# Module 2 – Browsable Intents (and Deeplinks)

One tap hyperlink to completely compromise the application

# Browsable Intents (and Deeplinks)

- One feature of the Android operating system is that an Activity can be launched via a hyperlink in web browsers
- Imagine a scenario where a victim taps a hyperlink and that one tap compromises the device
- That scenario can be achieved via Browsable Intents
- This module will go over what Browsable Intents are and how to craft one

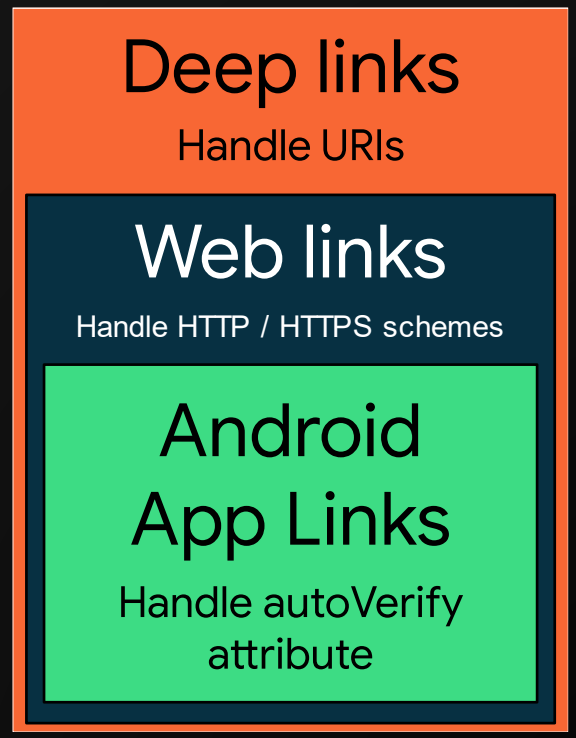



Image from the Android Studio developer documentation describing deep link capabilities

# Browsable Intents (and Deeplinks)

- A "Browsable Intent" is an Intent that can be used to launch an Activity via a hyperlink
- To achieve this, first an application must declare that it can be launched via a Browsable Intent
- This is achieved with the Intent filter category `android.intent.category.BROWSABLE` in the application's Manifest
  - NOTE: each individual Activity that wants to be launched via Browsable Intent must declare this Intent filter

- The term "DeepLink" and "Browsable Intent" are usually interchangeable terms
  - "DeepLink" is the term more widely used in both iOS and Android programming
    - https://developer.android.com/training/app-links/deep-linking
  - Technically, a "DeepLink" opens a "Browsable Intent"
  - For the purposes of this course, the term "Browsable Intent" will be used instead of "DeepLink" to avoid confusion

# Browsable Intents (and Deeplinks)

- Declaring a Browsable Intent is done in the Application's Manifest via the Intent Category `` `android.intent.category.BROWSABLE` ``
- For example, below is a screenshot that declares the Activity `` `com.sts.ryde.ui.SplashActivity` `` and a `` `Browsable` `` Category is added

- Additionally, some additional filters are added which requires that the incoming Intent must contain the following:
  - Data Uri must start with `` `ryde://open` ``
  - The Intent Action type must be `` `android.intent.action.VIEW` ``

```
<activity android:theme="@style/AppTheme.Launcher" android:name="com.sts.ryde.ui.SplashActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
    <intent-filter>
        <data android:scheme="ryde" android:host="open"/>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
    </intent-filter>
```

Android Developer documentation showing an example Manifest declaring an Activity with the Browsable Intent Category

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks)

- From a HTML perspective, a Browsable Intent looks just like a standard HTML hyperlink
- The main difference to keep in mind is the URI format of the hyperlink
  - For example, take the following HTML hyperlink for a Browsable Intent:

```
<a
href="intent://maliciouserection.com/path?yay=boo#Intent;package=com.maliciouserection.axolotl;action=android.intent.action.VIEW;scheme=theScheme;S.stringExtra=yayextrayay;end">HyperLink</a>
```

- When clicking a Browsable Intent, the hyperlink data is sent to the Android OS's "Intent Parser"
- This parser takes the hyperlink data and creates an Intent object based on the parsed data
- The source code for this parser can be found here:
  - https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/content/Intent.java
  - Search for the method `parseUriInternal(String, int)`

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks)

