Operating Systems Security

Rooting Android

Seong-je Cho

Spring 2019

Computer Security & Operating Systems Lab, DKU

Sources / References

- Android Device Rooting Lab, SEED Lab: A Hands-on Lab for Security Education.
  - https://seedsecuritylabs.org/Labs_16.04/Mobile/Android_Rooting/Android_Rooting.pdf

- Rooting Android, Created by: Mayank Talwar, Dignitas Digital.
- Root Access, by Derek Grove
- **Android Forensics** and Security Testing, Shawn Valle, Sep. 2012.
- Hacked, rooted and jailbroken – different approaches to accessibility, Bruce Darby, University of Eninburgh
- Rooting Android Devices, Lokendra Rawat, Student at Arya college of engineering &it

Contents

What is Root?

- An account that by default has access to all commands and files on a Linux or other Unix-like operating systems
  - Root account, root user, Super-User
  - Most privileged account on the system, has absolute power over it.

- Ability to modify the system in any way

- The ability to modify files in the root directory
  - Root directory: Only accessible by the tooted user

- Android devices do not allow their owners to have the root privilege on the device by default.
  - This is fine for normal customers, but for users who want to make deep customizations on their devices, this is too restrictive.

What is Rooting?

- The process of gaining the root privilege on Android devices
- **Process of allowing the users to attain privileged control (known as 'root access') within Android's subsystem.**
  - Allows full access and control of the OS
  - "rooting" an Android is similar to running a command with "sudo" OR "as root user" in Linux.
  - This can be done by installing an executable '**su**'

- Administrative or root access
  - Rooting Android device makes you super-user
  - Rooting your device turns you into its owner in the trust sense of the world, as it put you in charge of every aspect of your phone.
  - It allows you to extend the longevity of a device via custom ROMs, tweaks, and up to data patches

- Different methods for different phones

Android Device Rooting Lab

- SEED labs
  - https://seedsecuritylabs.org/Labs_16.04/Mobile/
  - https://seedsecuritylabs.org/Labs_16.04/Mobile/Android_Rooting/

Develop an OTA (Over-The-Air) package from scratch to root an Android device

- Two objectives of this lab
  1. Students will get familiar with the process of device rooting and understand why certain steps are needed
  2. Students will gain in-depth system knowledge
     - » Entire rooting involves many pieces of knowledge about the Android system and OS

- We will ask students to develop a complete rooting package from scratch, and demonstrate how to use the package to root the Android VM provided by us.
  - **SEEDUbuntu16.04 VM + Android 7.1 VM**
  - SimpleSU.zip

Modifying Android from inside

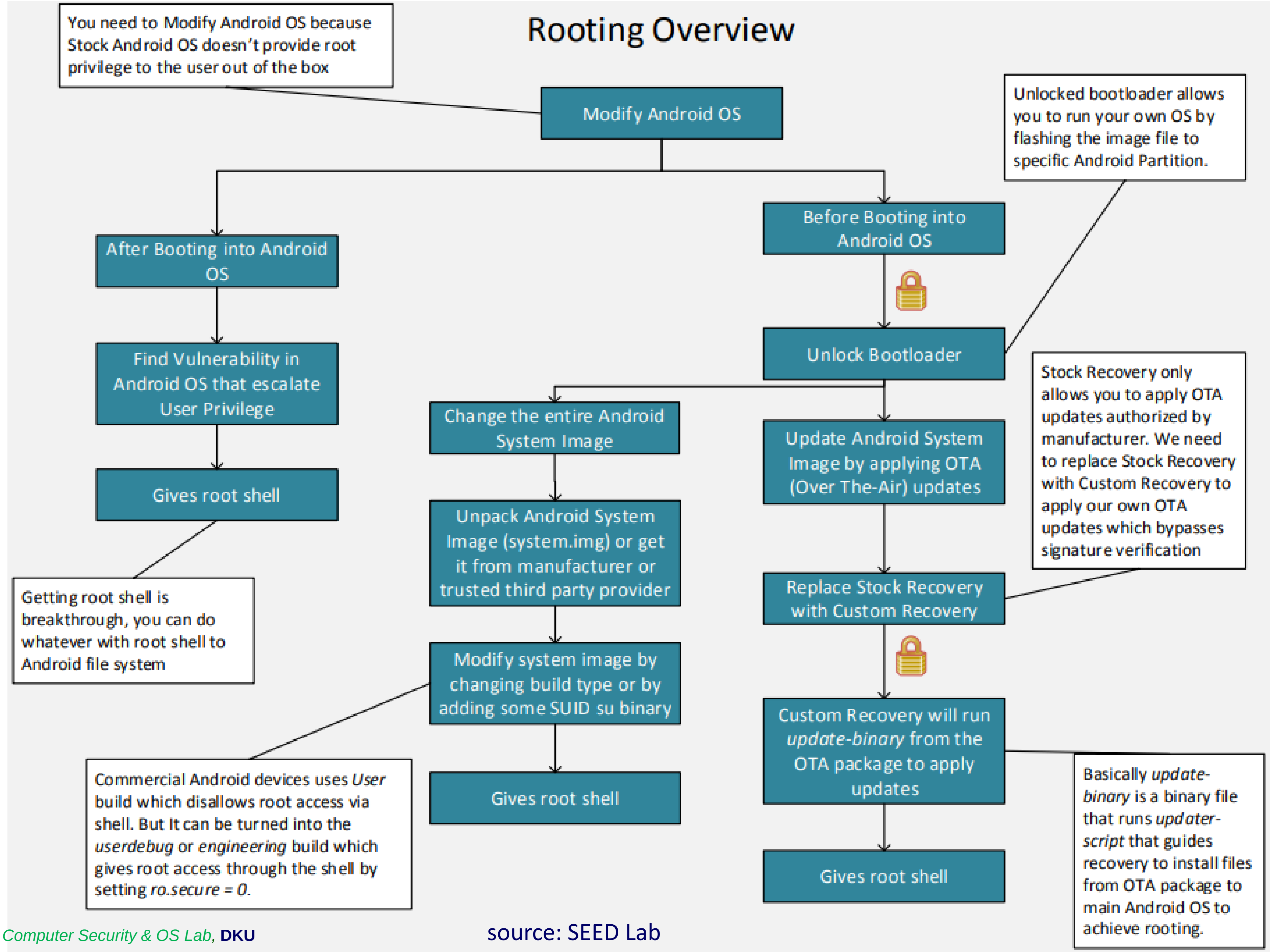
- rooting from inside Android means that the user has to gain the root privilege as a normal user.
  - there might exist vulnerabilities in the system.
  - If these vulnerabilities are inside the kernel or daemons running with the root privilege, users can exploit these vulnerabilities to gain the root privilege.

