# Apéndice A

# Distribución temporal

En este Anexo se muestra la división en tareas del proyecto así como el tiempo dedicado en cada una de ellas. Hay que remarcar que no son tiempos exactos ya que no se ha contabilizado la dedicación en cada una de las partes. Además, hay partes que no se han incluido debido a las modificaciones del propio proyecto durante su desarrollo y que, por tanto, no se han considerado vinculadas al proyecto final.

En primer lugar, se va a mostrar el gráfico general en la figura A.1 que está dividido en las diferentes fases del proyecto junto con el porcentaje de tiempo dedicado. En este gráfico no se muestran las subtareas realizadas en cada una de las fases ya que la intención era dar una visión global.

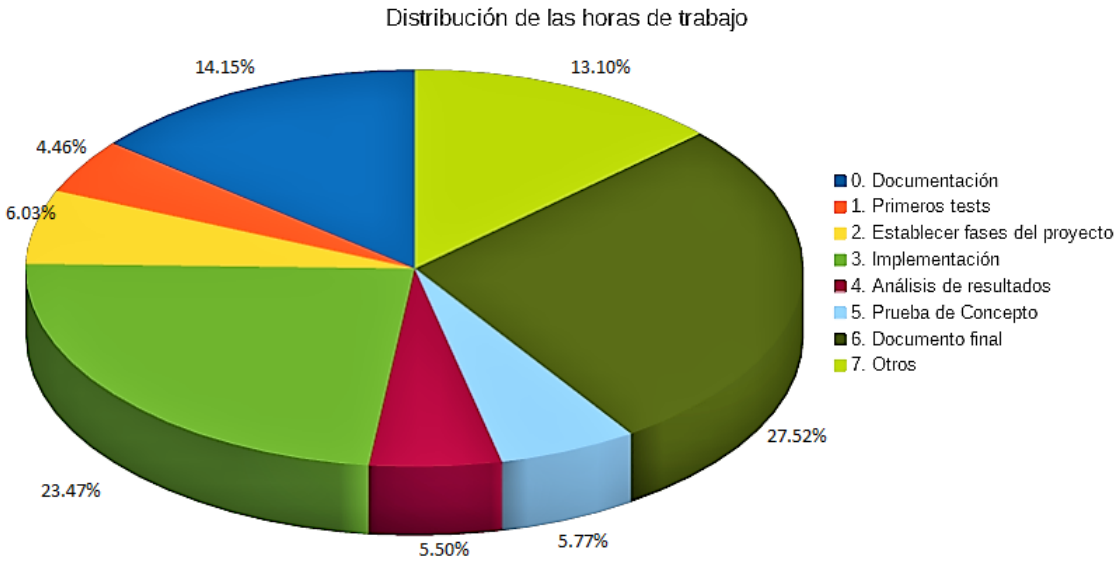


Figura A.1: Distribución de las horas de trabajo

Por otro lado, se muestran unas tablas con toda la información. La tablas con las

tareas se dividen en tres figuras. En la primera de ellas, la figura A.2, hace referencia a las primeras fases del proyecto, es decir, las fases de documentación con los proyectos considerados de interés y las primeras pruebas para familiarizar con el entorno Android ya que no se tenían conocimientos previos que alcanzasen tal profundidad. En la segunda tabla, en la figura A.3, están incluidas las tareas fundamentales del proyecto. Finalmente, en la figura A.4 se muestran las tareas finales relacionadas con la comprobación de resultados, la prueba de concepto, tiempo de redacción del presente documento y otra información de interés.

