brightsight®

the number one
security lab
in the world

Android Security

Domain Separation (B17)

Shiqi Li / Brightsight

Android Domain Separation

Android take advantages of security features offered by Linux Kernel:

- **Sandboxing**: Applications are entities managed by permissions (unlike traditional Linux where there are users)
  - **Processes isolation**: through permissions based on UID,GID
  - **Filesystem** (data directory): User resources isolation
  - **Shared Resource**: same UID
- **SELinux**

Android Sandboxing

User IDs (UID) are assigned when an Android package is installed by the package manager.

Note /data/system/package.xml file contains info about every installed application

```
<package name="com.example.hello" codePath="/data/app/Hello.apk" …
         userId="10051">
      <sigs count="1">
        <cert index="4"  key="3082030d30820…37cf0aa3a31243230f4e48f"/>
      </sigs>
</package>
```

PackageManager `packages.xml` file

Process Sandboxing

Process level

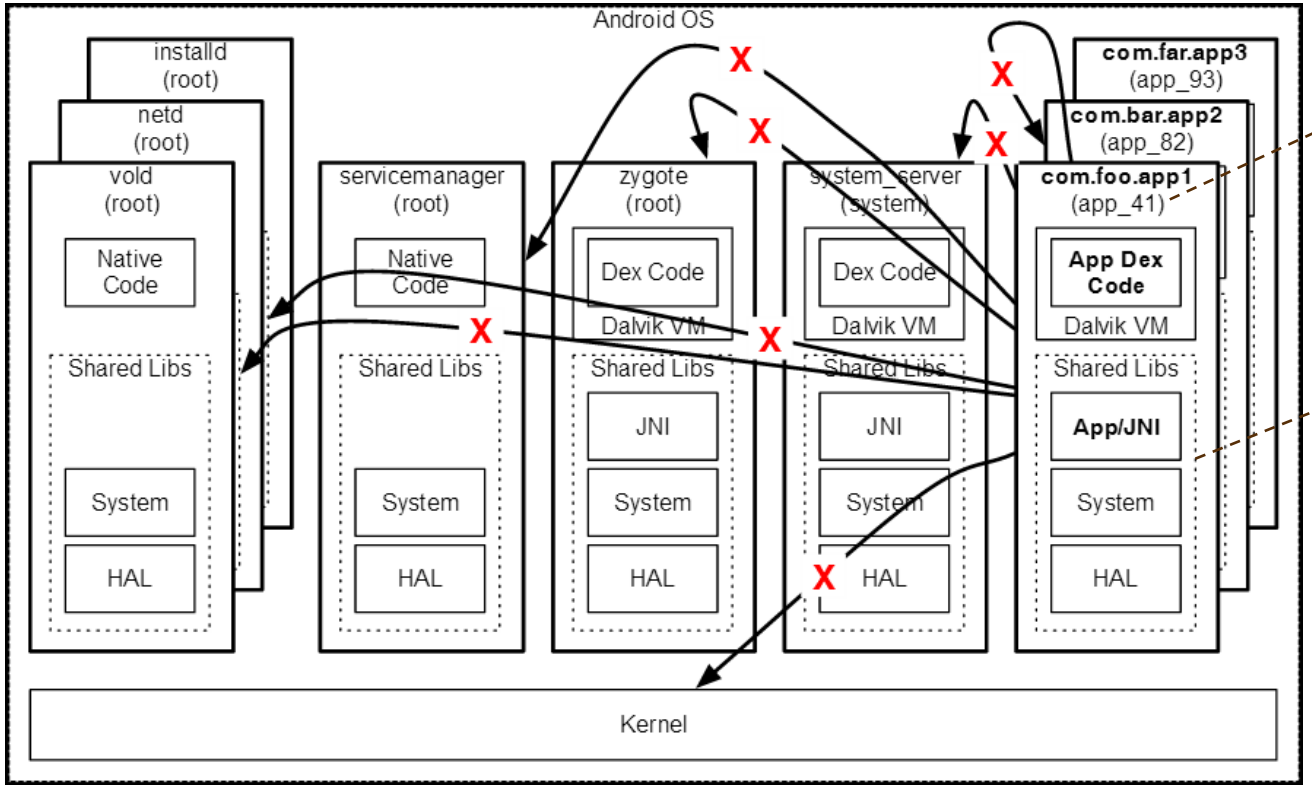
- An **UID** and **GID** are assigned to each application during installation
- Each app runs with a dedicated process using UID
- Systems daemons run under well-defined UID and only a few run as root user
- No /etc/passwd → android_filesystem_config.h
- System services: AID_APP (1000)
  - Apps: app_XXX or uY_aXXX
    - *e.g. UID 10037 → username u0_a37 → google_mail → com.google.android.email package*

