Mobile Security: Android Malware Analysis

Saurabh Kumar
Senior Research Scholar
IIT Kanpur
Date: 01/06/2022
MOTIVATION
Why Mobile Security?

- User activity
- Valuable data
- Always on
- Multiple Attack Surfaces
Why Android?

1. Almost completely open source

2. Global smartphone market share

<table>
<thead>
<tr>
<th>Period</th>
<th>Android</th>
<th>iOS</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020</td>
<td>84.1%</td>
<td>15.9%</td>
<td>0%</td>
</tr>
<tr>
<td>2021</td>
<td>83.8%</td>
<td>16.2%</td>
<td>0%</td>
</tr>
<tr>
<td>2022</td>
<td>84.1%</td>
<td>15.9%</td>
<td>0%</td>
</tr>
<tr>
<td>2023</td>
<td>84.4%</td>
<td>15.6%</td>
<td>0%</td>
</tr>
<tr>
<td>2024</td>
<td>84.7%</td>
<td>15.3%</td>
<td>0%</td>
</tr>
<tr>
<td>2025</td>
<td>84.9%</td>
<td>15.1%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Source: International Data Corporation (IDC), October 2021
Actors in the Android Ecosystem

- Tool chain (Cordova, App generator, …)
- Play Store
- Sideloading
- Alternate Store
- Ad Libs
- Third Party app
- Application Framework
  - Native libs (C / C++)
  - Android Runtime (Dalvik / ART)
  - Linux Kernel (modified)
- Online services
- Advertisement networks

Use Tools

App Developer

Publish app

Configure

Administrators

Platform vendors
## Security Impact of an Actor Over Others

<table>
<thead>
<tr>
<th>Actor</th>
<th>OS Developer</th>
<th>H/W Vendor</th>
<th>Library Providers</th>
<th>S/W Developer</th>
<th>Toolchain Providers</th>
<th>S/W Publisher</th>
<th>S/W Market</th>
<th>End User</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Developer</td>
<td>--</td>
<td>Partial</td>
<td>Full</td>
<td>Full</td>
<td>Partial</td>
<td>Full</td>
<td>Full</td>
<td>Full</td>
</tr>
<tr>
<td>H/W Vendor</td>
<td>None</td>
<td>--</td>
<td>Full</td>
<td>Full</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Full</td>
</tr>
<tr>
<td>Library Provider</td>
<td>None</td>
<td>None</td>
<td>--</td>
<td>Full</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Full</td>
</tr>
<tr>
<td>S/W Developer</td>
<td>None</td>
<td>None</td>
<td>Partial</td>
<td>--</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Full</td>
</tr>
<tr>
<td>Toolchain Providers</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Full</td>
<td>--</td>
<td>None</td>
<td>None</td>
<td>Partial</td>
</tr>
<tr>
<td>S/W Publisher</td>
<td>None</td>
<td>None</td>
<td>Partial</td>
<td>Partial</td>
<td>None</td>
<td>--</td>
<td>Partial</td>
<td>Full</td>
</tr>
<tr>
<td>S/W Market</td>
<td>None</td>
<td>None</td>
<td>Partial</td>
<td>Partial</td>
<td>None</td>
<td>None</td>
<td>--</td>
<td>Full</td>
</tr>
<tr>
<td>End User</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>--</td>
</tr>
</tbody>
</table>
Where to Improve Security?

- Tool Chain Provider
  - Uses tool chains
  - Includes libraries

- App developer
  - Publish App

- Markets
  - Install

- Installed Apps
  - Android API
  - Linux API

- Middleware
  - Retrofit Android’s Security

- Linux Kernel

- Android Platform

- Web Services
  - Use via internet

- 3rd party libraries
  - Use via internet

- Ad & Analytics Network

- Useable Security
  - Application Vetting

- Inline reference monitors
  - Retrofit Android’s Security
Motivation: Summary

- **Feature-rich smartphones** and **appification** have induced security research on various new aspects
- Android’s **open-source** nature has made Android very attractive to security researchers
- Android’s **market share** has made Android the **#1 target** for malware authors and cyber criminals
- Various actors in the **ecosystem** with (strong) influence on **security and privacy**
ANDROID BACKGROUND
Android Software Stack