| Fase | Tiempo dedicado (horas) | Porcentaje |
|---|---|---|
| **0. Documentación** | 54 | 14,15 % |
| 0.1 Android internals: A confectioner's cookbook | 30 | |
| 0.2 AppGuard – Enforcing User Requirements on Android apps | 2 | |
| 0.3 Slicing Droids: Program Slicing for Smali Code | 2 | |
| 0.4 Pscout: Analyzing the Android Permission Specification | 4 | |
| 0.5 On the Effectiveness of API-Level Access Control Using Bytecode Rewriting in Android | 2 | |
| 0.6 Towards Black Box Testing of Android apps | 2 | |
| 0.7 I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android applications | 2 | |
| 0.8 IRM Enforcement of Java Stack Inspection | 2 | |
| 0.9 Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware | 2 | |
| 0.10 SuSi: A Tool for the Fully Automated Classification and Categorization of Android Sources and Sinks | 2 | |
| 0.11 Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces | 2 | |
| 0.12 ARTDroid: A Virtual-Method Hooking Framework on Android ART Runtime | 2 | |
| **1. Primeros tests** | 17 | 4,46 % |
| 1.1 Test Java Reflection Smali | 6 | |
| 1.2 Test APIMonitor | 4 | |
| 1.3 Test Proguard | 3 | |
| 1.4 Tests Python | 4 | |

Figura A.2: Tareas fases iniciales

| Fase | Tiempo dedicado (horas) | Porcentaje |
|---|---|---|
| **2. Establecer fases del proyecto** | 23 | 6,03 % |
| 2.1 Decisión desarrollo scripts de descarga | 3 | |
| 2.1.1 Análisis de las APIs | 3 | |
| 2.2 Decisión Desarrollo de Apkdissector | 3 | |
| 2.2.1 Multithreading | 2 | |
| 2.2.2 Diseño de las bases de datos | 1 | |
| 2.3 Decisión desarrollo del analizador dinámico de permisos | 15 | |
| 2.3.1 Análisis forzado de permisos en Android | 15 | |
| 2.4 Prueba de Concepto | 2 | |
| **3. Implementación** | 89,55 | 23,47 % |
| 3.1 Bases de datos | 3,55 | |
| 3.1.1 Base de datos Pscout | 3 | |
| 3.1.2 Base de datos hashes | 0,55 | |
| 3.1.2.1 Integración base de datos Pscout | 0,5 | |
| 3.1.2.2 Añadir tabla para análisis | 0,05 | |
| 3.2 Implementación scripts de descarga | 20 | |
| 3.2.1 Implementación script Koodous | 16 | |
| 3.2.2 Implementación script Andrototal | 2 | |
| 3.2.3 Implementación script Virustotal | 2 | |
| 3.3 Implementación de Apkdissector | 54 | |
| 3.3.1 Androguard | 15 | |
| 3.3.2 Multithreading | 20 | |
| 3.3.3 Acceso a bases de datos | 10 | |
| 3.3.4 Resultados analizador en ficheros JSON | 6 | |
| 3.3.5 Resultados analizador en base de datos | 3 | |
| 3.4 Implementación analizador dinámico | 12 | |
| 3.4.1 Modificación del AOSP | 1 | |
| 3.4.2 Búsqueda kernel con SELinux permissive | 5 | |
| 3.4.3 Ejecución CTS | 6 | |

Figura A.3: Tareas del núcleo del proyecto

| Fase | Tiempo dedicado (horas) | Porcentaje |
|---|---|---|
| **4. Análisis de resultados** | 21 | 5,50 % |
| 4.1 Comprobación de resultados de Pscout | 2 | |
| 4.2 Comprobación de resultados tras CTS | 4 | |
| 4.3 Implementación script de comparación de resultados entre Pscout y el analizador dinámico | 15 | |
| **5. Prueba de Concepto** | 22 | 5,77 % |
| 5.1 Test de prueba con Frida | 12 | |
| 5.2 Implementación aplicación Android vulnerable | 4 | |
| 5.3 Implementación ficheros de hook para Frida | 6 | |
| **6. Documento final** | 105 | 27,52 % |
| 6.1 Escritura memoria final | 105 | |
| 6.1.1 Documento | 100 | |
| 6.1.2 Gestión de tiempo | 5 | |
| **7. Otros** | 50 | 13,10 % |
| 7.1 Intentos fallidos | 23 | |
| 7.1.1 Análisis limitaciones Pscout | 3 | |
| 7.1.2 Intento de integración de Slicing Droids | 10 | |
| 7.1.3 Intento de uso de ARTDroid | 3,5 | |
| 7.1.4 Intento de uso de ARTHook | 6,5 | |
| 7.2 Descarga de aplicaciones | 20 | |
| 7.2.1 Problemas con claves de la API de Koodous | 20 | |
| 7.3 Problemas con CTS | 7 | |
| **Tiempo total:** | **381,55** | **100,00 %** |

Figura A.4: Tareas finales y resultados

# Apéndice B

# Diccionario de términos

En este Anexo se muestra una explicación de cada uno de los conceptos empleados en este documento. Se recomienda la lectura del proyecto apoyándose en este apéndice para facilitar su comprensión.