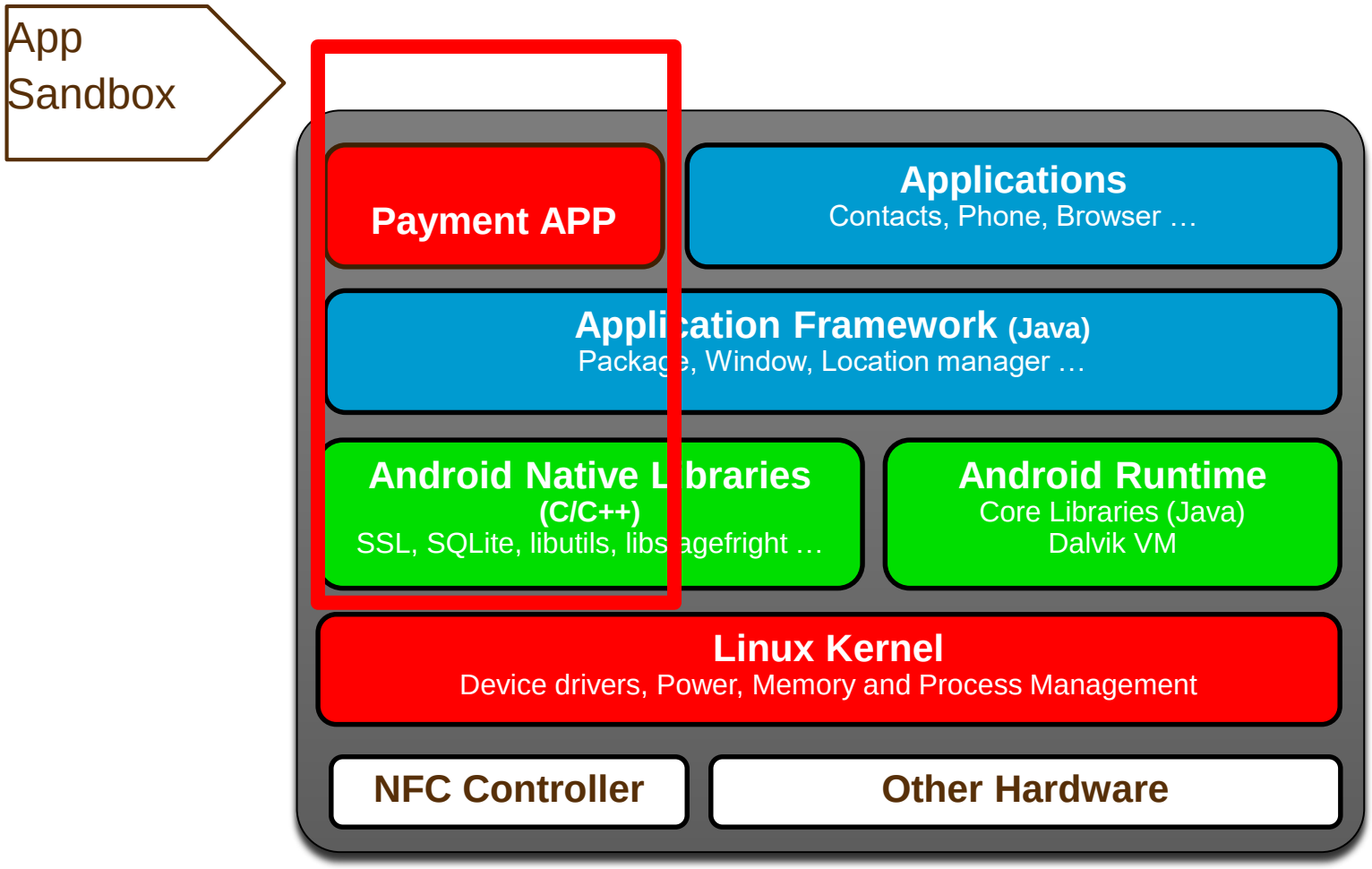
```
$ ps
--snip--
u0_a37    16973 182   941052  60800 ffffffff 400d073c S com.google.android.email❶
u0_a8     18788 182   925864  50236 ffffffff 400d073c S com.google.android.dialer
u0_a29    23128 182   875972  35120 ffffffff 400d073c S com.google.android.calendar
u0_a34    23264 182   868424  31980 ffffffff 400d073c S com.google.android.deskclock
--snip--
```

Application Sandboxing

By default, app cannot adversely affect other apps, processes

- E.g. read/write user data, modify other apps '/system' files and settings, access network, keep the device awake, etc.




