Security Model

Introduction

- The android OS platform includes a security model that was developed especially for mobile telephones. It spans through the entire application life cycle.

Deployment

- The application (the *.apk file) must be signed with a digital signature in order to be capable of installing it onto a device.
- This digital signature ensures that once an application was installed it will not be possible to upgrade it with a new version that comes from another developer.

Deployment

- When trying to upgrade an application with a new version it must be signed with the same signature that was used for signing the original version. Otherwise, the android phone won't allow the upgrade.
- The digital signature doesn't need to be purchased from a certificate authority, such as Verisign. The certificate is self signed.

Deployment

- The eclipse plug-in is capable of taking care after signing our APK file. It automatically signs it before deploying that file onto the emulator.
- The default certificate it uses for the emulator cannot be used for signing an application in order to deploy it on a real device.

Deployment

- The android platform tests whether the signature has already expired at install time only.
- When trying to update an application its signature has already expired the android platform won't allow us to complete the update.

The `keytool` Utility

- We can generate the required keystore for signing our application by using the keytool utility the JDK includes.

```
c:\keytool -genkey  -keystore c:\myrelease.keystore
                    -alias aliasname
                    -storepass xxxxxx
                    -keypass yyyyyy
                    -keyalg RSA
                    -validity nnnnn
```

The `keytool` Utility

`keytool`

This is the name of the utility we want to use. This utility is located within the 'bin' folder of our JDK.

`-getkey`

This tells the keytool utility that we want to generate a key.

`-keystore c:\filename.keystore`

This tells the name of the file in which we want the key to be stored.

The `keytool` Utility

`-alias aliasname`

This is the name of the keystore entry. This is the alias we will be able to use in order to access the key we create.

`-storepass xxxxxx`

This is the password that will be used to access the keystore.

`-keypass yyyyyy`

This is the password used to access the private key.

The `keytool` Utility

`-keyalg RSA`

This is the algorithm to be used when generating the key.

`-validity nnnnn`

This is the validity period. Specified in days. It should be long enough to support the entire lifespan of the application (including its updates).

The `keytool` Utility

- When executing this command we will be asked for various questions, which are part of the process of generating a keystore. Once completed, we shall get a digital certificate we can now use in conjunction with the `jarsigner` utility in order to sign our application.
- The `jarsigner` utility will use the digital certificate the keystore file holds.

The `jarsigner` Utility

```
c:\jarsigner     -keystore c:\mykeys.keystore
                 -storepass xxxxxx
                 -keypass yyyyyy
                 myapp.apk aliasname
```

The `jarsigner` Utility

`jarsigner`

This is the name of the utility we use in order to sign an apk file.

`-keystore c:\mykeys.keystore`

We should specify the name of the keystore file we want to use. The one that includes the key.

`-storepass xxxxxx`

We should specify the password required to access the keystore we want to use.

The `jarsigner` Utility

`-keypass yyyyyy`

We should specify the password of the alias we specified when creating the keystore now being used.

