistema no es només per *smarthphones* si no que serveix per altres dispositius com les recents *tablets*, navegadors per a cotxes, televisions, etc...

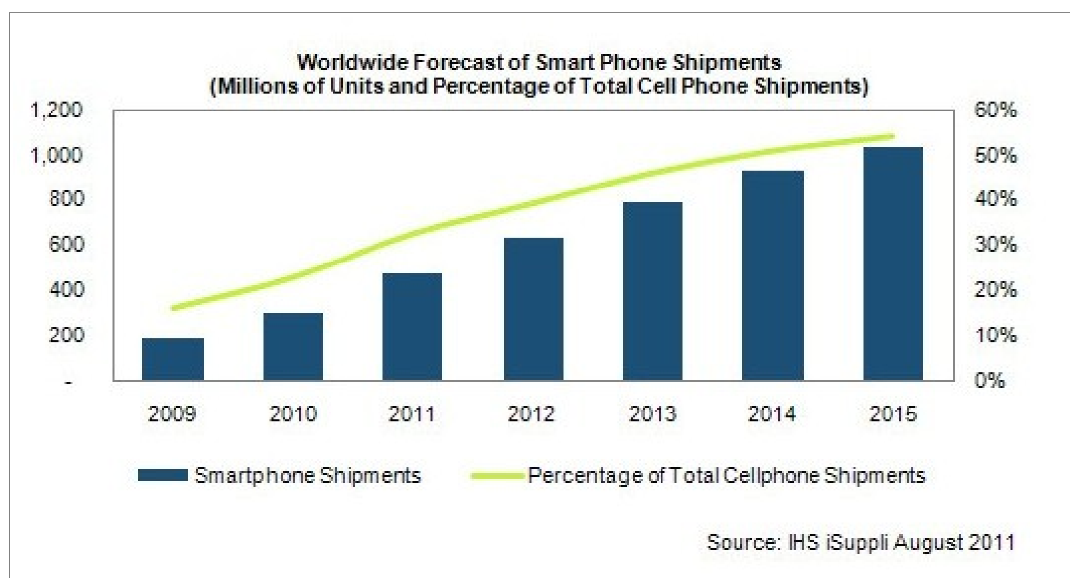


Ilustración 1: Evolució d'Android

Una de les característiques que fa molt atractiu aquests sistemes es que qualsevol desenvolupador pot crear aplicacions ja que disposen de molta documentació, llibreries i entorns de programació. Mitjançant els anomenats "*Markets*" , que no es altra cosa que un directori d'aplicacions, pot publicar la seva aplicació i accedir a tot aquest mercat de *smartphones*.

Aquesta facilitat comporta l'altre cara de la moneda, tal i com succeïx a Internet. Els anomenats "*ciberdelincuents*" poden crear tot tipus de *malware* i distribuir-lo a un gran nombre d'usuaris amb pocs o nuls coneixements sobre noves tecnologies. Aquest escenari es habitual al món d'Internet i ja comença a ser una realitat també al món Android. Per tant, Android és una plataforma creixent amb molt de potencial com a sistema operatiu per a molts dispositius.

Per això s'ha escollit aquesta plataforma per a desenvolupar aquest projecte. Donat l'area en que es presenta aquest projecte, el camí escollit és estudiar una vulnerabilitat dins d'aquesta plataforma.

L'elecció d'una vulnerabilitat i més concretament al *tapjacking*, és degut a dos factors:

- Una vulnerabilitat no recent permet que la l'aprenentatge sigui menys difícil, donat que ja existeix documentació.
- Aquesta vulnerabilitat, tot i no ser recent, encara afecta a un cert nombre de dispositius.

Com a ampliació del darrer punt, cal dir que Android, al igual que qualsevol altre software, es una plataforma que es va actualitzant amb noves versions. En aquest cas, Google, que és la companyia que es troba darrere d'aquesta plataforma, és la que publica periòdicament aquestes actualitzacions.

S'introdueixen millores a l'interfície gràfica (conegut com UI), noves classes, etc.. o fins i tot, quan són moltes millores, és presenten com a noves versions. A més de millores visibles, també s'introdueixen solucions als "*bugs*" o fallades que la comunitat troba i reporta.

El *tapjacking* fou reportat a Google poc abans de la presentació de la versió 2.3 d'Android coneguda com ***Gingebread.*** Per tant, a partir d'aquesta versió, la plataforma Android incorpora un mecanisme de seguretat per minimitzar aquest problema.

¿Però que passa amb les versions anteriors? Cada fabricant de telèfons mòbils o d'altres dispositius, incorpora una versió d'Android depenent de quina s'ajusti millor al propi aparell. És a dir, cada plataforma Android, al igual que un SO d'un ordinador (Android no deixa de ser un SO) té uns requeriments de hardware recomanables. Per tant, és el propi fabricant qui decideix quina versió d'Android utilitzar. A més, degut a la llicència del software, cada fabricant pot millorar, adequar, modificar, etc.. la versió d'Android que incorpora al dispositiu(per exemple *HTC* amb el seu *HTC Sense*). Això que hem explicat es coneix en el món Andorid com a **fragmentació.** Aquest es un dels problemes que pateix Android i que per exemple no pateix *Apple* amb el *iOS*, ja que aquest SO només s'executa en les diferents versions del *iPhone* que fabrica la pròpia *Apple*.

Per tant, aquesta llibertat que hi tenen els fabricants, és per un costat, un avantatge que ha fet que l'ús d'Android creixi exponencialment i per altre banda és un inconvenient per les actualitzacions. Google no té control sobre les versions que cada fabricant instal·la als seus dispositius. Son els propis fabricants els que han d'actualitzar els seus dispositius amb les millores que publica Google. Normalment aquestes actualitzacions s'alliberen en forma d'*OTA* (*Over-The-Air*)[11], és a dir, el propi dispositiu es descarrega l'actualització i s'instal·la. Però no sempre ho fan. Per tant, encara avui dia hi ha dispositius "vulnerables".

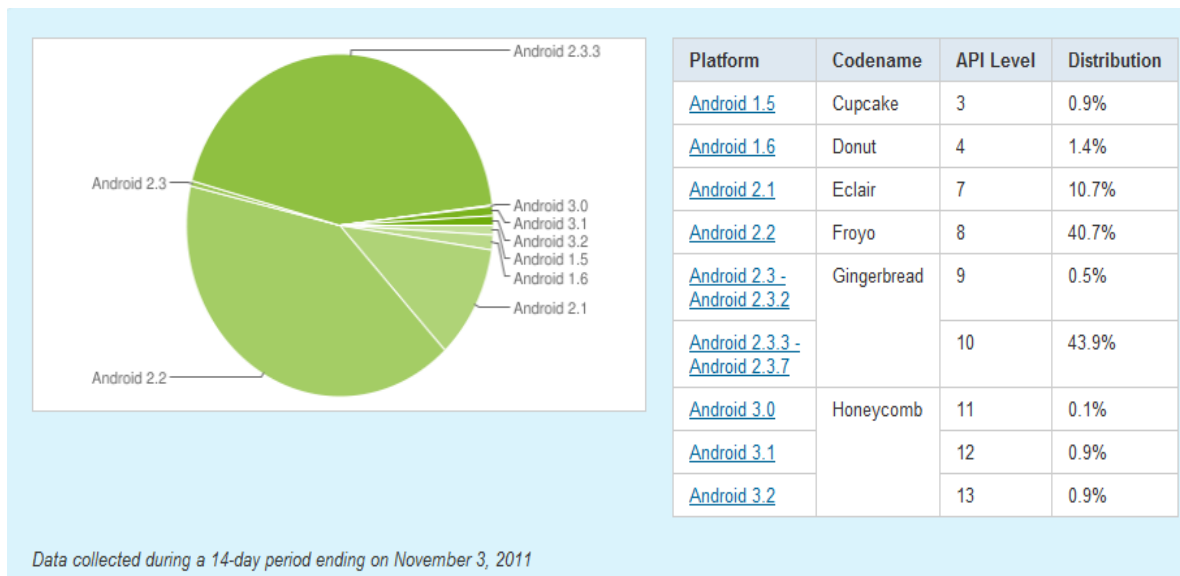
Segons el gràfic [7] encara hi ha un **53.7%** de dispositius que executen una versió vulnerable, com son:

- **Android 1.5**
- **Android 1.6 Donut**
- **Android 2.0/2.1 Eclair**
- **Android 2.2 Froyo**

Per tant, tot i que el *tapjacking* no afecta a les versions mès actuals de la plataforma Android, encara hi ha un nombre elevat de dispositius que no executen la darrera versió. El motiu, com ja hem comentat, es deu principalment a que no tots els dispositius poden executar la darrera versió. Cada fabricant actualitza els dispositius un cop ha fet proves i pot garantir que aquest podrà executar sense problemes la nova versió.

A més, també hi ha motius econòmics, ja que prefereixen presentar nous dispositius més potents i "obligar" als usuaris a canviar el seu telèfon mòbil vell per un de nou.

La següent figura mostra el nombre de versions segons les activacions que es detecten. Aquest gràfic es modifica constantment però en el moment inicial del projecte era aquesta la distribució:



Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.9%
Android 1.6	Donut	4	1.4%
Android 2.1	Eclair	7	10.7%
Android 2.2	Froyo	8	40.7%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7		10	43.9%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	0.9%
Android 3.2		13	0.9%

Ilustración 2: Nombre de dispositius per versions

2.2- Estat de l'art

Les tecnologies que intervendran al projecte seran:

- **Android 1.6 Platform**

  Android acompleix la "*retrocompatibilitat*" és a dir, que versions més noves poden fer servir aplicacions dissenyades per versions més antigues, però no a l'inrevés. Per tant, per aconseguir que l'aplicació funcioni en el màxim de dispositius, cal que sigui implementada amb el SDK més antic possible. D'aquesta manera serà compatible amb aquest SDK i amb SDK's posteriors. Per tant, l'elecció primera es fer servir la versió 1.6
  NOTA: Les instruccions per l'instalacció del SDK d'Android es troben en l'annex.

- **ADT Plugin for Eclipse**

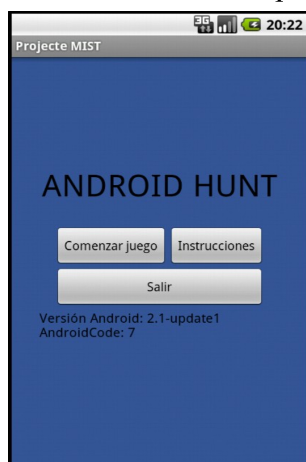
  Per al desenvolupament tant de la prova de concepte del *tabjacking* com per la segona l'aplicació, farem servir l'entorn de programació Eclipse en la seva versió per JavaEE, disponible a [8].
  Google, per la seva banda, ha posat a disposició dels desenvolupadors d'Android un *plugin* per aquest entorn de programació, disponible a [9].
  NOTA: Les instruccions per l'instalacció del *plugin* ADT es troben en l'annex.

2.3- Productes obtinguts

Per a aquest projecte s'han desenvolupat dues aplicacions completament funcionals e independents. Ambdues aplicacions es poden desplegar en qualsevol dispositiu amb Android com a sistema operatiu. Durant el seu desenvolupament s'ha fet servir un emulador disponible dintre del SDK d'Android. Les dues aplicacions són:

- **Android HUNT**

  Aquesta aplicació és una prova de concepte de la vulnerabilitat que detallarem en succesius apartats.



- **MIST**

  Aplicació que preten solventar la vulnerabilitat presentada.

2.4- Planificació del projecte

2.4.a- Contingut de les entregues parcials

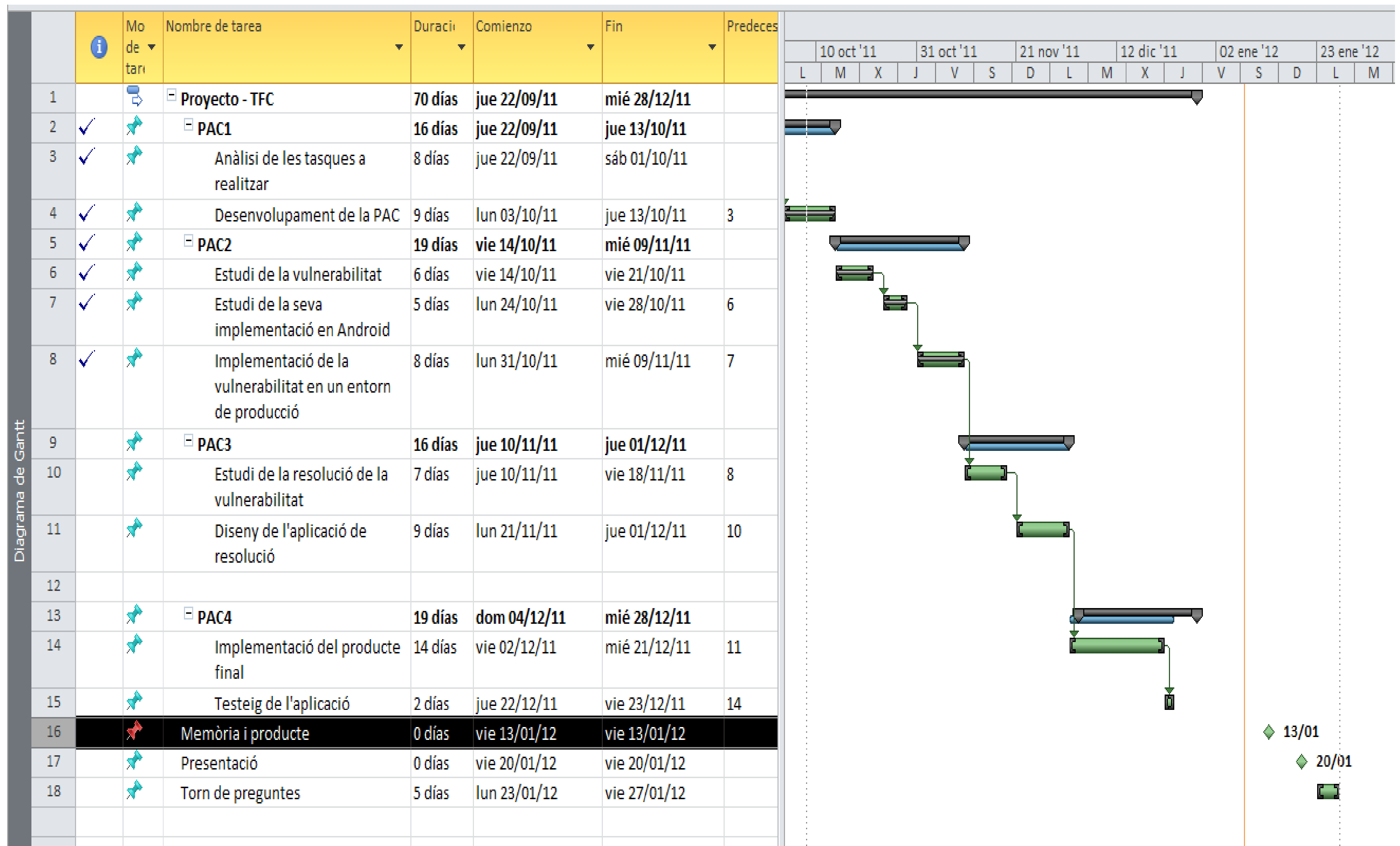
- **PAC1**
  - Pla de Treball i Estat de l'art
- **PAC2**
  - Documentació de la prova de concepte
  - Aplicació funcional per la plataforma Android
- **PAC3**
  - Estudi de la resolució de la vulnerabilitat
- **PAC4**
  - Documentació de la resolució
  - Aplicació funcional de la solució per la plataforma Android.