  - **`RageAgainstTheCage`**
    - This rooting exploit took advantage of **`RLIMIT_NPROC`** max, the value specifying how many processes a given UID (e.g. 2000) can run.
    - This exploit first uses "**`adb shell`**" to run a shell via the **`adb`** daemon
      1. Then, forks new processes until the number of processes have reached the limit and the fork will fail
      2. Kills the `adb` daemon
      3. Reconnects to it using "`adb shell`" to restart the `adb` daemon.
      4. When adb is started, it always has the root privilege, but it will drop the privilege to UID 2000 using `setuid(2000)`
      5. The UID 2000 has already used up its process quota, so the call will fail.
      6. The `adb` daemon fails to handle the failure correctly: instead of exiting, it keeps running, while retaining the root privilege.
      7. The "`adb shell`" command with give users a root shell

Modifying Android from outside

- Device can have a dual-boot configuration, allowing you to boot into any of these two OSes when the device powers on.
  - If we boot into this second OS, and become the root for this OS, we can mount the partition used by the Android system.
- most Android devices do have the second OS installed, and it is called *recovery OS*.
  - *Recovery OS* is placed on devices by vendors, who use the OS to update Android.
  - *Recovery OSes* do not give users a shell prompt, preventing users from running arbitrary commands.
  - Instead, *recovery OSes* take a **package** provided from outside (either from users or downloaded from the Internet)
    - The package contains the commands and files needed for updating the Android OS.
    - This mechanism is called Over-The-Air (OTA) update.
    - The package is called **OTA package**, which has a standard file structure that we will talk about later.
      - » Most recovery OSes only accept the packages made by the vendors.
      - » **The package that we use for rooting purposes will not come from the vendor of the device**.
      - » We need to find ways to bypass this access control.

Rooting Overview



You need to Modify Android OS because Stock Android OS doesn't provide root privilege to the user out of the box

Modify Android OS

After Booting into Android OS

Find Vulnerability in Android OS that escalate User Privilege

Gives root shell

Getting root shell is breakthrough, you can do whatever with root shell to Android file system

Before Booting into Android OS

Unlock Bootloader

Unlocked bootloader allows you to run your own OS by flashing the image file to specific Android Partition.

Change the entire Android System Image

Unpack Android System Image (system.img) or get it from manufacturer or trusted third party provider

Modify system image by changing build type or by adding some SUID su binary

Gives root shell

Commercial Android devices uses *User* build which disallows root access via shell. But It can be turned into the *userdebug* or *engineering* build which gives root access through the shell by setting *ro.secure = 0*.

Update Android System Image by applying OTA (Over The-Air) updates

Replace Stock Recovery with Custom Recovery

Stock Recovery only allows you to apply OTA updates authorized by manufacturer. We need to replace Stock Recovery with Custom Recovery to apply our own OTA updates which bypasses signature verification

Custom Recovery will run *update-binary* from the OTA package to apply updates

Gives root shell

Basically *update-binary* is a binary file that runs *updater-script* that guides recovery to install files from OTA package to main Android OS to achieve rooting.

source: SEED Lab

Reinstall recovery OS

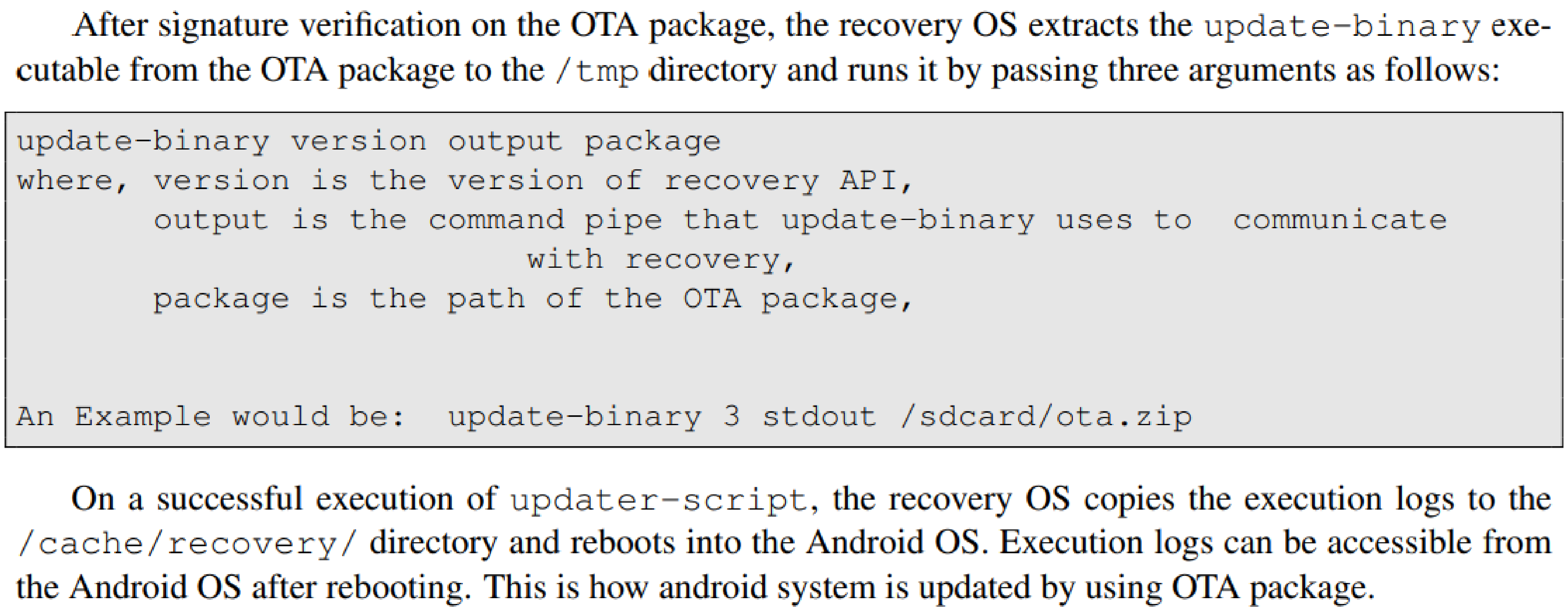
- Instead of bypassing the access controls of the *recovery OS*, the easiest way is to replace the entire *stock recovery OS* with *another recovery OS* (e.g., *custom recovery OS)* that does not have such access controls.
  - *custom recovery OS* will not include the signature verification part, so we can provide any OTA packages to the *recovery OS*.
- There is another access control that prevents us from replacing the *stock recovery OS*, and it is the bootloader.
  - Bootloader is a low level code that loads an OS or some other system software for the computer after the computer is powered on.
- When a bootloader is "**locked**", it will simply load one of the OSes that is already installed on the device, leaving no chances for users to modify any of the pre-installed OSes.

