Lab 8: Automated Patching & Analysis

Overview

In this lab, we are going to learn the automated analysis & patching of Android Apps it is important to have the skills to write the scripts that speed up the penetration testing , I will provide a script containing the important functionality, it will enumerate all classes and methods, and if you understand how it works, then you can use it directly or modify it based on your needs, there are two operations in this script which are dump and patch

Steps

1. In the terminal navigate to labs > automated_patching folder
    - you will find APk File which is hack.apk you can use it for this lab or open the Hack project in Android Studio and switch to the automated_patching branch
      `git checkout automated_patching`
      and build a new release app

2. Execute the script with dump operation
    - the script requires the package name of the app, and now the variable packageName is set to com.afif.hack so you don't have to change it for the hack app, but if you try another app then you have to replace it , you can extract the package name from the APK directly using aapt command
        - **for Windows**
            - you will find it in the path C:\Users\YourUser\AppData\Local\Android\Sdk\build-tools\versionNum\
              `aapt dump badging pathToFile.apk | findstr package`
        - **for Linux**
            - you will find it in the path /home/YourUser/Android/Sdk/build-tools/versionNum/
              `aapt dump badging pathToFile.apk | grep package`
    - run the script and pass the two arguments, dump operation, and the APK file
      `python patch_apk.py dump hack.apk`
    - when you run it, it will decompile the app and then enumerate the classes and the methods and print them in the console
    - this feature will give you a great overview of the app functionality

3. Execute the script with patch operation

    - now pass the operation name patch and the APK file
      `python patch_apk.py patch hack.apk`
    - this operation will decompile the app and then will patch the manifest debuggable attribute and extractNativeLibs attribute with value true after that in the smali files, it will patch each method with a log message, the log message contains the class name and the method name
    - check the patch effect in the manifest file, also in the smali files
    - now the app is modified so when you build and run it, each method will print its name and its class name when it executed
    - this logger, will enable you to track the app execution flow so you can know when these methods are executed

4. Build the app using apktool

    - `apktool b hack`
    - a new APK file will be generated in the hack > dist folder
    - so rename it to patched_hack.apk, and move it to the automated_patching folder

5. Sign the new patched_hack.apk

    - **for Windows**
        - `apksigner sign --v2-signing-enabled --ks ..\sign\certificate.jks --ks-key-alias hack --ks-pass pass:Test1234 --key-pass pass:Test1234 patched_hack.apk`
    - **for Linux**
        - `apksigner sign --v2-signing-enabled --ks ../sign/certificate.jks --ks-key-alias hack --ks-pass pass:Test1234 --key-pass pass:Test1234 patched_hack.apk`

6. Install the APK

    - delete the previous version of the hack app and install this version
      `adb install patched_hack.apk`
    - run the app and open Logcat, you will see the app in the debuggable processes because it becomes debuggable also each method will print its name when it is executed, that's great