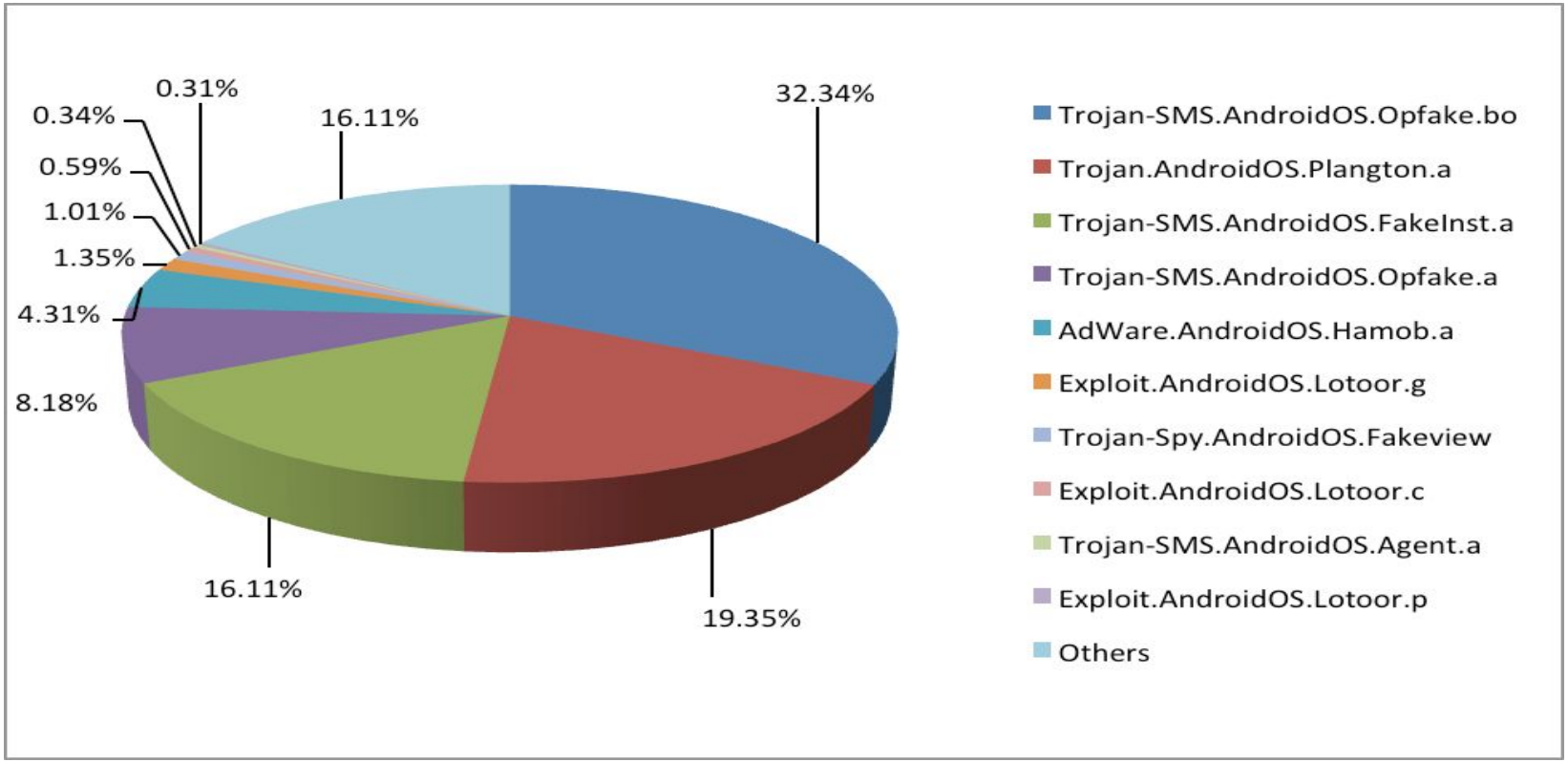
Android Security

Android & Malware

- Android is an Operating System broadly used on the mobile device and **It covers 60% of the OS market.**
- Android has been developed by Google engineering and **it is secure by design**. It is on the most attacked OS by malware.
- Report from Kaspersky lab said that the rate between **android malware and the other mobile malware is 9:1**

