



# Examen de la sécurité du système « Android »

Présenté par: Sabeur ETTIH



# Plan

- ▶ Introduction au Framework
- ▶ Etapes de Backdoorisation d'une application Android
- ▶ Démonstration





# Framework Droid Pentest

---

Introduction du concept



# Introduction

- ▶ Droid Pentest: un framework de preuve de concept.
- ▶ Son but est d'examiner le niveau de sécurité du système Android
- ▶ Étant le système plus populaire, Android est-il suffisamment sécurisé ?
- ▶ Étudier et voir à quel point est-il possible d'attaquer un téléphone intelligent qui tourne sous Android.



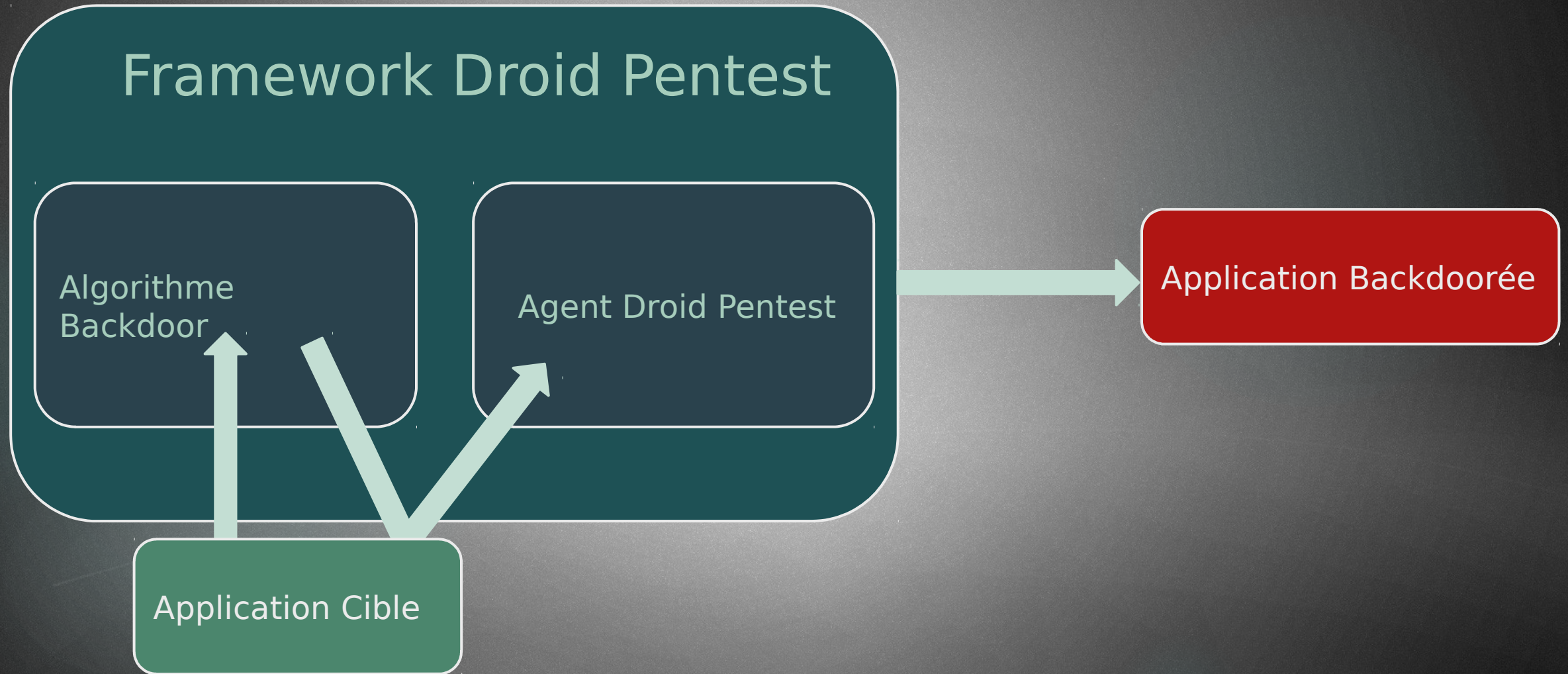
# Introduction

Le framework va permettre de:

- ▶ Générer des applications Android backdoorées.
- ▶ Contrôler le téléphone infecté à distance:
  - Repérer les déplacements de la victime;
  - Récupérer la liste de ses contacts
  - Récupérer ses messages textes (SMS)
  - Écouter ses conversations téléphoniques



# Introduction





# Introduction



UNIVERSITÉ  
LAVAL | LSI



Accéder au framework

Bienvenue à DroidPentest



# Introduction







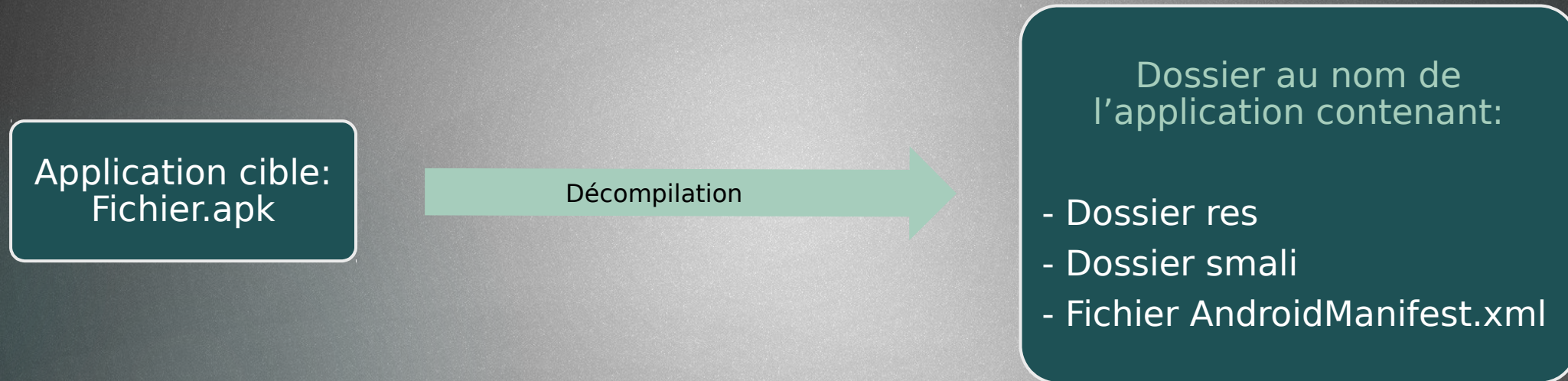
# Backdoorisation d'une application Android

---

Les différentes étapes de l'opération



### ► 1- Décompiler l'apk cible via l'outil Apktool





### ► 2- Recherche de l'activité principale dans AndroidManifest.xml

...