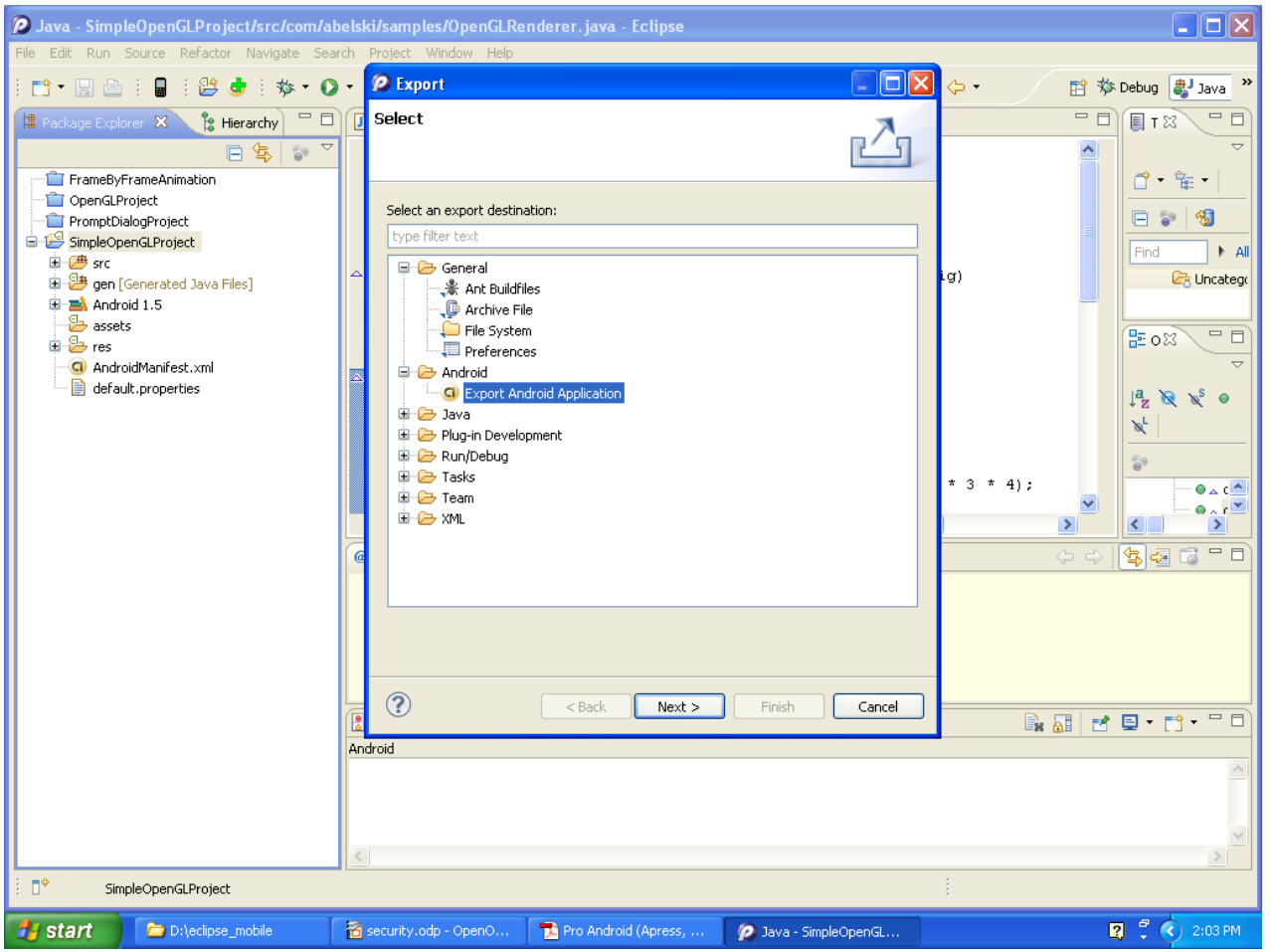
`myapp.apk aliasname`

We should specify the name of the apk file we want to sign following with the alias name been used when creating the digital certificate we are now using.

Deployment using The Eclipse

- Working with the Eclipse we can use the 'Export Android Application' that uses the `keytool` and the `jarsigner` utilities in order to sign our application. We can avoid using the `keytool` and the `jarsigner` from the command line.

Deployment using The Eclipse



Separated Processes

- Each application for the android platform is executed within a separated process. Each process has a unique and a permanent ID assigned when the application is installed.
- This separation prevents each application from accessing other applications directly.

Separated Processes

- Applications can still share information with each other by using predefined mechanisms such as content providers, services and starting new activities from within the running one.

