Mobile Security: Android Malware Analysis

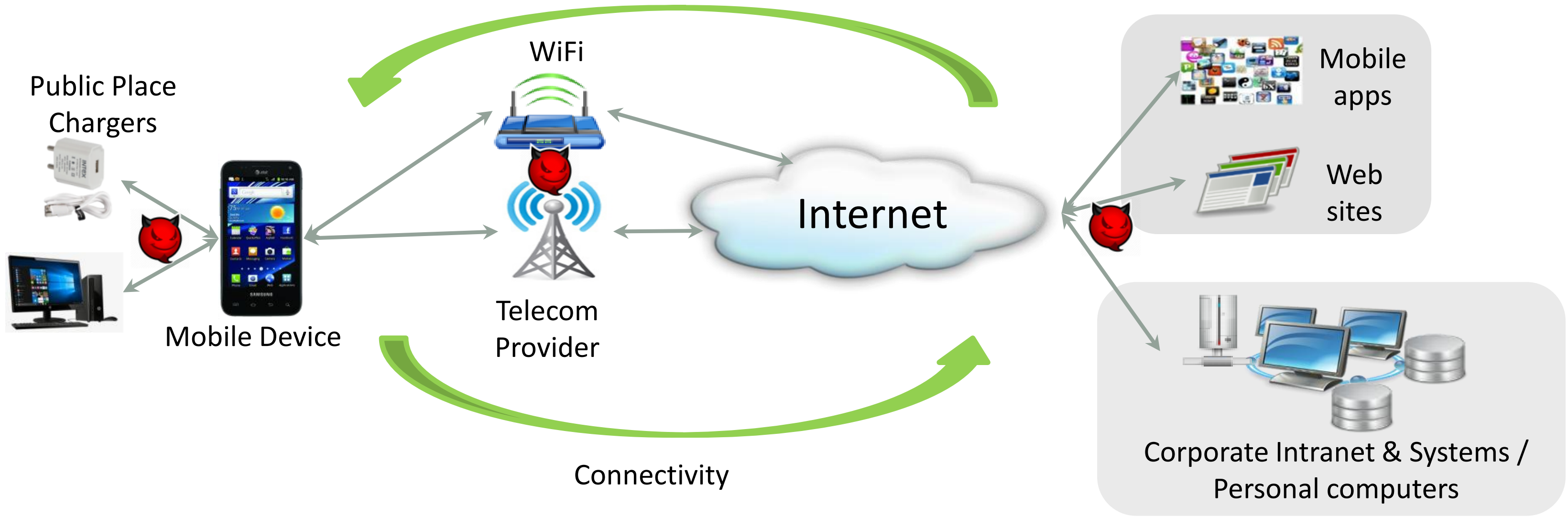
Saurabh Kumar
Senior Research Scholar
IIT Kanpur

Date: 01/06/2022

MOTIVATION

Why Mobile Security?

- ❑User activity
- ❑Valuable data
- ❑Always on
- ❑Multiple Attack Surfaces



Why Android?

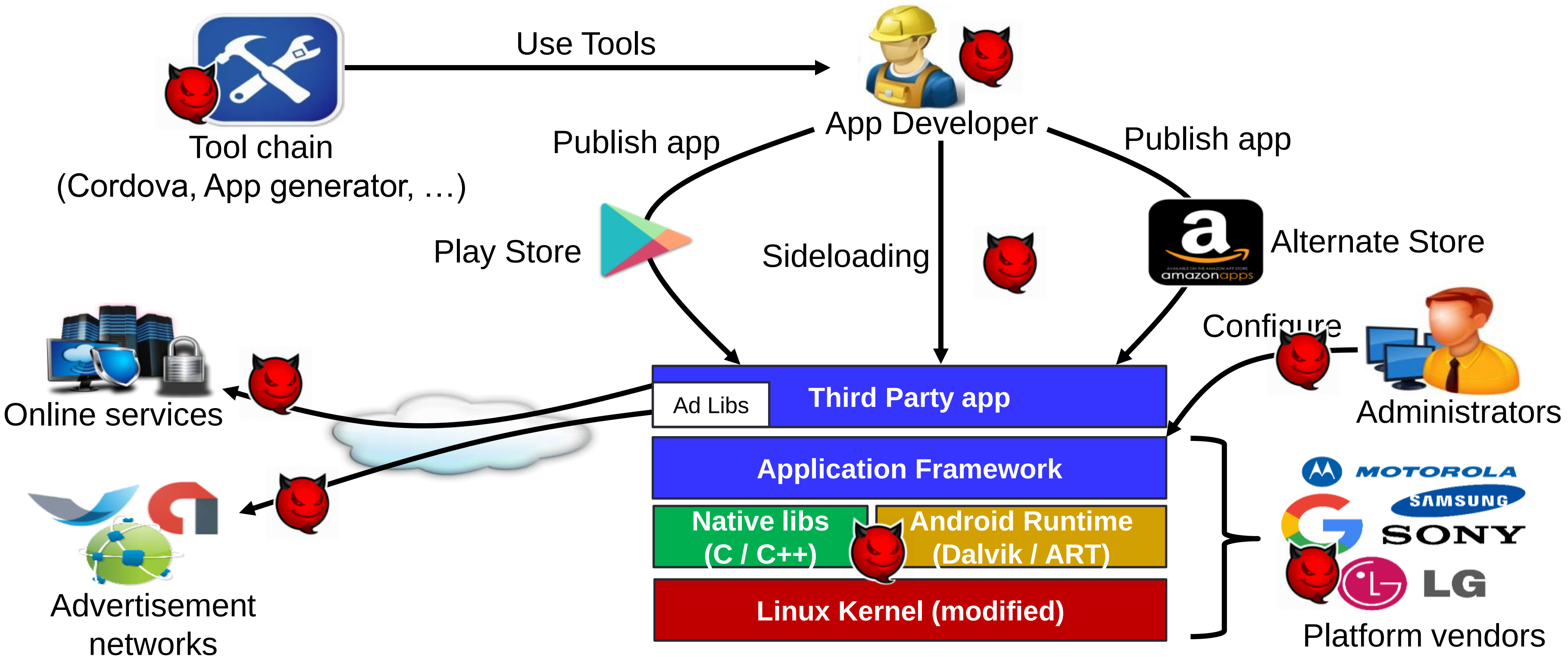
1. Almost completely open source

2. Global smartphone market share

Period	Android	iOS	Others
2020	84.1%	15.9%	0%
2021	83.8%	16.2%	0%
2022	84.1%	15.9%	0%
2023	84.4%	15.6%	0%
2024	84.7%	15.3%	0%
2025	84.9%	15.1%	0%