- If a bootloader can be **unlocked**, which allows users to install *custom OS* on the device, as well as replacing the existing ones.
  - The process is often referred to as **flashing** *custom OS*.

  - Most manufacturers do provide ways for users to unlock the bootloader on their devices.

OTA package

- Students need to build their own OTA package from the scratch in this lab.
- OTA package is just a zip file. → see the next slide



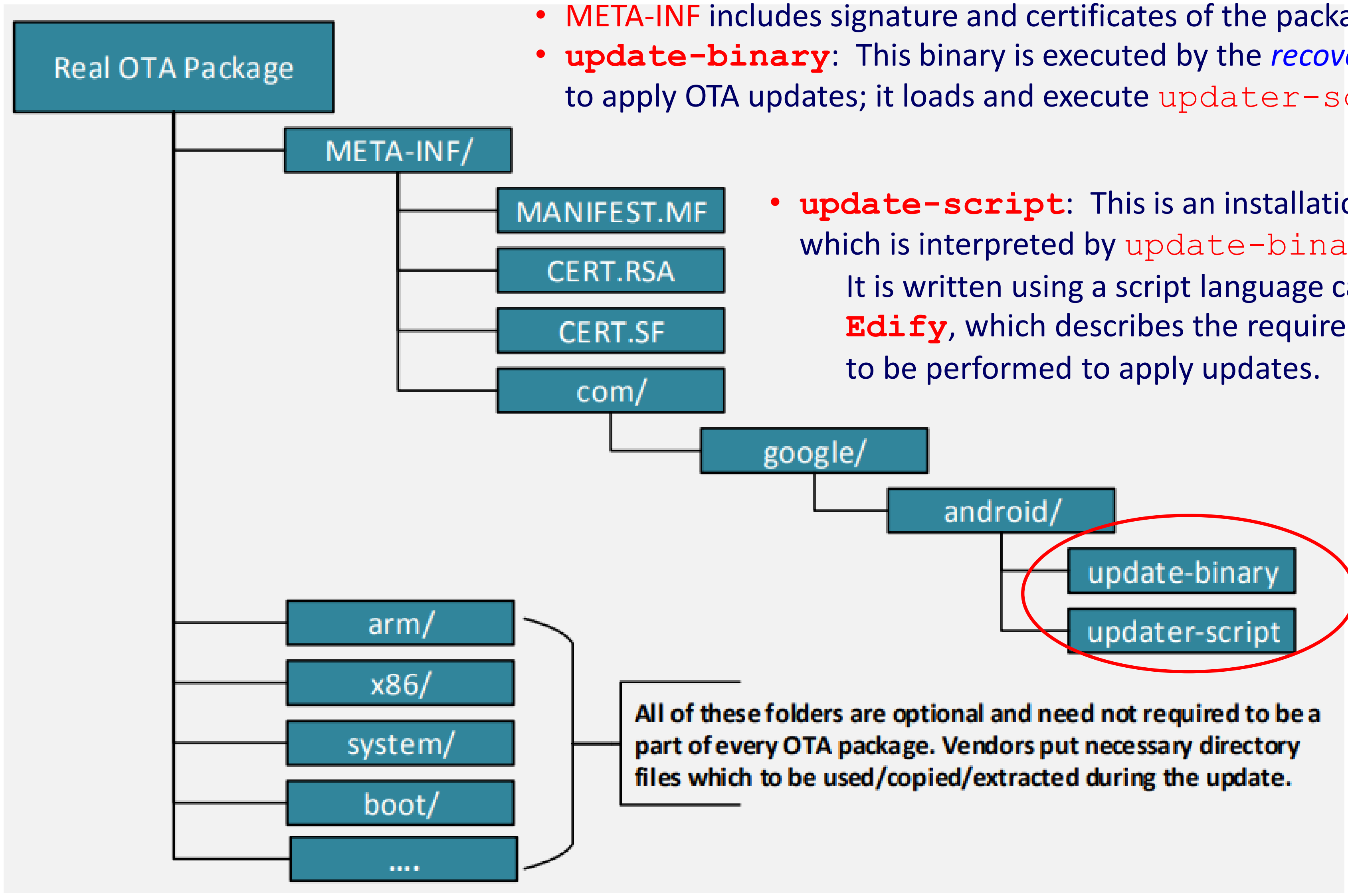
After signature verification on the OTA package, the recovery OS extracts the `update-binary` executable from the OTA package to the `/tmp` directory and runs it by passing three arguments as follows:

```
update-binary version output package
where, version is the version of recovery API,
       output is the command pipe that update-binary uses to  communicate
                          with recovery,
       package is the path of the OTA package,


An Example would be:  update-binary 3 stdout /sdcard/ota.zip
```

On a successful execution of `updater-script`, the recovery OS copies the execution logs to the `/cache/recovery/` directory and reboots into the Android OS. Execution logs can be accessible from the Android OS after rebooting. This is how android system is updated by using OTA package.

Structure of OTA package


Real OTA Package
META-INF/
MANIFEST.MF
CERT.RSA
CERT.SF
com/
google/
android/
update-binary
updater-script
arm/
x86/
system/
boot/
....
All of these folders are optional and need not required to be a part of every OTA package. Vendors put necessary directory files which to be used/copied/extracted during the update.

- META-INF includes signature and certificates of the package
- **update-binary**: This binary is executed by the *recovery OS* to apply OTA updates; it loads and execute `updater-script`.
- **update-script**: This is an installation script which is interpreted by `update-binary`.
  It is written using a script language called **Edify**, which describes the required action to be performed to apply updates.

Figure 2: OTA Structure

Lab Environment

- Assumption: Bootloader on the device can be unlocked, and the stock recovery OS can be replaced.
- Replacing the recovery OS in the VM environment is quite difficult, and it is not within the scope of this lab
- The Android VM that you download from our web site already has a custom recovery OS installed.

- We simply use Ubuntu 16.04 as the "*recovery OS*".
  - Technically, this is not a recovery OS; it is just another OS installed on the device.
  - We use this general-purpose OS to emulate what users can do using a custom recovery OS.
  - Students who have a physical Android device and want to know how to root a real device, they can follow our guidelines in Section 7. However, other than the step to replace the recovery OS, everything else is the same as what we do in this lab.

- **Our Android VM is already rooted**
  - Our VM build is actually a *userdebug* build not *user* build.
    - Several doors were built into the VM to allow root access.
  - In this lab, students are not allowed to use those doors to gain the root access; they have to use the OTA mechanism to achieve that

Lab Tasks

- Lab **Task 1**: Build a simple OTA package
  1. Step 1: Write the update script
  2. Step 2: Build the OTA Package
  3. Step 3: Run the OTA Package

- Task 2: Inject code via **`app_process`**
  1. Step 1. Compile the code.
  2. Step 2. Write the update script and build OTA package.