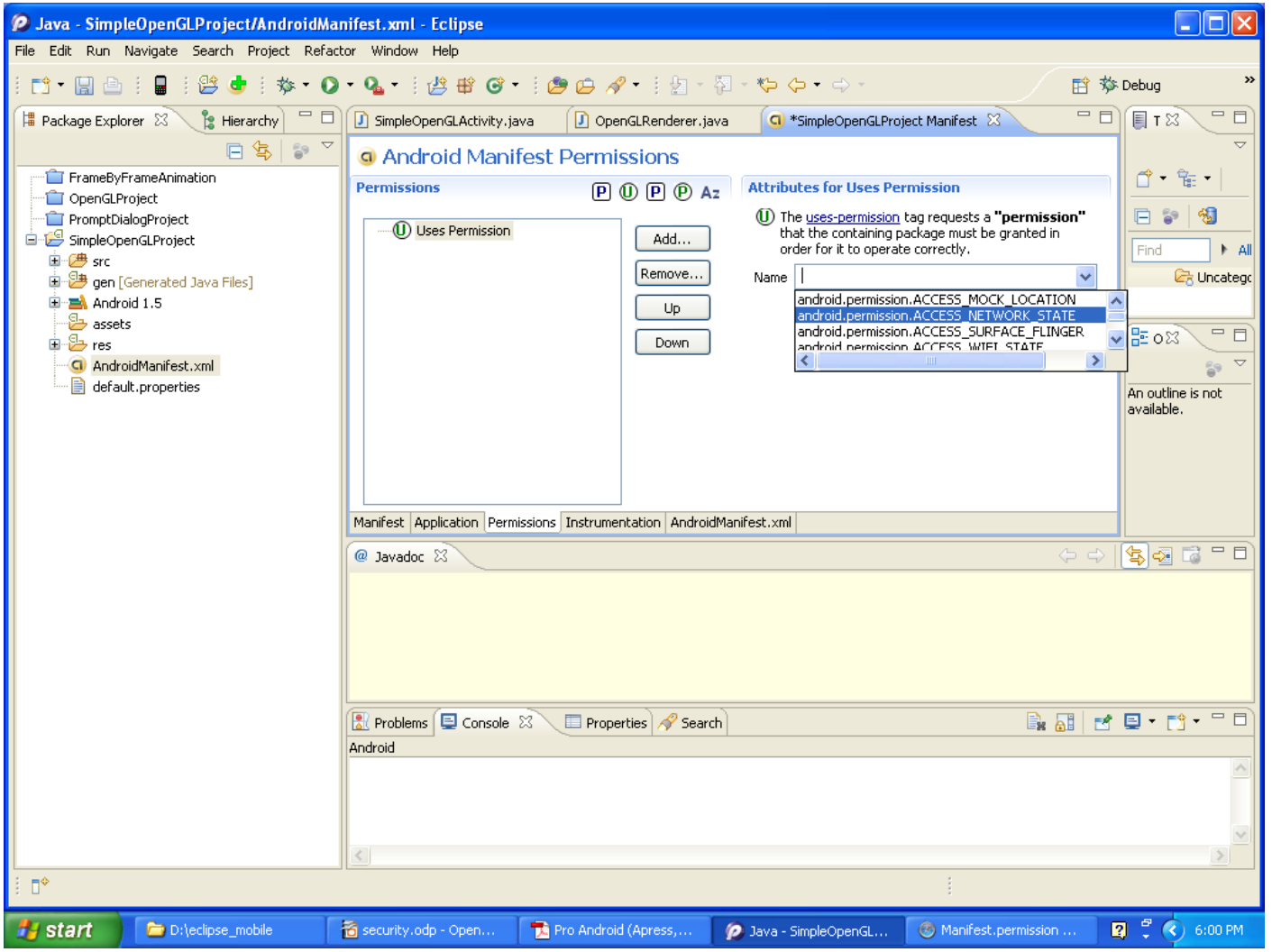
Declarative Permission Model

- The android platform implements a declarative permission model that protects sensitive features, such as the contacts list, the ability to send and receive SMS, the ability to make a phone call etc.
- In order to use any of these features and resources we must add the required (one or more) permissions to the `AndroidManifest.xml` file.