- At a high level, our example Browsable Intent / hyperlink can be broken down into the following:

```
<a
href="intent://maliciouserection.com/path?yay=boo#Intent;package=com.maliciouserection.axolotl;action=android.intent.action.VIEW;scheme=theScheme;S.stringExtra=yayextrayay;end">HyperLink</a>
```

  - NOTE: Google has documentation on how to craft a Browsable Intent
    - https://developer.chrome.com/docs/multidevice/android/intents/

- `intent://` – the Scheme of the hyperlink; declares that this hyperlink is an Intent type instead of the traditional HTTP/S type
- `maliciouserection.com/path?yay=boo` – the host/path of the Intent's Data Uri
  - More info on Intent Data Uris later in this module
- `#Intent;` – declare the end of the Data Uri and beginning of Intent configurations and extras
- `package=com.maliciouserection.axolotl;action=android.intent.action.VIEW;scheme=theScheme;S.stringExtra=yayextrayay;` – the Intent's configurations and extras
- `end` – the end of all the data that needed to be parsed

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks)

- Breaking down our example Browsable Intent / URI even further

```
<a
href="intent://maliciouserection.com/path?yay=boo#Intent;package=com.maliciouserection.axolotl;action=android.intent.action.VIEW;scheme=theScheme;S.stringExtra=yayextrayay;end">HyperLink</a>
```

  - One thing that should be noted is that since the URI starts with `intent://`, this hyperlink will ALWAYS be parsed by the Intent Parser
  - This is because the Intent Package in the Android OS, which contains the Intent Parser, has declared that it should always process URIs that start with `intent://`

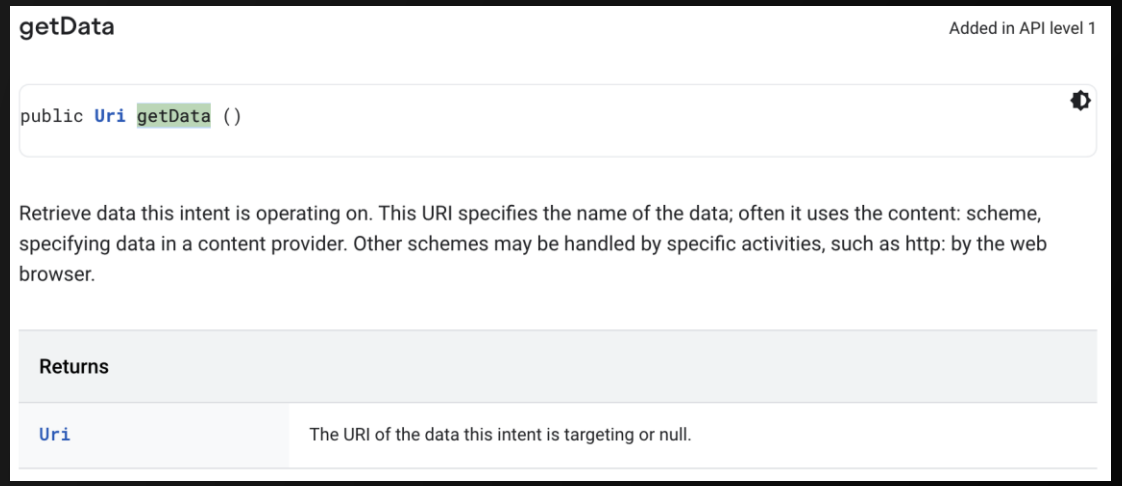
- `package` – the target package name that this Intent should be sent to
  - You can also declare a "component" along with a "package"
- `action` – the "action" that the Intent should use
- `scheme` – the scheme of the Data Uri that will be attached to the Intent
  - Again, more on Data Uris later
- `S.` – a String Extra that should be added to the Intent
  - Yes that is right, you can add Intent Extras to Browsable Intents
  - More on this later, too!