- Task 3: Implement **`SimpleSU`** for Getting Root Shell
  1. Background
  2. The Task

Task 1

Goals

1. How to inject a program into the Android OS from the recovery OS?   We then get Android to run the injected program using the root privilege.
   - Android의 /system 폴더에 dummy file을 생성 → 즉, /system/dummy에 “hello” 단어 저장

     ```
     echo hello > /system/dummy
     ```

   - 위 명령을 dummy.sh라는 shell script file에 저장
   - dummy.sh 는 다른 명령들도 포함해야 …
     - 예로, /system (분할)은 root 소유로 본래 read-only 허가만 있음. 그런데, dummy 파일을 생성할 수 있어야 함.

2. How to get our injected program to run automatically, and with the root privilege?

3. How to write a program that can give us the root shell?

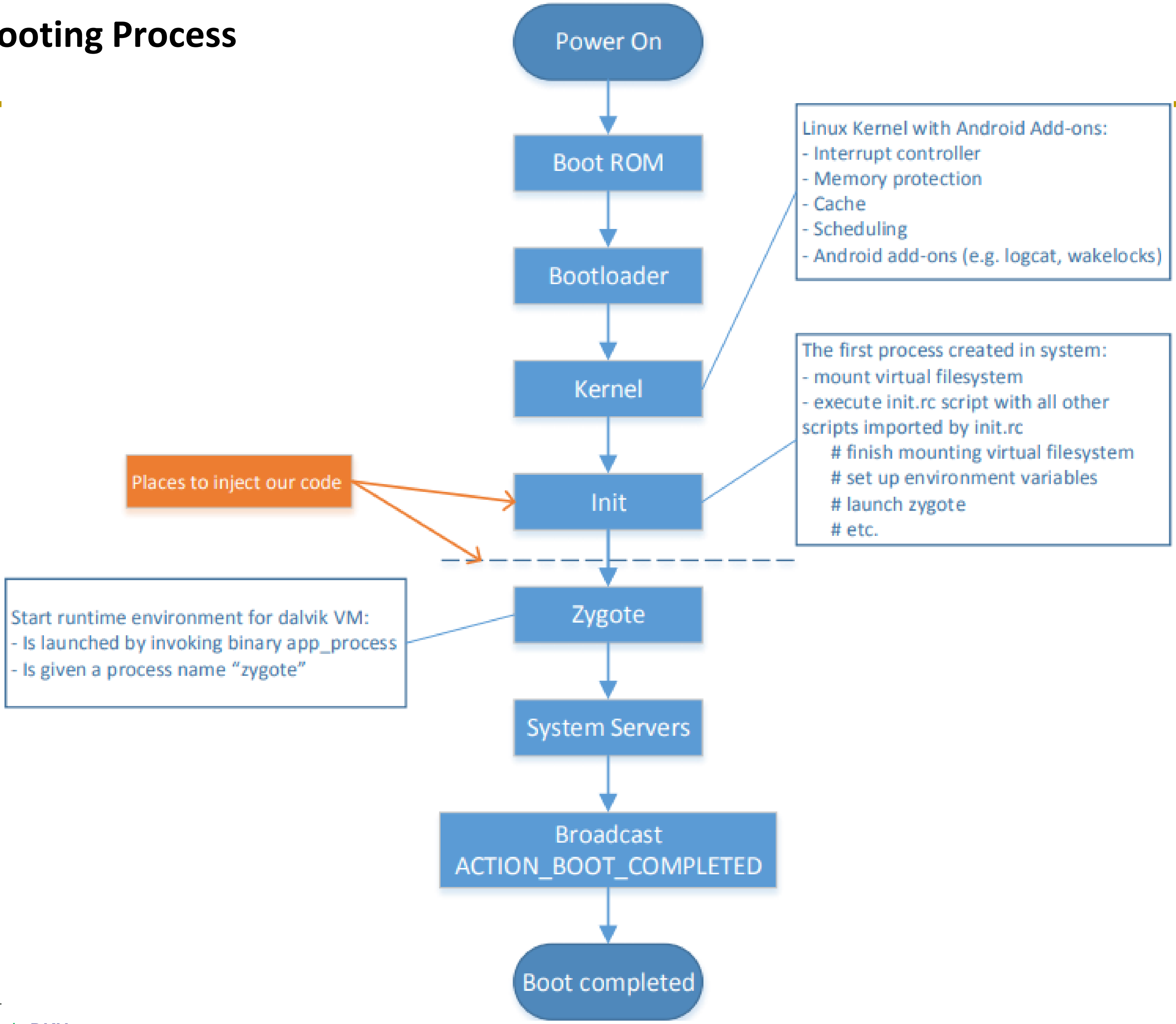
Step 1 of Task 1

Write the update script.

- `update-binary` file is just a simple script file in this Lab.
  - Our recovery OS (`Ubuntu`) does have `bash` installed

- `update-binary`의 목적 2가지
  1. dummy.sh 프로그램을 Android OS에 주입해야 함
     - dummy.sh을 어디에 위치시켜야 하는지, 이 프로그램의 permissions 설정은 어떻게 해야 하는지?
     - 즉 Android 분할에 위치시켜야 함. Android 분할은 recovery OS (Ubuntu VM)의 `/android` 디렉토리에 마운트되어 있음
  2. Android OS 설정 파일을 변경하여, Android 부팅 시에 dummy.sh가 루트권한으로 자동 실행되게 해야 함
     - 부팅 시, 초기화 과정에서 /system/etc/init.sh 파일이 수행됨.
       - » This is for Android-x86 build; for the ARM build, the file name is different.
     - dummy.sh를 실행하는 명령어를 init.sh에 포함시키면 됨. SEED Lab 설명서 6 페이지 중간에 sed 명령을 사용하는 코드가 있으며, 이를 `update-binary`에 반영해야.

Detailed Booting Process


Power On
Boot ROM
Bootloader
Kernel
Init
Zygote
System Servers
Broadcast ACTION_BOOT_COMPLETED
Boot completed
Places to inject our code
Linux Kernel with Android Add-ons:
- Interrupt controller
- Memory protection
- Cache
- Scheduling
- Android add-ons (e.g. logcat, wakelocks)
The first process created in system:
- mount virtual filesystem
- execute init.rc script with all other scripts imported by init.rc
# finish mounting virtual filesystem
# set up environment variables
# launch zygote
# etc.
Start runtime environment for dalvik VM:
- Is launched by invoking binary app_process
- Is given a process name "zygote"

Step 2 of Task 1

Build the OTA Package.