Con el objetivo de dividir los conceptos que componen el documento, se presentan separados en dos secciones: términos de Android y términos adicionales.

## B.1. Términos de Android

- **Activity:** Sección de la funcionalidad de una aplicación Android para que el usuario realice una tarea concreta. Debe definirse en el fichero AndroidManifest.
- **Android Application Package (APK):** Formato de las aplicaciones de Android.
- **Android Debug Bridge (ADB):** Herramienta disponible para acceder a un dispositivo Android.
- **Android Interface Definition Language (AIDL):** Interfaz para definir el esquema de comunicación entre procesos.
- **Android Open Source Project (AOSP):** Proyecto *open source* de Android desarrollado por Google.
- **Android Virtual Device (AVD):** Emulación de un dispositivo Android real.
- **Content Provider:** Gestor de acceso a un conjunto de datos estructurado. Se deben definir antes de su uso en el fichero AndroidManifest.
- **Compatibility Test Suite (CTS):** Test ofrecido por Google para los fabricantes con el fin de cumplir la compatibilidad con el Android Open Source Project.
- **Dalvik Executable (DEX):** Ficheros compilados de una aplicación Android.

- **Google Summer of Code (GSOC):** Concurso de Google en el que estudiantes desarrollan un proyecto con el apoyo de un mentor de Google.
- **Service:** Opción que ofrece Android para la realización de tareas largas sin interacción del usuario. Debe definirse previamente en el fichero AndroidManifest.
- **Software Development Kit (SDK):** Conjunto de herramientas de software.

## B.2. Términos adicionales

- **Application Programming Interface (API):** Definición de herramientas y funciones para desarrollar una aplicación.
- **Call Flow Graph (CFG):** Grafo útil para gestionar el proceso de invocaciones seguidas por una determinada función.
- **Comma-Separated Values (CSV):** Formato de ficheros de texto compuestos por diferentes campos separados por comas.
- **eXtensible Markup Language (XML):** Formato de fichero de texto con una estructura definida mediante *tags*.
- **First In First Out (FIFO):** Tipo de cola en el que el primer elemento en añadirse es el primero en salir.
- **Graphical User Interface (GUI):** Interfaz gráfica de cualquier aplicación para ser más amigable para el usuario.
- **Hypertext Transfer Protocol Secure (HTTPS):** Protocolo de seguridad basado en certificados para dotar de seguridad a la navegación web.
- **Inline Reference Monitor (IRM):** Modificación de un software o aplicación con el fin de aplicar unas políticas de seguridad.
- **International Mobile Station Equipment Identity (IMEI):** Identificador de un dispositivo móvil.
- **Inter-Process Communication (IPC):** Función básica de un sistema operativo para comunicación de procesos.
- **JavaScript Object Notation (JSON):** Formato de fichero de texto con una estructura definida fácilmente analizable. Similar al formato XML.
- **Ley Orgánica de Protección de Datos (LOPD):** Ley española de protección de datos de carácter personal.
- **Proof of Concept (PoC):** Exposición de los resultados obtenidos para un proyecto.

- **Remote Procedure Call (RPC):** Software para ejecución de funciones de código localizadas en otra máquina.
- **Short Message Service (SMS):** Servicio de envío de mensajes de texto.
- **Subscriber Identity Module (SIM):** Tarjeta empleada para identificar a los usuarios de una red telefónica.
- **Uniform Resource Identifier (URI):** Identificador para localizar un cierto recurso.
- **Uniform Resource Locator (URL):** Identificador de un recurso web.