- Default apps
  - Contacts
  - SMS

- Third party apps
  - paytm
  - linkedin

- Application Framework

- Native libs (C / C++)
- Android Runtime (Dalvik / ART)

- Linux Kernel (modified)
Application Packages (APK)

- APK is simply a packaging format like **JAR**, **ZIP** and **TAR**
- Component of Application
  - Activity
  - Content Provider
  - Services
  - Broadcast Receiver
- Native Code (C/C++ shared libraries)
- Resources
- META-INFO
- Application Manifest
ANDROID SECURITY ARCHITECTURE

• Package Integrity
• Sandboxing
• Permission and Least Privilege
Package Integrity: Package Manifest

- Created with jarsigner
- META-INF
  - Manifest.mf, Cert.sf, Cert.{RSA,DSA}

File

**Manifest.mf**
- Manifest-Version: 1.0
- Built-By: Generated-by-ADT
- Created-By: Android Gradle 3.0.1
- Name: res/mipmap-hdpi-v4/ic_launcher.png
- SHA1-Digest: 2zkIQdtvIXqEHSTVOVuwBQ18als=

**Cert.sf**
- Signature-Version: 1.0
- Created-By: 1.0 (Android)
- SHA1-Digest-Manifest: h9xNIIN3bQiTJ8RQyPUWBojRKD8=
- X-Android-APK-Signed: 2
- Name: res/mipmap-hdpi-v4/ic_launcher.png
- SHA1-Digest: L8RpX5x8pChJbcqml+hMt9D9CQ=

Certificate  Cert.sf signature

CERT.{RSA,DSA}
Verifying of package manifest

Chain of trust:

PKI

Package certificate in Cert.{RSA,DSA}

Manifest.mf

Cert.sf

Files
ANDROID SECURITY ARCHITECTURE

- Package Integrity
- Sandboxing
- Permission and Least Privilege
Sandboxing

- The application sandbox **specifies** which system **resources** the application is allowed to access.
- An **attacker** can only perform **actions** defined in the sandbox.
Application Isolation by Sandboxing

- Each Application is **isolated** in its own **environment**
  - Applications can access only its own **resources**
  - Access to **sensitive resources** depends on the application’s **rights**
- **Sandboxing** is enforced by **Linux**
Application sandbox

- Isolation: Each installed App has a separate user ID

Diagram:
- App Code (Classes.dex)
- Core libraries
- JNI
- Native Code (*.so)
- syscalls
- Kernel
- UID A
Application sandbox

- **Isolation**: Each installed App has a separate user ID
  - Each App lives in its own sandbox
ANDROID SECURITY ARCHITECTURE

- Package Integrity
- Sandboxing
- Permission and Least Privilege
Android Permission System

- **Access rights** in Android’s application framework
  - Permissions are required to *gain* access to
    - System interfaces (Internet, send SMS, etc.)
    - System resources (logs, battery, etc.)
    - Sensitive data (SMS, contacts, etc.)
  - Currently more than 140 default permissions defined in Android

- Permissions are *assigned* to sandbox

- Application developers can also *define* their *own* permissions
Android Permission: Example

App B (has permission P)

App C (has not permission P)

App A (Permission P) Service
Permissions’ Protection Level

- Normal
- Dangerous
- Signature
- SignatureOrSystem
Dynamic Permissions (≥ Android 6.0)

- App developers must **check** if their apps hold required **dangerous** permission, otherwise request them at runtime.
- User can **grant** permissions at runtime and also **revoke** once granted permissions again.

Is the requested permission reasonable?