- slide 12 (Fig. 2)에 나타난 구조로 OTA package를 구성해야
  - 그러나, (signature, optional files과 같은), 본 과제에 불필요한 파일들을 생성할 필요는 없음.

- dummy.sh 파일을 OTA 패키지의 임의 위치에 배치할 수 있음.
  - 단, 그 위치는 update-binary 내의 command와 일치해야 함

- 필요한 파일들을 구성한 후에, zip 명령어를 사용하여 zip 파일 생성

```
zip -r my_ota.zip ./
```

- **결과 보고서에** OTA 패키지의 파일 구조를 나타내고 설명해야 함.
  - unzip -l 명령을 사용

Step 3 of Task 1

<u>Run the OTA Package.</u>

- 실제로는 생성한 OTA 패키지를 recovery OS에 제공하면 됨.

- 본 과제에서는 Ubuntu를 recovery OS로 사용
  - 따라서, recovery OS 기능을 흉내내야 함.
  - 즉, unzip 명령을 사용하여, OTA 패키지를 수작업으로 압축해제하고, META-INT/com/google/android 폴더로 가서 update-binary 파일을 찾아 실행하면 됨.
  - 제대로 했다면, Android가 업데이트 됨.
  - 업데이트된 Android OS를 부팅하여 /system 폴더에 dummy 파일이 생성되었는지 확인.

- 결과보고서에는, 해당 screenshot들을 포함해야 함.

- **For more info, see "Android Device Rooting Lab"**
  - **https://seedsecuritylabs.org/Labs_16.04/Mobile/Android_Rooting/Android_Rooting.pdf**

Technical Terms & Rooting Techniques

Background (Technical terms)

- **ROM**
  - Data cannot be modified, or can be modified only slowly or with difficulty
  - A ROM is a modified version of Android.
    - It may contain extra features, a different look, speed enhancements, or even a version of Android that hasn't been released for your phone yet.

- **STOCK**
  - "**Stock**" means the version of Android that came with your phone – e.g., if you want to get rid of your ROM and return your phone to factory settings, you might say you're "going back to stock."

  - Stock ROM is the default ROM (Backup)

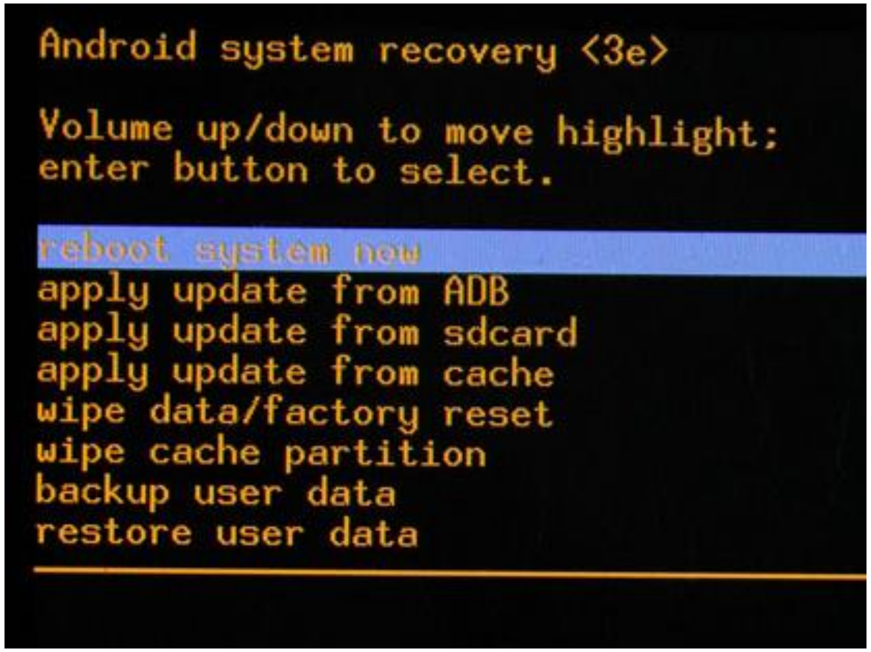
Background

- **Kernel**
  - Component of OS
  - Manages communication b/w S/W & H/W
  - Custom Kernel can speed up phone
  - Custom Kernel can improve battery life
- **Flashing**
  - Installing something on your device, whether it can be a ROM, a kernel, or a recovery that comes in the form of a zip file
  - Done through recovery or through ADB(Android Debug Bridge)
- **Brick**
  - To brick your phone is to break it during flashing or other acts.
  - There is always a small risk with flashing, and if your phone becomes unable to function – that is, it basically becomes a brick.

Background

- **Bootloader**
  - the lowest level of S/W on your phone
  - runs all the code to start OS
  - Security Checkpoint for different partitions
  - Locked Bootloader keeps phone safe
  - Bootloader verify signature of system image before booting

- **Recovery OS**
  - the S/W on your phone that lets you make backups, flash ROMs, and perform other system-level tasks.
  - Wiping cache/data



```
Android system recovery <3e>

Volume up/down to move highlight;
enter button to select.

reboot system now
apply update from ADB
apply update from sdcard
apply update from cache
wipe data/factory reset
wipe cache partition
backup user data
restore user data
```

Background

- **Nandroid**
  - From most third-party recovery modules, you can make backups of your phone called Nandroid backups.
  - It's essentially a system image of your phone: Everything exactly how it is right now.
  - Using **Nandroid backup**, it is possible to install costom OS

- **ADB (Android Debug Bridge)**
  - It's a command line tool for your computer that can communicate with an Android device you've connected to it
  - It's part of the Android
    - Software Development Kit (SDK)