# Apéndice C

# Androguard

En este Anexo se muestra un ejemplo de análisis de una aplicación empleando androlyzer.py que permite observar aspectos básicos como pueden ser los permisos o las *activities*. A bajo nivel, Androguard lo que hace es decodificar la aplicación, leer el fichero Manifest, con formato XML, y filtrar la información requerida. Lo bueno de esta herramienta es que ofrece una línea de comandos para hacer las diferentes consultas.

```
In [1]: a,d,dx = AnalyzeAPK("ffdb94d60214020248b8
0e3d464af1bab6e4a0413d7e55cb405cd0d9a97ed43a.apk",
decompiler="dad")

In [2]: a.get_permissions()
Out[2]:
['android.permission.WAKE_LOCK',
 'android.permission.INTERNET',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.WRITE_SETTINGS']

In [3]: a.get_activities()
Out[3]:
['com.ta3alam.ibriya.fiasra3wa9t.Hebrew.Audio95212',
 'com.ta3alam.ibriya.fiasra3wa9t.Hebrew.Dashboard_000',
 'com.google.android.gms.ads.AdActivity',
 'com.google.android.gms.ads.purchase.InAppPurchaseActivity']
```

# Apéndice D

# Análisis de AppGuard

En el presente Anexo se va a realizar un análisis de una aplicación simple que requiere el IMEI del dispositivo. Para comprobar el proyecto de AppGuard se plantean dos modos de invocación: normal y mediante Java Reflection [Ora]. Para ilustrarlo de forma más clara se muestra una invocación Java y, después, se analiza el código Smali vinculado a dicho fragmento de código una vez compilado.

En el siguiente fragmento de código se observa el tratamiento de una llamada al método *getDeviceId()* de la clase *TelephonyManager* que require tener el permiso *READ_PHONE_STATE* declarado en el AndroidManifest.

```
1 TelephonyManager tm = (TelephonyManager) getSystemService
2     (this.TELEPHONY_SERVICE);
3 imei = tm.getDeviceId();
```

Invocación al método *getDeviceId()*.

A continuación, en el fragmento de código inferior se observa como aparece la anterior llamada en el código smali. La información más importante se encuentra en la primera instrucción, encargada de realizar la invocación. En ella, se muestra que el objeto *TelephonyManager*, creado anteriormente, llama a su método *getDeviceId()*, seguido del tipo de objeto que devuelve y donde lo va a almacenar, es decir, devuelve un String y lo almacena en v1. Las siguiente instrucciones se encargan de establecer el valor obtenido por la operación realizada en la Activity que contiene la llamada, se mueve el registro v1 a v2 (línea 3) y se almacena lo que contiene dicho registro en la variable , de tipo String, en *MainActivity* (línea 4).

```
1 invoke-virtual {v1},Landroid/telephony/TelephonyManager;
2     ->getDeviceId()Ljava/lang/String;
3 move-result-object v2
4 iput-object v2, p0,Lcom/example/sergio/zeroapp/MainActivity;
5     ->:Ljava/lang/String;
```

Invocación al método *getDeviceId()* en lenguaje Smali.

Java ofrece la posibilidad de realizar las llamadas a métodos de una determinada clase de forma genérica. Esto se conoce como *Java Reflection* y, por medio de la clase Method, es posible, partiendo de una objeto determinado, crear un objeto *Method* capaz de obtener a través de un String el método de interés. Para invocarlo se emplea el método *invoke()* lo que dará un resultado igual que si se hace siguiendo la forma común. En el siguiente fragmento de código se observa un ejemplo.

```
1  TelephonyManager tm = (TelephonyManager) getSystemService(
2  this.TELEPHONY_SERVICE);
3  try {
4      Method m = TelephonyManager.class.
5          getMethod("getDeviceId", null);
6      try {
7           imei = (String) m.invoke(tm, null);
8      }
9      [..]
10 }
11 [..]
```

Invocación al método *getDeviceId()* con Java Reflection.

Este procedimiento tiene implicaciones en el bajo nivel ya que, en vez de aparecer en el código smali la invocación a *getDeviceId()*, aparecerá el nombre del método en un registro que se pasará en la invocación lo que supone un problema para los analizadores estáticos. Para ilustrar un ejemplo de sus implicaciones, se va a analizar su comportamiento en las aplicaciones mostradas anteriormente tras ser modificadas por APIMonitor, herramienta pública que sigue el esquema planteado por Appguard.