...

```
<activity android:name="com.example.android.apis.view.MapViewDemo"
android:label="MapView">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

...



### ▶ 3- Suppression de l'intent filtre "LAUNCHER"

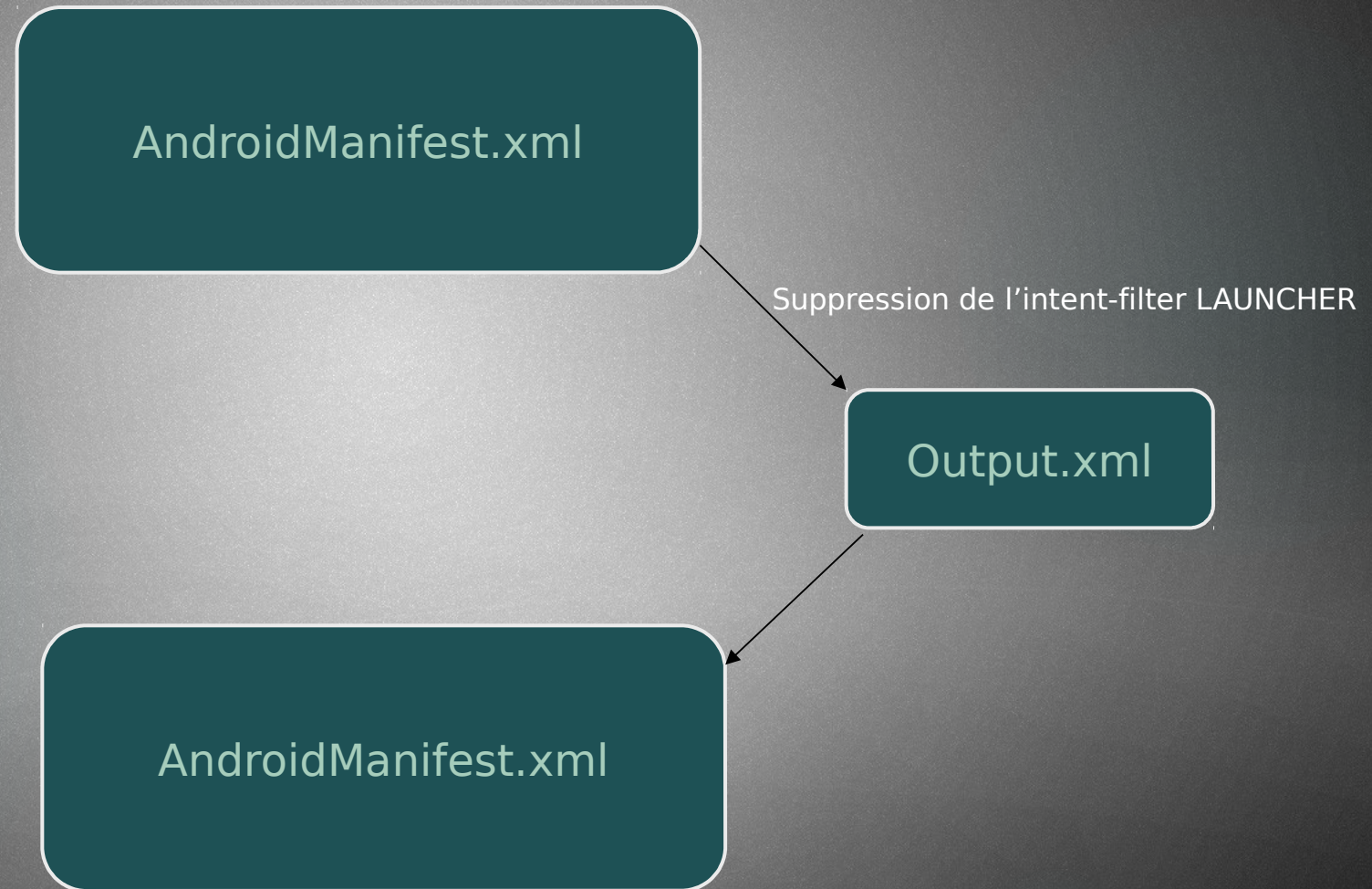
<

```
<activity android:name="com.example.android.apis.view.MapViewDemo"  
android:label="MapView">
```

```
</activity>
```



### ► 4- Générer un nouveau fichier AndroidManifest.xml





### ► 5- Injection dans l'activité principale de l'agent

Un code permettant de lancer l'application cible lors du démarrage de l'agent (à partir de son activité principale)

AgentDP/src/com/ulaval/droidpentest/MainActivity.java

```
public class MainActivity extends Activity {  
    @Override  
    void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        Log.i("Agent:", "L'agent a été lancé");  
        ...  
        ...  
        setContentView(R.layout.activity_main);  
  
        finish();  
    }  
}
```

```
Intent intentApp = new Intent(getApplicationContext(), ".$mainapp.".class);  
\nstartActivity(intentApp); \n
```



- ▶ 6- Création d'un répertoire pour l'application cible sous le src de l'agent

```
/AngetDP
-- /src
-- /com
  - /droidpentest
  - /AppCible
-- /bin
--/assets
--/res
...
```

← Création du dossier



- ▶ 7- Création de la classe principale de l'application à backdoorer

```
/AngetDP
-- /src
-- /com
-- /droidpentest
-- /AppCible
    MainActivityAppCible.java
-- /bin
-- /assets
-- /res
...
```

← Création du fichier java



- ▶ 8- Injection d'un code permettant de simuler la présence de la classe afin d'éviter des erreurs de compilation

AgentDP/src/com/AppCible/MainActivityAppCible.java