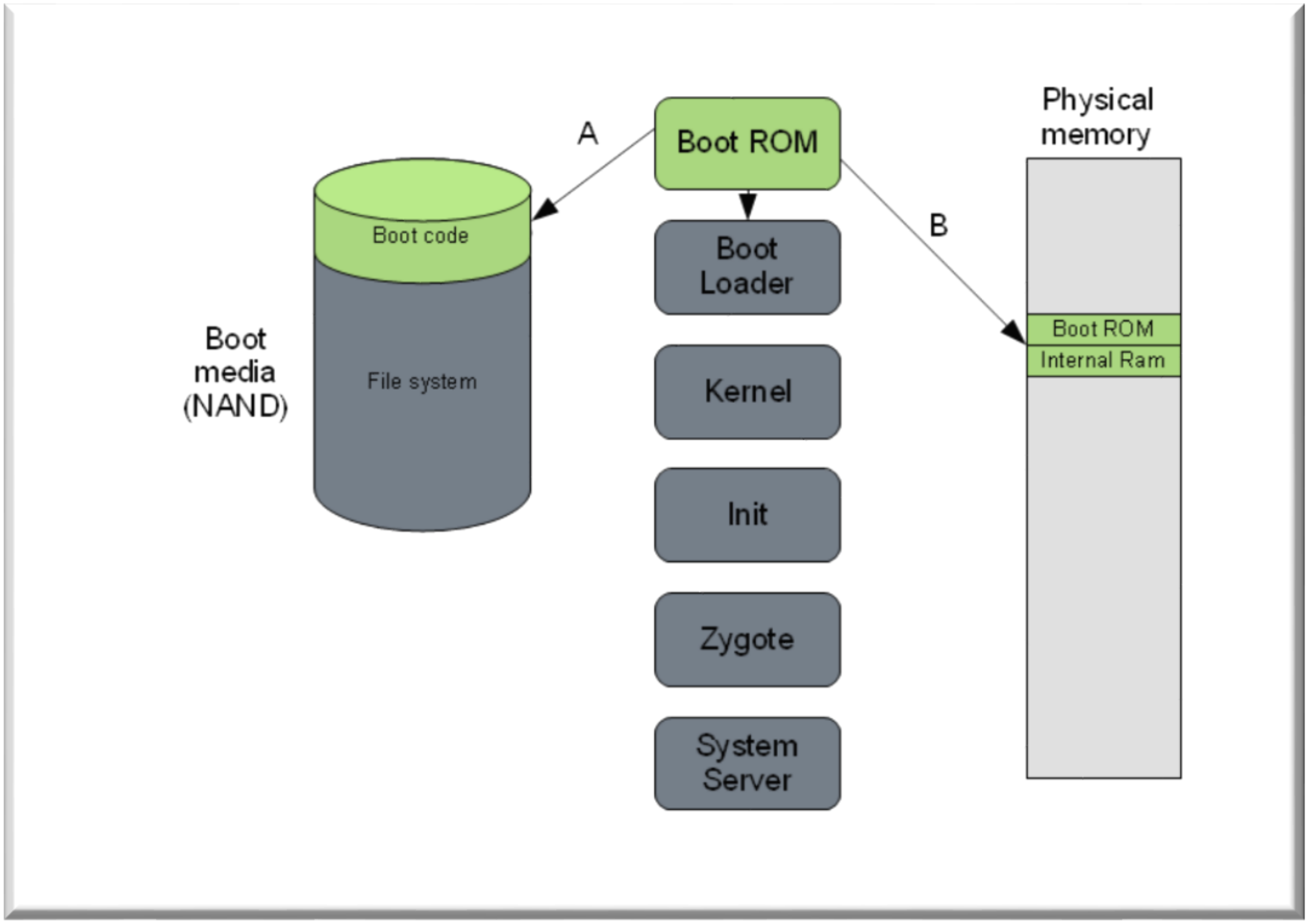
ROM & Boot Loaders

- ROM varies by manufacturer
- Contains boot process
- seven key steps to the Android boot process:
  1. Power on and on-chip boot ROM code execution
  2. The boot loader
  3. The Linux kernel
  4. The init process
  5. Zygote and Dalvik
  6. The system server
  7. Boot complete

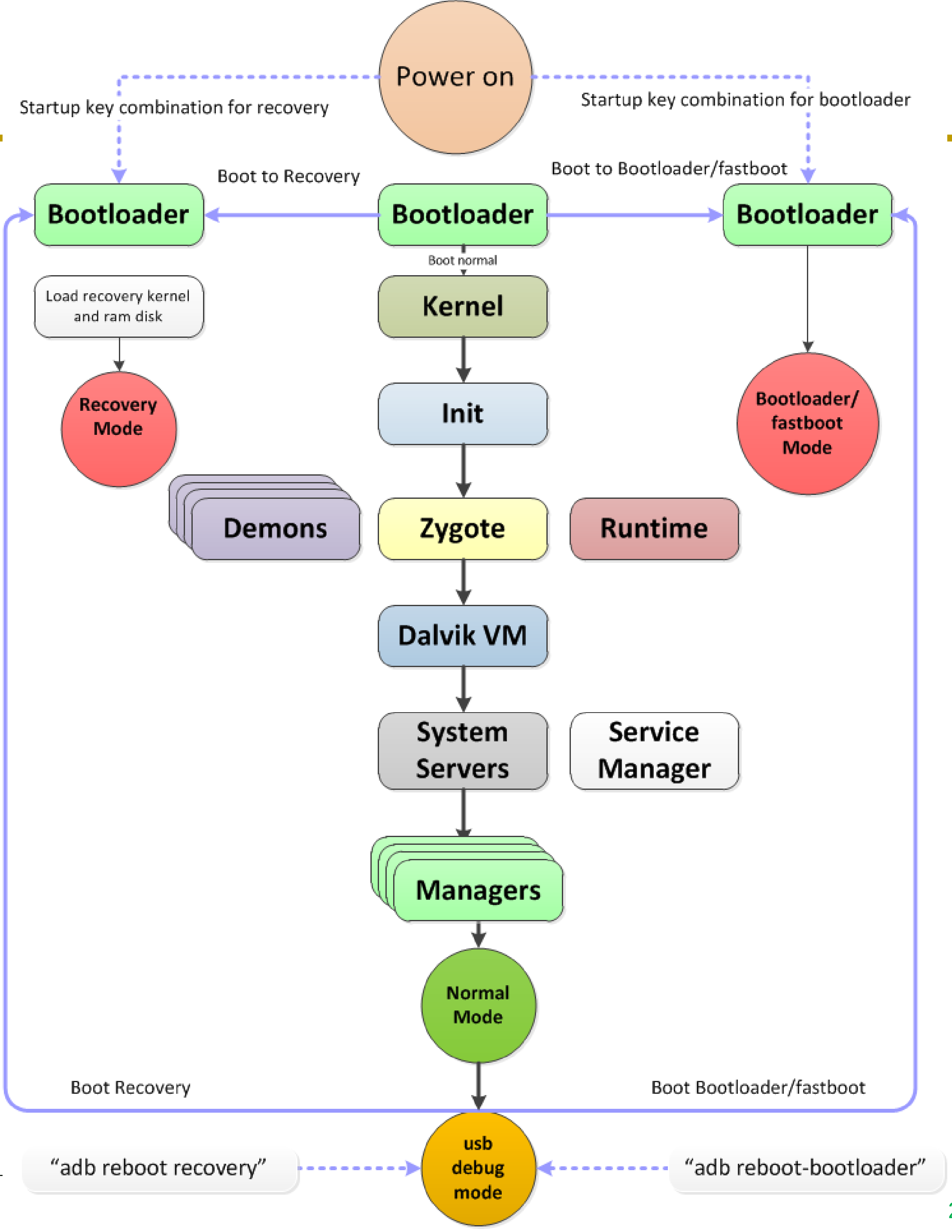
- **Boot loader** – separate from Linux kernel; the boot loader has two distinct stages: the initial program load (IPL) and the second program loader (SPL).
- **Init** - starts key system and user processes; similar to the /etc/init.d scripts found on traditional Linux devices; init.rc is typically located on the root file system and provides the kernel with the details on how to start core services.