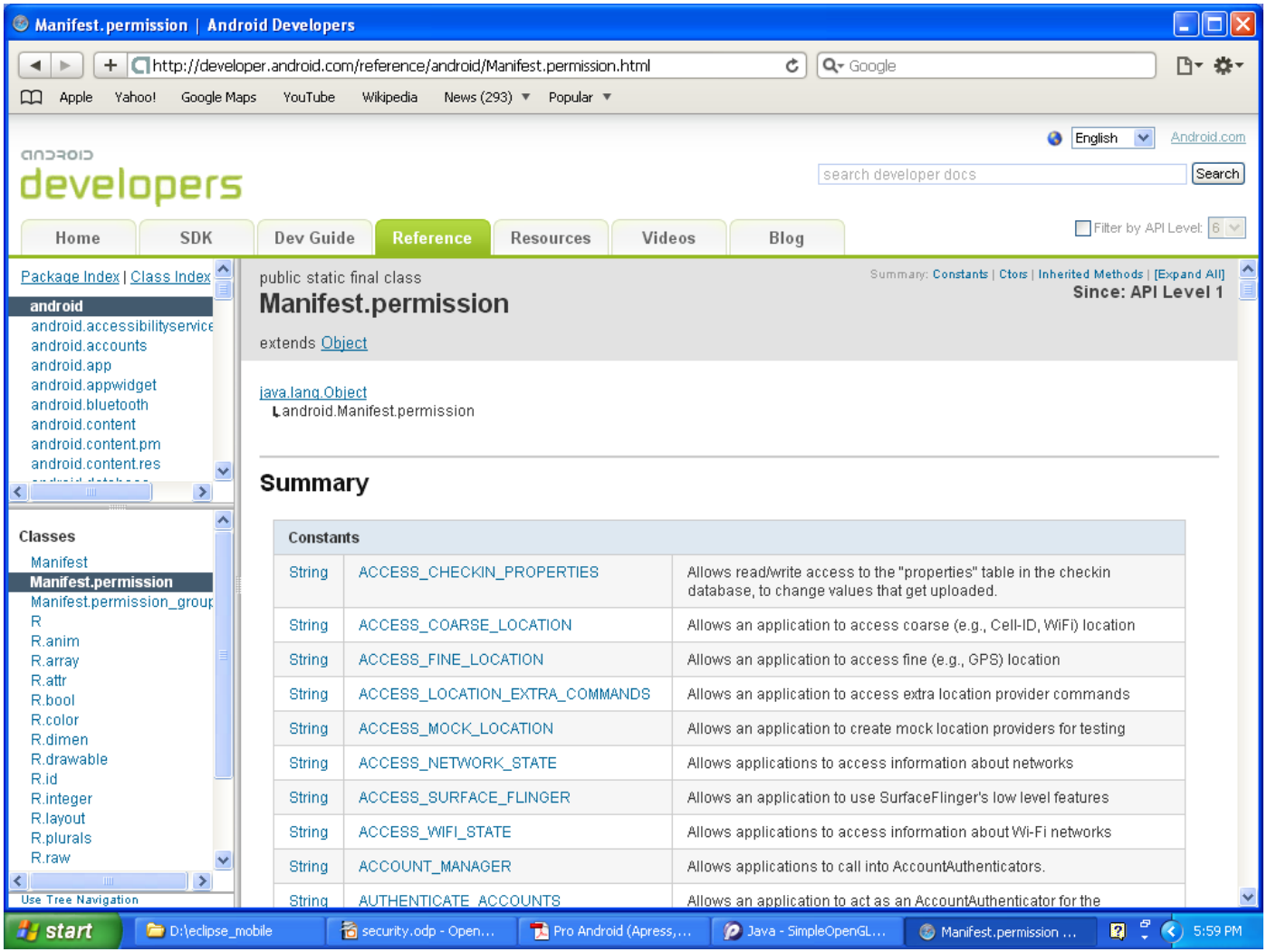
Declarative Permission Model

- When installing an application the android platform either grants or denies the requested permissions based on the signature of the .apk file and/or the user's settings and feedback.
- Each permission is defined as a final static integer variable within the `android.Manifest.permission` final static inner class.

Declarative Permission Model



Declarative Permission Model



Security Model

Introduction

- The android OS platform includes a security model that was developed especially for mobile telephones. It spans through the entire application life cycle.

Deployment

- The application (the *.apk file) must be signed with a digital signature in order to be capable of installing it onto a device.
- This digital signature ensures that once an application was installed it will not be possible to upgrade it with a new version that comes from another developer.

Deployment

- When trying to upgrade an application with a new version it must be signed with the same signature that was used for signing the original version. Otherwise, the android phone won't allow the upgrade.
- The digital signature doesn't need to be purchased from a certificate authority, such as Verisign. The certificate is self signed.

Deployment

- The eclipse plug-in is capable of taking care after signing our APK file. It automatically signs it before deploying that file onto the emulator.
- The default certificate it uses for the emulator cannot be used for signing an application in order to deploy it on a real device.

Deployment

- The android platform tests whether the signature has already expired at install time only.
- When trying to update an application its signature has already expired the android platform won't allow us to complete the update.

The `keytool` Utility

- We can generate the required keystore for signing our application by using the keytool utility the JDK includes.

```
c:\keytool -genkey  -keystore c:\myrelease.keystore
                    -alias aliasname
                    -storepass xxxxxx
                    -keypass yyyyyy
                    -keyalg RSA
                    -validity nnnnn
```

The `keytool` Utility

`keytool`

This is the name of the utility we want to use. This utility is located within the 'bin' folder of our JDK.

`-getkey`

This tells the keytool utility that we want to generate a key.

`-keystore c:\filename.keystore`

This tells the name of the file in which we want the key to be stored.

The `keytool` Utility

`-alias aliasname`

This is the name of the keystore entry. This is the alias we will be able to use in order to access the key we create.

`-storepass xxxxxx`

This is the password that will be used to access the keystore.

`-keypass yyyyyy`

This is the password used to access the private key.

The `keytool` Utility

`-keyalg RSA`

This is the algorithm to be used when generating the key.

`-validity nnnnn`

This is the validity period. Specified in days. It should be long enough to support the entire lifespan of the application (including its updates).