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks)

- One concept that should be discussed about Intents (and by extension, Browsable Intents) is that they can hold Uri data
  - https://developer.android.com/reference/android/content/Intent#getData()
- A “Uri” is exactly what it sounds like; a URI
- Example URIs
  - https://yay.com
  - ftp://boo.localhost
  - smb://someserver

- Uris have schemes, hosts, Uri-paths, etc.
- Example
  - `https://maliciouserection.com/somepath`
    - `https://` = scheme
    - `maliciouserection.com` = host
    - `/somepath` = Uri-path



getData

Added in API level 1

```
public Uri getData ()
```

Retrieve data this intent is operating on. This URI specifies the name of the data; often it uses the content: scheme, specifying data in a content provider. Other schemes may be handled by specific activities, such as http: by the web browser.

| Returns | |
|---|---|
| Uri | The URI of the data this intent is targeting or null. |

Android Developer documentation about the `getData()` public method

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks)

Storing a Uri in Intents is easy

```
Intent intent = new Intent();
intent.setData(Uri.parse( uriString: "https://yay.com"));
```

Retrieving the URI is easy

```
Intent intent = getIntent();
Uri uri = intent.getData();
```

You can then also take apart the URI

```
String scheme = uri.getScheme();
String host = uri.getHost();
String path = uri.getPath();
```

Some other notable things to do with Uris:

- `getPathSegments()` - get how many different path segments
  - Example – http://yay.com/pathsegment/pathsegmentagain
  - There are 2 path segments above
- `getPort()` - gets the port if defined
  - Example – http://yay.com:80
  - The port is 80

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks)

- Back to Browsable Intents, if we were to convert our Browsable Intent into an Android Intent object:

```
<a href="intent://maliciouserection.com/path?yay=boo#Intent;package=com.maliciouserection.axolotl;action=android.intent.action.VIEW;scheme=theScheme;S.stringExtra=yayextrayay;end">HyperLink</a>
```

- Package – `com.maliciouserection.axolotl`
- Action – `android.intent.action.VIEW`
- Data – `theScheme://maliciouserection.com/path?yay=boo`
- String Extra – `yayextrayay`

```
Intent intent = new Intent();
intent.setPackage("com.maliciouserection.axolotl");
intent.setAction("android.intent.action.VIEW");

Uri uri = Uri.parse( uriString: "theScheme://maliciouserection.com/path?yay=boo");

intent.setData(uri);

intent.putExtra( name: "stringExtra", value: "yayextrayay");
```

Java code showing how the Browsable Intent would be converted into an Intent object

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks) - Example

- We will now use Axolotl to better demonstrate how Browsable Intents function
- On Axolotl's main menu, tap:
  - "Exercise Modules"
  - "Browsable Intents (and Deeplinks)"
- A blank activity will appear with some text
  - The launched activity is programmed via the Java class `com.maliciouserection.axolotl.example.activity.intents.browsableIntent`


MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks) - Example

- Use your favorite Android application decompiler to decompile Axolotl
- Open the `manifest.xml` file and confirm that the Activity `com.maliciouserection.axolotl.example.activity.intents.browsableIntent` is exported
  - Note that the Intent filter declares a Browsable Intent Category and a Uri scheme value `` `axolotlbrowsablescheme` ``

```
<activity android:name="com.maliciouserection.axolotl.example.activity.intents.getIntent" android:exported="true"/>
<activity android:name="com.maliciouserection.axolotl.example.activity.intents.browsableIntent" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="axolotlbrowsablescheme"/>
    </intent-filter>
</activity>
<activity android:name="com.maliciouserection.axolotl.example.activity.nfc.nfcIntent" android:exported="true">
```

Axolotl's manifest.xml with the exported Activity highlighted


MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks) - Example

- Now we are going to craft a Browsable Intent hyperlink, host it on a webserver, and browse to it on the Android device
- Setup `adb` reverse port forwarding so that if you browse to http://localhost:8000 on your Android device, the traffic is forwarded to your workstation on TCP port 8000
  - `adb reverse tcp:8000 tcp:8000`
- On the same computer, create an `index.html` file with the source code on the right and place it in a directory that you will host your web server from
  - Notice how the source code contains a Browsable Intent hyperlink