## D.1. Implicaciones para AppGuard

Para ilustrar el tratamiento que hace el *rewriter*, lo mejor es observar directamente el código Smali. En un primer lugar, se va a analizar la aplicación con una invocación sin uso de *Reflection* al método *getDeviceId()*. Si se observa el código del fragmento de abajo aparece la invocación a dicho método tras la modificación del código que aplica APIMonitor: en vez de partir del objeto TelephonyManager que ofrece Android se emplea el modificado por ellos, indicado por emplear el paquete Ldroidbox en la línea 9.

```
1  invoke-virtual {p0, v2}, Lcom/example/sergio/zeroapp/
2      MainActivity;->getSystemService(Ljava/lang/String;
3      )Ljava/lang/Object;
4
5  move-result-object v1
6
7  check-cast v1, Landroid/telephony/TelephonyManager;
```

```
8
9  invoke-static {v1}, Ldroidbox/android/telephony/
10     TelephonyManager;->getDeviceId(Landroid/telephony
11     /TelephonyManager;)Ljava/lang/String;
```

Invocación al método *getDeviceId()* con APIMonitor en código Smali.

Con ello, se concluye que efectivamente interfiere en la llamada original y la modifica. Pero... ¿qué pasaría si se invoca a través de *Reflection*? En la siguiente sección de código se observa el proceso de invocación a *getDeviceId()* con *Reflection*. En la línea 12 se observa el registro en el que se carga el nombre del método en forma de String. Ese registro se emplea más tarde en la línea 16 que es la encargada de crear el objeto *Method*. Ese objeto invoca al método establecido en el registro anterior, visible en la línea 29, que obtendría el IMEI del dispositivo y, por último, en la línea 37, se introduciría el valor en la variable llamada *imei*.

Como se observa en la línea 29, teniendo en cuenta que se ha empleado APIMonitor para asegurar la llamada a *getDeviceId()*, no aparece la llamada a través de DroidBox al método modificado. Esto se debe a que no considera Java Reflection y ha conseguido evitar la invocación segura.

```
1  invoke-virtual {p0, v4}, Lcom/example/sergio/zeroapp/
2      MainActivity;->getSystemService(Ljava/lang/String;
3      )Ljava/lang/Object;
4
5  move-result-object v3
6
7  check-cast v3, Landroid/telephony/TelephonyManager;
8
9  :try_start_0
10 const-class v4, Landroid/telephony/TelephonyManager;
11
12 const-string v5, "getDeviceId"
13
14 const/4 v6, 0x0
15
16 invoke-virtual {v4, v5, v6}, Ljava/lang/Class;->getMethod
17     (Ljava/lang/String;[Ljava/lang/Class;
18     )Ljava/lang/reflect/Method;
19
20 :try_end_0
21 .catch Ljava/lang/NoSuchMethodException;
22 {:try_start_0 .. :try_end_0} :catch_1
23
24 move-result-object v2
```

```
25
26  const/4 v4, 0x0
27
28  :try_start_1
29  invoke-virtual {v2, v3, v4}, Ljava/lang/reflect/Method;
30      ->invoke(Ljava/lang/Object;[Ljava/lang/Object;
31      )Ljava/lang/Object;
32
33  move-result-object v4
34
35  check-cast v4, Ljava/lang/String;
36
37  iput-object v4, p0, Lcom/example/sergio/zeroapp/
38      MainActivity;->:Ljava/lang/String;
```

Invocación al método *getDeviceId()* con Java Reflection y APIMonitor en código Smali.

¿Qué implicaciones tienen estos resultados en el análisis estático de aplicaciones Android? Teniendo en cuenta que es posible saltarse las políticas de seguridad haciendo que no sean relevantes, las conclusiones son claras y negativas. Para evitar este tipo de invocaciones se emplea los CFG, comentados en el capítulo 3.2.

Los CFG permiten analizar el flujo de uso de un determinado registro dentro del código smali por lo que, en caso de encontrar Java Reflection, se podría controlar el registro que contiene el nombre del método que se va a invocar y evitar comportamientos de este tipo dentro de los analizadores estáticos.

# Apéndice E

# Consultas a la base de datos

A continuación se van a mostrar una serie de consultas a la base de datos creada y poblada con los *scripts* mencionados anteriormente.