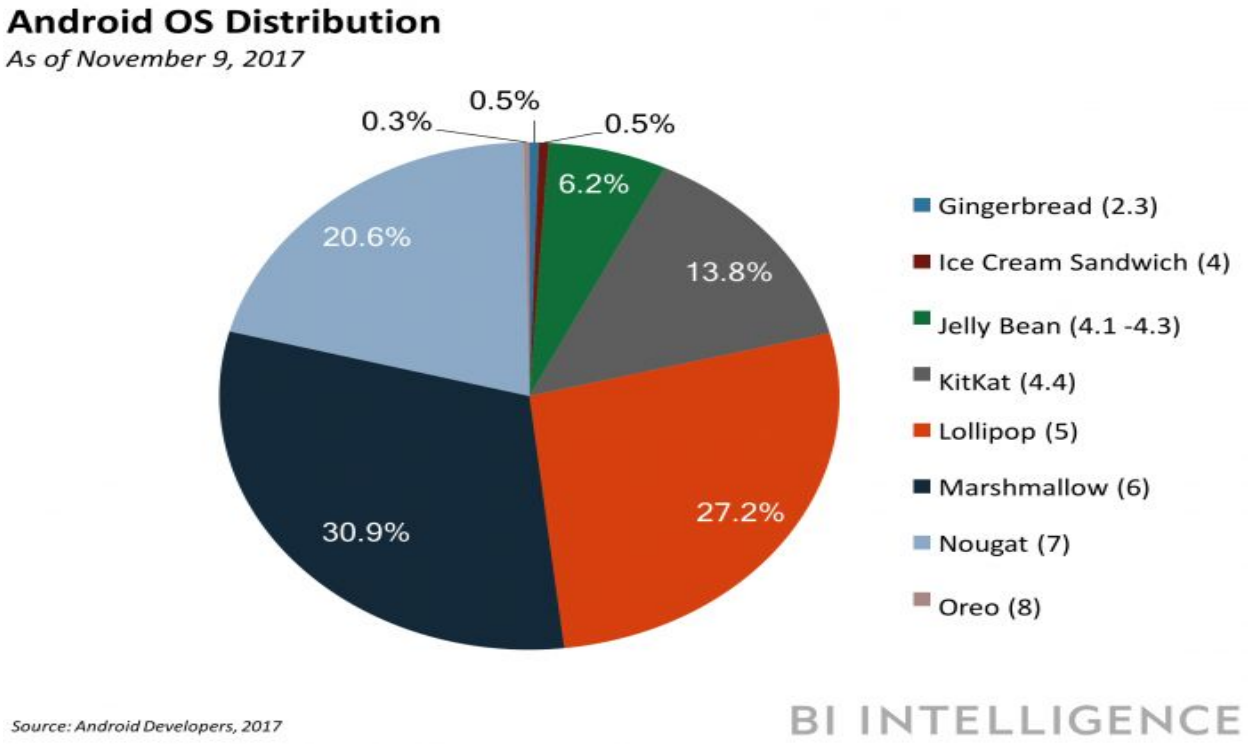
Android & Malware



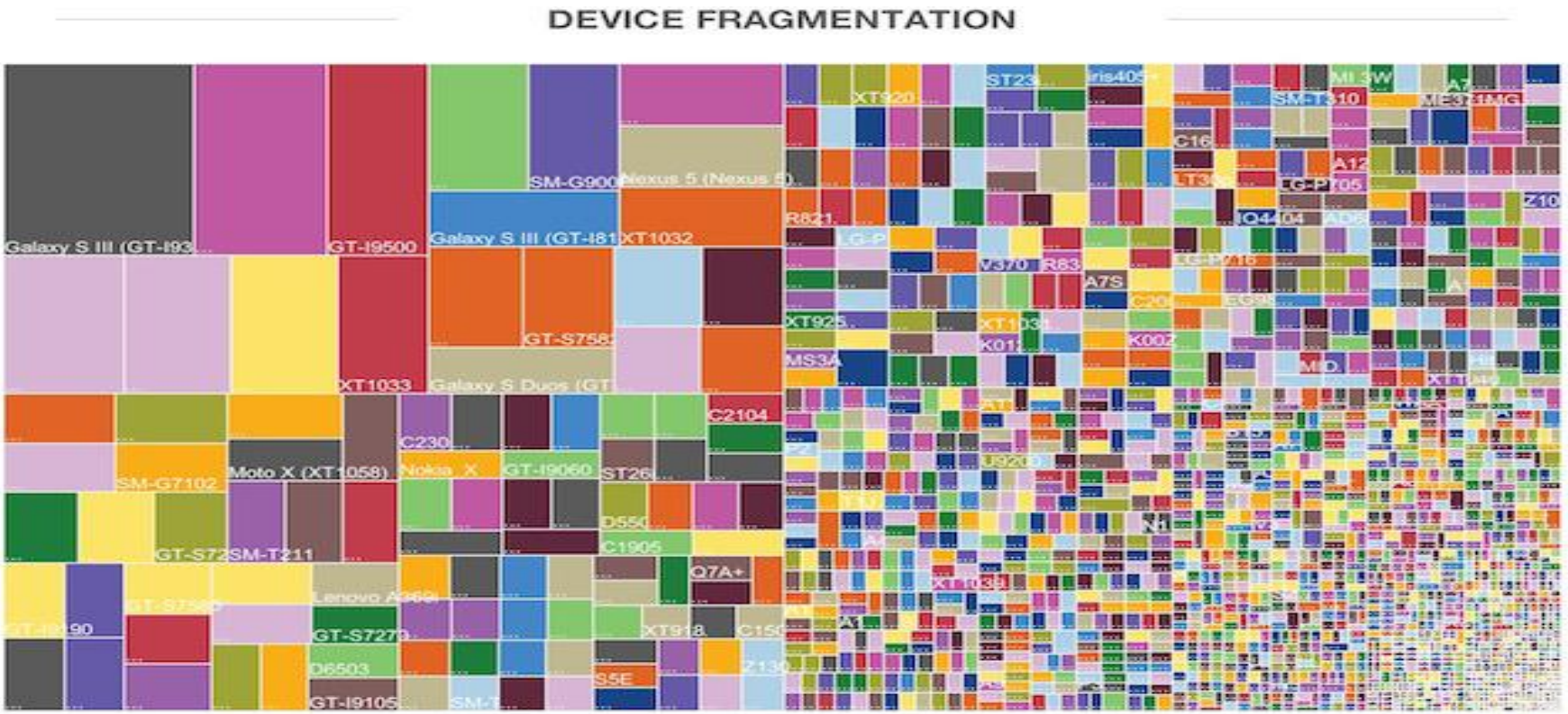
Android Fragmentation Problem

- One of the main problem of Android Os is the **fragmentation.**
- Android fragmentation — when older versions of operating systems remain on active mobile devices.
- Over half of all Android devices, or a billion, are more than two years out of date, according to Dan Luu. That's up from 42% of devices over two years out of date in 2014.