```
package com.example.android.appcible;  
  
import android.app.Activity;  
  
public class MainActivityAppCible extends Activity  
{  
  
}
```



- ▶ 9- Mettre à jour le projet AgentDP via sdk/tools/android et compiler via ant

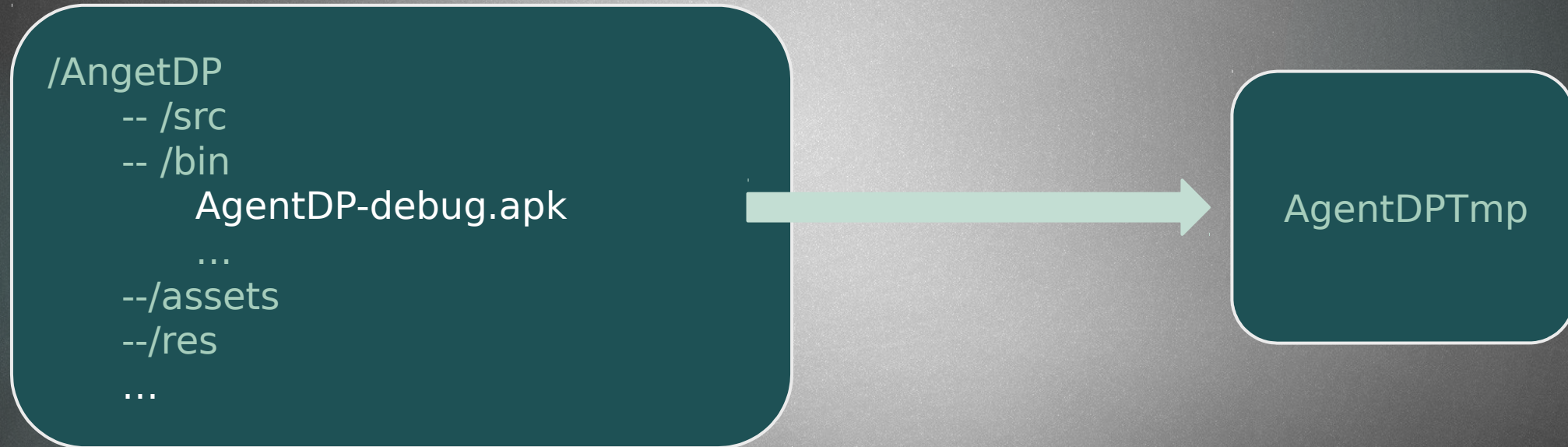


Agent Droid Pentest