Should I adjust some permissions?
ANDROID VULNERABILITIES

- Architecture Based
- Software Based
- Hardware Based
Vulnerability Classification

- Android Vulnerability
  - Architecture
  - Software
    - Operating System
  - Hardware
    - Original Equipment Manufacturer (OEM)
    - Third Party App
ANDROID VULNERABILITIES

- Architecture Based
- Software Based
- Hardware Based
Application-Level Privilege Escalation Attacks

Malicious App + Confused Deputy App = Confused Deputy Attack

Malicious App + Malicious App = Collusion Attack
Collusion Attack

- Variants:
  - Apps communicate directly
  - Apps communicate via covert channels in Android

Malicious apps collude in order to merge their respective permissions
ANDROID VULNERABILITIES

- Architecture Based
- Software Based
- Hardware Based
Dirty COW

- Existed in the Linux Kernel for **9 years**
- A **local** Privilege Escalation Vulnerability
- Exploits a race condition in the implementation of the **copy-on-write** mechanism
- Turns a **read-only** mapping of a file into a writable mapping

Android malware ZNIU exploits **DirtyCOW vulnerability**

Media Projection Service Issue

Vulnerabilities

Android issue allows attackers to capture screen and record audio on 77% of all devices

Over-privileged Apps

- Many apps request permissions that their **functionality** does not require
- Suspected root cause: API **documentation/naming** convention
  - Solution: API Permissions Maps
    - Can be integrated into lint tools
A privileged app is fooled into misusing its privileges on behalf of another (malicious) unprivileged app.

Example:
- Unauthorized phone calls
- Various confused deputies in system apps
Confused Deputy Introduce by OEMs

- Several **confused deputies** found in Samsung devices’ **firmware**
  
  - One deputy running with system privileges provided **root shell service** to any app

![Diagram showing various functionalities related to a backdoor]

- Access to mail account
- Camera
- Microphone
- GPS Location
- Internet
- Contacts
- SMS & MMS
- Access to SD card
ANDROID VULNERABILITIES

- Architecture Based
- Software Based
- Hardware Based
Broadcom Wi-Fi SoC Flaw

**BIZ & IT —**

Android devices can be fatally hacked by malicious Wi-Fi networks

Broadcom chips allow rogue Wi-Fi signals to execute code of attacker's choosing.

**DAN GOODIN - 4/6/2017, 1:16 AM**

ADVANCED THREAT
## Dynamic Code Loading: Techniques and Risks

<table>
<thead>
<tr>
<th>Techniques</th>
<th>API</th>
<th>Risk</th>
<th>Code Injection Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class loader</td>
<td>DexClassLoader</td>
<td>No checking of <strong>Integrity</strong> or <strong>Authenticity</strong></td>
<td>Attacker can control loaded code</td>
</tr>
<tr>
<td>Package Context</td>
<td>createPackageContext</td>
<td><strong>No verification</strong>: App from same developer</td>
<td>Attacker can install app</td>
</tr>
<tr>
<td>Native Code</td>
<td>Java Native Interface</td>
<td><strong>No restrictions</strong> on location</td>
<td>Manipulate the native code to inject code</td>
</tr>
</tbody>
</table>
Android Instant App

Insta...thing

March 07, 2017

Phishing Attacks Against Android Instant Apps

by Camilo Reyes

Android dev...

By Paul Miller


Source: https://blog.jscrambler.com/phishing-attacks-against-android-instant-apps/
MALWARE ANALYSIS

- Analysis Techniques and Their Limitations
Why Malware Analysis?

This data-stealing Android malware infiltrated the Google Play Store, infecting users in 196 countries

First Android Clipboard Hijacking Crypto Malware Found On Google Play Store

Android banking malware hitting more users than ever


Fake banking apps could be more effective than banking Trojans

Source: https://thehackernews.com/2019/02/beauty-camera-android-apps.html
In every 10 seconds, a new Android malware is born.