Source: International Data Corporation (IDC), October 2021

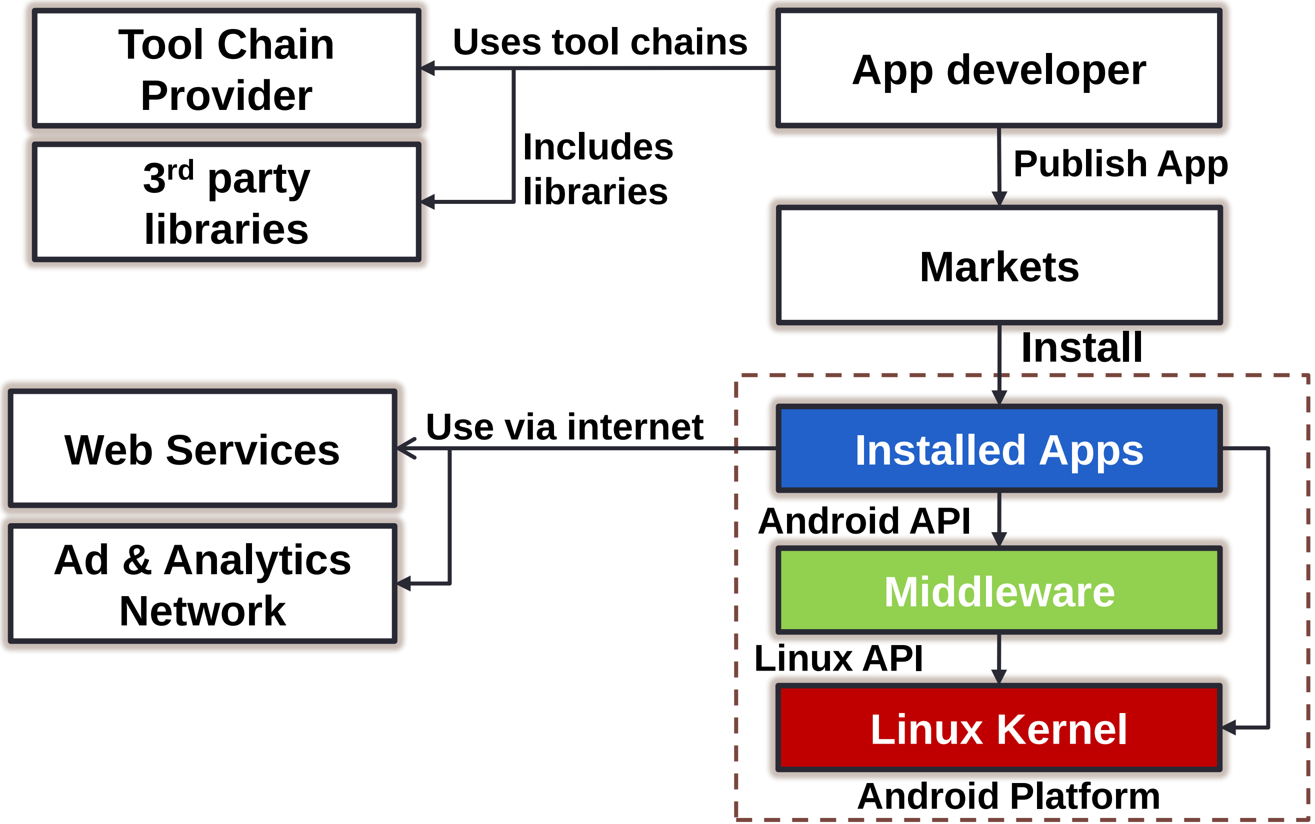
Actors in the Android Ecosystem



Security Impact of an Actor Over Others

Actor	OS Developer	H/W Vendor	Library Providers	S/W Developer	Toolchain Providers	S/W Publisher	S/W Market	End User
OS Developer	--	Partial	Full	Full	Partial	Full	Full	Full
H/W Vendor	None	--	Full	Full	None	None	None	Full
Library Provider	None	None	--	Full	None	None	None	Full
S/W Developer	None	None	Partial	--	None	None	None	Full
Toolchain Providers	None	None	None	Full	--	None	None	Partial
S/W Publisher	None	None	Partial	Partial	None	--	Partial	Full
S/W Market	None	None	Partial	Partial	None	None	--	Full
End User	None	None	None	None	None	None	None	--

Where to Improve Security?



Useable Security

Application Vetting

Inline reference monitors

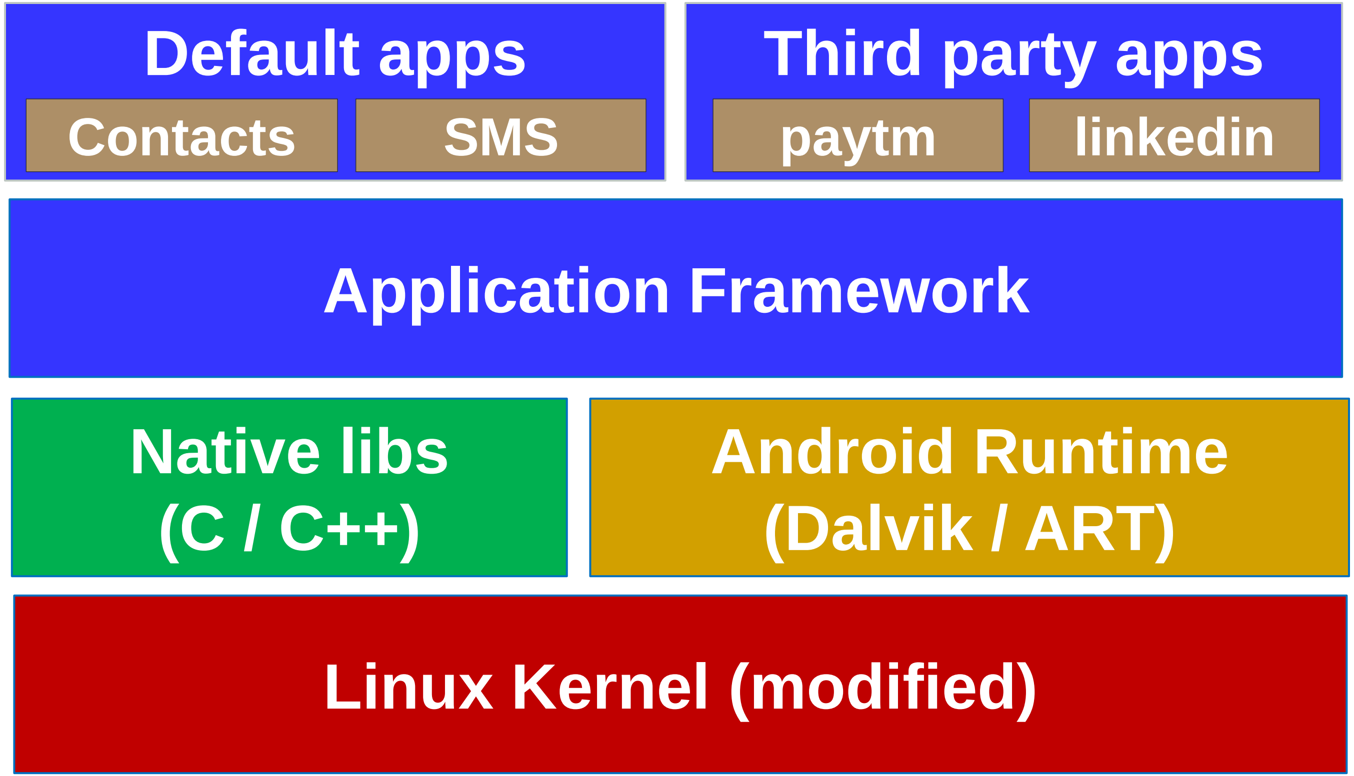
Retrofit Android’s Security

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



Application Packages (APK)