2.4.b- Entregues no presentades

En el primer pla de treball hi havia inclosos algunes entregues que finalment no han estat incloses al projecte final:

- Estava previst publicar l'aplicació final al *Android Market*, però donat que finalment no supera uns mínims de qualitat per resoldre la vulnerabilitat presentada i donat que un compte de desenvolupador a l'*Android Market* no es gratuït, finalment he desestimat incloure aquesta part al projecte final.
- Estava previst, un cop publicada l'aplicació al *Android Market*, crear una plana web per publicitar l'aplicació. Donat que el pas anterior no s'ha inclòs finalment, aquest pas tampoc s'ha fet, donat que depenia del anterior.

2.4.c- Diagrama de Gantt



	Mo de tar	Nombre de tarea	Duraci	Comienzo	Fin	Predeces
1		**Proyecto - TFC**	**70 días**	**jue 22/09/11**	**mié 28/12/11**	
2	✓	**PAC1**	**16 días**	**jue 22/09/11**	**jue 13/10/11**	
3	✓	Anàlisi de les tasques a realitzar	8 días	jue 22/09/11	sáb 01/10/11	
4	✓	Desenvolupament de la PAC	9 días	lun 03/10/11	jue 13/10/11	3
5	✓	**PAC2**	**19 días**	**vie 14/10/11**	**mié 09/11/11**	
6	✓	Estudi de la vulnerabilitat	6 días	vie 14/10/11	vie 21/10/11	
7	✓	Estudi de la seva implementació en Android	5 días	lun 24/10/11	vie 28/10/11	6
8	✓	Implementació de la vulnerabilitat en un entorn de producció	8 días	lun 31/10/11	mié 09/11/11	7
9		**PAC3**	**16 días**	**jue 10/11/11**	**jue 01/12/11**	
10		Estudi de la resolució de la vulnerabilitat	7 días	jue 10/11/11	vie 18/11/11	8
11		Diseny de l'aplicació de resolució	9 días	lun 21/11/11	jue 01/12/11	10
12						
13		**PAC4**	**19 días**	**dom 04/12/11**	**mié 28/12/11**	
14		Implementació del producte final	14 días	vie 02/12/11	mié 21/12/11	11
15		Testeig de l'aplicació	2 días	jue 22/12/11	vie 23/12/11	14
16		Memòria i producte	0 días	vie 13/01/12	vie 13/01/12	
17		Presentació	0 días	vie 20/01/12	vie 20/01/12	
18		Torn de preguntes	5 días	lun 23/01/12	vie 27/01/12	

Ilustración 3: Diagrama de Gantt

3- Plataforma Android

3.1- Història d'Android

Android com a plataforma es va iniciar oficialment el 22 d'Octubre de 2008 als EUA, amb el llançament del telèfon móbil **G1 de T-Mobile.** Aquella **versió 1.0** ja disposava de:

- Finestra de notificacions desplegable per fer accessible l'informació.
- Widgets a la pantalla d'inici per mostrar tot tipus d'informació.
- Integració amb Gmail
- Android Market com a botiga per la distribució d'aplicacions.

Al Febrer de 2009 es va llençar la **versió 1.1.** Únicament es van corregir *bugs* de la versió anterior i es va implementar les actualitzacions OTA ("*over the air").* Aquest tipus d'actualitzacions permet actualitzar el dispositiu sense necessitat de cap ordinador.

La següent versió d'Android es la **versió 1.5** alliberada el 30 d'Abril d'aquest mateix any. Aquesta versió ja fou batejada amb el nom d'un postre amb una inicial seguint l'ordre alfabètic, mètode que Google continua fent servir. Aquesta versió 1.5 fou batejada amb el nom de ***Cupcake.*** Aquesta versió ja incorpora:

- Teclat virtual en substitució dels teclats físics del dispositiu amb predicció de text..
- Millora dels *widgets* amb l'inclusió de *widgets* de tercers.
- Millora del porta-papers.
- Reproducció i grabació de vídeo amb diversos formats.
- Suport per a Bluetooth A2DP

La següent versió és **Android 1.6 Donut**. Alliberada el 30 de Setembre. Aquesta versió dóna suport a les xarxes CDMA, que permet la seva expansió als mercats asiàtics. Aquesta versió incorpora:

- Suport per a diferents resolucions de pantalla, adaptant-se als diferents dispositius.
- Cerca dintre del propi dispositiu, com contactes o correus.
- Millora del Android Market.

Al Novembre del 2009 s'allibera la **versió 2.0 Eclair** amb el llançament mundial del *Motorola Droid* que va impulsar definitivament Android com a plataforma per a dispositius mòbils. Posteriorment es llença la **versió 2.1** que manté el mateix nom ja que no es considera una nova versió si no una millora o correcció d'errors de la **2.0**

Aquesta versió incorpora:

- Suport per a diferents comptes per gestionar el dispositiu.
- Google Maps Navigation que permet fer servir el dispositiu com a navegador per automòbils.
- Suport per *Microsoft Exchange*
- Fons de pantalla animats
- Noves pantalles de bloqueig del terminal

La següent **versió 2.2** s'allibera al Maig del 2010 amb el nom en clau de **Froyo.** El telèfon mòbil **Nexus One** va ser el primer dispositiu en incorporar aquesta nova versió. Aquesta versió incorpora com a novetat:

- Funcionalitat de *Hotspot*, que permet al dispositiu actuar com a router WI-FI.
- Suport per *Adobe Flash*
- Millores en la resta de funcionalitats.

A partir d'aquí les versions comencen en intervals més grans. La següent versió és **Android 2.3 GingerBread.** Aquesta és la darrera versió dirigida únicament a dispositius mòbils. El seu inici va lligat amb el llançament del següent mòbil propietat de Google amb la colaboració de *Samsung,* el Nexus S. Aquesta versió d'Android incorpora:

- Suport per *Near Field Communication (NFC)*
- Redisseny de l'interfície d'usuari
- Canvi del sistema d'arxius de YAFFS a ext4
- Suport per múltiples càmeres, incloses les càmeres frontals.
- Millores d'anteriors funcionalitats.

La següent versió és la versió **Android 3.0 Honeycomb**. Aquesta versió incorpora suport per a *tablets.* Aquesta versió fou presentada amb la nova tableta *Motorola Xoom.* Aquesta versió incorpora:

- Escriptoris 3D amb *widgets* redissenyats
- Suport per a perifèrics mitjançant els ports USB
- Redisseny de l'interfície amb l'inclusió de botons virtuals en substitució dels físics.

La darrera versió presentada fa poc temps és la versió actual **Android 4.0 Ice Cream Sandwich.** Aquesta versió ha sigut presentada amb el nou mòbil de Google *Galaxy Nexus* i s'espera que els terminals de gama alta el vagin incorporant mitjançant actualitzacions via OTA.

3.2- Android com a Sistema Operatiu

Android es un paquet de software basat en codi obert per a telèfons móbils, tabletes, etc.. creat per Google y la *Open Handset Alliance*. És una plataforma basada en un kernel Linux que inclou interfícies d'usuari, aplicacions, biblioteques de codi, etc.. Les aplicacions que és dissenyen per Android s'escriuen en el llenguatge de programació Java, ja que Android proporciona la màquina virtual *Dalvik* que utilitza els serveis del nucli basat en *Linux* per proporcionar un entorn d'allotjament per a les aplicacions.

A continuació, el diagrama de l'arquitectura d'Android[12] que il·lustra els seus components:

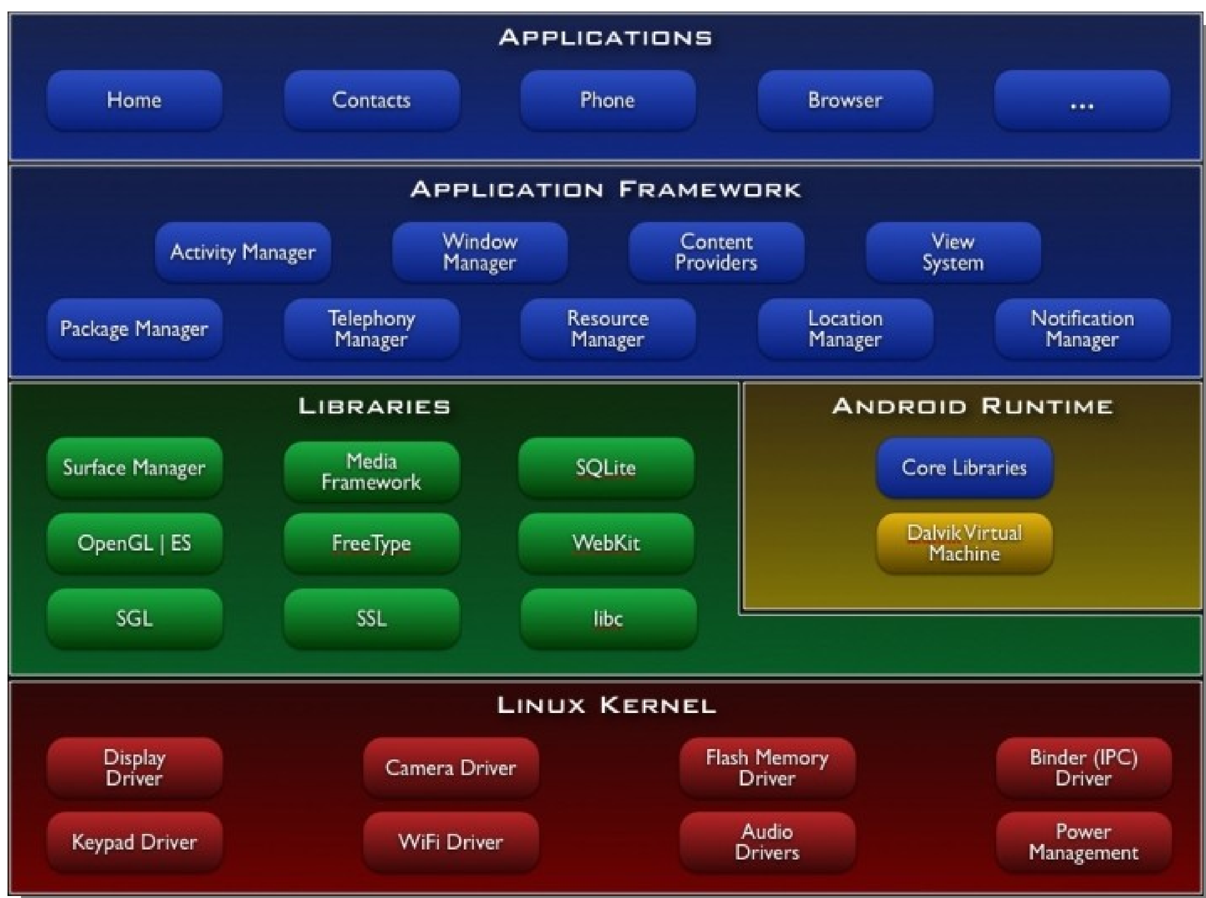


Ilustración 4: Arquitectura d'Android

Hi ha quatre components bàsics per construir aplicacions Android:

- **`Activities` (Activitats)**

Les activitats representen una única pantalla amb la qual l'usuari interactua. Normalment les aplicacions tindran una o més activitats, però no es obligatori. Una d'aquestes activitats és l'activitat principal i és l'activitat que es carrega a l'inici. Les interfícies es poden declarar tant mitjançant codi Java com mitjançant arxius xml.

- ***Services* (Serveis)**

Un servei és un component que s'executa en segon pla. No té interfície i es troba enllaçat a una activitat. És semblant als demonis de *Linux* o els serveis de Windows.

- ***Content Provider* (Proveïdor de Contingut)**

És un component que permet intercanviar dades entre aplicacions. Les aplicacions, degut a les restriccions de seguretat imposades, només poden accedir a les seves pròpies dades. Per tant, per poder intercanviar dades s'utilitza aques component.