The `keytool` Utility

- When executing this command we will be asked for various questions, which are part of the process of generating a keystore. Once completed, we shall get a digital certificate we can now use in conjunction with the `jarsigner` utility in order to sign our application.
- The `jarsigner` utility will use the digital certificate the keystore file holds.

The `jarsigner` Utility

```
c:\jarsigner     -keystore c:\mykeys.keystore
                 -storepass xxxxxx
                 -keypass yyyyyy
                 myapp.apk aliasname
```

The `jarsigner` Utility

`jarsigner`

This is the name of the utility we use in order to sign an apk file.

`-keystore c:\mykeys.keystore`

We should specify the name of the keystore file we want to use. The one that includes the key.

`-storepass xxxxxx`

We should specify the password required to access the keystore we want to use.

The `jarsigner` Utility

`-keypass yyyyyy`

We should specify the password of the alias we specified when creating the keystore now being used.

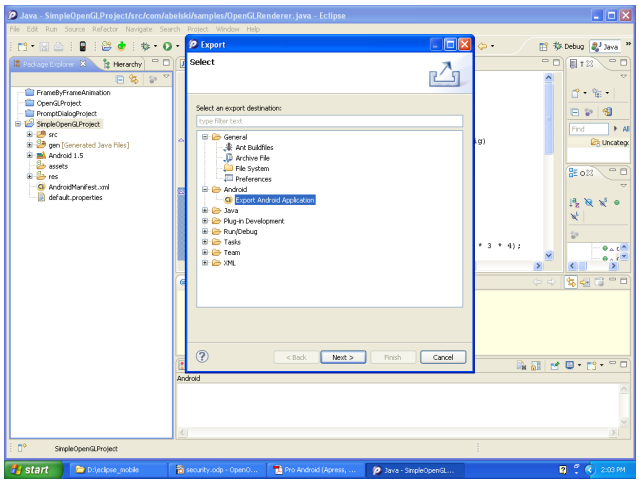
`myapp.apk aliasname`

We should specify the name of the apk file we want to sign following with the alias name been used when creating the digital certificate we are now using.

Deployment using The Eclipse

- Working with the Eclipse we can use the 'Export Android Application' that uses the `keytool` and the `jarsigner` utilities in order to sign our application. We can avoid using the `keytool` and the `jarsigner` from the command line.

Deployment using The Eclipse



Separated Processes

- Each application for the android platform is executed within a separated process. Each process has a unique and a permanent ID assigned when the application is installed.
- This separation prevents each application from accessing other applications directly.

Separated Processes

- Applications can still share information with each other by using predefined mechanisms such as content providers, services and starting new activities from within the running one.

Declarative Permission Model

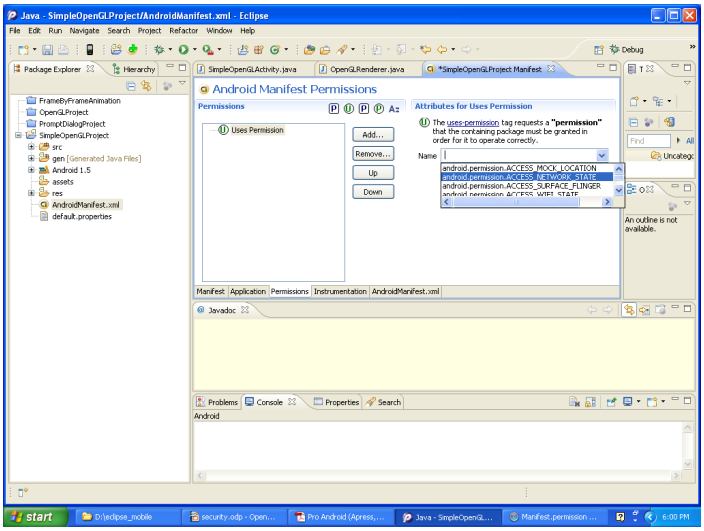
- The android platform implements a declarative permission model that protects sensitive features, such as the contacts list, the ability to send and receive SMS, the ability to make a phone call etc.
- In order to use any of these features and resources we must add the required (one or more) permissions to the `AndroidManifest.xml` file.

Declarative Permission Model

- When installing an application the android platform either grants or denies the requested permissions based on the signature of the .apk file and/or the user's settings and feedback.
- Each permission is defined as a final static integer variable within the `android.Manifest.permission` final static inner class.

You can find a detailed list of the available permissions at
http://developer.android.com/reference/android/Manifest.permission.html

Declarative Permission Model



Declarative Permission Model