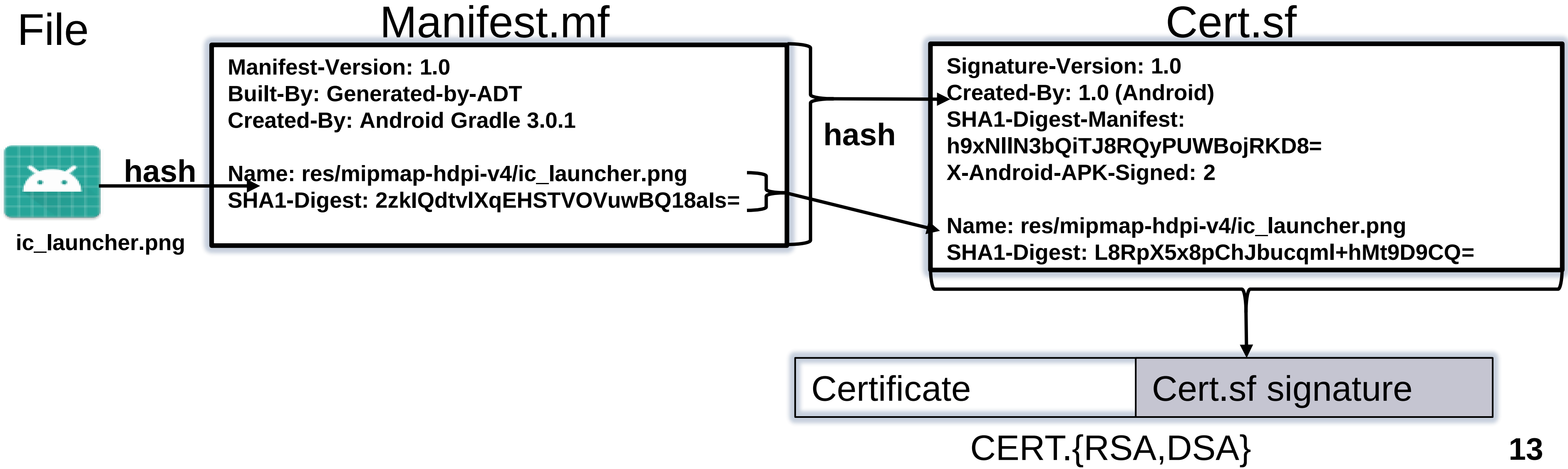
- APK is simply a packaging format like **JAR**, ZIP and TAR
- Component of Application
  - Activity
  - Content Provider
  - Services
  - Broadcast Receiver
- Native Code (C/C++ shared libraries)
- Resources
- META-INF
- Application Manifest

Android Security Architecture

- Package Integrity
- Sandboxing
- Permission and Least Privilege

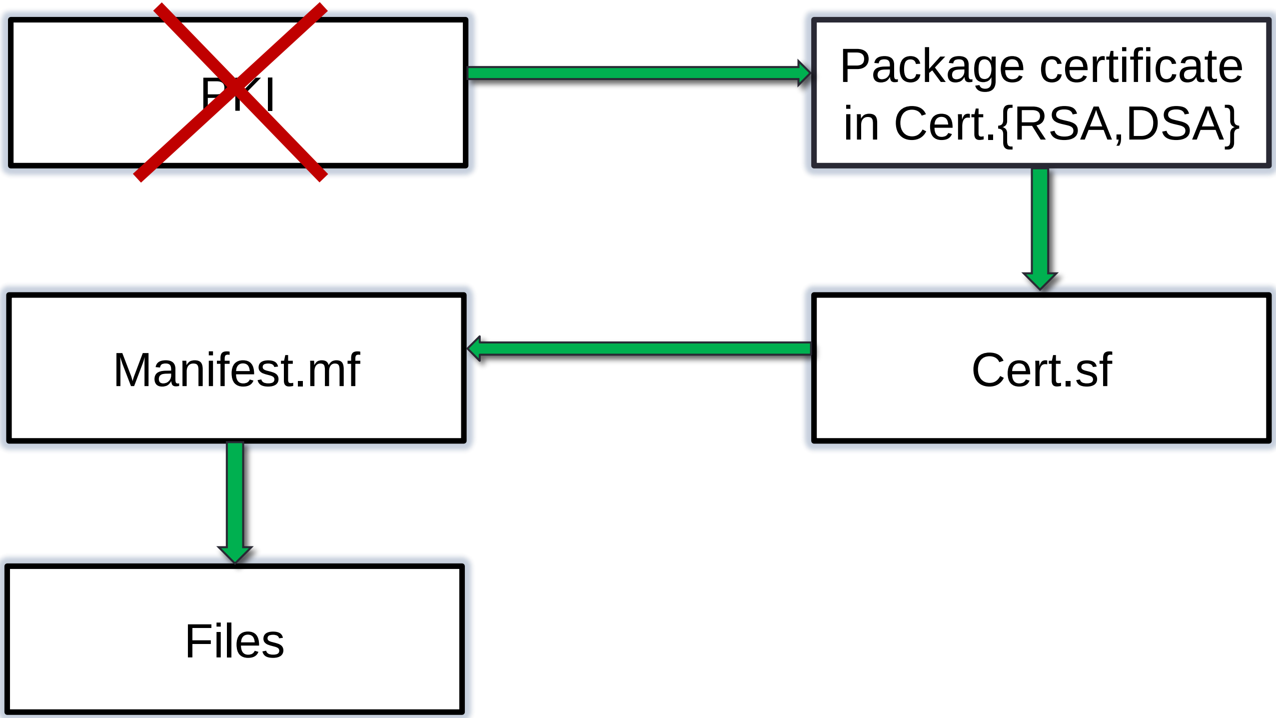
Package Integrity: Package Manifest

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



Verifying of package manifest

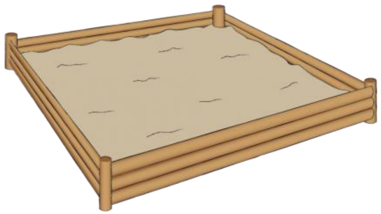
Chain of trust:



Android Security Architecture

- Package Integrity
- Sandboxing
- Permission and Least Privilege

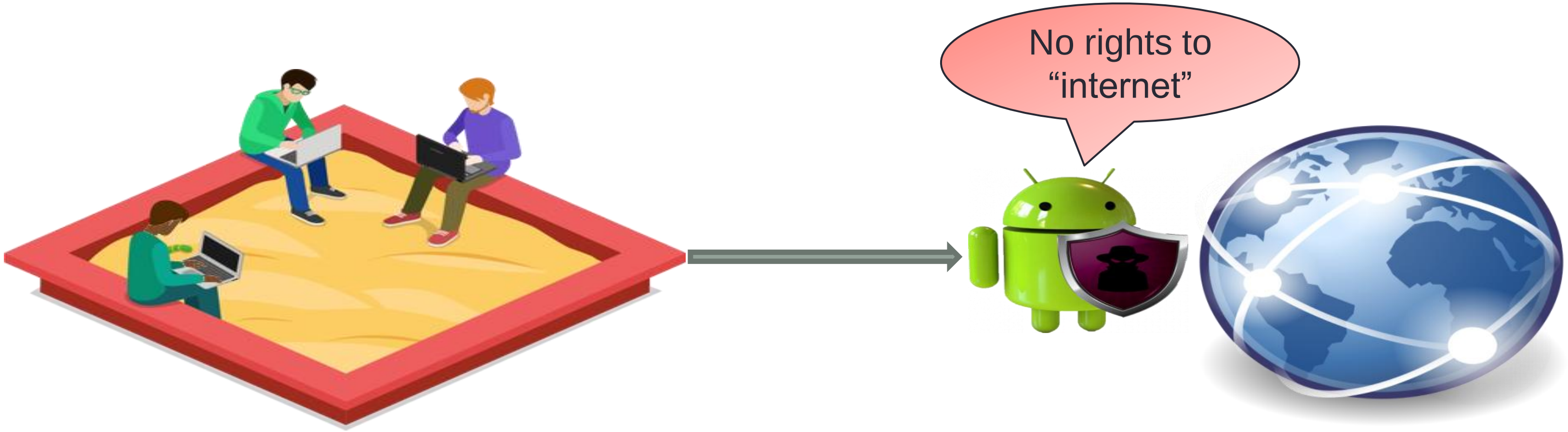
Sandboxing



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

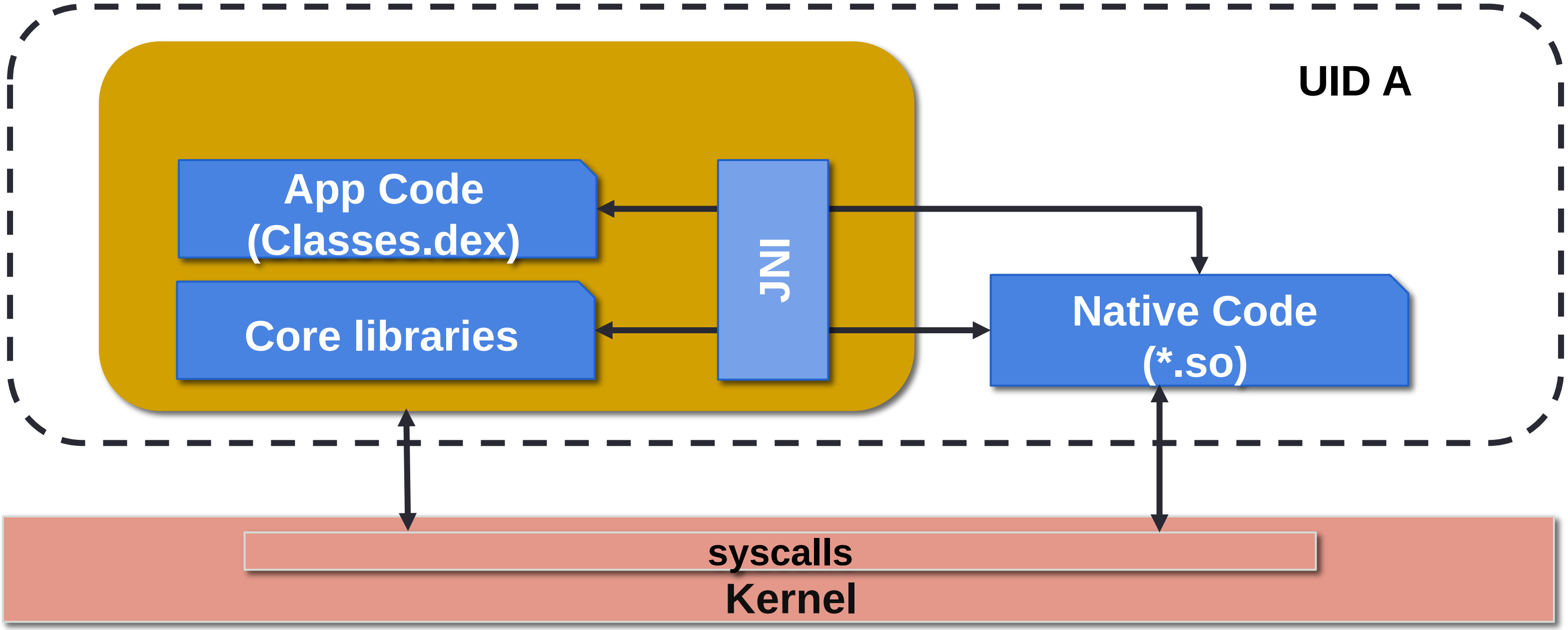
Application Isolation by Sandboxing

- 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**



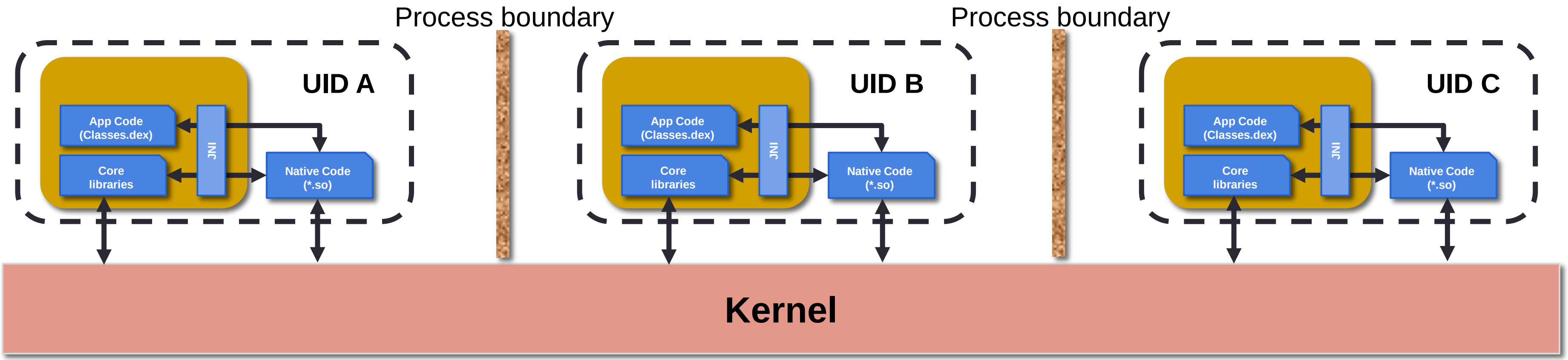
Application sandbox

- Isolation: Each installed App has a separate user ID



Application sandbox

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



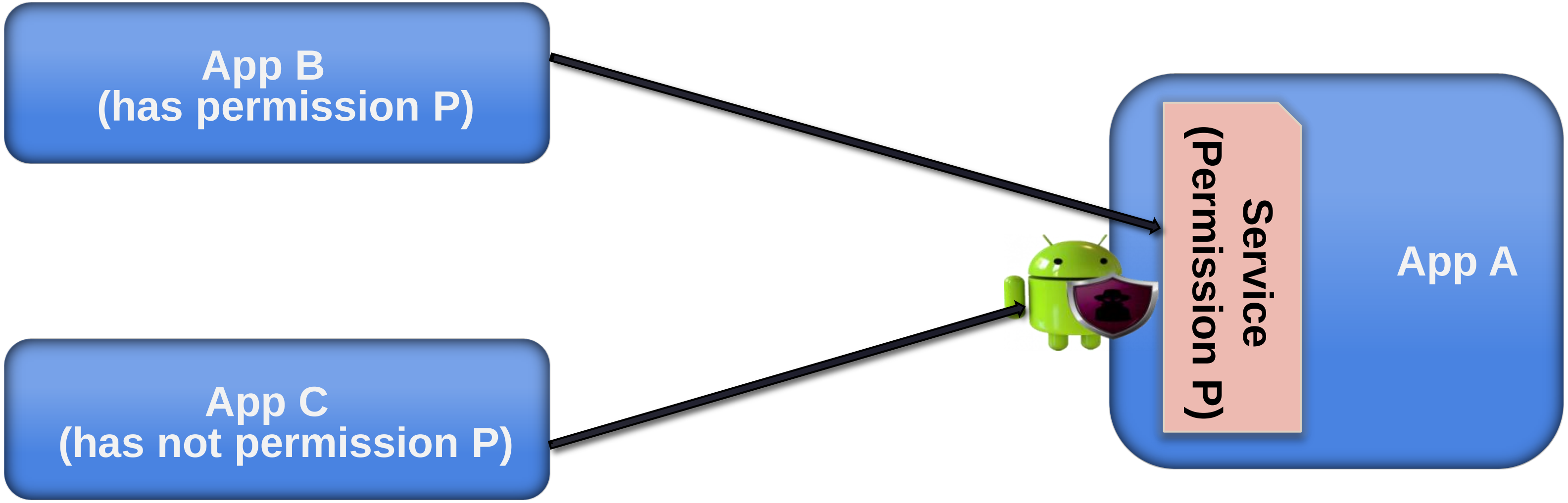
Android Security Architecture

- 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

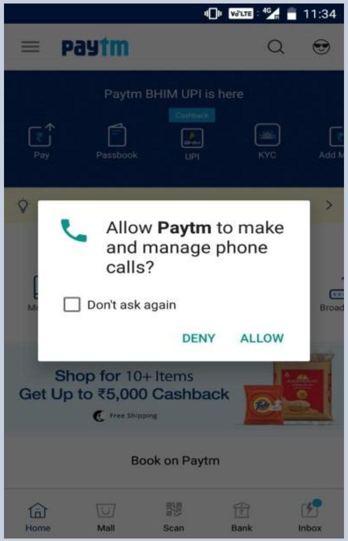


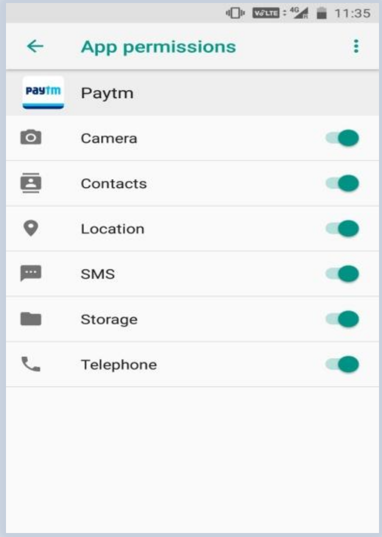