Source: "The Android boot process from power on" by Mattias Björnheden of the Android Competence Center at Enea

ROM & Boot Loaders



Android bootloader, fast mode and recovery mode



Recovery Mode

- Designed as an avenue for manufacturers to deliver and apply system updates
- Recovery partitions offer shell access and root permissions
- When booting into recovery mode, pass codes are circumvented

- Not on all stock or manufacture provided ROM’s
- Not all recovery partitions are equal

- Most custom ROM’s and some manufacturer ROM’s contain a recovery partition in the NAND flash

- Devices without user accessible recovery partition will have a screen similar to this when attempting the recovery mode key combination
- Reboot in Recovery (T-Mobile MyTouch 4G with CyanogenMod 7.10 ROM)
- Press and hold power off button
- Choose Reboot from Phone options menu
- Choose Recovery from Reboot phone menu

Recovery Mode

- Recovery Mode Techniques

Device	Key Combination
Motorola Droid X	Power off. Hold Home and press power butt on. Release power. When (!) displays release Home. Press Search button. (needs more re search)
HTC Incredible	Hold volume down and press power button. Use volume down to select recovery and pre ss power button.

- **Passcode Circumvention Recap**
  - If device is on and passcode protected, connect to USB and attempt ADB access.
  - If pattern lock is present (and you have access to lighting and camera), attempt smudge attack.
  - If those fail, attempt to reboot into recovery mode.
  - If device is off, attempt boot into recovery mode.

  - More advanced techniques include flashing the recovery partition with a custom ROM, or utilizing exploits in boot loaders. Many possibilities exist; time limits us from diving into each.

Why Root?

- To improve the performance of an old phone
- Rooting enables all the user-installed apps to run privileged commands that are typically unavailable to the devices in their stock configuration (Ex- LUCKY_PATCHER).
- After you have a root access over your phone, you could browse your Android root folder and make any changes to the system files on your phone.

☞ **Rooting can give you a lot of advantages**

- By rooting your Android device and getting super-user access, things can only get better for you.

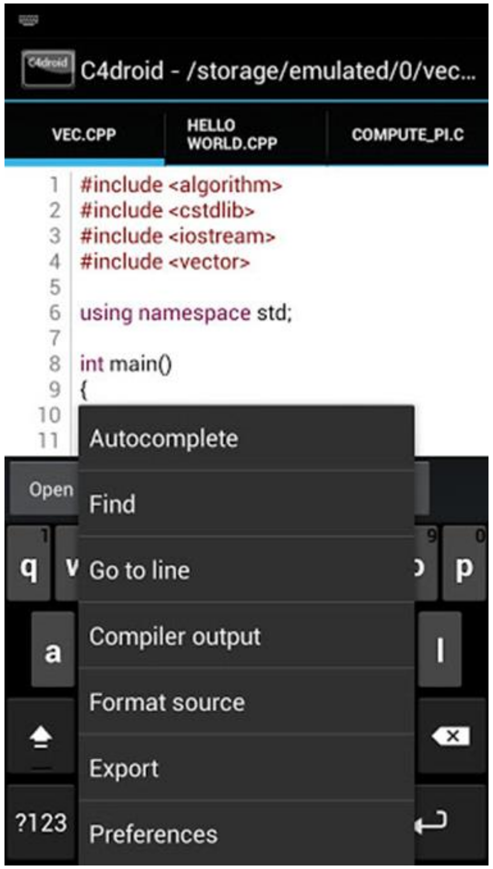
Benefits of SU(SuperUser) Abilities

- **Back-up and Tethering**

- **Custom ROMs**
  - Able to defy GUI restrictions
  - IDEs (Interactive Development Environment)
    - IDEs provide a centralized software in which a programmer can modify, compile, deploy and debug software
    - The IDE to be most flawless is C4droid
    - Programming by phone or computer, the results are the same.
  - Emulators
  - Access to more tools and utilities

- **Performance Optimization**

- **Early updates**



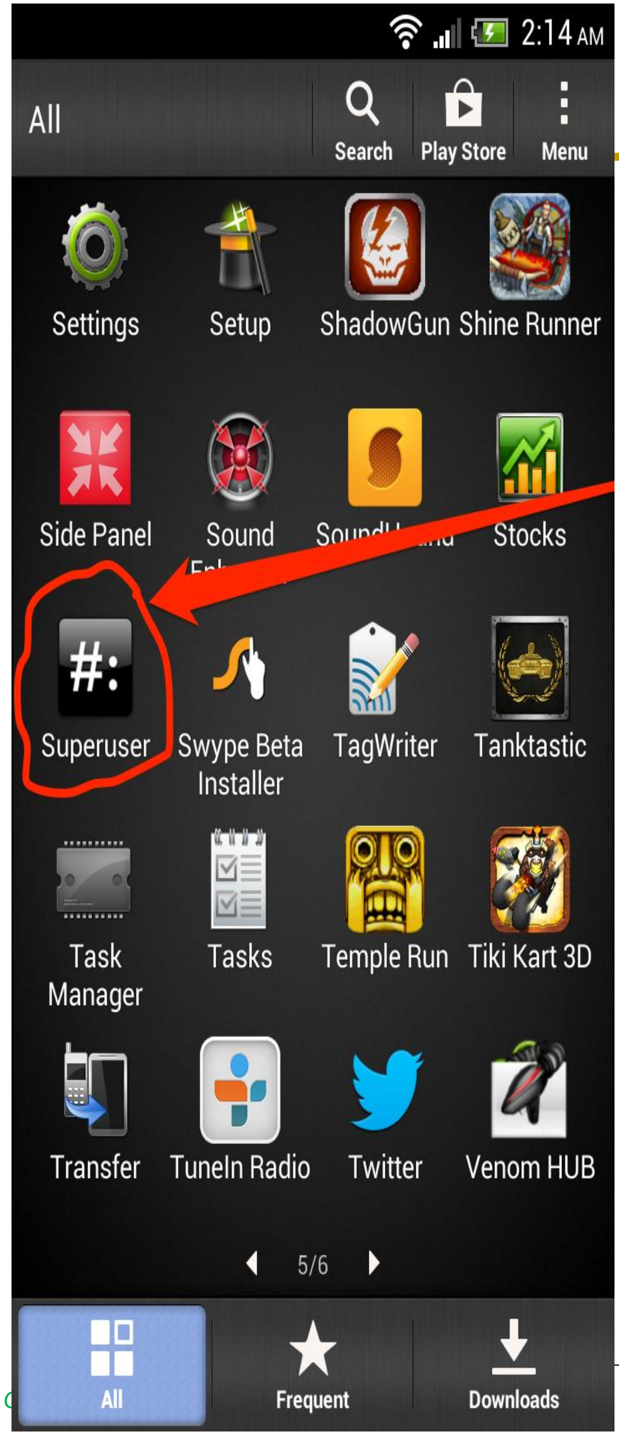
```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <vector>

using namespace std;

int main()
{
```