```
android update project --name AgentDP --path AgentDP/  
ant -f AgentDP/build.xml clean debug
```

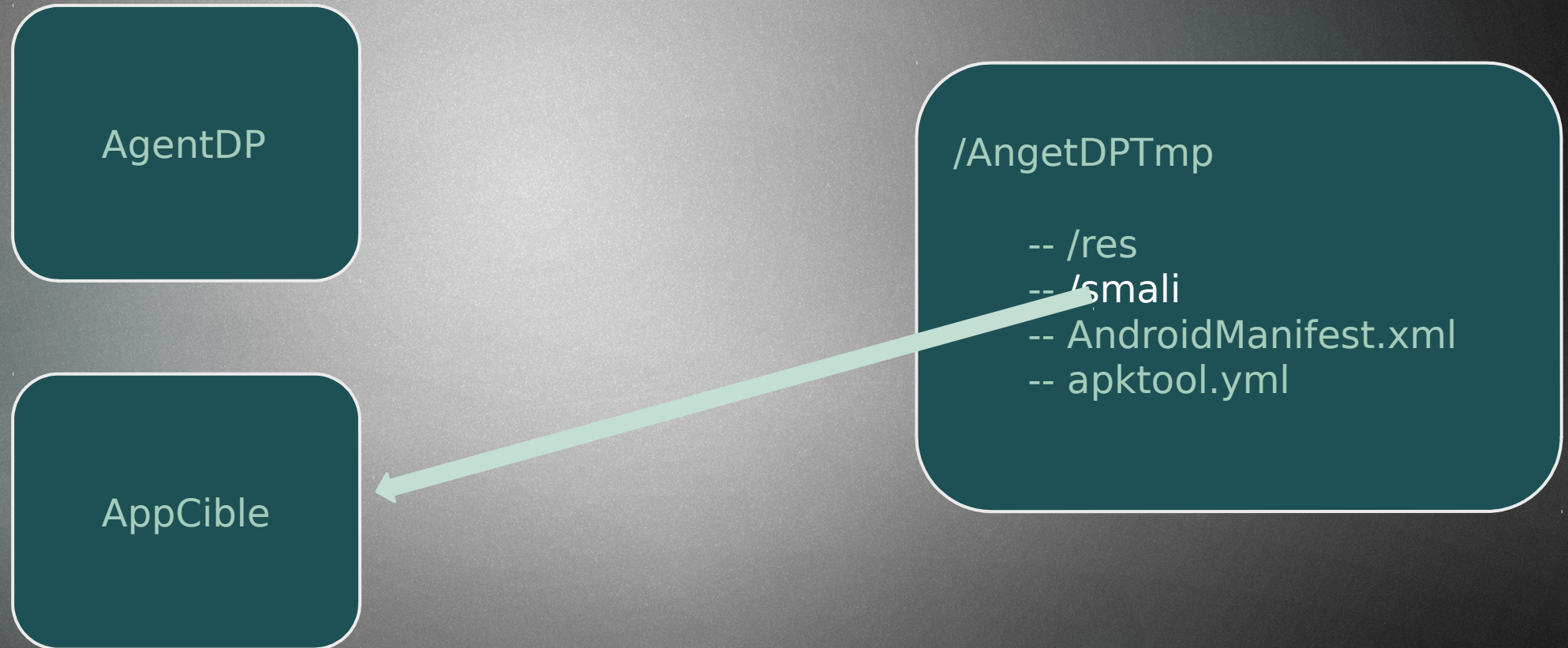


- ▶ 10- Décompiler le fichier apk situé dans AgentDP/bin/AgentDP-debug.apk dans un dossier temporaire





- ▶ 11- Déplacer le package en format smali du dossier temporaire sous le répertoire smali de l'application cible(décompilée précédemment)





- ▶ 12- Fusion des fichiers AndroidManifest.xml de l'Agent et l'application à backdoorer (Mettre le tout dans le dossier de décompilation de l'application à backdoorer)

12-a Injection du contenu de la balise <applications> (activités et services).

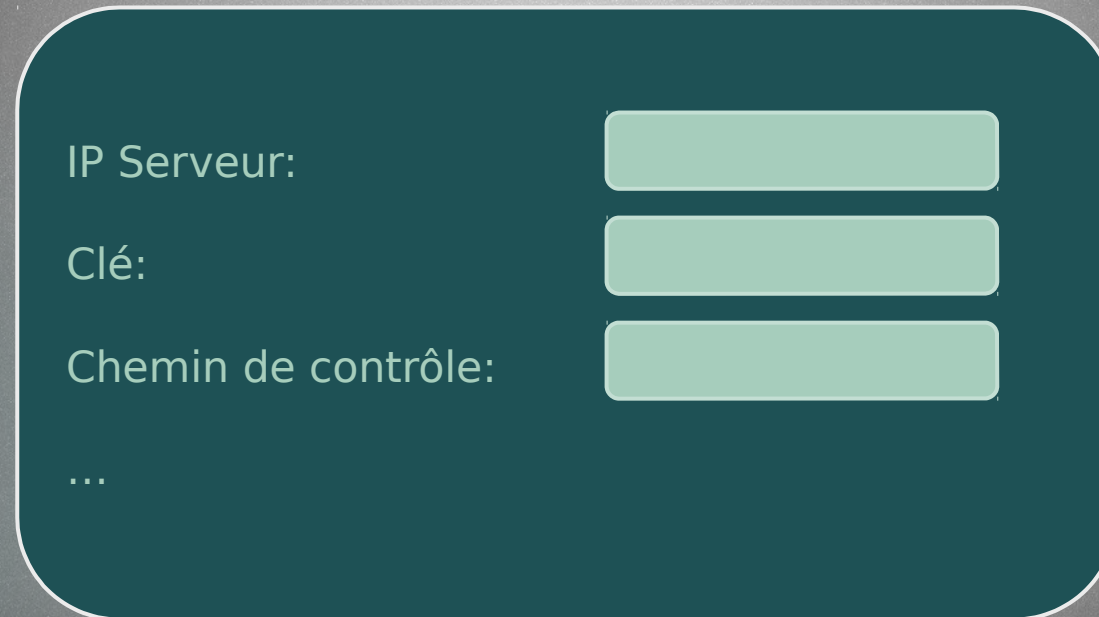
```
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission
android:name="android.permission.READ_CONTACTS"/> <uses-
permission android:name="android.permission.INTERNET"/>
...
<application>
  <activity android:name="com.ulaval.droidpentest.MainActivity">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category
android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
  </activity>
  <service android:name="WebUploadService"> </service>
...
```

AndroidManifest.xml

AppCible



- ▶ 13- Introduire les paramètres de configuration de l'agent : @IP du serveur, l'url, le numéro de control...



IP Serveur:

Clé:

Chemin de contrôle:

...



- ▶ 14- Injecter les valeurs introduites dans le fichier string.xml sous le répertoire /res/values du dossier de l'application à backdoorer

Valeurs  
introduites  
dans  
l'étape 13-