## E.1. Consulta tabla hashes

En esta consulta se quiere mostrar la estructura de los elementos de la tabla *hashes*. Observando la consulta, se comprueba la versatilidad que ofrece la distinción mediante tipos a la hora de conocer de que base de datos proviene el *hash*, mostrándose junto con un valor que indica si está descargada o no.

```
sqlite> select * from hashes where type = 'andrototal' limit 5;
00002d74a9faa53f5199c910b652ef09d3a7f6bd42b693755a233635c3ffb0f4
|1|andrototal|0
0000350c0792f61ee513f40bd9a42d09144cc6a3c4f2171f812ef415a9a51640
|1|andrototal|0
00017d87a5a7a738061e3e40cf29cc9ac68d114451566d88ad8051015fa9f671
|1|andrototal|0
000198ad556a1f50cc285c76ae8f8d1b2472808f3ccaa0b429a60984912ce88f
|1|andrototal|0
0001bbdf3489bbaa27df4af9d52b77028a1b7315b528a1e6bce8b63b0d3384a8
|1|andrototal|0
```

## E.2. Consulta tabla analyzed

Para conocer las aplicaciones que han sido analizadas, es necesario consultar la tabla *analyzed*. En ella, se muestran los permisos que una determinada aplicación contiene en su *Manifest*. De esta forma, se pueden comprobar facilmente los permisos que tiene una determinada aplicación y, posteriormente, realizar estadísticas sobre esos permisos.

```
sqlite> select * from analyzed where hash = '0e06661024b010880f
099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880' LIMIT 5;
```

```
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
com.android.launcher.permission.INSTALL_SHORTCUT
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.permission.GET_TASKS
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.intent.category.HOME
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.permission.RESTART_PACKAGES
0e06661024b010880f099f2f6b4a2bdaddd2358b47977a882ebda66851d3e880|
android.permission.REORDER_TASKS
```

## E.3. Consulta tabla pscout

La tabla *pscout* contiene la información que ofrece el proyecto Pscout. Para poblarla se ha elaborado un *parser* de sus ficheros y, a continuación, se ha integrado en la base de datos construida. La estructura es la misma que contiene los ficheros de este proyecto en los que se observa un mapeo de los permisos.

```
sqlite> select * from pscout where permission =
'android.permission.INTERNET' limit 5;
31808|com/android/server/ConnectivityService|reportInetCondition|
(II)V|android.permission.INTERNET|5.1.1

31809|com/android/server/NsdService|getMessenger|
()Landroid/os/Messenger;|android.permission.INTERNET|5.1.1

31810|com/android/server/ConnectivityService|reportBadNetwork|
(Landroid/net/Network;)V|android.permission.INTERNET|5.1.1

31811|com/android/browser/BrowserWebView|<init>|
(Landroid/content/Context;)V|android.permission.INTERNET|5.1.1

31812|com/android/browser/BrowserWebView|<init>|
(Landroid/content/Context;Landroid/util/AttributeSet;
ILjava/util/Map;Z)V|android.permission.INTERNET|5.1.1
```