Benefits of SU(SuperUser) Abilities

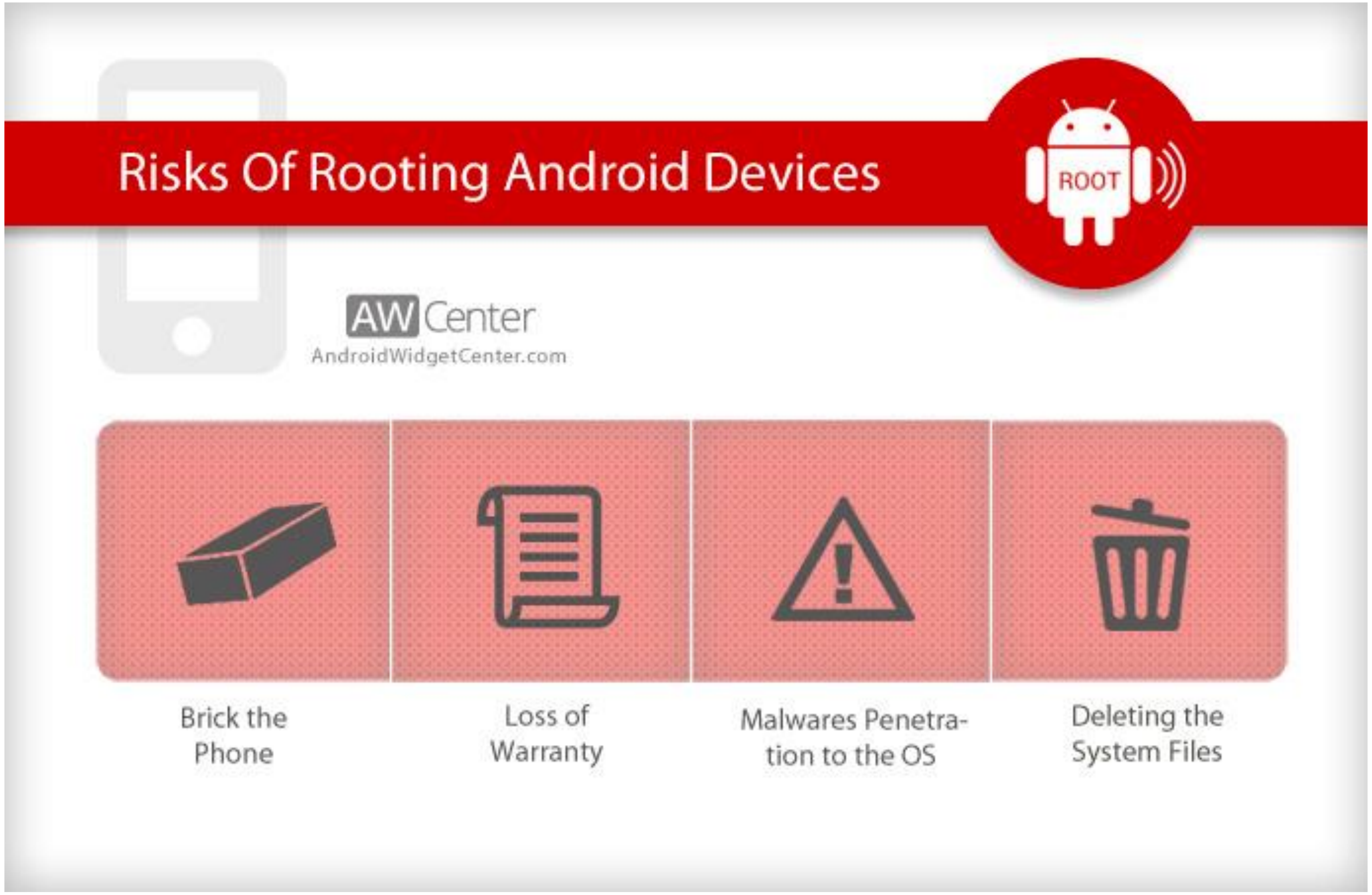
- **Emulators**
  - Software that duplicates the desired system in another system
    - The Guest (duplicated system) must be a less intricate system so the host system can process it
  - Often for convenience purposes, having multiple systems integrated into one device
  - Can be used for professional manners such as a TI calculator emulator. Or could be used to re-live the childhood memories with emulators dating from the Atari to the PlayStation one



Rooting & Why root?

- This is usually how you can see if an android phone is rooted
- **Advantages**
  - Control your CPU: Scaling on-demand
  - Flash a custom ROM: CyanogenMod 10.1
  - Flash a custom Kernel
  - Remove pre-installed bloatware
  - Backup for seamless transitions
    - Titanium Backup for your Android: Scheduled, Automated, …
  - Use low end device with high end performance
  - Completely block advertisements in any app
  - Automate everything with tasker
  - Dual Boot ROMs
  - Move apps to SD card (Partition SD card)

Risks of Rooting


Risks Of Rooting Android Devices
ROOT
AW Center
AndroidWidgetCenter.com
Brick the Phone
Loss of Warranty
Malwares Penetration to the OS
Deleting the System Files

- Root makes the device vulnerable to many exploits
- Problems with official updates
- Gaining root will change data on the device, possibly altering evidence. (in Forensics)

Precautions

1. **Before doing anything - Make a backup before flashing a new ROM**
   - Backup your Android device (Nandroid using recovery like – TWRP)
     - Make a backup of your current ROM
     - SMS, call logs text messages etc. can be erased so don't take chances!
   - Battery is a must 70% or above

2. **Install necessary driver for your Android device**
3. **Find a suitable rooting method**
   - Look for exact match for your phone model number which searching for guides, ROMs, Kernel, etc.
4. **Every device is different but the basic premise is you need to**
   - unlock bootloader
   - flash a custom recovery which will allow you to
   - install Superuser APK

5. **Know how to unroot**

   If anything goes wrong you'll be able to reflash a ROM you know works.

Detecting a rooted device

- */system/xbi/su* file

  $ which su

- Monitoring */system* partition
  - Unmount, mount, …

- Blacklisted rooting traits
  - File paths, package names, commands, keywords

- …