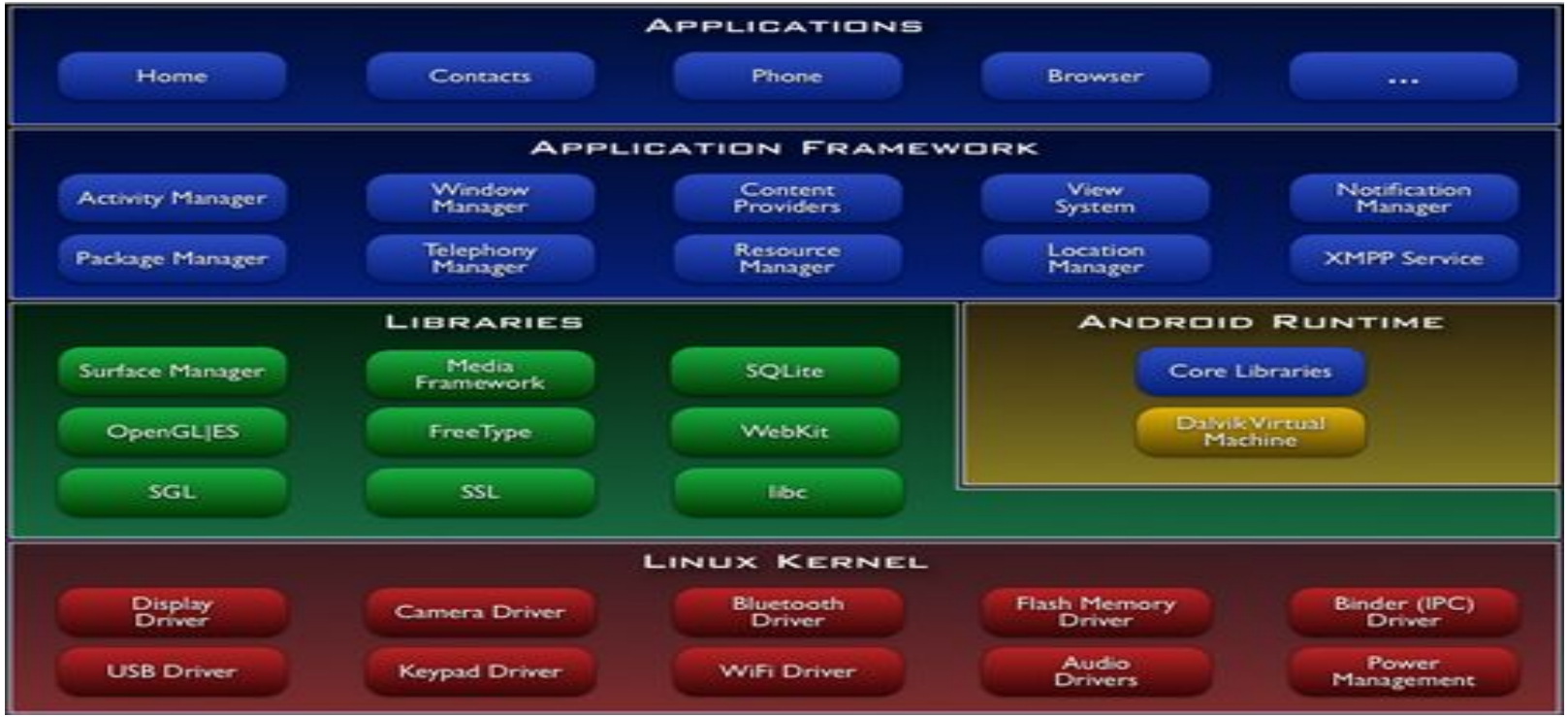
Android OS Distribution



Android Fragmentation



Android Architecture



Why this new app model?

Desktop apps

- - Hard to install apps?
- - No isolation at permission level
- + Sharing is easy

Web apps

- + Easy to install apps
- + Isolation
- - No API for devices (e.g., Phone call)
- - Limited data sharing
- - JavaScript

- **The android apps are written in Java, why?** Usability, portability, security? Why this not happen for IoS system?

Android Apps Components

- Each Android Apps can have four components:
  - **Activity**: This component is in charge to handle the GUI for interact with the user.
  - **Broadcast receivers:** This component is in charge to receive message from the other part of the system.
  - **Content Providers**: We can think as a database that stored the data application and allow or deny other app to query such data.
  - **Services**: This components are executed in background for example monitoring your location or polling some data from Internet SMS etc.

Android Apps (Manifest)

- All the four components are java classes that developer writes.
- **Manifest** is another part of the Android App, it is an XML file that described all the components and how the other part of the system should interact with this application.
- **Manifest** defines label in terms of what this application is able to do and who else can interact with the components of this application.
- **Labels** define privileges for the application. For the components defined the requirements that another part of the system should have in order to communicate with a particular components.

Permessi Android

IPC Android App (Intent)

- When an application wants to communicate with the another app defines an **Intent**. The intent has **three main** important things:

1. **Name of the component** to which you want to send a message.
2. **Action** that you want the component to take.
3. **Data (MIME Type)** that you want to send to this other component.

IPC Android (Reference Monitor)

- Application communicate each other by using classic UNIX pipe mechanism with the **Reference Monitor (RM) that is in charge to check all the permission of the App and deliver the message**.
- **Explicit Intent**: the message has to go to this component.
- **Implicit Intent:** you do not know which is the final component destination the RM chooses the final application with the right permission.
- **Bind Intent**: I want to direct connect to this application (avoid RM bottleneck)

Broadcast Receiver

- **When an app sends a broadcast message, it wants to control who is gonna receive the message**, otherwise an app can register for any broadcast message in the system and receive all of them.
- **Android system permit to set up a label along with the broadcast message** that specified the permission that a received app needs to have in order to read the message**.**
- In order to authenticate the source, the apps that wants to receive the message **stick a label on broadcast receiver that represents the permission that the sender needs to have** in order to receive the message.

Android Permission

- The **isolation among apps** is achieve by using Linux permission. The system assigned for each apps a different UID.
- **Not all the operations with the system are mediated through the RM** (Intent) For example:
    - Networking (API Enforcement)
    - SD card
    - File System (DAC enforcement)
    - Devices

Android Permission

- **All these operations are orchestrated by the system using UID e GID**
- For each resources the **system predefined e label string and set up a Group ID** that can be associate to an app for using that particular resources.
- For example to **use Internet the app needs to belong to a certain GID,** and Linux Kernel allows to use the set of the system call used for communication for that app (**API enforcement**).