```
/AppCible
-- /res
    /drawable
    /layout
    /values
        values.xml
        string.xml
        public.xml
-- /smali
-- AndroidManifest.xml
-- apktool.yml
```

A diagram illustrating the injection of values into an Android application. On the left, a teal rounded rectangle contains the text 'Valeurs introduites dans l'étape 13-'. A light green arrow points from this box to a larger teal rounded rectangle on the right. This rectangle contains a directory tree structure. The path '/AppCible' is at the top. Below it is '-- /res', followed by a list of subdirectories: '/drawable', '/layout', and '/values'. Under the '/values' directory, three files are listed: 'values.xml', 'string.xml', and 'public.xml'. Below these are '-- /smali', '-- AndroidManifest.xml', and '-- apktool.yml'. The light green arrow points directly to the 'string.xml' file, indicating the target of the injection.



- ▶ 15- Corriger un bug au niveau de l'outil Apktool: remplacer "@\*android:style/" par "@android:style/" dans le fichier styles.xml sous /res/values

@\*android:style/

Changé par

@android:style/

/AppCible

-- /res

/drawable

/layout

/values

values.xml

string.xml

public.xml

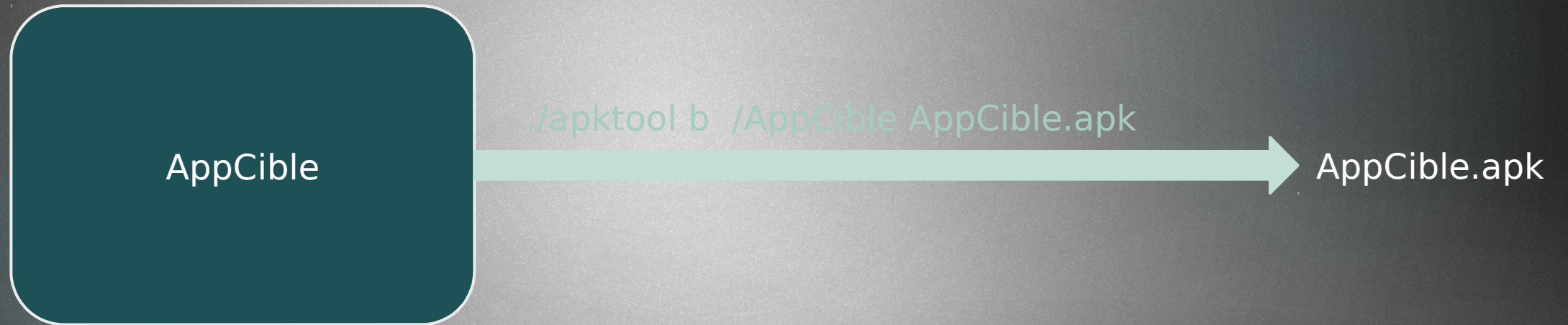
-- /smali

-- AndroidManifest.xml

-- apktool.yml

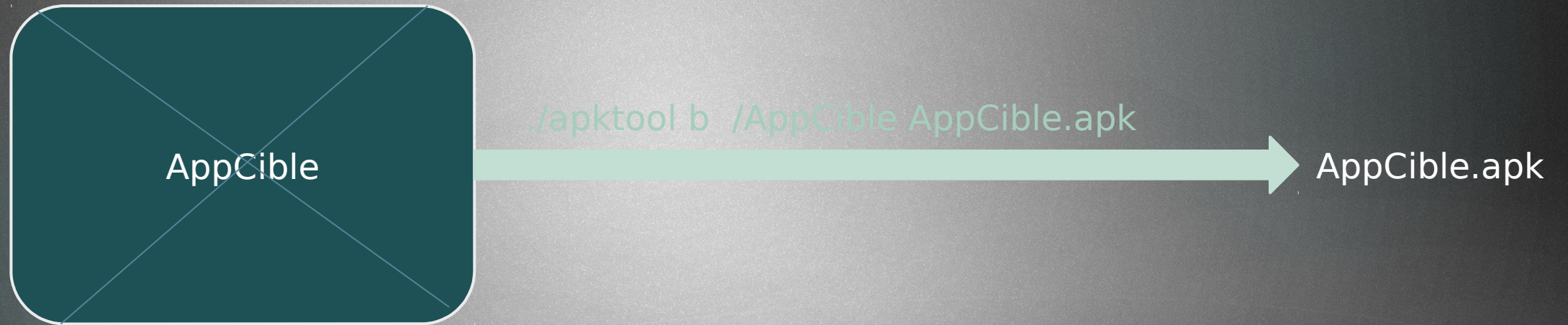


- ▶ 16- Compiler le dossier de l'application à backdoorer via l'outil apktool





- ▶ 17- Supprimer le dossier en question





## ► 18- Décompiler le fichier apk obtenu





### ► 19- Remapping du fichier public.xml pour corriger un bug dans l'outil apktools (décalage au niveau des valeurs hexadécimales des éléments ajoutés)

19-a Lecture des valeurs des id dans le fichier public.xml sous le répertoire res/values


19-b Recherche des valeurs actuelles dans le fichier  
/smali/com/ulaval/droidpentest/R\$string.smali

19-c Recherche dans tous les fichiers sources smali de l'agent les occurrences des éléments ajoutés et les remplacer par les valeurs mentionnées dans le public.xml

AppCible/res/values/public.xml

```
<public type="string" name="key" id="0x7f050001" />
<public type="string" name="controlnumber"
id="0x7f050002" />
<public type="string" name="controlIP" id="0x7f050003" />
...
```

```
/AppCible
-- /res
-- smali
-- AndroidManifest.xml
-- apktool.yml
```

A horizontal arrow points from the 'id' attribute of the 'controlnumber' element in the left box to the 'smali' directory entry in the right box, indicating the mapping of the resource ID to the smali file.



- 20- Corriger de nouveau le bug au niveau de l'outil Apktool: Remplacer "@\*android:style/" par "@android:style/" dans le fichier styles.xml sous /res/values

@\*android:style/

Changé par

@android:style/

/AppCible

-- /res

/drawable

/layout

/values

values.xml

string.xml

public.xml

-- /smali

-- AndroidManifest.xml

-- apktool.yml



- ▶ 21- Compiler le dossier de l'application à backdoorer et signer le fichier apk obtenu en sorti

AppCible

```
jarsigner -verbose -sigalg MD5withRSA -digestalg SHA1  
-keystore .android/debug.keystore AppCible.apk androiddebugkey
```

AppCible.apk  
(Backdoorée)





# Démonstration

---

Fonctionnement du Framework et de l'agent