# Apéndice F

# Estadísticas de permisos

| Permiso | Porcentaje | Cuenta |
|---|---|---|
| android.permission.INTERNET | 97,43 % | 2011 |
| android.permission.ACCESS_NETWORK_STATE | 92,97 % | 1919 |
| android.permission.WRITE_EXTERNAL_STORAGE | 82,03 % | 1693 |
| android.permission.READ_PHONE_STATE | 71,75 % | 1481 |
| android.permission.ACCESS_WIFI_STATE | 64,53 % | 1332 |
| android.permission.WAKE_LOCK | 57,95 % | 1196 |
| android.permission.ACCESS_COARSE_LOCATION | 41,96 % | 866 |
| android.permission.RECEIVE_BOOT_COMPLETED | 41,09 % | 848 |
| android.permission.GET_TASKS | 40,41 % | 834 |
| android.permission.VIBRATE | 39,92 % | 824 |
| android.permission.ACCESS_FINE_LOCATION | 37,69 % | 778 |
| android.permission.SYSTEM_ALERT_WINDOW | 33,67 % | 695 |
| com.android.launcher.permission.INSTALL_SHORTCUT | 31,59 % | 652 |
| android.permission.CHANGE_WIFI_STATE | 27,13 % | 560 |
| android.permission.SEND_SMS | 25,15 % | 519 |
| android.permission.WRITE_SETTINGS | 23,35 % | 482 |
| android.permission.MOUNT_UNMOUNT_FILESYSTEMS | 22,92 % | 473 |
| android.permission.GET_ACCOUNTS | 22,72 % | 469 |
| android.permission.RECEIVE_SMS | 22,34 % | 461 |
| com.google.andrroid.c2dm.permission.RECEIVE | 20,40 % | 421 |
| android.permission.CHANGE_NETWORK_STATE | 19,86 % | 410 |
| android.permission.READ_EXTERNAL_STORAGE | 19,67 % | 406 |
| android.permission.READ_SMS | 19,38 % | 400 |
| android.permission.CALL_PHONE | 15,75 % | 325 |
| android.permission.READ_CONTACTS | 14,63 % | 302 |
| android.permission.CAMERA | 14,24 % | 294 |
| android.permission.READ_LOGS | 14,10 % | 291 |
| android.permission.WRITE_SMS | 13,91 % | 287 |
| android.permission.RESTART_PACKAGES | 13,66 % | 282 |



Figura F.1: Tabla de estadísticas de permisos

# Apéndice G

# Código de la prueba de concepto

## G.1. Código Android

### G.1.1. MainActivity

```
1  package com.example.t4k3d0wn.exampleapp;
2
3  import android.bluetooth.BluetoothAdapter;
4  import android.bluetooth.BluetoothManager;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.content.pm.PackageManager;
8  import android.support.v7.app.AppCompatActivity;
9  import android.os.Bundle;
10 import android.telephony.TelephonyManager;
11 import android.util.Log;
12 import android.view.View;
13 import android.widget.Button;
14 import android.widget.EditText;
15 import android.widget.TextView;
16 import android.widget.Toast;
17
18 import org.w3c.dom.Text;
19
20 public class MainActivity extends AppCompatActivity {
21
22     private TelephonyManager tm;
23     private BluetoothAdapter ba;
24
25     private Button imeiButton, checkButton, webViewButton,
   ↪  bluetoothButton;
```

```
26      private TextView textIMEI;
27      private EditText editPIN;
28
29      private String imei;
30
31
32      @Override
33      protected void onCreate(Bundle savedInstanceState) {
34          super.onCreate(savedInstanceState);
35          setContentView(R.layout.activity_main);
36
37          textIMEI = (TextView) findViewById(R.id.textView);
38          checkButton = (Button) findViewById(R.id.sendPIN);
39          editPIN = (EditText) findViewById(R.id.editText);
40
41          //Get IMEI button and hide until unlock with PIN
42          imeiButton = (Button) findViewById(R.id.button);
43          imeiButton.setVisibility(View.INVISIBLE);
44
45          webViewButton = (Button) findViewById(R.id.webviewButton);
46          webViewButton.setVisibility(View.INVISIBLE);
47
48          bluetoothButton = (Button) findViewById(R.id.bluetoothButton);
49          bluetoothButton.setVisibility(View.INVISIBLE);
50          bluetoothButton.setOnClickListener(new View.OnClickListener()
    ↪ {
51              public void onClick(View view) {
52                  modifyBluetooth();
53              }
54          });
55
56          tm = (TelephonyManager)
    ↪ getSystemService(Context.TELEPHONY_SERVICE);
57          ba = BluetoothAdapter.getDefaultAdapter();
58      }
59
60      public void changeActivity(View view){
61          Intent intent = new Intent(getApplicationContext(),
    ↪ WebViewActivity.class);
62          startActivity(intent);
63      }
64
65      public void getIMEI(View view){
```

```
66          imei = tm.getDeviceId();
67          Log.e("IMEI",imei);
68          if(!imei.contains("Este")){
69              imei = imei.substring(0,3) + "***************";
70          }
71
72          Toast.makeText(view.getContext(),imei,
    ↪ Toast.LENGTH_LONG).show();
73          updateTV(imei);
74      }
75
76      public void updateTV(String s){
77          textIMEI.setText(s);
78      }
79
80      public void modifyBluetooth(){
81
82          if(ba.isEnabled()){
83              ba.disable();
84          }
85          else{
86              ba.enable();
87          }
88      }
89
90      public void checkPIN(View view){
91          String text = editPIN.getText().toString();
92          if(!text.equals("")){
93              if(text.equals("1234")){
94                  Toast.makeText(view.getContext(), "PIN OK",
    ↪ Toast.LENGTH_LONG).show();
95                  hidePinVerification();
96              }
97              else{
98                  Toast.makeText(view.getContext(), "WRONG PIN",
    ↪ Toast.LENGTH_LONG).show();
99              }
100         }
101     }
102
103     private void hidePinVerification(){
104         editPIN.setVisibility(View.INVISIBLE);
105         checkButton.setVisibility(View.INVISIBLE);
```

```
106             imeiButton.setVisibility(View.VISIBLE);
107             webViewButton.setVisibility(View.VISIBLE);
108             bluetoothButton.setVisibility(View.VISIBLE);
109         }
110
111     }
```