- ***Broadcats Receivers* (Receptors d'emissió)**

Són components que responen a determinats esdeveniments emesos pel sistema o per altres aplicacions.

Aquests són els quatre components bàsics d'Android per la construcció d'aplicacions. Un altre element molt important és el que s'anomena ***Intent* (intenció).** Són missatges asíncrons que el sistema "llença" constantment per notificar diversos esdeveniments. Aquests *intents* també poden ser emesos per les aplicacions. Un exemple d'*intent* seria:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("tel://666999666"));
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

Aquest *intent* defineix una acció, que en aquest cas es *Intent.ACTION_VIEW* i sobre quines dades es vol es vol realitzar l'acció amb format *URI*. Existeixen dos tipus d'*intents*:

- *Intents explícits*

  Aquest tipus d'*intents* indiquen explícitament quin component ha d'atendre l'*intent*.

- *Intents implícits*

  Aquest *intents* no defineixen qui ha d'atendre l'*intent*.

Android detecta aquest llançament en el cas dels *intents implícits* i busca quin component, que prèviament ho ha declarat mitjançant els ***<intent - filter>*** al seu fitxer *AndroidManifest.xml*, el pot executar. Per exemple, l'aplicació que declares el següent ***<intent - filter>*** podria atendre l'*intent* anterior:

```
<intent-filter android:priority="10">
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:scheme="tel" />
</intent-filter>
```

El filtre s'estableix per acció, per categoria i per tipus de dades. Perquè un component sigui seleccionat, l'***<intent - filter>*** ha de coincidir amb l'*intent*. Per exemple, l'*intent* anterior defineix la seva acció com . L'***<intent - filter>*** defineix la mateixa acció, per

tant l'acció coincidiria. Si tots els elements coincideixen, el component pot ser seleccionat.

Aquest tema és força ampli, donat que hi ha molts casos diferents i aspectes ha tenir en compte. Per a més informació es pot consultar [19] Els conceptes d'intents implícits i intents explícits és important perquè com es veurà al **capítol 7,** afectaran a a la solució proposada.

Un altre element que cal presentar ja que és un element clau per explicar la vulnerabilitat, tot i que aprofundirem més endavant, són els anomenats ***Toast.*** Un *Toast* és un element gràfic semblant a un globus i que permet mostrar informació al usuari. En Android, tots els elements gràfics com poden ser els botons, els camps de text, etc.. hereten de la classe *View*. Per tant, hereta molts mètodes que tenen d'altres elements.

Finalment, una altre característica de la plataforma Android que cal presentar donat que hi té relació amb la vulnerabilitat que presentarem més endavant son els **permisos** en Android. Android, donat que la plataforma fa servir un kernel de Linux, hereta el sistema de permisos que es fa servir en Linux, ampliant la seva funcionalitat. Els permisos són un mecanisme per restringir determinades operacions específiques que hi podem realitzar les aplicacions. Per defecte i com a part central del disseny de l'arquitectura, les aplicacions no disposen dels permisos de forma implícita i cal atorgar-li els permisos de forma explícita. D'aquesta manera, cada aplicació no interfereix amb altres aplicacions ni amb el propi sistema.

Exemples de permisos poden ser:

- Accedir íntegrament a la xarxa (Internet)
- Enviar i/o rebre trucades o missatges SMS/MMS
- Accedir a parts del dispositiu (càmera, Bluetooth, etc...)

La llista completa de permisos es pot trobar a [16]. Aquest permisos es defineixen dintre del fitxer *AndroidManifest.xml*. Aquest fitxer defineix tots els aspectes i característiques d'una aplicació. Cada aplicació disposa del seu fitxer únic i personal. Un exemple de definició de permisos seria:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.FLASHLIGHT"/>
```

Quan es defineixen aquest permisos que l'aplicació necessita per funcionar, ja sigui completament o per a determinades funcionalitats, en el moment d'instal.lar l'aplicació s'accepten aquests permisos. És a dir, l'aplicació necessita de certs permisos que es defineixen al *AndroidManifest.xml*. Quan s'instal.la l'aplicació, automàticament el sistema li concedeix aquests permisos de forma exclusiva a aquesta aplicació. Per tant és important per part de l'usuari final conèixer el sistema de permisos i només concedir permisos a aquelles aplicacions de confiança.

L'*Android Market* és la botiga d'aplicacions, que com vàrem presentar al capítol anterior, ja va ser presentat amb la versió inicial d'Android. L'*Android Market* inicialment estava concebut com una altre aplicació, tot i que recentment Google ha desenvolupat una plana web

disponible a [20].

En un inici, la llista de permisos requerits per una aplicació no estaven suficientment clars per a l'usuari mig. Amb la darrera actualització, Google ha inclòs una pestanya on s'especifica molt clarament els permisos requerits per una aplicació. Especialment queden remarcats aquells permisos que poden tenir un cost per l'usuari, com poden ser **l'enviament de missatges SMS/MMS i/o trucades telefòniques.**

A la següent figura tenim un exemple de sol.licitud de permisos en la plana web del `Market`:

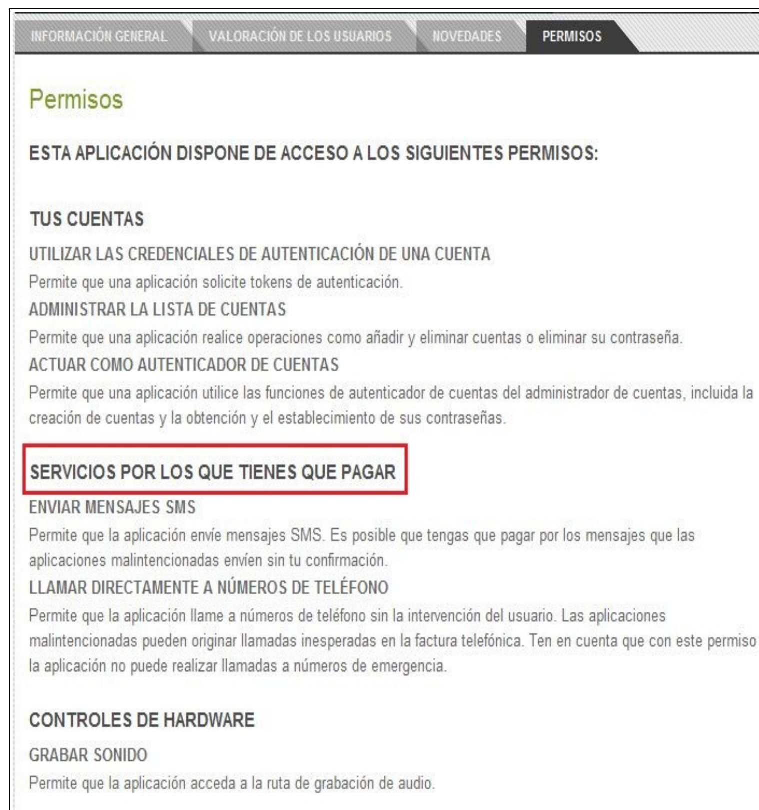


Ilustración 5: Permisos en l'Android Market

Com veurem més endavant, el tema de permisos és important, no tan sols com a coneixement de la plataforma Android, si no que és important per entendre l'abast de la vulnerabilitat que presentarem tot seguit.

4- Vulnerabilitat

4.1- Introducció

La vulnerabilitat en la plataforma Android que presentarem en el següents capítols té el seu origen en una tècnica maliciosa que va ser presentada per Jeremiah Grossman i Robert Hansen l'any 2008.

Aquesta tècnica va ser anomenada *clickjacking* (segrest de clic)[2][3] i té la seva àrea d'efecte al món web.

Aquesta tècnica consisteix en enganyar a l'usuari perquè faci clic en una determinada part d'una web especialment manipulada sense que se'n adoni.

Bàsicament consisteix en carregar un *iframe* opac i maliciós a pantalla completa i la web legítima en una capa per sota. L'usuari només veu la capa opaca, que es visible i interactua amb aquesta, però realment està interactuant amb la capa no visible que hi ha per sota. Només cal, amb codi *Javascript* per exemple, capturar les pulsacions, redirigi-lo a d'altres llocs, carregar scripts maliciosos, etc..

El nivell d'afectació queda als coneixements dels “*ciberdelincuents*”.

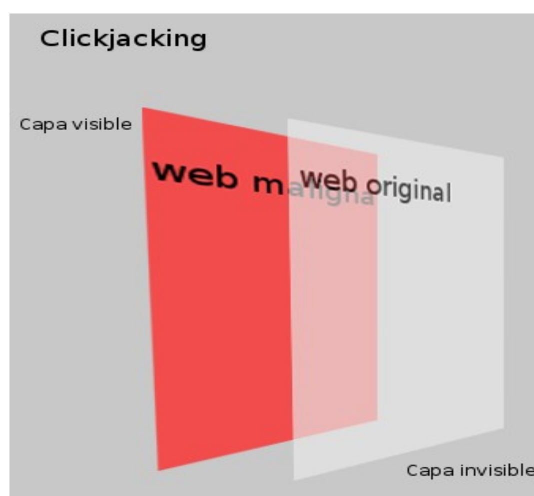


Ilustración 6: Clickjacking

Aquesta tècnica es catalogada com greu, degut a que afecta a l'estructura bàsica del propi funcionament de la web. Els llenguatges web, que es basen en protocols i estàndards, permeten crear transparències i capes i no es pot controlar si són malicioses o legítimes.

A partir d'aquesta tècnica, es va traslladar el mateix concepte a la plataforma Android i que ha donat peu a la vulnerabilitat que presentem a continuació.

4.2- Tapjacking

Un cop presentada la tècnica *clickjacking*, presentarem la seva homònima dintre de l'ecosistema Android. En aquest cas rep el nom de *tapjacking*[4] i el seu funcionament es molt semblant, tot i que en aquest cas no esta relacionat amb el món web.

La motivació que possiblement va desencadenar que aquesta tècnica ja coneguda s'intentés adaptar a la plataforma Android (i que finalment es va aconseguir) es que quan un usuari vol instal.lar una aplicació a Android, ja sigui desde l'*Androd Market* oficial o desde altres repositoris (totalment desaconsellable, per altre banda), es demana prèviament a l'usuari que accepti certs permisos definits dintre de la pròpia aplicació, com ja es va detallar en el capítol anterior dedicat a l'arquitectura Android.

Quan una aplicació ha de realitzar certes tasques, ha de tenir els permisos adients per fer-ho, ja que si no el sistema directament no li ho permet. Per tant, aquest permisos es declaren a l'aplicació i quan l'usuari la vol instal.lar, se li mostren perquè els accepti o els rebutgi. No hi ha terme mig. No pot acceptar uns i negar altres. Si l'usuari detecta massa permisos, no instal.la l'aplicació (tot i que diversos estudis demostren que l'usuari no llegeix prou i instal.la qualsevol cosa).

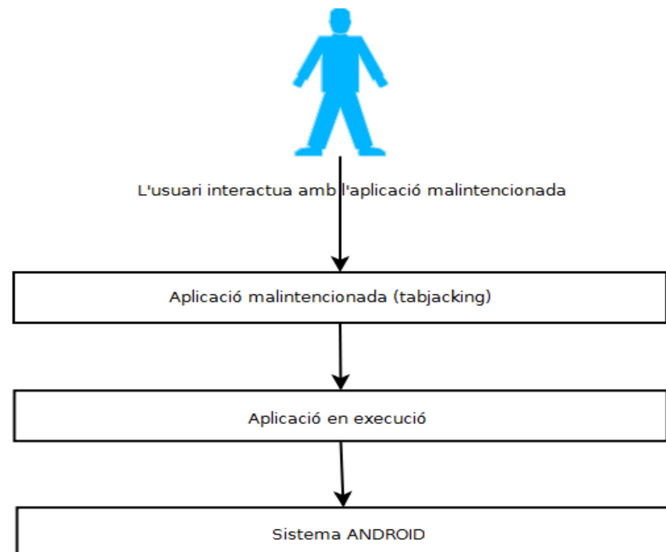


Ilustración 7: Esquema Tapjacking

Per tant, la motivació és intentar evitar aquestes restriccions del sistema de seguretat de la plataforma.

És a dir, la idea es que si no ho podem fer perquè el sistema de permisos no ens deixa, farem que sigui el propi usuari el que faci allò que no podem. Això s'aconsegueix aplicant la tècnica del clickjacking a la plataforma Android, que va ser la base d'inspiració per la vulnerabilitat sobre la qual versa aquest projecte i que ha rebut el nom de ***tapjacking*****.**

4.3- Anàlisi del tapjacking

Com ja hem comentat, la tècnica bàsicament, ja sigui el clickjacking o el tapjacking, consisteix en tenir dues capes, una visible i un altre invisible de cara a l'usuari, de forma que l'usuari cregui que interactua amb una de les capa visible però veritablement està interactuant amb la capa invisible.

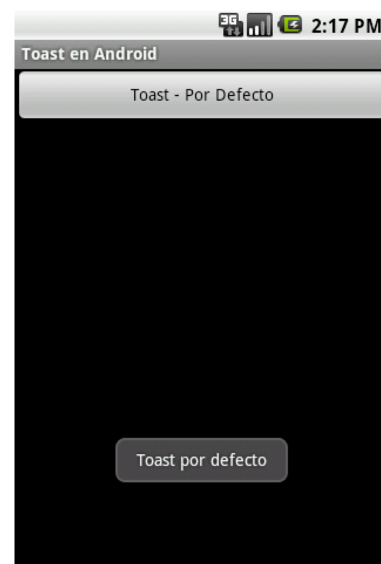


Ilustración 8: Element Toast en Android

En el món web, l'element o capa que es fa opaca sóls ser un *iframe*. En Android, i com ja és preveu, l'element gràfic que fa de capa opaca és el ***Toast*****.**

Normalment, aquest element s'utilitza per mostrar informació al usuari. És la seva utilitat bàsica i per això s'hauria de fer servir. Com es pot veure al gràfic, solen ser de la mateixa mida que el text que mostrem i tenen una duració determinada.

La motivació de fer servir aquest element, que a priori no sembla pràctic, es per una qualitat que només disposa aquest element i que en el fons és una petita errada per part de

Google. El *Toast* és un element gràfic i com tots els elements hereta de la classe pare *View*. Tots els elements que hereten de *View* disposen de funcions per definir la seva amplada i/o altura, el seu fons ja sigui amb un color o una imatge i la seva opacitat (a més d'altres característiques). Per tant, tot i que normalment tenen l'aspecte que és veu a l'imatge, el podem fer tant gran com per cobrir tota la pantalla del dispositiu. A més, el podem fer opac per ocultar el que hi hagi a sota i fins i tot li podem definir una imatge com a *background*.

Tot això no es especial dels *Toast*. Qualsevol element que hereti de *View* té aquestes qualitats. Llavors, ¿qué el fa diferent? Doncs que per Android és com si no hi hagués cap element. És a dir, no captura els esdeveniments com pot ser un *clic* que es faci sobre ell. Per tant, ja tenim el nostre element perfecte que farà de capa opaca per ocultar la pantalla principal.

Però, com hem comentat, els *Toasts* tenen una duració limitada. ¿Com podem eliminar aquesta restricció? Avançant el que veurem al cas pràctic, podem executar el *Toast* en un altre fil d'execució (*Thread*) diferent del fil d'execució de l'intefície principal. Per tant, tenim dos fils, un que executa la pantalla principal i altre que va executant el *Toast* per sobre de forma ininterrompuda. És a dir, el va recarregant sense parar, com si fos una pel·lícula i cada *Toast* fos un fotograma.

Per tant, el que farem és dissenyar un *Toast* que simuli un joc on l'usuari hagi d'anar fent *clic* en determinades posicions. Aquest *Toast* s'executarà en un fil d'execució i al mateix temps, construirem un *intent* com serà enviar un SMS o fer una trucada. Android capturarà aquest *intent* i executa l'aplicació corresponent, que queda totalment oculta al usuari.

A la prova de concepte que es desenvoluparà en els següents capítols, quedarà pal.lès fins a quin punt pot arribar la perillositat d'aquesta tècnica o vulnerabilitat. La prova de concepte es desenvoluparà tenint en compte només dos tipus d'intents o accions diferents, que ja detallarem, però es podria aplicar per a molts tipus d'accions diferents.

És a dir, aquesta tècnica ens permet evitar el sistema de seguretat de la plataforma Android. Per tant, totes aquelles accions que es va decidir que per la seva naturalesa les aplicacions havien de sol.licitar un permís explícit si les volien fer servir, ara passen a no necessitar-ne si apliquem la tècnica del tapjacking.

Aquesta darrera reflexió ens demostra fins a quin punt pot arribar a ser potencialment perillosa aquesta tècnica i les seves possibilitats.

4.4- La solució de Google

Google, per solucionar el *Tapjacking* va incloure en la versió 2.3 d'Android un mecanisme per filtrar els esdeveniments de tipus *Touch* (contacte) quan l'element de tipus *View* estigui tapat per un altre element. Per tant, i degut a que tots els elements gràfics hereten de *View*, tots incorporen aquest mecanisme.

Per activar aquest mecanisme, només cal fer servir la funció ***setFilterTouchesWhenObscured(boolean)*** sobre l'element gràfic si es declara de forma procedimental, és a dir, mitjançant codi Java o fent servir l'etiqueta ***<elementGrafic android:filterTouchesWhenObscured=”true” />*** en el fitxer *.xml* si es declara de forma declarativa. Per exemple:

```
<Button android:text="Button"
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:filterTouchesWhenObscured="true">
</Button>
```

- Fragment extret de la pàgina web d'Android (*Views ->Security*)[13]:

“Sometimes it is essential that an application be able to verify that an action is being performed with the full knowledge and consent of the user, such as granting a permission request, making a purchase or clicking on an advertisement. Unfortunately, a malicious application could try to spoof the user into performing these actions, unaware, by concealing the intended purpose of the view. As a remedy, the framework offers a touch filtering mechanism that can be used to improve the security of views that provide access to sensitive functionality.

To enable touch filtering, call setFilterTouchesWhenObscured(boolean) *or set the android:filterTouchesWhenObscured layout attribute to true. When enabled, the framework will discard touches that are received whenever the view's window is obscured by another visible window. As a result, the view will not receive touches whenever a toast, dialog or other window appears above the view's window.”*

“A vegades, es essencial que una sol·licitud sigui capaç de verificar que una acció es realitza amb ple coneixement i consentiment de l'usuari, com per exemple la concessió d'un permís, fer una compra o seleccionar un anunci. Per desgracia, una aplicació malintencionada podria intentar suplantar l'usuari en la realització d'aquestes accions, ocultant aquestes accions. Com a solució, el marc de treball ofereix un mecanisme de filtrat de contacte, que pot ser utilitzat per millorar la seguretat de les vistes que proporcionen accés a aquestes funcions sensibles.
Per habilitar el filtratge, faci servir setFilterTouchesWhenObscured(boolean) o estableixi l'atribut de disposició android:filterTouchesWhenObscured en verdader. Quan estigui

> *activat, el marc de treball descartarà els contactes que es rebin quan la ventana estigui obscurida per un altre. Com a resultat la vista no rebrà els contactes que faci sobre un Toast, una finestra de diàleg o qualsevol altre finestra per sobre."*

Fent servir qualsevol dels dos mètodes, tant si és procedimental amb codi Java o declaratiu mitjançant etiquetes al fitxer *.xml* del *Layout*, el que s'aconsegueix és que si l'element gràfic sobre el que activem aquest filtre es troba "*tapat*" per un altre element gràfic, no rep l'esdeveniment *Touch*.

Per analitzar possibles solucions, vaig navegar pel codi font d'Android per trobar com esta implementat. Els dos mètodes **get** i **set** de la classe *View* es troben a continuació, tal qual es troben al codi font d'Android 2.3:

```
/**
     * Gets whether the framework should discard touches when the view's
     * window is obscured by another visible window.
     * Refer to the {@link View} security documentation for more details.
     *
     * @return True if touch filtering is enabled.
     *
     * @see #setFilterTouchesWhenObscured(boolean)
     * @attr ref android.R.styleable#View_filterTouchesWhenObscured
     */
    @ViewDebug.ExportedProperty
    public boolean getFilterTouchesWhenObscured() {
        return (mViewFlags & FILTER_TOUCHES_WHEN_OBSCURED) != 0;
    }

    /**
     * Sets whether the framework should discard touches when the view's
     * window is obscured by another visible window.
     * Refer to the {@link View} security documentation for more details.
     *
     * @param enabled True if touch filtering should be enabled.
     *
     * @see #getFilterTouchesWhenObscured
     * @attr ref android.R.styleable#View_filterTouchesWhenObscured
     */
    public void setFilterTouchesWhenObscured(boolean enabled) {
        setFlags(enabled ? 0 : FILTER_TOUCHES_WHEN_OBSCURED,
                FILTER_TOUCHES_WHEN_OBSCURED);
    }
```

Aquest codi es troba dins del fitxer *View.java* que es pot trobar a la carpeta *sources/android/view/View.java*

Aquests mètodes serveixen per assignar i recuperar la variable *FILTER_TOUCHES_WHEN_OBSCURED*. Dintre de la classe *View* trobem el mètode que filtra els esdeveniments:

```
/**
     * Filter the touch event to apply security policies.
     *
     * @param event The motion event to be filtered.
     * @return True if the event should be dispatched, false if the
event should be dropped.
     *
     * @see #getFilterTouchesWhenObscured
     */
    public boolean onFilterTouchEventForSecurity(MotionEvent event) {
        if ((mViewFlags & FILTER_TOUCHES_WHEN_OBSCURED) != 0
                && (event.getFlags() &
MotionEvent.FLAG_WINDOW_IS_OBSCURED) != 0) {
            // Window is obscured, drop this touch.
            return false;
        }
        return true;
    }
```

Per tant, si un objecte de tipus `View` i descendents rep un esdeveniment i es troba activat aquesta *flag,* retorna *false* i no rep l'esdeveniment.

Aquesta és la solució més elegant, però per això cal accedir al codi font, fer les modificacions pertinents i recompilar el codi. Una opció hagués sigut *"cuinar"* una ROM, com es denomina a re-compilar el codi font d'Android amb les modificacions pertinents per part d'un usuari particular però l'instal.lació d'un ROM requereix de coneixements que queden fóra de l'usuari normal.

5- Aplicació ANDROID HUNT

5.1- Prova de concepte

En aquest apartat i com a pas previ a l'estudi i implementació d'una aplicació que minimitzi els efectes del *tapjacking* en dispositius vulnerables, he implementat un cas pràctic d'aplicació que explota tot allò que hem explicat anteriorment i que ens servirà, tant per entendre realment com funciona i fins a quin punt pot ser perillós, com per provar si la solució final es veritablement útil. Aquesta aplicació rep el nom de **ANDROID HUNT.**

Per qüestions únicament temporals, només s'ha implementat únicament dos tipus de tests:

- Missatge SMS a un número especial
- Trucada telefònica a un número especial.

Però no només es poden realitzar aquestes dues accions fent servir el *tapjacking*. La veritable potència d'aquesta vulnerabilitat radica en que qualsevol tipus d'acció que es pugui

llençar fent servir un `Intent`, component que ja hem presentat anteriorment, pot ser susceptible de ser utilitzada. Com a exemple, es podria fer:

- Obrir la finestra `Ajustes->Tarjeta SD y memoria` i formatar la targeta de memòria.
- Obrir l'`Android Market` i descarregar de forma silenciosa una aplicació malintencionada (amb tot tipus de permisos) amb un virus incorporat i instal.lar-la.
- Etc...

5.1.a- Arquitectura

- Diagrama de classes de l'aplicació

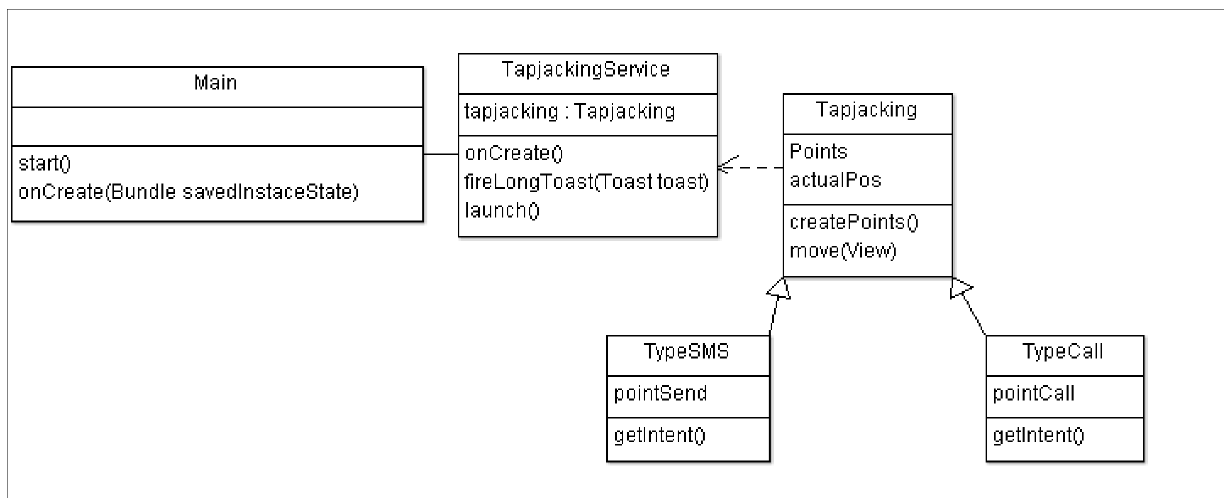


Ilustración 9: Diagrama classes ANDROID HUNT

La classe `Main` és la classe principal, ja que es defineix així al fitxer `AndroidManifest.xml`. Aquest fitxer serveix per definir els components de l'aplicació, com per exemple les *activities* o els *services,* definir quin es el component inicial, els permisos que requereix l'aplicació, etc...

Continuant amb la nostra aplicació, la classe `Main` es defineix com la classe principal que inicia l'aplicació i carrega l'interfície gràfica a partir d'un fitxer .xml. A més, controla el menú de les opcions i serveix per codificar els diàlegs, que serveixen per mostrar informació a l'usuari, al estil dels *alerts* en Javascript.

La part important d'aquesta classe es que inicia la classe `TapjackingService` i li passa com a atribut una instancia del tipus de test que volem que s'executi per sota. En el constructor dels objectes `Tapjacking` s'aprofita per crear els punts on es dibuixaran els gràfics. Les responsabilitats de la classe `Main` terminen aquí ja que qui controla tot es la classe `TapjackingService`.

La classe `TapjackingService` és, com indica el seu nom, un component de tipus `Service`

com vam explicar al capítol dedicat a la plataforma Android. Aquest tipus de components no necessiten cap pantalla per ser executats i per tant continuen executant-se fins i tot quan l'usuari està fent d'altres coses, ja que s'executen en segon pla. Per tant és el component ideal per als nostres propòsits.

Un cop la classe *`Main`* inicia el *`Service`*, aquest executa el mètode ***`onCreate()`***, que es el mètode que s'executa per defecte en els components. En aquest mètode es construeix l'interfície gràfica d'un segon fitxer xml. També es construeix un objecte de tipus *`Toast`* i se li assigna aquesta interfície.

Un cop tenim tot preparat, es crida al mètode ***`fireLongToast(Toast toast)`*** que pinta el *`Toast`* per pantalla. Els *`Toasts`* tenen una duració limitada. **Per això, l'executem en un nou fil d'execució**. D'aquesta manera el podem re-pintar cada cop i no interfereix amb el fil d'execució principal. Aquest es un aspecte clau alhora de desenvolupar la prova de concepte. Si no ho féssim així, no podríem crear la capa invisible per sota, donat que els components (activities, services, etc...) en Android, inclosa la seva interfície s'executant en el mateix fil. Per poder tenir dues capes, una visible per l'usuari i l'altre invisible s'han d'executar en fils diferents.

Finalment, dintre del mètode ***`fireLongToast(Toast toast)`*** fem una crida al mètode ***`Launch()`*** per llançar l'activitat *`Tapjacking`* que també s'executa en el seu propi fil d'execució. Per tant, cada activitat s'executa en el seu propi fil, fent cadascú la seva tasca.

Android disposa un mecanisme per intercanviar dades entre fils diferents. Mitjançant una instancia de la classe *`Handler`* es pot fer arribar dades o missatges d'un fil a un altre, amb els mètodes ***`dispatchMessage(Message msg)`*** i ***`sendMessage(Message msg)`***.

5.1.b- L'aplicació en funcionament

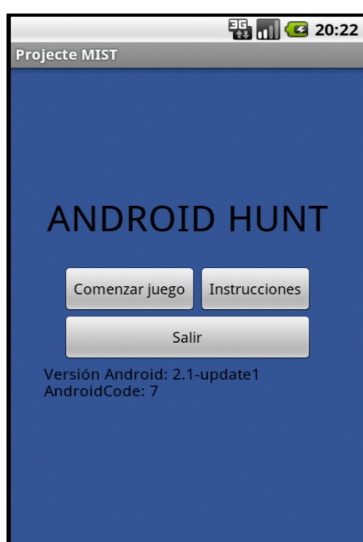


Ilustración 10: Pantalla inicial

Aquesta aplicació s'ha dissenyat com un senzill joc on l'usuari ha d'intentar "*enxampar*" una determinada icona, que en el nostre cas es un petit androide i ha d'ignorar un altre icona, que en el nostre cas es una poma. És un joc senzill, que segurament no seria *Top Ten* en descàrregues a l'*Android Market* però no és aquest el seu propòsit. Es tracta de generar un capa invisible per sota i "*animar*" a l'usuari a que faci clic allà on vulguem.

El seu funcionament és molt senzill. Un cop l'usuari el descarrega, el pot instal·lar al seu dispositu. L'usuari en aquest moment no sospita res, ja que no li hem demanat cap permís. O almenys cap permís sospitós per a un joc. Un cop instal·lat, l'executa i li apareix la pantalla inicial, com qualsevol altre aplicació. A l'esquerra hi tenim la pantalla inicial, amb tres botons. Un per iniciar el joc, un per veure les instruccions i un altre per finalitzar el joc i sortir al

Home.

L'usuari, mitjançant el botó "`Menú`", pot accedir a les opcions del joc, tot i que en aquest cas, son opcions per configurar l'aplicació per als nostres propòsits, com son:

- Definir la velocitat d'aparició dels icones.
- La opacitat de la capa `Toast`. D'aquesta manera es pot observar el que passa a sota.
- El tipus de test, si és SMS o trucada. És a dir, que activitat executarem sense que l'usuari se n'adoni.

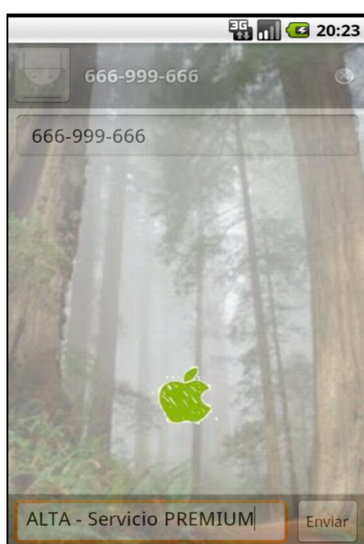


Ilustración 11: Tapjacking amb SMS

Ilustración 12: Tapjacking amb trucada

Per iniciar el joc, no cal definir les opcions abans ja que per defecte s'executa a velocitat alta, amb una opacitat mitjana i s'executa el test per SMS.

A més dels botons, a la pantalla inicial es mostra la versió d'Android sobre la que s'executa l'aplicació. Aquestes dades són només de caire informatiu per distingir visualment sobre quina plataforma s'està executant.

Un cop l'usuari inicia el joc, es carrega una nova pantalla on hi apareix una imatge de fons i sobre aquesta imatge hi van apareixen les icones de forma aleatòria. L'usuari ha de seleccionar amb el dit (el que es coneix com a esdeveniment `Touch`) l'imatge de l'androide i ha de deixar passar la poma. En l'imatge de l'esquerra es pot veure un moment de l'execució del joc. Com es pot apreciar, l'element `Toast` es mig transparent amb una imatge d'un bosc i més per sota es veu la pantalla d'enviament de missatges SMS.

Tornant al joc, hi ha un detall que cal analitzar una mica més. L'explicació de que l'usuari hagi de seleccionar només una imatge i no l'altre és degut a que per una banda volem que el joc sigui mínimament atractiu i per això li anem mostrant imatges de pomes fins que en un moment donat li mostrem l'imatge de l'androide. I per l'altre banda hi tenim la raó de més pes. Com que l'element `Toast` no captura els esdeveniments, si deixem que l'usuari vagi pitjant en tota la pantalla, ens pot fer malbé l'engany. En el cas de l'enviament de SMS no es veu tan clar, però en el cas de la trucada sí, ja que al darrere es troben tots els botons que simulen un teclat telefònic. Si anés tocant tota la pantalla modificaria el número de telèfon al que volem marcar.

A la següent figura es veu l'aplicació executant el test per les trucades. A més, en aquesta imatge també es veu el moment

en que l'icona que volem que seleccioni l'usuari es troba sobre el botó de realitzar la trucada.

El joc no finalitza quan l'usuari ha seleccionat l'androide, si no que continua realitzant l'iteració tantes vegades com hi te definit al codi font. Un cop finalitzat, torna a la pantalla inicial.

Un altre detall important es que hem definit per codi que l'androide apareixi **dues vegades**. Això es així per assegurar-nos de que l'activitat de sota hagi estat carregada. Tot i que es pot intentar controlar que tot estigui coordinat, i de fet s'intenta al codi, el sistema Android es qui decideix quan s'executent les aplicacions. Per tant, la solució triada es que hi apareixi dos cops. Si no s'encerta a la primera, encara hi ha una segona oportunitat.

A continuació es mostra una iteració del joc on hem seleccionat una opacitat total. D'aquesta manera es com s'hauria d'executar perquè l'usuari no veiés el que està succeint per sota. En aquest cas, però, triarem el test de les trucades, ja que com marcarem a l'imatge, sí que es pot detectar que alguna cosa esta passant. Efectivament, apareix a l'àrea de notificacions de que s'ha iniciat una trucada.

Ilustración 13: Tapjacking en acció

Amb aquesta prova de concepte hem aconseguit els nostres propòsits. Hem fet que l'usuari executi una aplicació que no li ha demanat cap permís especial i hem aconseguit executar dos tipus de tests diferents:

- Un enviament de missatge SMS a un número especial
- Una trucada de telèfon a un altre número especial.

Per qüestions temporals només hem implementat aquests dos tipus de test però l'aplicació es troba preparada, tal i com s'ha dissenyat, per executar d'altres tipus de tests. Nomes cal implementar un classe que hereti de la classe pare `Tapjacking` e introduir sobre quins punts volem que aparegui l'icona que ha de seleccionar l'usuari. Aquest punt correspondrà a l'acció que volem fer i per la qual no disposem de permisos.

6- Aplicació MIST

6.1- Introducció

En les anteriors apartats hem explicat en que consisteix el *tapjacking* i hem fet una prova de concepte real on es pot veure aquesta vulnerabilitat en acció. La part final del projecte consistirà en dissenyar i implementar una aplicació que eviti que una aplicació malintencionada faci servir aquesta vulnerabilitat.

En aquests darrers apartats detallarem la solució implementada entorn el punt de vista funcional i el punt de vista pràctic alhora de servir com a solució real. Finalment parlarem dels problemes amb que hem topat i de d'altres possibles vies de solució.

6.2- Diagrama de classes

L'aplicació s'ha dissenyat fent servir els diversos components que proveu la plataforma Android. En concret hem fet servir els següents elements:

- ***Activitity***: Cada pantalla de l'aplicació es una *activity* diferent. L'interfície gràfica es defineix dintre d'un fitxer .xml.
- ***Service:*** Alhora de fer servir el Widget, es fa servir un component Service com a pasarel.la de comunicació entre el Widget i l'aplicació.
- ***TabActivity:*** Aquest es un tipus d'activitat que permet gestionar les tabs o pestanyes.
- ***ListActivity***: Aquest es un tipus d'activitat per crear de forma ràpida llistat d'elements. Mitjançant un objecte de tipus *cursor* es pot gestionar els esdeveniments que es poden capturar.
- ***AppWidgetProvider:*** Aquesta es la classe que s'ha d'estendre alhora de crear un Widget.

El diagrama de classes es mostra a continuació:

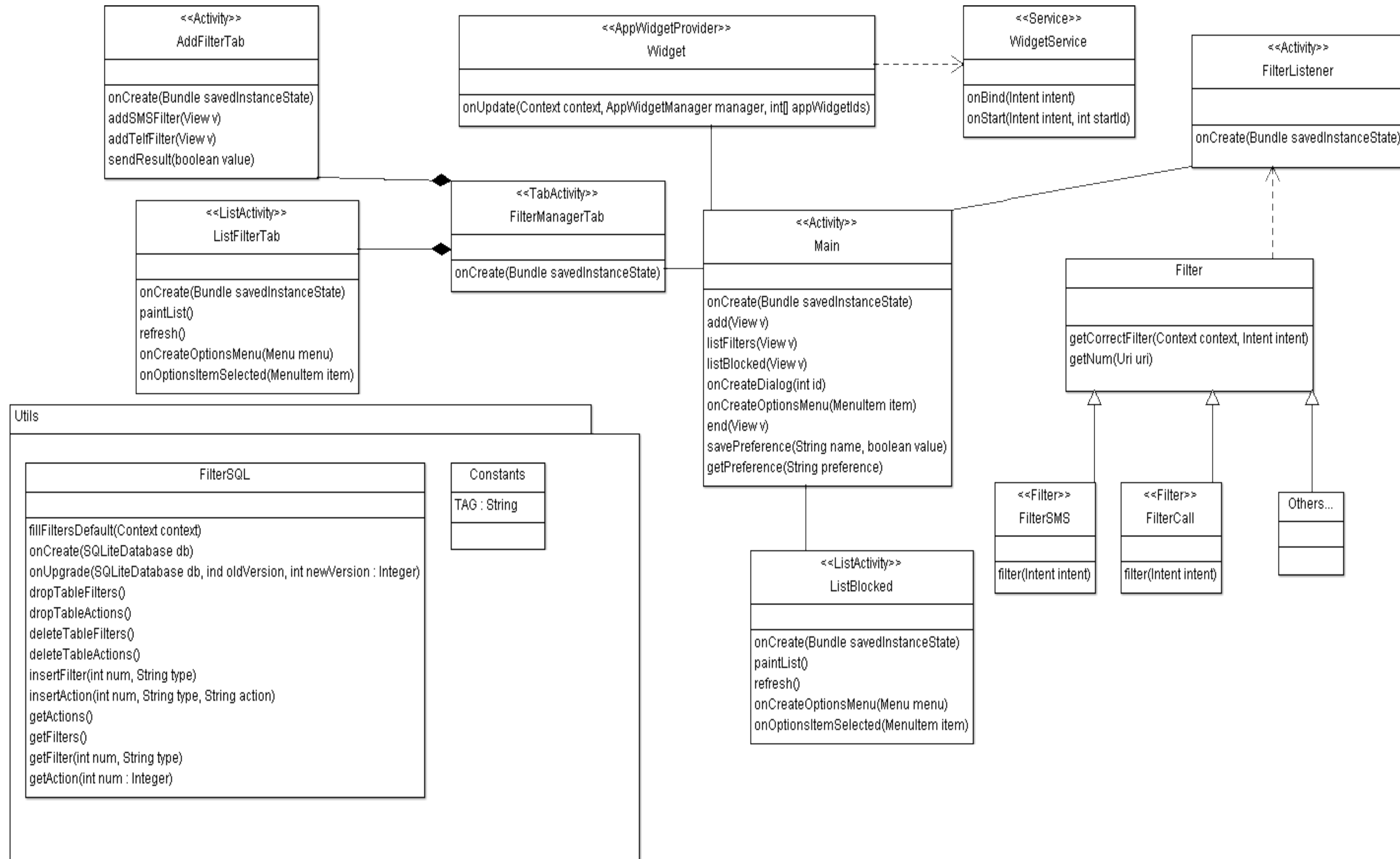


Ilustración 14: Diagrama de classes Projecte MIST

6.2.a- Classe Main

Nom	Main.java
Package	com.mist
Layout associat	main.xml

Aquesta es la classe principal d'entrada a l'aplicació. Es així donat que al fitxer *AndroidManifest.xml* es defineix com:

```
<application
  android:icon="@drawable/mist_icon"
  android:label="@string/app_name" >
  <!-- Activity principal -->
<activity
      android:label="@string/app_name"
      android:name=".Main" >
      <intent-filter >
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
  </activity>
```

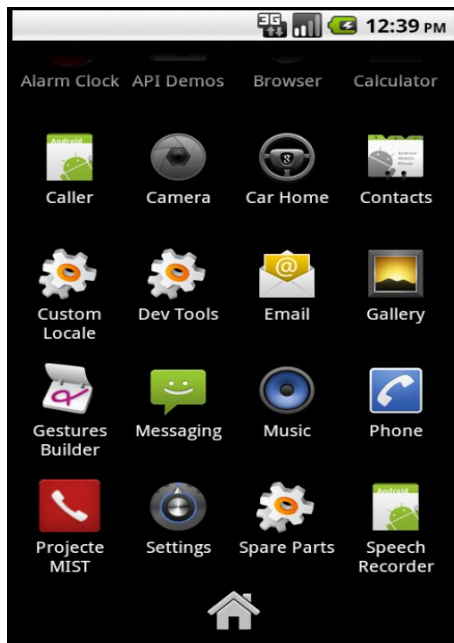


Ilustración 15: Pantalla "Home"

En un principi l'aplicació estava plantejada per no tenir activitat principal. En Android no és un requisit indispensable que una aplicació tingui punt d'entrada. Al definir dintre de l'etiqueta ***<application>*** els elements ***icon*** *i* ***label*** Android crea l'icona i l'etiqueta al *Home*, que no deixa de ser una altre activitat del propi sistema. En el *AndroidManifest.xml* podem definir les activitats, serveis, etc.. de la nostra aplicació, però només una d'aquestes activitats serà l'activitat principal i que Android executarà quan la seleccionem desde el *Home*. Per exemple, en la figura es pot veure l'aplicació dintre de l'emulador d'Android

La definició d'aquest punt d'entrada en l'aplicació és només per motius de comoditat. D'aquesta manera podem accedir a l'aplicació per configurar els filtres, veure les accions bloquejades, esborrar les dades, etc.. com ja anirem detallant.

Un cop arrenca l'aplicació, la pantalla inicial es la que es mostra en la figura següent. Desde aquesta finestra es poden realitzar les accions següents:

Ilustración 16: Pantalla inicial

- **Afegir filtres.** Permet que l'usuari pugui afegir nous filtres. Un filtre representa una acció que ha de ser bloquejada.
- **Llistat de filtres.** Veure una llista dels filtres que hi ha activats.
- **Llistat d'accions.** Veure una llista de les accions realitzades, com per exemple bloquejar una trucada a un número amb filtre
- **Sortir**. Permet sortir i tancar l'aplicació.

A més, si l'usuari fa click en el botó "*Menú*", apareix el menú que ja vam veure en l'anterior aplicació. Hi ha dues opcions dintre d'aquest menú:

- **Opcions**. Permet seleccionar quins tipus de filtres volem aplicar (funció finalment no implementada) i si volem que l'aplicació carregui a l'inici els filtres per defecte.
- **Info**. Carrega un quadre de diàleg amb informació sobre l'aplicació.

6.2.b- Classe FilterManagerTab

Nom	**FilterManager.java**
Package	**com.mist**
Layout associat	**filter_manager.xml**

Les accions de *"**Afegir filtres**" i "**Llistat de filtres**"*, donat que tenen en comú l'objecte *filtre*, s'ha optat per implementar-les fent servir pestanyes o *tabs*, donat que estèticament es més vistos. Per tant, tant seleccionant qualsevol de les dues accions, s'executa la classe ***FilterManagerTab.***

Aquesta classe hereta de la classe *TabActivity*. Es només una classe contenidora per les *tabs* o pestanyes. Cada pestanya es considera una activitat separada, amb el seu propi *Layout*. Per tant, aquesta classe contenidora únicament gestiona les pestanyes, definint les seves propietats i indicant a Android quina *Activity* ha de pintar en cada pestanya.

6.2.c- Classe AddFilterTab

Nom	AddFilterTab.java
Package	com.mist
Layout associat	add_filters.xml

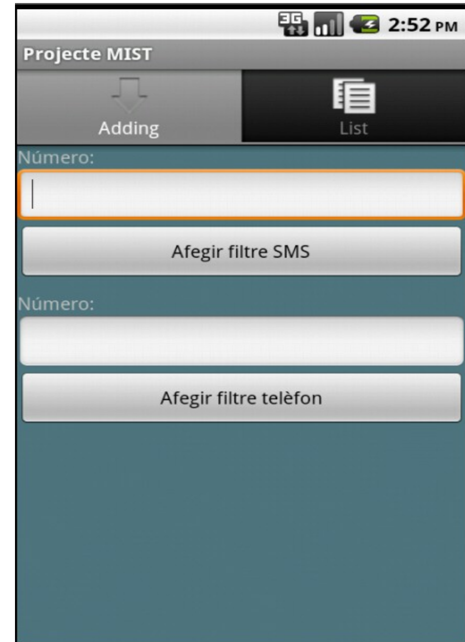


Ilustración 17: Afegir filtre

Aquesta classe hereta de la classe *Activity*. Per tant, es una activitat amb el seu propi layout associat. L'única particularitat es que s'executa dintre d'una pestanya gestionada per la classe anterior.

En aquesta activitat l'usuari pot afegir filtres de tipus numèric, és a dir, filtres com són els SMS o trucades on s'ha d'informar el número que volem filtrar, tal i com es veu a la figura següent. D'aquesta manera l'aplicació guarda aquest filtres en la base de dades. Tant si es produeix algun error al guardar el filtre com si es correcte, es mostra un element *Toast* informant d'aquest fet a l'usuari.

Guardar els filtres es degut a que quan es produeix un determinant esdeveniment que l'aplicació captura, ho compara amb cada filtre. Si hi ha coincidència, l'aplicació genera una notificació en la barra de notificacions i guarda aquest bloqueig en la base de dades. Més endavant detallarem l'emmagatzematge de les dades.

6.2.d- Classe ListFilterTab

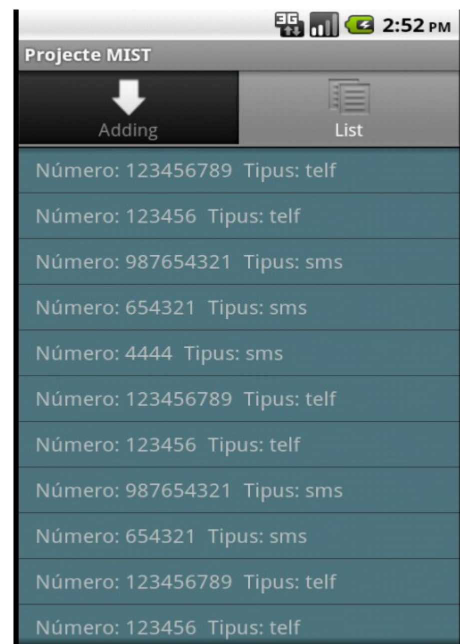


Ilustración 18: Llistat de filtres

Aquesta classe, que hereta de *ListActivity* és l'altre activitat que forma part del conjunt de pestanyes. La funció d'aquesta activitat és mostrar a l'usuari un llistat dels filtres que hi ha activats.

El motiu de que aquesta classe hereti de *ListActivity* i no d'*Activity* és que Android proveeix d'un mecanisme per facilitar precisament construir aquest tipus de llistats. Només cal heretar d'aquesta classe pare i construir un objecte de tipus *Adapter* concret. Hi ha de diversos tipus amb les seves característiques. En aquest cas he fet servir un objecte de tipus *ArrayAdapter*.

Com a paràmetres requereix del context de l'aplicació, el *Layout* on mostrarà la llista i un array amb els valors. En el *Layout* tenim el component, que en aquest cas és només un ***<TextView>***, que Android anirà omplint amb el text i repetint per cada element de l'array d'elements que

volem llistar.

Com que només hi tenim activats els filtres per trucades i SMS, l'informació es mostra en format llista. Si tinguéssim d'altres filtres, caldria estudiar la forma de mostrar aquesta informació ja que d'altres filtres no requereixen un llistat de números. Tan sols un activat/desactivat és suficient.

6.2.e- Classe ListBlocked

Nom	**ListBlocked.java**
Package	**com.mist**
Layout associat	**block_list.xml**

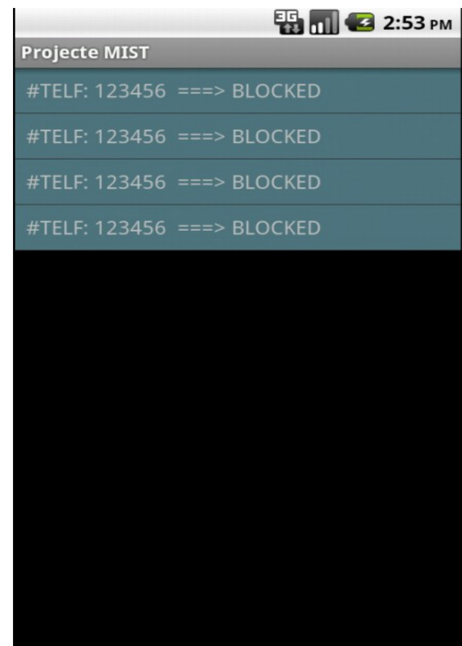


Ilustración 19: Accions

Aquesta és una classe molt semblant a l'anterior classe. També hereta de *`ListActivity`*. La seva funció és mostrar un llistat de les accions realitzades per l'aplicació.

La idea inicial era mostrar tant les accions bloquejades com les accions permeses, és a dir, les accions que ha interceptat l'aplicació però que no han estat bloquejades. Finalment aquesta darrera funció ha estat posposada per la següent versió donat els problemes que pateix l'aplicació en aquest sentit.

Una altre element de caire informatiu inclòs als requeriments de la pròxima versió és mostrar la hora (i/o data) del esdeveniment. Aquesta dada ja es guarda en la base de dades junt amb la resta de dades (com tipus i número) però no es mostra en el llistat.

6.2.f- Classe Widget

Nom	**Widget.java**
Package	**com.mist.widget**
Layout associat	**mist_widget.xml**

Aquesta classe és la classe que gestiona el *`widget`* i el seu cicle de vida. Un *`widget`* és un element gràfic ancorat al escriptori del terminal que permet mostrar informació al usuari o interactuar amb l'aplicació "propietària" del *`widget`*.

Un dels requisits de disseny en aquesta aplicació era incloure un *`widget`* que permetés a l'usuari poder activar o desactivar l'aplicació. La seva utilitat estava íntimament lligada al fet de que en un principi l'aplicació només estava ideada per ser un servei sense interfície gràfica

i per tant, sense opció a poder-lo activar/desactivar o per afegir o modificar opcions. Però com que l'aplicació finalment té interfície gràfica, la seva utilitat ha quedat mermada.

Tornant a la seva implementació, aquesta classe per gestionar el *widget* hereta de la classe *AppWidgetProvider*, que és la classe que Android ens proveeix per gestionar els *widgets*. Aquesta classe pare té, al igual que la classe *Activity*, diversos mètodes que son cridats per la plataforma Android en determinats moments del cicle de vida del *widget*, com per exemple el mètode ***onCreate(...)*** quan es crea o el mètode ***onDestroy(...)*** quan es destrueix.

Per al nostre cas només hem fet servir el mètode ***onUpdate(...)*** Aquest mètode es cridat a continuació dels mètodes de creació del *widget* i a posteriori de forma freqüent segons ho haguem definit. Com a consell l'interval no hauria de ser menor de 30 minuts, ja que no hem d'oblidar que estem parlant de terminals amb bateria.

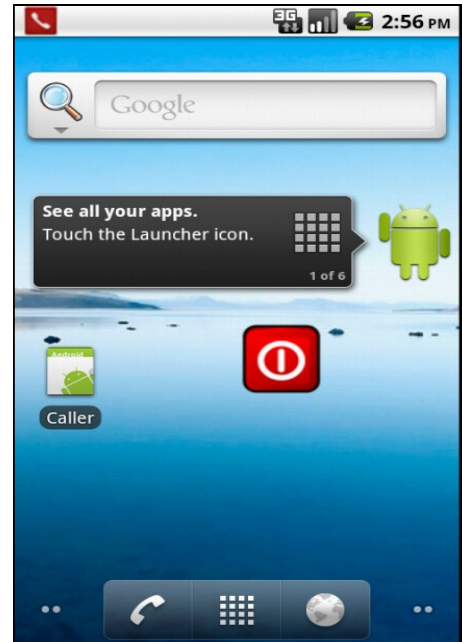


Ilustración 20: Widget a l'escriptori

Com que el *widget* en el nostre cas es un botó (el botó vermell a la figura), el que fem en aquest mètode es recuperar l'identificador del nostre *widget*, ja que cadascú té el seu propi identificador i assignar-li un *Listener* com a qualsevol altre botó i l'intent que ha de enviar quan l'usuari faci click en el *widget*.

La classe que rebrà l'intent és la classe *WidgetService*, que hereta de la classe *Service*. Com que només ha d'activar o desactivar els filtres, sense necessitat de cap interfície, aquesta era la millor opció. Com a qualsevol altre classe que hereti de *Service*, tenim el mètode ***onStart(...)*** que Android cridarà per arrancar el servei. Dintre d'aquest mètode hem implementat la lògica per activar o desactivar l'aplicació.

6.2.g- Classe Constants i classe FilterSQL

Nom	**Constants.java**
Package	**com.mist.utils**
Layout associat	-------------

Nom	**FilterSQL.java**
Package	**com.mist.databse**
Layout associat	---------------

Aquestes dues classes es troben dintre del paquet *utils* perquè són classes de suport a l'aplicació. La classe *Constants* conté totes les constants que és fan servir a l'aplicació. És una forma de tenir-les accessibles i controlades.

La classe *FilterSQL* és la classe encarregada de interactuar amb la base de dades que

incorpora Android. Aquesta classe hereta de la classe `SQLiteOpenHelper` que proveu Android per aquestes tasques. En aquesta classe tenim mètodes per afegir filtres i accions, per recuper-los, per esborrar la base de dades, etc...

La plataforma Android proveu de forma nativa el motor de base de dades `SQLite` per emmagatzematge de dades. És petita i lleugera, ideal per dispositius amb espai limitats.

Hem fet servir dues taules per l'aplicació. Una per guardar els filtres i un altre per guardar les accions generades per l'aplicació. Totes dues taules són les que proporcionen les dades per als dos llistats que hem vist anteriorment. La definició de les taules és:

- **Filter(*_id*** integer primary key autoincrement**, *num*** integer**, *type*** text**)**
- **List(*_id*** integer primary key autoincrement**, *num*** integer**, *type*** text**, *action*** text**, *time*** long**)**

La plataforma Android guarda aquesta base de dades, juntament amb qualsevol altre arxiu al sistema d'arxius del dispositiu. Normalment es trobarà en `/data/data/nom_del_paquet/databases`

En l'aplicació **MIST**, donat que el *package* definit en l'arxiu `AndroidManifest.xml` és `com.mist`, com es pot veure a la captura següent, la base de dades es trobarà en **`/data/data/com.mist/databases/filter.db`**

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.mist"
  android:versionCode="1"
  android:versionName="1.0" >
```