Permissions' Protection Level

- 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

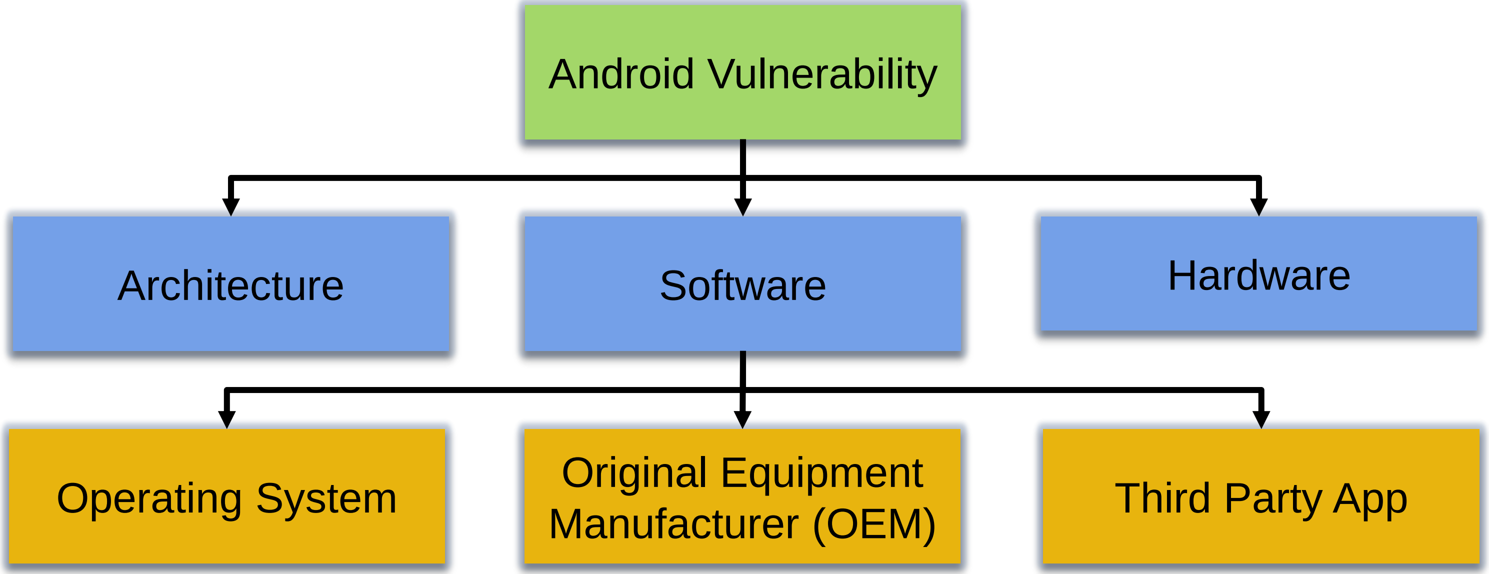




Android Vulnerabilities

- Architecture Based
- Software Based
- Hardware Based

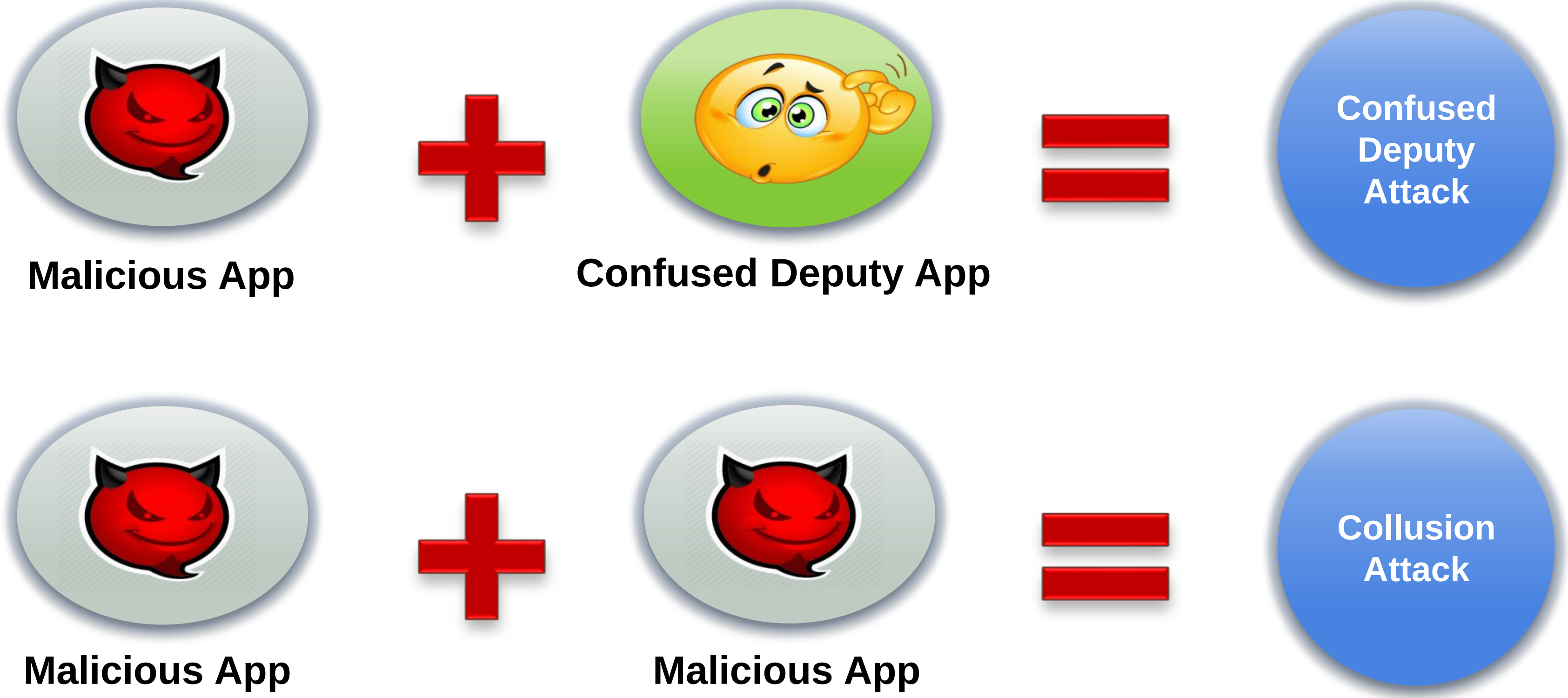
Vulnerability Classification



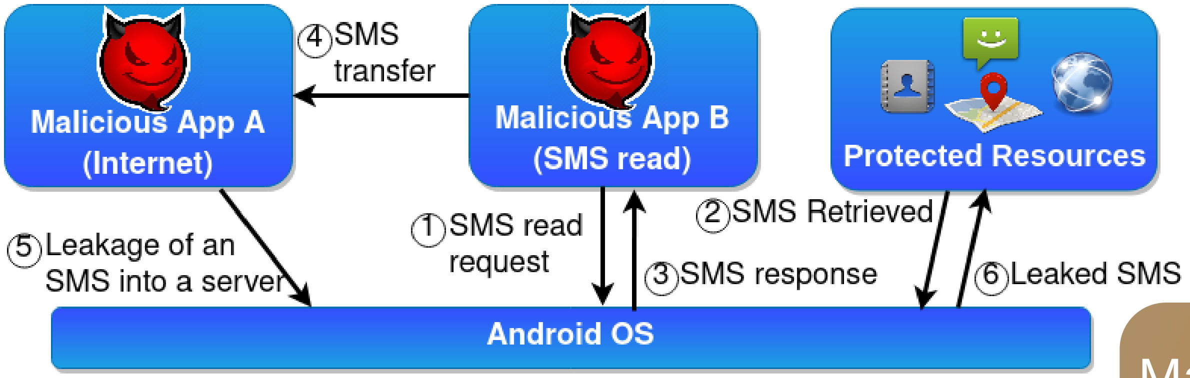
ANDROID VULNERABILITIES

- Architecture Based
- Software Based
- Hardware Based

Application-Level Privilege Escalation Attacks



Collusion Attack



Malicious apps **collude** in order to **merge** their respective **permissions**

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

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

29 SEP 2017 0

Android, Google, Malware, SophosLabs, Vulnerability

