**Updating Android Permissions**

As policy is updated by the Internal Review Board (IRB) the policy document will be analysed with natural language processing tools to create a list of policies that an Android research application must comply with. The Android Manifest specifies permissions, and permission levels for the app, users grant these permissions when the app is installed.

Table 1 illustrates the process used to update existing apps

Table 2 illustrates an example policies deduced from the IRB policy document

Table 3 illustrates how a policy will be implemented in the app

**Pseudocode (language Kotlin)**

CICI Figure 2: “The University will replace the ID of your device with a pseudo random code, to help protect your identity.”

Add required permission to the manifest

|  |
| --- |
| 1. <manifest> 2. <!-- Required to access telephony services access --> 3. <uses-permission android:name=”android.permission.READ\_PHONE\_STATE”/> 4. </manifest> |

Table

|  |
| --- |
| 1. // function generating randomized device ID based on device characteristics 2. fun randomized\_device\_id(): Int { 3. // obtain Internation Mobile Equipment Identity 4. // TelephonyManager object required for getImei method 5. val tm: TelephonyManager = (TelephonyManager) 6. getSystemService (Context.TELEPHONY\_SERVICE) 7. val deviceId: String = tm.getImei() 8. // get device TAC 9. val tac: Int = get\_TAC(deviceId) 10. // generate six digit pseudorandom number 11. val serial: Int = generate\_Randoms(0,9) 12. // Return concatenation of tac, serial, check\_sum 13. return tac.toString() + serial + check\_sum(tac,serial) 14. } // end fun randomized\_device\_id(): Int |

Table

CICI Figure 3: “Some example rate-limiting and obfuscating policies as shown in Table 3”

|  |
| --- |
| 1. // limit the accelerometer 2. fun rate\_limited\_accelerometer(pauseTime: Float) { 3. // pause the code for time threshold 4. sleep\_until(next\_access\_time) 5. // update the time threshold for next access 6. next\_access\_time = pauseTime + getTime() 7. // return the original get\_accelerometer 8. return get\_accelerometer() 9. }// end fun rate\_limited\_accelerometer(pauseTime: Float) |

Table

CICI Figure 4: “To ensure data is encrypted before being saved, the PRI in [Table] 4 will be inserted in the program to replace a regular write() system call with encrypted\_write(), whenever data is saved.”

|  |
| --- |
| 1. // define a function to encrypt data on save 2. fun encrypted\_write(fd: File, data: String, pub\_key: String){ 3. // encrypt data with the researcher’s public key 4. encrypted\_data = encrypt(data, count, pub\_key) 5. // save encrypted data to a file 6. count = data.length 7. fd.writer(data) 8. }// end fun encrypted\_write(fd: File, data: String, pub\_key: String) |

Table

How the CICI\_Android modification script will be run to cleanup prior runs, decompile a given APK, apply new policy content to the APK, and update the manifest and code to exhibit the desired behaviour.

|  |
| --- |
| // 1. Clean-up any prior runs of the script  >> CICI\_Android cleanup  // 2. Run APKTool to decompile the application  >> CICI\_Androidtest\_app.apk  // 3. Find AndroidManifest.xml OR relevant Kotlin/Java snippet in the decompiled app  >> CICI\_Androidtest\_app policy\_content    // 4a. Update the manifest so that location does not use NETWORK\_PROVIDER  // information  <manifest ... >  ...  ~~<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>~~  <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/> </manifest>  //4b. Update the code snippet defining the Location() to have a set accuracy  // of 100 meters.  public Location createLocation(double lat, double lng, float accuracy) {  // Create a new Location  Location newLocation = new Location(LocationManager.GPS\_PROVIDER);  …  newLocation.setAccuracy(100);  …  }  // 5. Recompile and sign the APK  >> CICI\_Android -r test\_app |

Table

**Policy List**

In this example the policy states that GPS\_Location cannot be precise, and that no wifi access is permissible.

|  |
| --- |
| <policies>  <gps\_location>imprecise</gps\_location>  <wifi>imprecise</wifi>  </policies> |

Table

**Android Manifest**

Code in **RED** will be removed, code in **BLUE** will be inserted. Because GPS\_location must be imprecise, and WiFi access is not allowed, these will be removed. GPS location can be imprecise, and Android allows for approximate location, therefore we will insert a coarse location access.

|  |
| --- |
| *<?*xml version="1.0" encoding="utf-8"*?>* <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="edu.uccs.andreasslovacek.decompiletestproject">     <application  android:allowBackup="true"  android:icon="@mipmap/ic\_launcher"  android:label="@string/app\_name"  android:roundIcon="@mipmap/ic\_launcher\_round"  android:supportsRtl="true"  android:theme="@style/AppTheme">  <activity  android:name=".MainActivity"  android:label="@string/app\_name"  android:theme="@style/AppTheme.NoActionBar">  <intent-filter>  <action android:name="android.intent.action.MAIN" />   <category android:name="android.intent.category.LAUNCHER" />  </intent-filter>  </activity>  </application>    *<!-- Grants the app access to the users exact location-->* **<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>**  *<!-- Grants the app access to the users approximate location-->* **<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>**  *<!-- Grands the app access to managing all aspects of wifi connectivity-->* **<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"/>**   </manifest> |

Table