Rooting device

- **Rooting** operation permit to acquire the high privilege on the device:
  - **Advantages**:
    - ROM substitution (e.g., software with new features inside)
    - Setting of the HW parameters (CPU frequency, voltage, etc.)
    - New software configuration parameters (add new users etc.)
  - **Disadvantages:**
    - All the applications installed on the phone can abuse of the high privileges
    - If the phone is compromised the attacker can execute command with the high privileges.

Kakao Talk

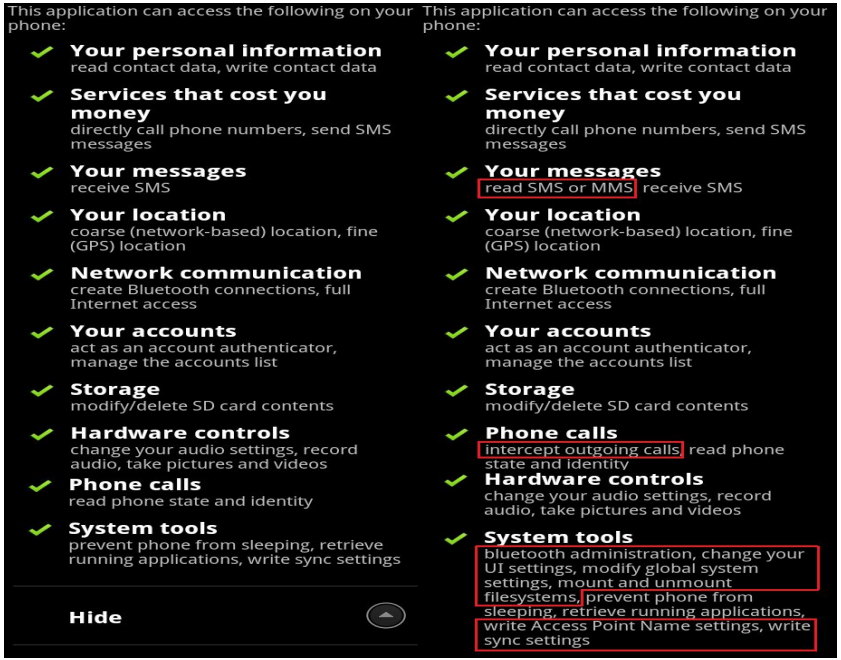
- The first step is the **repack of a famous application to give to the user as a free version**.

- **Kakao Talk is a chat application used in the china community.**

- The attack vector is represented by an **e-mail with attach the free version of the app to install on the phone**.

Caso Kakao Talk

- Permissions required from the malware:



Caso Kakao Talk

- The malicious application stores in a crypted version: the agenda, the SMS history and the log of the phone call.
- The application then connects to a **Command & Control and download information like URL and credential**.
- Moreover it was filtering out SMS based on some particular string (password, bank etc.) and it was sending the geo locations to physically individuate the device.

Defensive System: Google Bouncer

- **Google Bouncer** is a defensive mechanism based on scanning of the application published on the market.
- We have three scanning techniques:
  - **Static Analysis of the application to look for known vulnerabilities/backdoor etc.**
  - **Execution of the app in a virtual environment.**
  - **Analysis of the data submitted by the developers**

Security Analysis Google Bouncer

- **Google Bouncer** has been tested by a group of security researchers:
  - **Bouncer analyzes the application immediately when it submitted on the market.**
  - **Bouncer is looking for suspicious patterns (e.g., /system/bin, /bin etc.)**
  - **Bouncer is composed by an automatic stimulator for a GUI.**
  - **Bouncer check the app only if they come from certain IP addresses.**
  - **Bouncer use for testing only predefined set of IP addresses.**

Security Analysis Google Bouncer

- The security researchers create a malicious app that is able to elude the bouncer system:
  - The application disable its own functionalities if it recognizes is running under the bouncer IP addresses set.
  - The application embed a downloader that download a java application from internet when it is running.

Security Analysis Google Bouncer

- Bouncer Issues:
    - Code Coverage
    - Emulation detection
    - Predefined IP Blocks.
- Bouncer improvements:
    - Improve the static analysis
    - Improve the dynamic analysis