DFRWS2014 Forensic Challenge

Mobile Malware Analysis
Mobile Malware Analysis

- **Challenge:**
  - Demonstrate effective methods and develop open source tools for analyzing mobile malware

- **Desired capabilities**
  - Extracting metadata and components
  - Decompiling mobile malware
  - Decoding data associated with mobile malware
  - Behavioral scanners running on localhost
  - Identifying potentially malicious functions

- **Two submission categories**
  - Practitioner Submissions
  - Researcher/Developer Submissions
Submissions: Android, Android, Android, Android

- **Practitioner**
  - Singapore: Darell Tan, Sufatrio, Tong-Wei Chua
    Agency for Science, Technology and Research, Institute for
    Infocomm Research

- **Researcher/Developer**
  - South Korea: Nikolay Akatyev, Hojun Son
  
  - United States: Zhaoheng Yang and Ibrahim Baggili
    University of New Haven, Cyber Forensics Research and
    Education Group
  
  - South Korea: Dongwoo Kim and Wootak Jung
    Chungnam National University, Information Security Lab
Assessment of Submissions

Main criteria for judging:

- Novelty
- Practicality
- Amount of extracted information
- Usefulness of extracted information
- Development effort (R&D category only)

Judges: Eoghan Casey, Justin Grover, Mark Guido, Jared Ondricek
Assist: Morgan Marquis-Boire (Citizenlab) & Sebastian Porst (Google)
Practitioner

- APKTool
- VirusTotal androapkinfo.py
- Androguard API visualization
- FlowDroid, SuSi
- Dynamic: ApkAnalyzer, ApkInspector, logcat, strace, tcpdump
- Decompile: Dex2jar, Procyon
Eclipse Plug-in (FlowDroid, APK2Java)

https://github.com/SeoulTech/Manal/wiki/Getting-started
Reverse Engineered source code automatically imported into an Eclipse project!

New Suspect Analysis Perspective worked great

The judges were unable to populate the Leak Sources pane

The judges were unable to populate the Suspect List pane
Android Malware INvestigation Tool (MINT)

- Decompile Android Malware using APKTool
- Provide GUI to extracted data
- Calculate “danger scores”
  - Uses app permissions and machine learning association rules
  - Generated permission association rules using Weka
  - Danger from Association Rule Score (DFA)
  - Danger from Permissions (DFP)
MINT – Backdoor
MINT – Banking Malware
Dump-Code

Problem Statement:
- Malware using encryption, dynamic class loading, anti-tamper and anti-debugging, making it more difficult and time-consuming to reach the main executable code with existing tools and methods

Solution: Extract executable code from memory
- Insert dump-code into the dvmDexFileOpenFromFd function
- Run on Android platform 4.0.1
- Replace /dalvik/vm/DvmDex.cpp with Dump-Code
- Run malware sample on the modified platform
- Optimized executable file (Odex) saved to SDCard
- Disassemble and assemble using Baksmali & Smali tools
- Decompile as usual, e.g., dex2jar, Apktool
Dump-Code Process

- Dump odex file loaded by main process of malicious app
  - Touching child processes not desirable/effective
Odex Extraction from Memory

- Processes Used in Comparison
  - Typical Static Analysis
    - Unzipped APK
    - Ran dex2jar
    - Opened in jd-gui
  - Odex Extraction Method
    - Installed & executed malware in emulator
    - Android OS dumped odex file to sdcard
    - Ran baksmali & smali
    - Ran dex2jar
    - Opened in jd-gui
Provided Sample: Result Sizes Different

```
jgrover@malware:~/Downloads/Bangcle
jgrover@malware:~/Downloads/Bangcle$ ls -l
total 5820
-rw-rw-r-- 1 jgrover jgrover 10306 Aug 3 03:34 adobe_classes_dex2jar.jar
-rw-rw-r-- 1 jgrover jgrover 1940381 Aug 3 03:36 adobe_classes_dex2jar_odex.jar
drwxrwxr-x 8 jgrover jgrover 4096 Aug 3 03:47 AdobeFlash
-rw-rw-r-- 1 jgrover jgrover 3999682 Aug 3 03:33 AdobeFlash.apk
jgrover@malware:~/Downloads/Bangcle$
```
New Sample: Result Sizes Similar

jgrover@malware:~/Downloads/GreatAppLuck$ ls -l
total 13340
drwxrwxr-x 6 jgrover jgrover  4096 Aug  3  03:44 GreatAppLuck
-rw-rw-r-- 1 jgrover jgrover 10497651 Jul 29  07:27 GreatAppLuck.apk
-rw-rw-r-- 1 jgrover jgrover 1574192 Aug  3  02:31 GreatAppLuck_classes_dex2jar.jar
-rw-rw-r-- 1 jgrover jgrover 1578585 Aug  3  03:21 GreatAppLuck_classes_dex2jar_odex.jar
Provided Sample: Typical Static Method

Missing Files
Provided Sample: Odex Method

```
package com.madabai.junit;

import android.test.AndroidTestCase;

class HttpClientUtilsTest extends AndroidTestCase {
    public void testSaveBank()
    {
        HashMap localHashMap = new HashMap();
        localHashMap.put("bankModel.bankM2", "2013-12-06 02:26:18");
        localHashMap.put("bankModel.bankM3", "name");
        localHashMap.put("bankModel.bankM4", "946494-4976846");
        localHashMap.put("bankModel.bankM5", "+8613904484217");
        localHashMap.put("bankModel.bankM6", "名字111sss");
        localHashMap.put("bankModel.bankM7", "49704940464064");
        localHashMap.put("bankModel.bankM8", "4944");
        localHashMap.put("bankModel.bankM9", "bankM9");
        localHashMap.put("bankModel.bankM10", "bankM10");
        localHashMap.put("bankModel.bankM11", "4946");
        localHashMap.put("bankModel.bankM12", "58.245.112.25");
        String str1 = "";
        int i = 1;
        while (true)
        {
            if (i > 35)
            {
                localHashMap.put("bankModel.bankM13", str1.substring(0, i) + str1.substring( str1.length() - i , str1.length()));
            }
        }
    }
```