6.2.h- Classe FilterListener

Nom	**FilterListener.java**
Package	**com.mist**
Layout associat	**-------------**

Aquesta classe hereta de la classe `Activity` però no es una activitat normal. Aquesta classe és l'encarregada de *"escoltar" i "capturar"* els esdeveniments que volem filtrar. Per fer això, al arxiu `AndroidManifest.xml` aquesta `activity` es defineix:

```
<!-- Activity per filtrar -->
<activity
  android:label="@string/app_name"
  android:name=".FilterListener" >
  <intent-filter android:priority="10">
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:scheme="sms" />
  </intent-filter>
  <intent-filter android:priority="10">
       <action android:name="android.intent.action.VIEW" />
       <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="tel" />
  </intent-filter>
</activity>
```

Cada ***<intent - filter>*** representa l'intent que volem que la nostra aplicació capturi. Quan l'aplicació malintencionada emeti un *intent*, la plataforma Android comprovarà quines *activities* poden atendre aquest *intent* mirant aquest ***<intent - filter>***.

Com que hi haurà més d'una aplicació que poder atendre l'*intent*, ja que com a mínim, a més de la nostra aplicació hi haurà l'aplicació per defecte de la pròpia plataforma Android, el sistema li demanarà a l'usuari que seleccioni quina vol que s'executi.

Un altre dels problemes del projecte era assignar que per defecte s'executés la nostra aplicació, en comptes de l'aplicació del sistema. D'aquesta forma l'aplicació podria començar a actuar de forma transparent per l'usuari, però això no es possible ja que òbviament seria un problema de seguretat. Una aplicació malintencionada podria suplantar una aplicació legítima afegint-se com a aplicació per defecte. El mètode que feia això es troba ***deprecated.*** Concretament, el mètode es trobava a ***PackagerManger.setPreferredActivities(....)***

Quan la classe *FilterListener* capturi l'intent, el passarà a la classe *Filter*, que serà l'encarregada de comprovar si existeix un filtre per aquest *intent*. Un cop l'hagi comprovat, retornarà el tipus de filtre adequat si existeix a la classe *FilterListener*. Un cop tenim el filtre, simplement li demanem que comprovi si s'ha de bloquejar o no. Depenent del resultat, aquesta classe *FilterListener* emetrà una notificació com es pot veure a la figura. És la mateixa notificació que rebem quan rebem un missatge de text, per exemple.

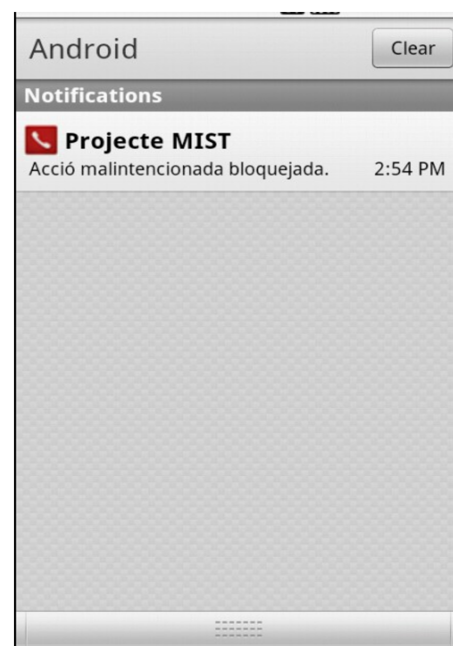


Ilustración 21: Notificacions en Android

Només cal definir un icona que apareixerà a la barra de notificacions més un text. A més, cal informar també dos textos diferents que apareixen al desplegar la barra de notificacions. Finalment, es defineix un *intent* que

s'executarà si l'usuari selecciona la notificació de la barra de notificacions. En aquest cas, hem seleccionat que l'activitat que s'executi sigui el llistat d'accions, ja que si la notificació indica que s'ha produït un bloqueig d'una acció, el més normal es que vulgui veure aquest bloqueig.

6.2.i- Classe pare Filter i descendents

Nom	**Filter.java**
Package	**com.mist.filter**
Layout associat	**-------------**

- **Classe FilterCall**

Nom	**FilterCall.java**
Package	**com.mist.filter**
Layout associat	**-------------**

- **Classe FilterSMS**

Nom	**FilterSMS.java**
Package	**com.mist.filter**
Layout associat	**-------------**

La classe *Filter* és la classe que rebrà l'*intent* a comprovar desde la classe *FilterListener*. Aquesta classe simplement analitza l'intent rebut i retorna el tipus adient de *Filter*, com seria un *FilterCall o un FilterSMS*. Per afegir nous filtres, només cal implementar una nova classe filla amb el tipus de filtre i modificar el mètode ***getCorrectFilter(...)*** perquè retorni el nou filtre segons l'*intent* rebut. L'acció de comprovar si s'ha de bloquejar l'*intent* recau en el mètode ***filter(Intent intent)*** de cada classe *Filter*.

7- Valoració final de la solució proposada

Com ja s'ha parlat al capítol 4.4, Google implementa la solució directament al codi font de la plataforma Android a partir de la versió 2.3. Tot i que possiblement hagués sigut una opció vàlida una solució semblant per a les versions anteriors, el que s'ha intentat en aquest projecte, a banda de presentar aquesta vulnerabilitat i fer una prova de concepte, és intentar implementar una solució en forma d'aplicació tenint en compte que l'instal.lació és infinitament molt mès senzilla per a l'usuari normal que no pas instal.lar una ROM modificada.

Per tant, tenint en ment aquesta idea, és va iniciar aquest projecte sense sàpiguer inicialment si la plataforma Android permetria desenvolupar una aplicació amb els requeriments desitjats. Un cop implementada, la valoració final de la solució proposada és que **no podem evitar la vulnerabilitat.** És a dir, aquesta solució és vàlida si és modifica el problema perquè s'adequï a la solució. I òbviament, aquest no és un plantejament correcte. Si hem de modificar el problema perquè la solució sigui vàlida, no es una solució completa. Es només una part de la solució.

El motiu d'aquesta afirmació ve donada perquè l'aplicació **MIST** detecti els *intents* que sabem que són potencialment perillosos mitjançant els **<intent - filter>** definits al *AndroidManifest.xml* de l'aplicació, cal que l'aplicació malintencionada emeti els *intents* de forma ímplicita. És a dir, que per exemple per activar el dial per fer una trucada, tingui el següent codi:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("tel://666999666"));
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

D'aquesta manera, com es pot veure a la figura següent, el sistema demana a l'usuari que seleccioni quina aplicació vol executar quan detecta l'*intent* anterior, donat que internament, el sistema té definits dues *activities* que poden atendre l'*intent*.

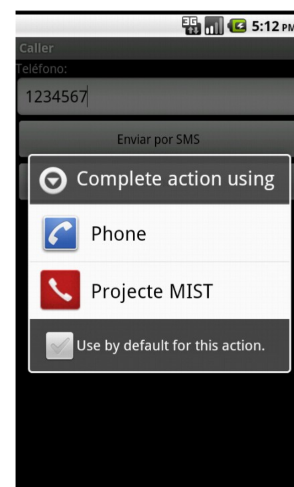


Ilustración 22: Selecció d'aplicació

Aquest tipus d'*intents* són els que s'anomenen ***intents implícits***. I aquest tipus d'*intens* podrien ser "*capturats*", és a dir, que el sistema doni l'opció d'executar la nostra aplicació o que s'executi per defecte si l'usuari ho marca. Per tant, si l'aplicació malintencionada fa servir aquests *intents*, la nostra aplicació podria actuar com si fos una mena de tallafocs, que és el que es pretenia.

Però no només és pot fer servir aquest *intent* per activar una altre *Activity*. La plataforma Android també proveu el que s'anomenen ***intents explícits***. És a dir, un tipus d'*intent* on li demanem explícitament que executi l'*activity o Service* que li passem.

Per exemple, aquest seria l'*intent* anterior per activar el dial amb la forma explícita:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("tel:"+telfNum.getText().toString()));
intent.setClassName("com.android.contacts","com.android.contacts.DialtactsActivity");
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

En aquest *intent* fem servir el mètode ***intent.setClassName(String PackageName, String ClassName)***, és a dir, li estem demanant que sigui el component amb el nom de paquet *"com.android.contacts"* i nom de classe *"com.android.contacts.DialtactsActivity"* qui rebi l'*intent*. En aquest cas el sistema Android no ha de mirar qui ho pot rebre. Directament executa aquesta *Activity*.

Per tant, l'aplicació **Mist** pot ser cridada quan els *intents* que s'emetin son de forma implícita. Si son de forma explícita, l'aplicació és ignorada donat que com hem explicat, el sistema Android activa directament l'activitat demanada i la resta d'activitats no en reben cap notificació.

Per tant, si modifiquem el problema, com hem comentat abans, per a que siguin emesos de forma implícita, l'aplicació funciona però si son emesos de forma explícita, no ho detecta. Aquesta és l'explicació de que modificar el problema per adequar-lo a la solució fa que no sigui una solució completa o vàlida.

Per tant, sembla a priori que no és possible implementar una solució al mateix nivell que el problema donat les limitacions de la plataforma. Només implementant una solució a més baix nivell (com ho ha fet Google) pot solucionar la vulnerabilitat.

Encara que fer una afirmació tan categòrica en aquest sector és arriscada. Segurament aprofundint molt més en la plataforma Android potser és podria implementar una solució vàlida i que acomplís els requeriments que volíem al inici del projecte.

8- Annex: Instal.lació de les aplicacions

8.1- Aplicacions Android (SDK d'Android)

En Android, les aplicacions es distribueixen en fitxers en format *apk*. Es un fitxer comprimit que conté tots els fitxers necessaris per executar l'aplicació tant en un emulador com en un dispositiu real. Per poder empaquetar un projecte Android en format *apk,* cal signar-lo. Si no es signa no es pot instal·lar, encara que sigui un emulador. Si es per tasques de depuració, Android permet signar les *apk* amb una clau de depuració. Si es per publicar l'aplicació al *Android Market*, cal signar-lo amb la clau de desenvolupador.

Per instal.lar les aplicacions en un emulador, haurem de seguir els següents passos:

1. Instal·lar la última versió del SDK d'Android que es pot trobar a l'adreça http://developer.android.com/sdk/index.html
2. Un cop instal·lat, entrar dintre de la carpeta d'instal·lació i executar el fitxer ***SDK Manager.exe***
3. Descarregar les *Tools->Android SDK Tools, Tools->Android SDK Platform Tools* i *Android 2.1(API 7) -> SDK Platform*.
4. Un cop instal·lat, tanquem l'aplicació i obrim el ***AVD Manager*** per crear un nou emulador.
5. Arranquem l'emulador.
6. Per instal·lar l'aplicació a l'emulador, desde el directori *tools/*, escrivim la comanda en una finestra de comandes:
   adb install <path_to_your_bin>.apk
7. Un cop finalitzat el procés, arrencarà l'aplicació.

8.2- Altres

A continuació detallarem d'altres aspectes que poden ser interesants.

8.2.a- Instal.lar *plugin ADT* en Eclipse

- Per instal.lar el plugin ADT en Eclipse cal prèviament tenir instal.lat el propi Eclipse:
  1. Descarregar la versió "*Clàssic*" recomanada desde [8]
  2. Descomprimir el fitxer *zip* descarregat en l'ubicació on vulguem.
  3. Executar el fitxer *Eclipse.exe*

- Un cop tenim Eclipse preparat, el següent pas es instal.lar el *plugin* ADT:
  1. Arrencar Eclipse i seleccionar *`Help > Install new Software...`*
  2. En el quadre de diàleg que apareix, seleccionar **ADD** i afegir les següents dades:
     1. **Name:** ADT *Plugin*
     2. **URL:** `https://dl-ssl.google.com/android/eclipse/`
  3. Fer clic en **OK**
  4. En el quadre de diàleg Available Software, seleccionar el recuadre de *`Developer Tools`* i fer clic en **Next**.
  5. En la següent finestra, apareix un resum. Fer clic en **Next**
  6. Llegir i acceptar la llicència. Fer clic en **Next**.
  7. Quan l'instal.lació hagi finalitzat, reiniciar l'Eclipse.