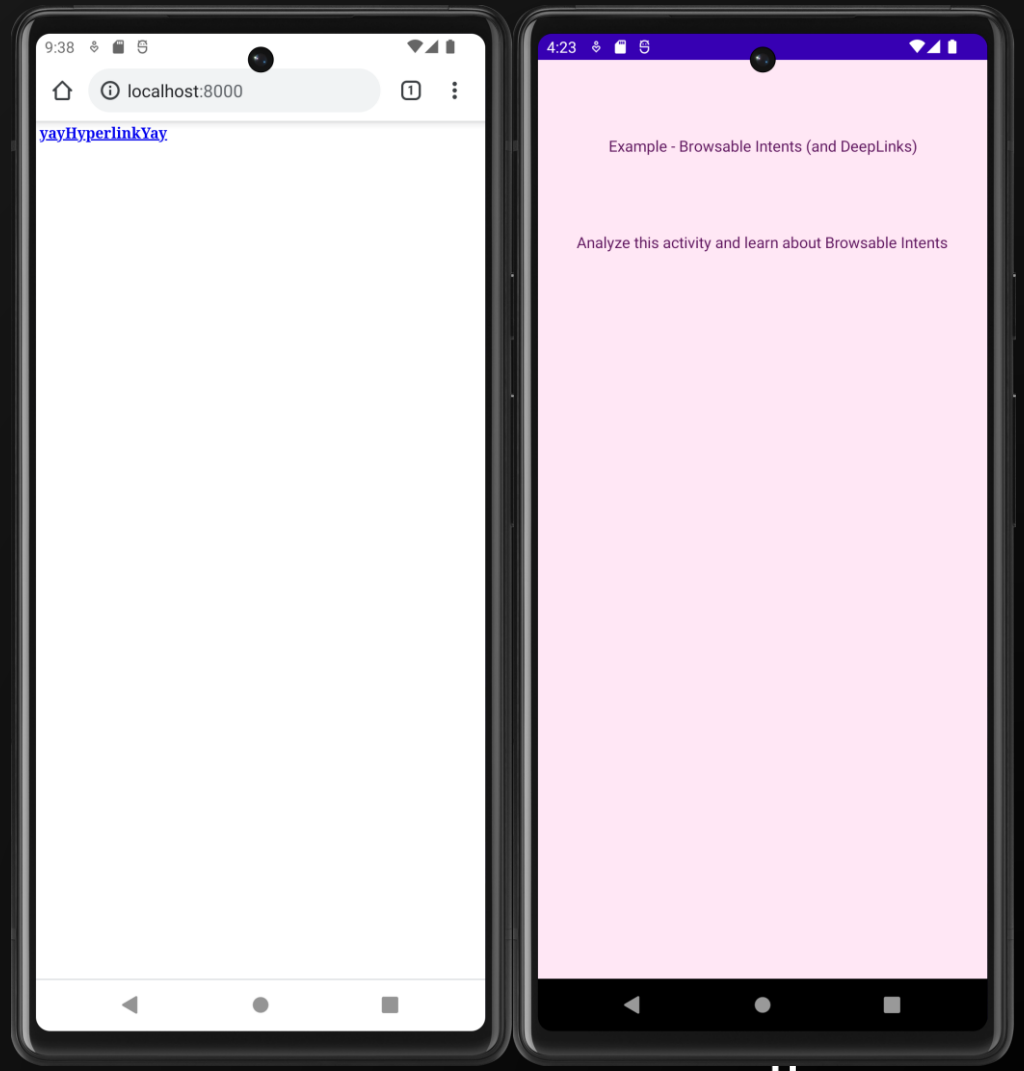
Source code for `index.html`:

```
<html>
<h1>
<a
href="intent://#Intent;package=com.maliciouserect
ion.axolotl;component=com.maliciouserection.axolo
tl.example.activity.intents.browsableIntent
;action=android.intent.action.VIEW;scheme=axolotl
browsablescheme;end;">yayHyperlinkYay</a>
</h1>
</html>
```

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks) - Example

- Using a command prompt window and Python, browse to where you stored `index.html` and start a Python web server
  - `python3 -m http.server`
  - By default, Python will start a web server on all network interfaces and listen on TCP port 8000
- On your Android device, use Chrome to browse to http://localhost:8000
- Tap the "yayHyperlinkYay" link and the Activity `browsableIntent` should launch
- So we now have a working Browsable Intent hyperlink, what next?




MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks) - Example

- Decompiling `com.maliciouserection.axolotl.example.activity.intents.browsableIntent` reveals that the Activity processes some Intent Extras and a Data Uri
- To get past the highlighted area below, a Browsable Intent must be created which meets the following criteria:
  - String Extra `yaystringyay` must not be null
  - Data Uri must not be null

```
private void theMainMethod() {
    Intent yayintentyay = getIntent();
    if (yayintentyay.getStringExtra("yaystringyay") != null && yayintentyay.getData() != null) {
        Uri yayuriyay = yayintentyay.getData();
        String yaystringyay = yayintentyay.getData().toString();
        this.text.setText("theMainMethod - getIntent().getData().toString(): " + yaystringyay);
        int yayintyay = yayintentyay.getIntExtra("yayintyay", 0);
        if (yayintyay > 0 && yayuriyay.getHost() != null && Objects.equals(yayuriyay.getHost(), "axolotlexamplehost")) {
            aSecondMethod(yayintentyay);
        }
    }
}
```

Decompiled example Activity `browsableIntent`

MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks) - Example

- Let's add an Extra String value `yaystringyay`
- In Browsable Intent hyperlinks, Intent Extras can be defined by a single character, key, and value
  - String
    - `S.<key>=<value>`
  - Boolean
    - `B.<key>=<value>`
  - Byte
    - `b.<key>=<value>`
  - Char
    - `c.<key>=<value>`
  - Double
    - `d.<key>=<value>`
  - Float
    - `f.<key>=<value>`
  - Integer
    - `i.<key>=<value>`
  - Long
    - `l.<key>=<value>`
  - Short
    - `s.<key>=<value>`

- Since we want to add a String Extra `yaystringyay`, we should update index.html to do so

- Updated source code for `index.html`:

```
<html>
<h1>
<a
href="intent://#Intent;package=com.maliciouserect
ion.axolotl;component=com.maliciouserection.axolo
tl.example.activity.intents.browsableIntent
;action=android.intent.action.VIEW;scheme=axolotl
browsablescheme;S.yaystringyay=test123;end;">yayH
yperlinkYay</a>
</h1>
</html>
```