Source: https://nakedsecurity.sophos.com/2017/09/29/android-malware-zniu-exploits-dirtycow-vulnerability/

Media Projection Service Issue

Vulnerabilities

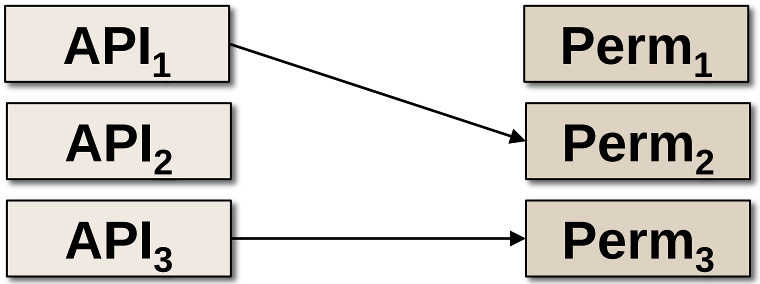
Android issue allows attackers to capture screen and record audio on 77% of all devices

November 20, 2017 Eslam Medhat 14 Views 0 Comments android, MediaProjection

Source: https://latesthackingnews.com/2017/11/20/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**



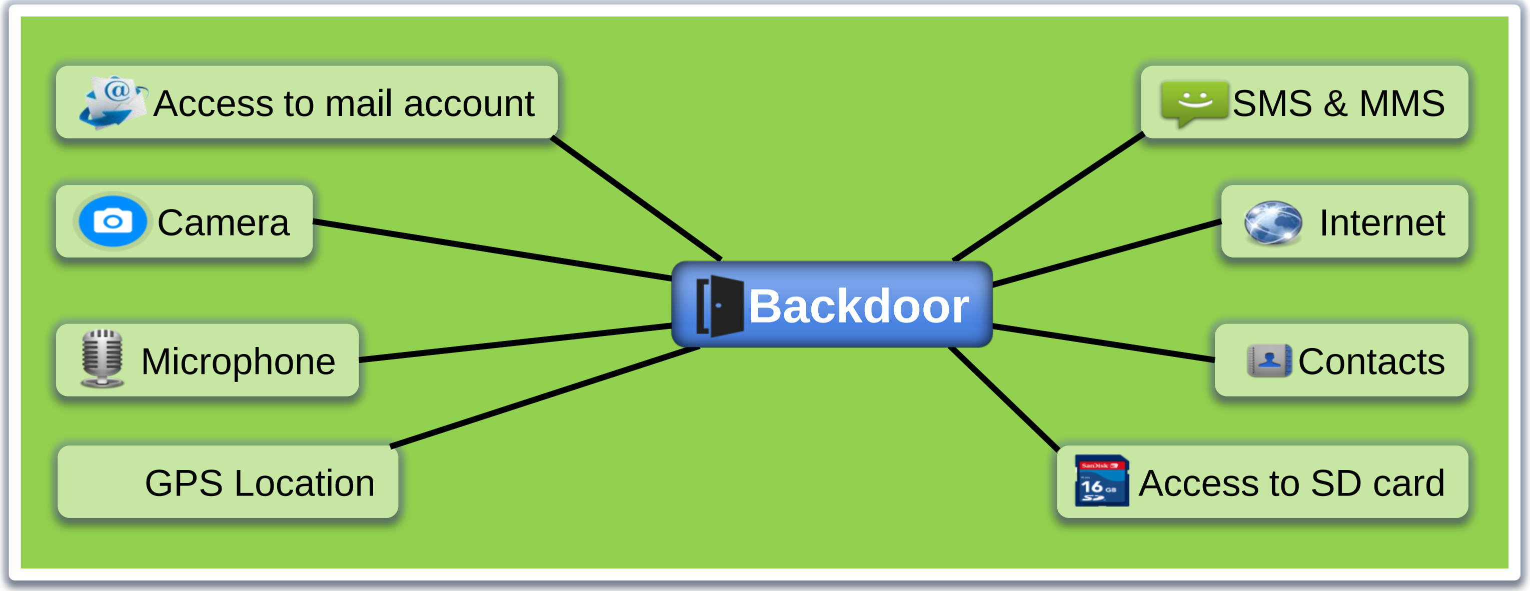
Confused Deputy Attack

- 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



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

Source: https://arstechnica.com/information-technology/2017/04/wide-range-of-android-phones-vulnerable-to-device-hijacks-over-wi-fi/