### G.1.2. WebViewActivity

```
1   package com.example.t4k3d0wn.exampleapp;
2
3   import android.os.Bundle;
4   import android.support.v7.app.AppCompatActivity;
5   import android.support.v7.widget.Toolbar;
6   import android.view.View;
7   import android.webkit.WebView;
8   import android.widget.Button;
9   import android.widget.EditText;
10
11  public class WebViewActivity extends AppCompatActivity {
12
13      private String url = "";
14      private WebView w;
15      private EditText newurl;
16      private Button refreshButton;
17
18      @Override
19      protected void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21          setContentView(R.layout.webview);
22
23          w = (WebView) findViewById(R.id.webviewId);
24          w.getSettings().setJavaScriptEnabled(true);
25
26          newurl = (EditText) findViewById(R.id.newUrl);
27          refreshButton = (Button) findViewById(R.id.
28              refreshButton);
29      }
30
31      public void refreshWebView(View view){
32          String text = newurl.getText().toString();
33          if(text.contains("http://")){
34              url = text;
35          }
```

```
36          else{
37              url = "http://" + text;
38          }
39          updateURL(url);
40      }
41
42      private void updateURL(String url){
43          w.loadUrl(url);
44      }
45  }
```

## G.2. Código Python

```
1   import frida
2   import sys
3
4   package_name = "com.example.t4k3d0wn.exampleapp"
5
6
7   def get_messages_from_js(message, data):
8       print(message)
9       print (message['payload'])
10
11
12  def instrument_load_url():
13
14      hook_code = """
15      Java.perform(function () {
16          // Function to hook is defined here
17          var BA = Java.use('android/bluetooth/BluetoothAdapter');
18
19          // Whenever button is clicked
20          BA.isEnabled.implementation = function (v) {
21              send("hook - isEnabled");
22
23              return true;
24          };
25
26          var TM = Java.use('android/telephony/TelephonyManager');
27          TM.getDeviceId.overload("int").implementation = function(v) {
28              send("hook - getDeviceId");
29
30              return "Este no es tu IMEI";
```

```
31          };
32
33          var MainActivity =
    ↪   Java.use('com.example.t4k3d0wn.exampleapp.MainActivity');
34
35          // Whenever button is clicked
36          MainActivity.checkPIN.implementation = function (v) {
37              // Show a message to know that the function got called
38              send('Call - CheckPIN');
39
40              this.hidePinVerification();
41
42          };
43
44          var WebViewActivity =
    ↪   Java.use('com.example.t4k3d0wn.exampleapp.WebViewActivity');
45
46          // Whenever button is clicked
47          WebViewActivity.refreshWebView.implementation = function (v) {
48
49              // Show a message to know that the function got called
50              send('Call - refreshWebView');
51
52              this.updateURL("http://192.168.1.39");
53          };
54      });
55      """
56      return hook_code
57
58  process = frida.get_usb_device().attach(package_name)
59  script = process.create_script(instrument_load_url())
60  script.on('message',get_messages_from_js)
61  script.load()
62  sys.stdin.read()
```