App
Sandbox
Payment APP
Applications
Contacts, Phone, Browser …
Application Framework (Java)
Package, Window, Location manager …
Android Native Libraries
(C/C++)
SSL, SQLite, libutils, libstagefright …
Android Runtime
Core Libraries (Java)
Dalvik VM
Linux Kernel
Device drivers, Power, Memory and Process Management
NFC Controller
Other Hardware

Benefits of Application Sandboxing

In 2008 the Web browser vulnerability discovered

- By Charlie Miller, Mark Daniel, and Jake Honoroff of ISE
- The first commercial Android phone, the T-Mobile G1 by HTC, shipped with the vulnerability

The impact:

"...Upon visiting the malicious site, the attacker can run any code they wish with the privileges of the web browser application...

The Android security architecture is very well constructed and the impact of this attack is somewhat limited by it. ... attacker will have access to any information the browser may use, such as cookies ... saved passwords, etc. ... However, [he] can not control other, unrelated aspects of the phone, such as dialing the phone directly. This is in contrast, for example, with Apple's iPhone which at that time does not have this application sandboxing feature..."

http://securityevaluators.com/knowledge/case_studies/android/index.php

File Sandboxing

File level

- Applications have their own **private data directory**
- Permissions apply when **reading/writing** on data directories
- Application can use MODE flags to give access to other applications (*but direct access is discouraged and deprecated from Android 4.2+*)
  - MODE_WORLD_READABLE
  - MODE_WORLD_WRITABLE

```
# ls -l /data/data/com.google.android.email
drwxrwx--x u0_a37   u0_a37          app_webview
drwxrwx--x u0_a37   u0_a37          cache
drwxrwx--x u0_a37   u0_a37          databases
drwxrwx--x u0_a37   u0_a37          files
--snip--
```

Shared ID in Sandboxing

Shared user ID	• Applications can run with the same UID • They can share files and run in the same process • Non-system apps can share UID, but they need to be signed by the same code signing key
Application UID management	• All app UID are managed in /data/system/packages.xml

```
# grep ' 10012 ' /data/system/packages.list
com.android.keyguard 10012 0 /data/data/com.android.keyguard platform 1028,1015,1035,3002,3001
com.android.systemui 10012 0 /data/data/com.android.systemui platform 1028,1015,1035,3002,3001
```

Application Permissions

Application permissions

- Defined in the **androidManifest.xml** file.
- Requested at install time
- Permissions cannot be revoked