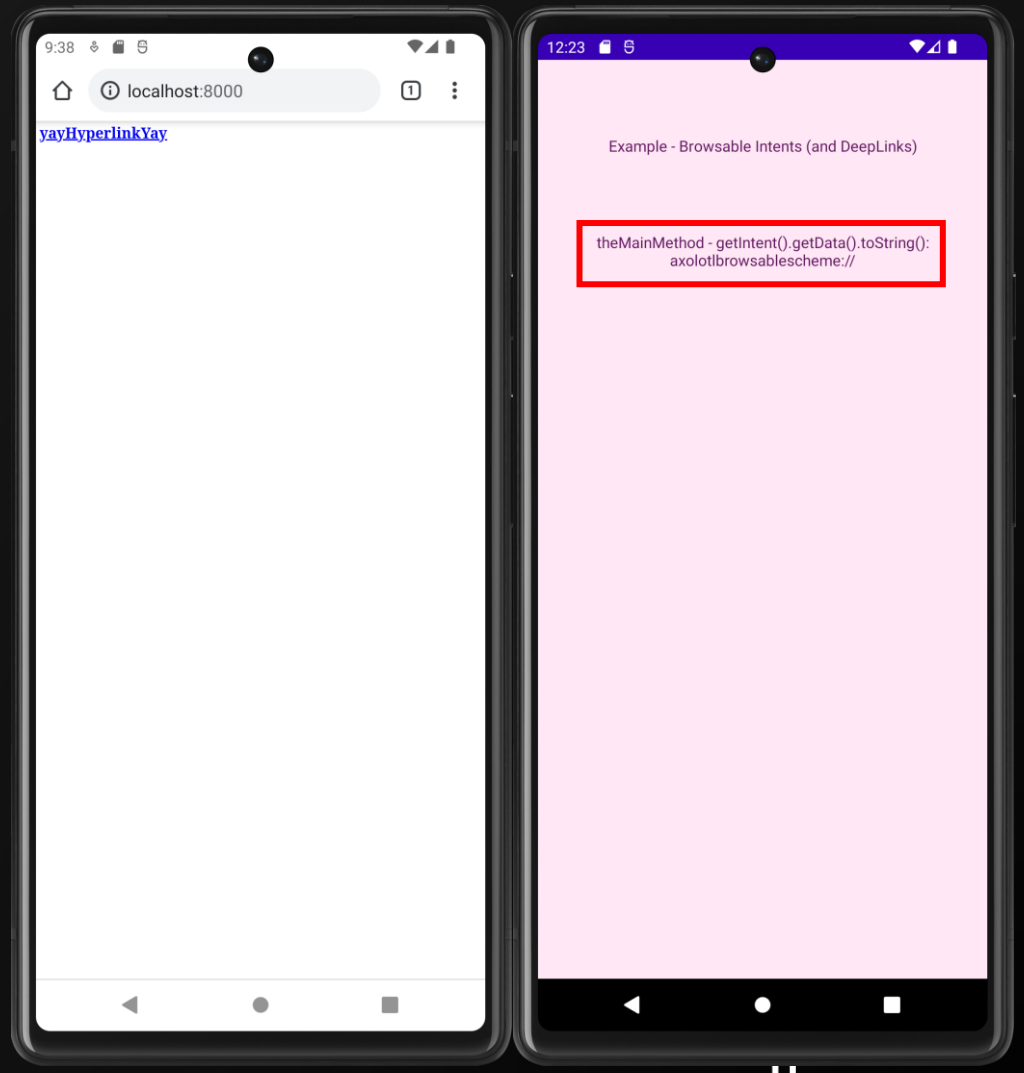
MALICIOUS ERECTION LLC

# Browsable Intents (and Deeplinks) - Example

- After updating your `index.html` file, on your Android device, refresh http://localhost:8000
- Tapping yayHyperlinkYay should now bring up the `browsableIntent` Activity with more text
- If you read the Activity's source code, you should see that the Activity will run `getData()` on the received Intent and output the result to the Activity's text
  - Play around with the Browsable Intent hyperlink and try to make `getData()` output other text

```
private void theMainMethod() {
    Intent yayintentyay = getIntent();
    if (yayintentyay.getStringExtra("yaystringyay") != null && yayintentyay.getData() != null) {
        Uri yayuriyay = yayintentyay.getData();
        String yaystringyay = yayintentyay.getData().toString();
        this.text.setText("theMainMethod - getIntent().getData().toString(): " + yaystringyay);
```

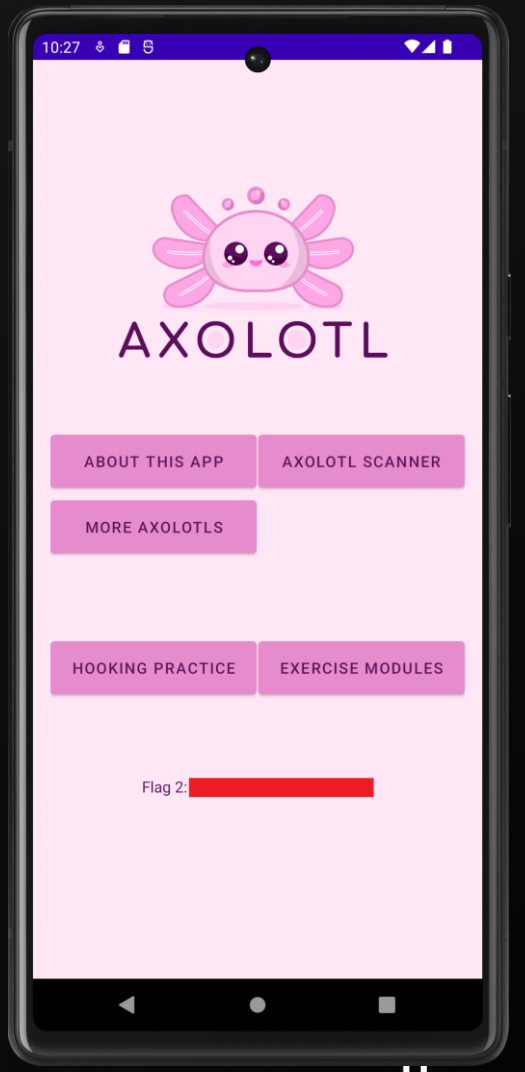
Decompiled example Activity `browsableIntent`




MALICIOUS ERECTION LLC

# Module 2 Exercise

- The Activity `com.maliciouserection.axolotl.example.activity.intents.browsableIntent` had the method `` `aSecondMethod()` `` which would get called when:
  - The Intent Integer Extra `` `yayintyay` `` is greater than 0
  - The Intent's Data Uri has a host value of `` `axolotlexamplehost` ``
- Craft a Browsable Intent which calls `` `aSecondMethod()` ``
- Capture The Flag - The activity `com.maliciouserection.axolotl.MainActivity` contains the method `` `showFlag2()` ``
  - Craft a Browsable Intent which will show Flag 2 on Axolotl's `` `MainActivity` ``




MALICIOUS ERECTION LLC