Advanced Threat

Dynamic Code Loading: Techniques and Risks

Techniques	API	Risk	Code Injection Vector
Class loader	DexClassLoader	No checking of **Integrity** or **Authenticity**	Attacker can control loaded code
Package Context	createPackageContext	**No verification**: App from same developer	Attacker can install app
Native Code	Java Native Interface	**No restrictions** on location	Manipulate the native code to inject code

Android Instant App

GOOGLE \ HANDS-O

Insta ... inating
thing

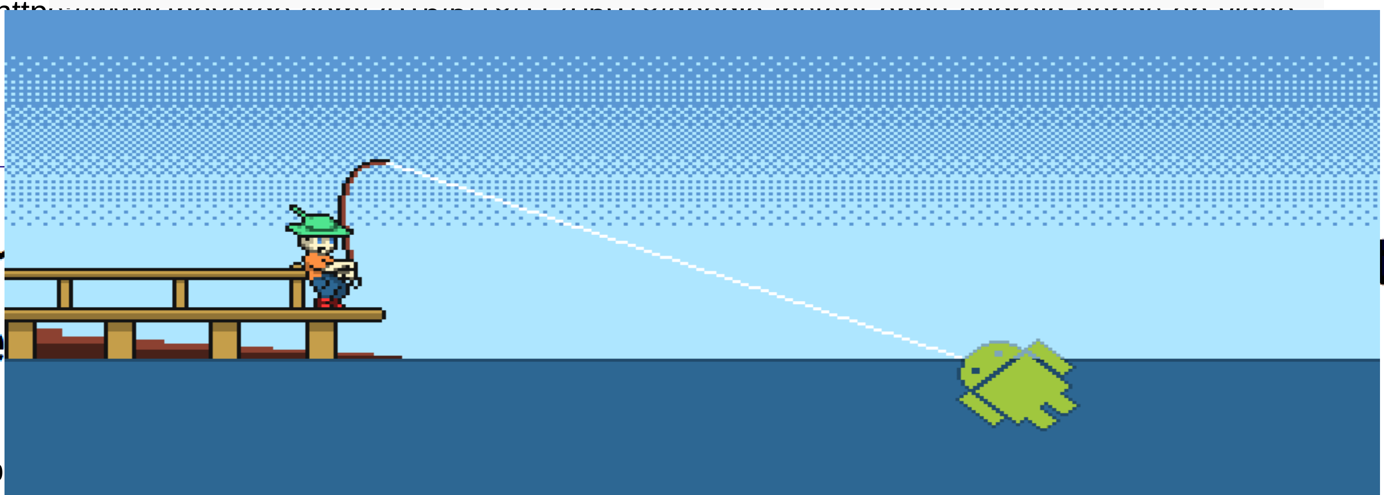
By Dieter Bohn |

Source: http

March 07, 2017

Phishing Attacks Against Android Instant Apps

by Camilo Reyes



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

GOOGLE \ TECH

Andr ... to all
deve

By Paul Miller |

Source: http

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

Source: https://www.techradar.com/news/android-banking-malware-hitting-more-users-than-ever

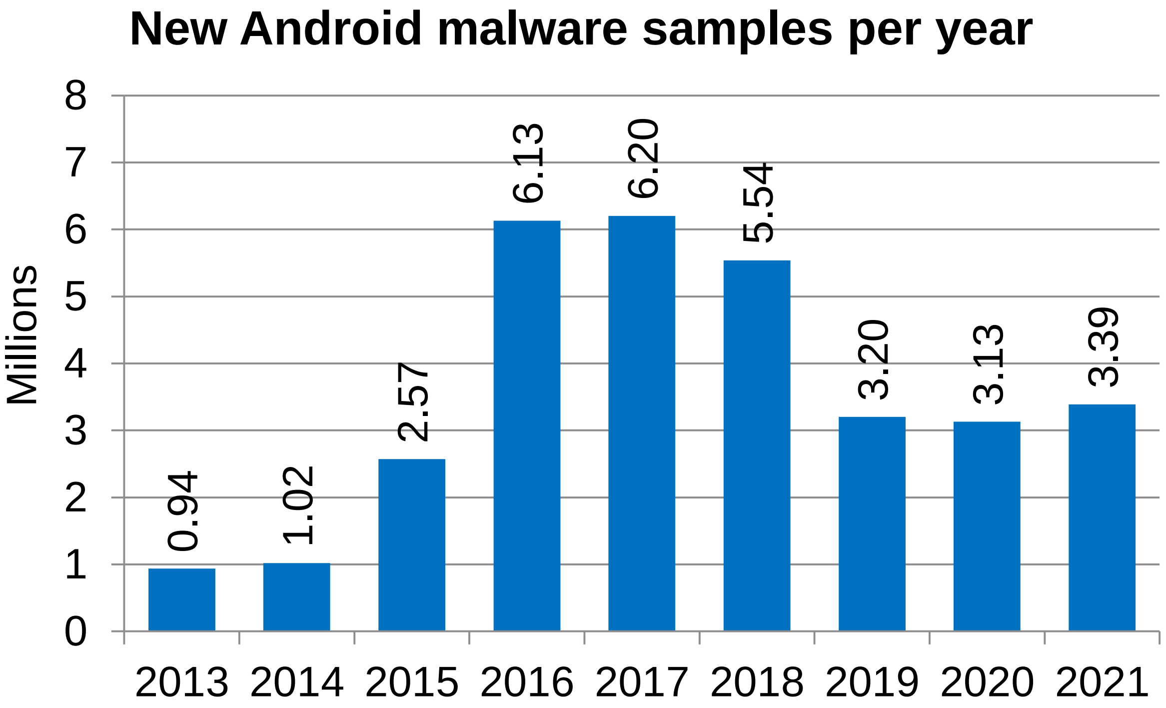
By Anthony Spadafora 22 days ago Internet

Fake banking apps could be more effective than banking Trojans

February 04, 2019 Swati Khandelwal

Source: https://thehackernews.com/2019/02/beauty-camera-android-apps.html

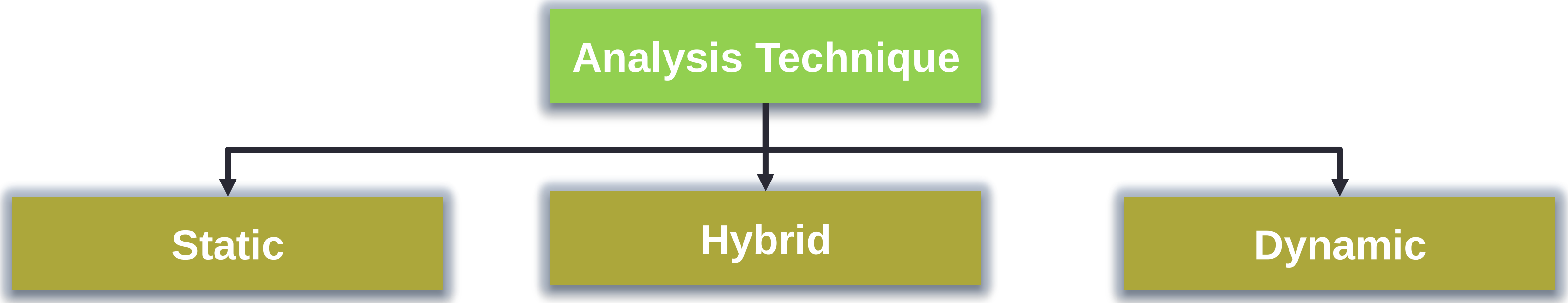
Android Malware Statistics



- In every 10 seconds, A new Android malware is born.