Source: AV-TEST malware statistics report Jan 2022
Analysis Techniques

- Static
- Hybrid
- Dynamic
Malware Analysis

- Many work has been proposed
- Deployed on
  - Server
  - Real Device
- Offline analysis can be bypassed
- On a real device, existing offline method cannot be used
  - High resources requirement
## Analysis Techniques Challenges

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static</strong></td>
<td>• Dynamically Loaded Code</td>
</tr>
<tr>
<td></td>
<td>• Crypto API</td>
</tr>
<tr>
<td></td>
<td>• Java Reflection</td>
</tr>
<tr>
<td></td>
<td>• False positive (permission based)</td>
</tr>
<tr>
<td></td>
<td>• Network based activity</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>• False positive (Anomaly based)</td>
</tr>
<tr>
<td></td>
<td>• Code Coverage</td>
</tr>
<tr>
<td></td>
<td>• 20 times slowdown system if used in real device</td>
</tr>
<tr>
<td><strong>Hybrid</strong></td>
<td>• Data Dependency ACG</td>
</tr>
<tr>
<td></td>
<td>• Logic based triggers</td>
</tr>
<tr>
<td></td>
<td>• Obfuscation and reflection</td>
</tr>
</tbody>
</table>
# Malware Analysis Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Method</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aurasium [Xu et al. 2012]</td>
<td>Dynamic – detect API misuse</td>
<td>Native code, java refl.,</td>
</tr>
<tr>
<td>SmartDroid [Zheng et al. 2012]</td>
<td>Statically find activity path + dynamic to find triggers</td>
<td>Native code</td>
</tr>
<tr>
<td>Jin et al. [2013]</td>
<td>Dynamic (SDN traffic monitoring)</td>
<td>Encrypted traffic</td>
</tr>
<tr>
<td>SAAF [Hoffmann et al. 2013]</td>
<td>Static (smali) auto and optional manual</td>
<td>Reflection, native code</td>
</tr>
<tr>
<td>RiskMon [Jing et al. 2014]</td>
<td>Dynamic + machine learning + API monitor</td>
<td>Colluding apps</td>
</tr>
<tr>
<td>Drebin [Arp et al. 2014]</td>
<td>Static (features from Manifest + dex code) + machine learning</td>
<td>Colluding apps, Obfuscation, Dynamic code, Native code</td>
</tr>
<tr>
<td>DroidSafe [Gordan et al. 2015]</td>
<td>Static information flow + hooks + calls that start activity</td>
<td>Cross-layer, Emulation-detection</td>
</tr>
</tbody>
</table>
## Malware Analysis Frameworks Cont..

<table>
<thead>
<tr>
<th>Framework</th>
<th>Method</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wang et al. [2016]</td>
<td>Static + machine learning + permissions + APIs</td>
<td>Dynamic code loading, Native code, obfuscation</td>
</tr>
<tr>
<td>DroidSeive [Guillermo et al. 2017]</td>
<td>Static + machine learning + multiple location features</td>
<td>Dynamic code loading</td>
</tr>
<tr>
<td>IntelliAV:[Ahmadi, et al. 2017]</td>
<td>Static + machine learning + API Call, Components statistics</td>
<td>Dynamic code loading, Native code, obfuscation</td>
</tr>
<tr>
<td>TinyDroid [Chen et al. 2018]</td>
<td>Static + Opcode + machine learning</td>
<td>Dynamic code loading, Native code</td>
</tr>
<tr>
<td>Fatima et al. [2019]</td>
<td>Static + machine learning + permissions</td>
<td>Dynamic code loading, Native code, repackaging attack</td>
</tr>
</tbody>
</table>
Analysis Techniques used in Different Area

- Malware: 50 Static, 36 Dynamic, 14 Hybrid
- Grayware: 50 Static, 39 Dynamic, 11 Hybrid
- Vulnerable: 58 Static, 27 Dynamic, 15 Hybrid
CHALLENGES AND FUTURE DIRECTIONS
Challenges in Android Security

- Android Instant Apps
- Device Fragmentation
- Cheap Devices
- Colluding Apps
- Platform Sensing Malware
Future Research Direction

- Basic simple app Analysis to analyze whole system
- Consider dynamically loaded code that is not bundled with installed packages
- Analyze code of different forms and from different languages
  - Native (C/C++), Obfuscated Code
- Colluding malware analysis
- Stealthy Dynamic Analyzer
Thank You