Amb aquestes passes, tindrem preparat l'entorn Eclipse amb el *plugin* ADT per poder desenvolupar aplicacions, arrancar l'emulador, etc..

8.2.b- Importar projecte Android en Eclipse

El codi font de les aplicacions es troba dintre de la carpeta del projecte. Dins de cada carpeta es troba el fitxer ***`.project`***. En aquest fitxer l'Eclipse inclou tota l'informació de cada projecte per poder ser importat en altres IDE's.

Per importar un projecte dins de l'Eclipse, cal fer:

1. Seleccionar dins d'Eclipse *`File->Import`*
2. A la finestra que s'obre, buscar la cadena *`Existing Projects into Workspace`* i fer clic en "*`Next`*".
3. En la següent finestra, seleccionar "*`Browse`*" i buscar la carpeta on es troba el projecte.
4. Un cop tenim el projecte, marcar-lo en la finestra de sota i fer clic en "*`Finish`*".

D'aquesta manera, importarem el projecte dins del *`workspace`* definit.

8.2.c- Base de dades *SQLite* en Android

Tot i que Google proveu del *`plugin ADT`* per Eclipse per desenvolupar aplicacions Android, aquest *`plugin`* no conté cap eina (si més no si la conté es troba força oculta) per poder visualitzar còmodament les bases de dades *`SQLite`* de les nostres aplicacions. Cal fer servir eines externes per poder accedir a una base de dades.

Com que *`SQLite`* no és un tipus de base de dades adient per aplicacions web ni per entorns empresarials, no hi ha molts clients per gestionar bases de dades *`SQLite`*.

Per aquest projecte hem fet servir l'aplicació *`SQLiteBrowser`*, que es pot descarregar desde [17]. Només cal descomprimir el fitxer *`zip`* descarregat i executar i executar el fitxer ***`SQLite`***

Database Browser 2.0 b1

Per poder accedir als fitxers *.db que contenen la base de dades cal seguir els següents passos:

1. Dintre d'Eclipse, accedir a la pestanya *DDMS*
2. Si tenim diversos dispositius activats com poden ser emuladors i/o dispositius físics, cal seleccionar el dispositiu al qual volem accedir seleccionat-lo a la finestra `Devices`. D'aquesta manera podrem accedir al seu sistema de fitxers.
3. En la finestra de la dreta, seleccionar la pestanya `File Explorer` i navegar fins la directori exclusiu que disposa cada aplicació.

   L'aplicació **MIST** disposa del directori ***`/data/data/com.mist/`***
4. Seleccionar el fitxer que volem i fer click en el botó de dalt a la dreta de la pestanya amb forma de disquet.
5. Seleccionar on volem desar el fitxer al nostre ordinador.

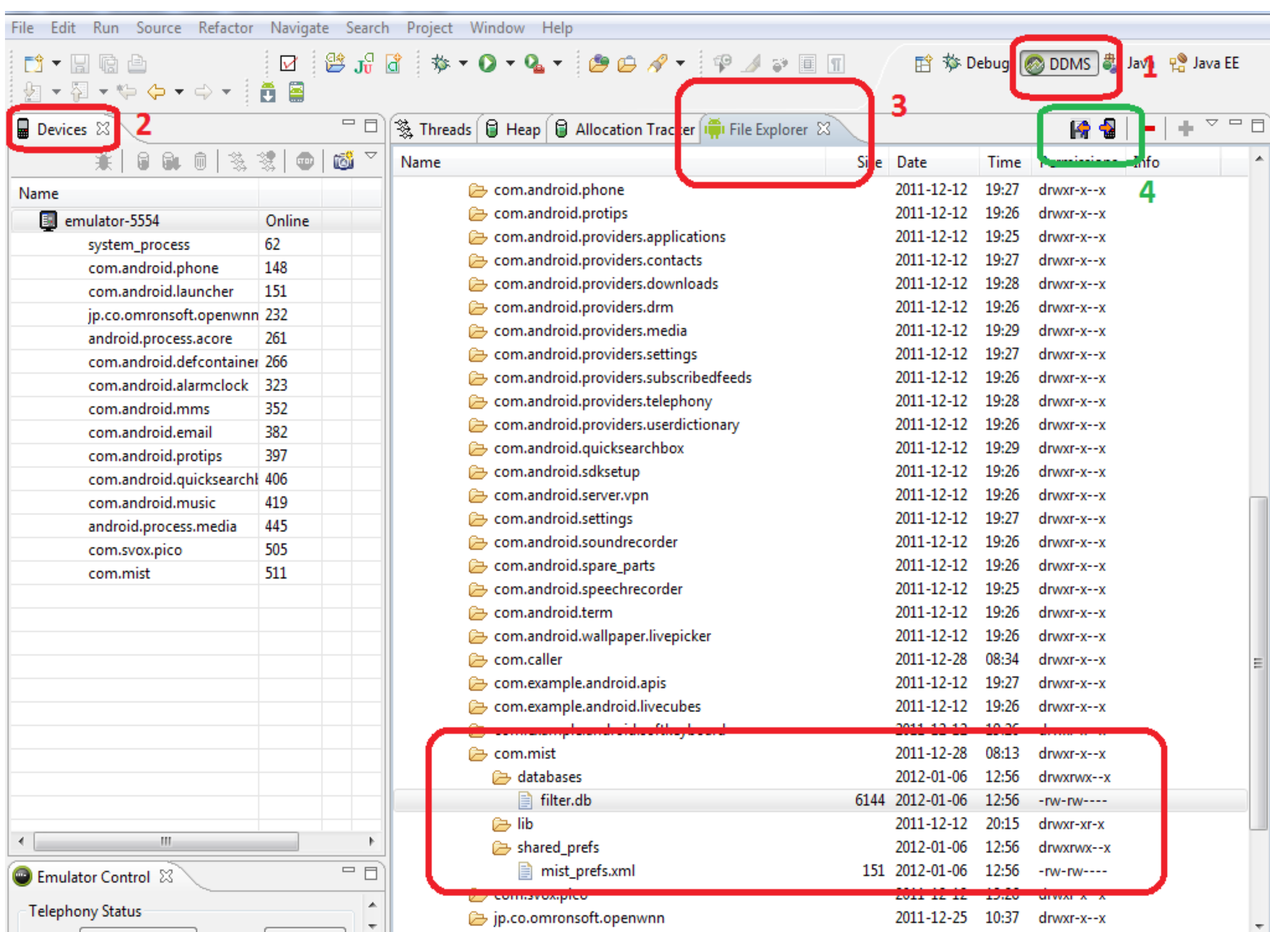


Ilustración 23: File Explorer en Eclipse

D'aquesta forma tenim accessible el fitxer per poder-lo veure, editar, etc... Aquest mètode val per a qualsevol fitxer, com per exemple el fitxer de preferències.

També podem desar un fitxer desde el nostre ordinador cap al dispositiu Android. Només cal fer les mateixes passes però en comptes de seleccionar l'icone amb forma de disquet, cal seleccionar l'icona amb forma de telèfon mòbil.

Un cop tenim el fitxer de base de dades, hem de:

- Obrim l'aplicació ***SQLiteBrowser***
- Seleccionar *File->Open Database* o **Ctrl + o**
- Seleccionar el fitxer que hem desat desde Eclipse

Podrem navegar per les diferents taules que conté la base de dades:

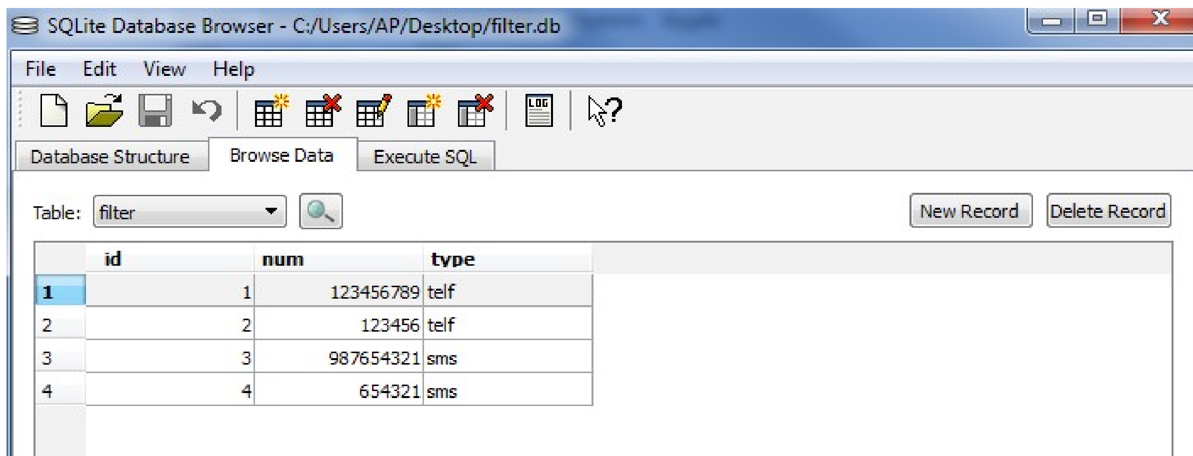


Ilustración 24: Taula Filter en SQLiteBrowser

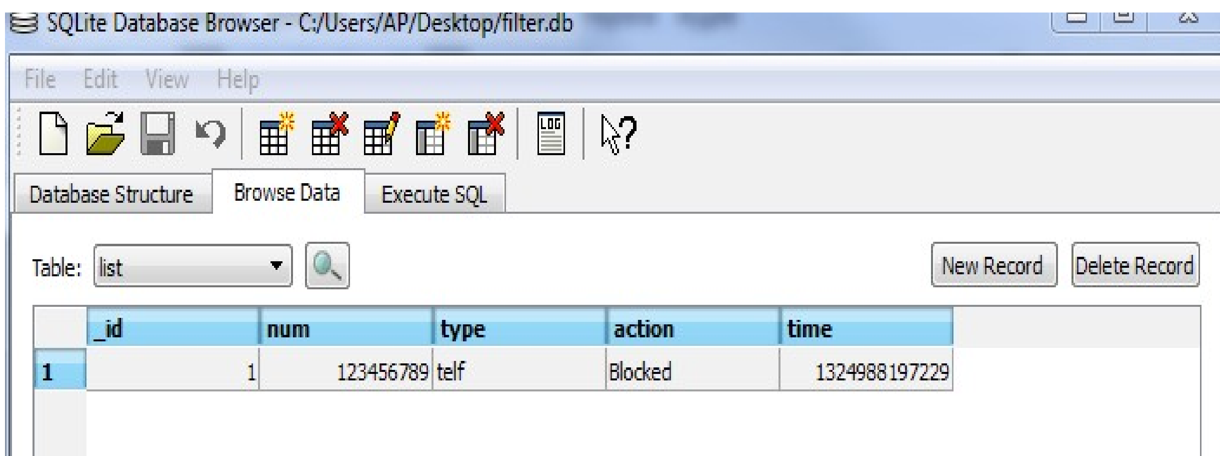


Ilustración 25: Taula List en SQLiteBrowser

9- Bibliografia

- **Martín Sierra, Antonio J.** (2006) Programador Certificado JAVA2. (Curso práctico). Madrid: Ed. Ra-Ma
- **Frank Ableson, Charlie Collins, Robi Sen.** (2009) Guía para desarrolladores Android. Madrid: Ed. Manning
- **Ed Burnette.** (2010). Programación Android. Madrid: Ed. Anaya Multimedia
- **Joan Ribas Lequerica.** (2010) Desarrollo de aplicaciones Android (Manual imprescindible) Madrid: Ed. Anaya Multimedia

10- Enllaços d'interès

- [1] - http://goo.gl/Yx3KS
- [2] - http://www.dragonjar.org/clickjacking.xhtml
- [3] - http://rooibo.wordpress.com/2008/10/05/clickjacking-a-fondo-y-con-ejemplos/
- [4] - http://goo.gl/nEbeR
- [5] - http://developer.android.com/guide/topics/security/security.html
- [6] - http://developer.android.com/guide/topics/fundamentals/services.html
- [7] - http://developer.android.com/resources/dashboard/platform-versions.html
- [8] - http://www.eclipse.org/downloads/
- [9] - http://developer.android.com/sdk/eclipse-adt.html#installing
- [10] - http://code.google.com/intl/es-ES/appengine/
- [11] - http://en.wikipedia.org/wiki/Over-the-air_programming
- [12] - http://developer.android.com/guide/basics/what-is-android.html
- [13] - http://developer.android.com/reference/android/view/View.html
- [14] - http://www.ocu.org/sms-de-tarificacieoacute-n-adicional-s476044.htm
- [15] - http://goo.gl/SJMde
- [16] - http://developer.android.com/reference/android/Manifest.permission.html
- [17] - http://sqlitebrowser.sourceforge.net/
- [18] - http://goo.gl/Hjk4T
- [19] - http://developer.android.com/guide/topics/intents/intents-filters.html
- [20] - https://market.android.com/?hl=es

11- Agraïments

El camí que vaig iniciar ja fa uns quants anys ja toca al seu final. Amb la presentació d'aquest projecte que neix amb tota la ilussió del món, i que vol ser el punt culminant de tot el treball, tot l'esforç i perquè no, de totes les alegries i satisfaccions que he rebut durant el meu trajecte dins la UOC, finalitza aquesta etapa que ha marcat un abans i un desprès en la meva vida.

Desde aquí, vull donar les gràcies a tots els consultors, professors i persones que treballen a la UOC per la seva fantàstica feina. Agraïment extensible al consultor Carles Ares que m'ha donat suport en aquest projecte.

Finalment, agraïments per a tota la meva família i molt especialment a la meva mare, que ens ha deixat sense veure acomplert el seu somni de dirigir-se al seu fill com a "enginyer". Ho hem aconseguit, mare.