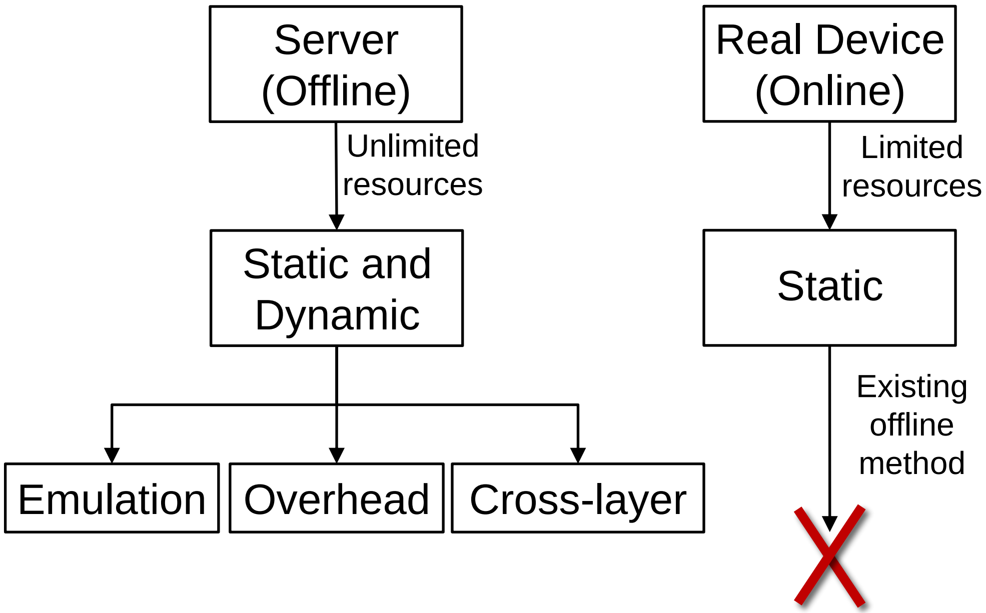
Source: AV-TEST malware statistics report Jan 2022

Analysis Techniques



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

Techniques	Challenges
Static	• Dynamically Loaded Code • Crypto API • Java Reflection • False positive (permission based) • Network based activity
Dynamic	• False positive (Anomaly based) • Code Coverage • 20 times slowdown system if used in real device
Hybrid	• Data Dependency ACG • Logic based triggers • Obfuscation and reflection

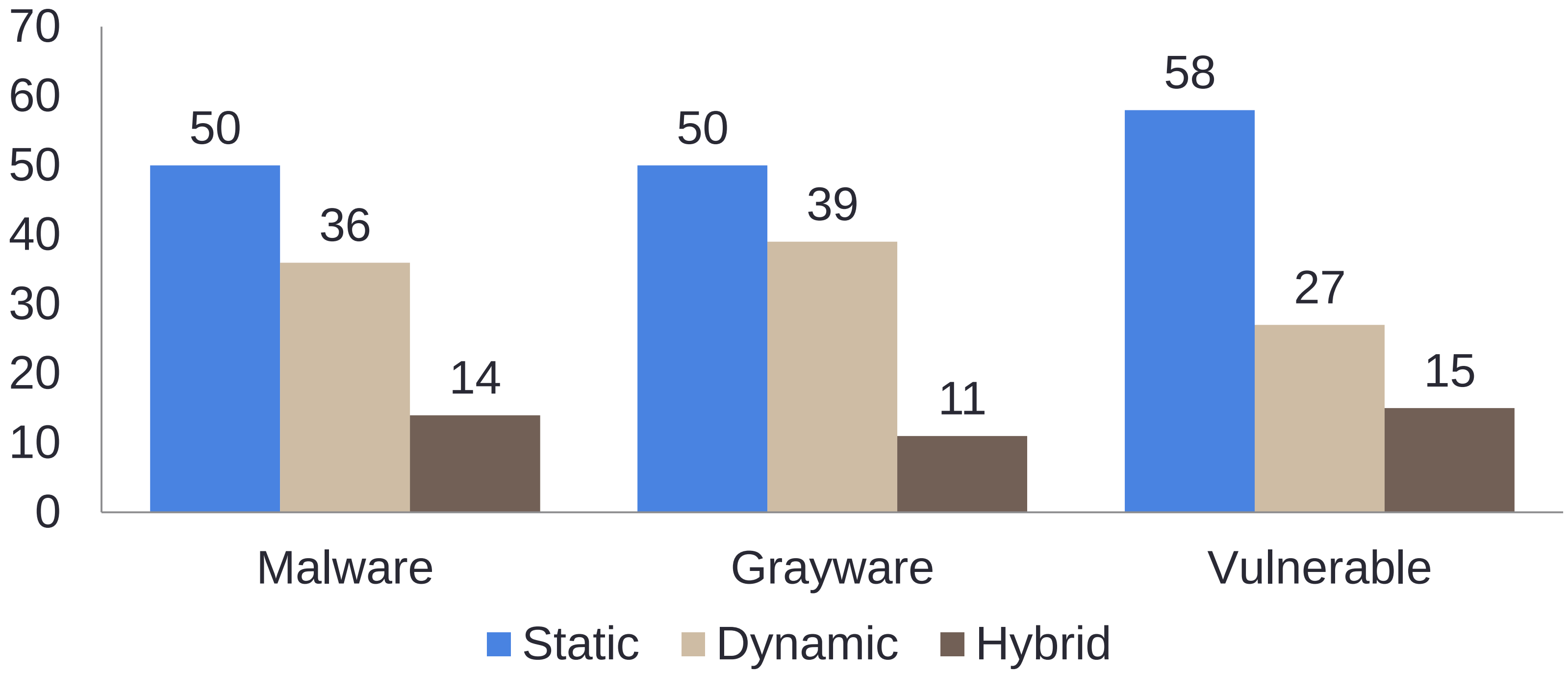
Malware Analysis Frameworks

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

Malware Analysis Frameworks Cont..

Framework	Method	Limitation
Wang et al. [2016]	Static + machine learning + permissions + APIs	Dynamic code loading, Native code, obfuscation
DroidSeive [Guillermo et al. 2017]	Static + machine learning + multiple location features	Dynamic code loading
IntelliAV:[Ahmadi, et al. 2017]	Static + machine learning + API Call, Components statistics	Dynamic code loading, Native code, obfuscation
TinyDroid [Chen et al. 2018]	Static + Opcode + machine learning	Dynamic code loading, Native code
Fatima et al. [2019]	Static + machine learning + permissions	Dynamic code loading, Native code, repackaging attack

Analysis Techniques used in Different Area



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