Mandatory Access Control in Android

SELinux

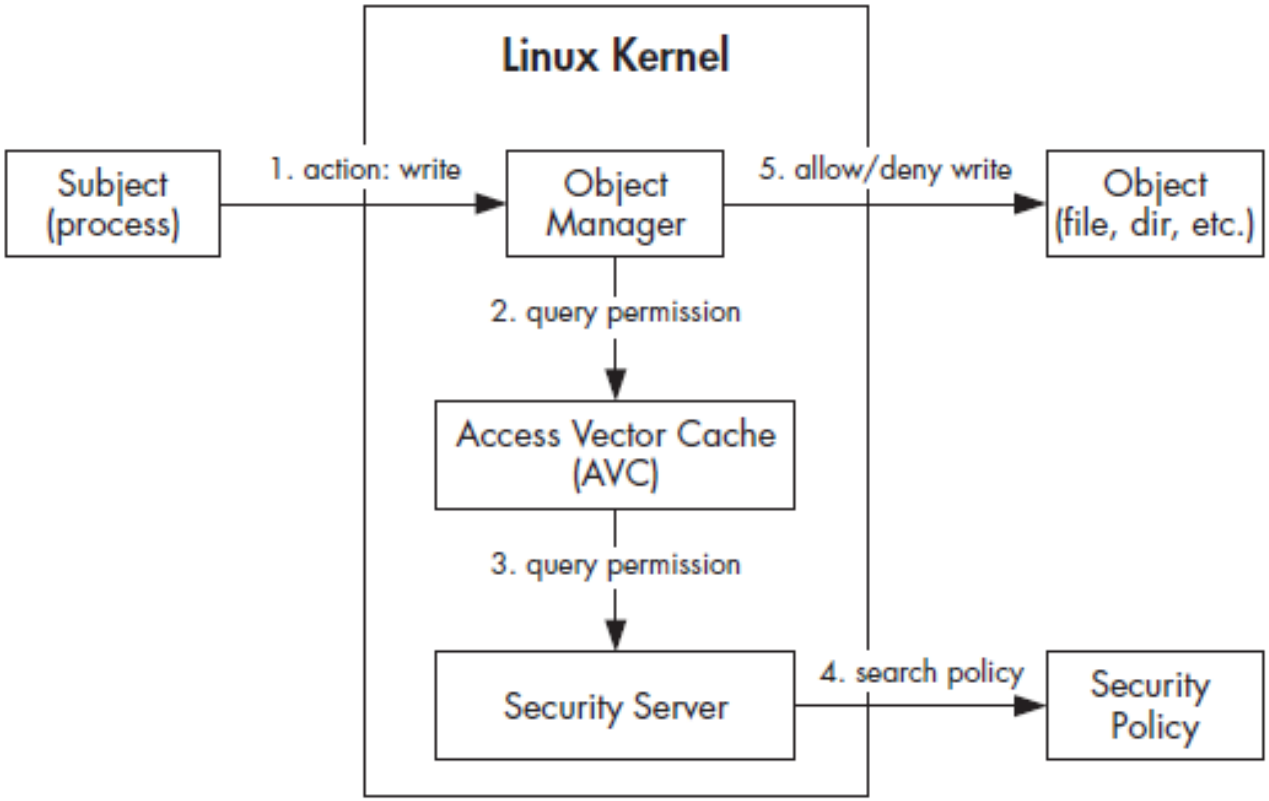
- Traditional Linux: DAC (discretionary access control)
  - Once privileges are given, they can be transferred
  - Apps could give all access privileges to others
- SELinux: **MAC (Mandatory access control).**
  - It can **isolate Core system daemons and user applications** on different domains

SELinux

- Used from Android 4.3 up
- Introduces MAC
- Default denial access control based on minimal privilege
- Modes
  - Permissive (denied permissions are logged but **not enforced**)
  - Enforcing (denied permissions are logged and enforced)
- A 4.3 Permissive mode ➔ A 4.4 Partially enforcing ➔ **A 5.0 Full enforcing mode**

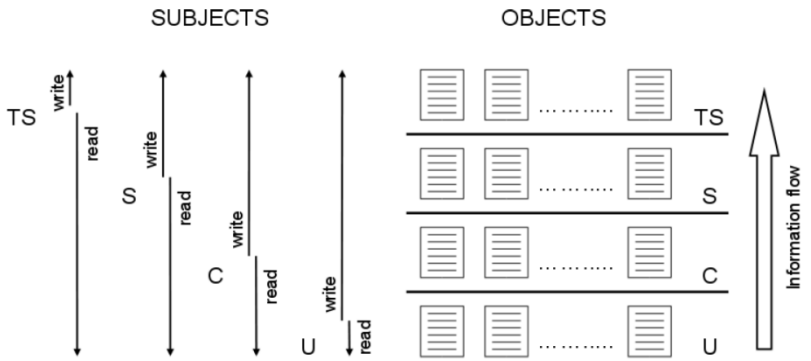
- Defines Domains (Set of identically labeled processes) to be treated equally by security policy
- Have to create & maintain security policies

SE LINUX


Linux Kernel
Subject (process)
1. action: write
Object Manager
5. allow/deny write
Object (file, dir, etc.)
2. query permission
Access Vector Cache (AVC)
3. query permission
Security Server
4. search policy
Security Policy

Mandatory Access Control (MAC)

Processes ➔ Subjects
System resources ➔ Objects



(Bell-La Padula model)

- **MAC is applied if DAC permits**
- **Bell-La Padula model**
  - Confidentiality safeguard
- **Android app-level MAC**
- In DAC permissions cannot be revoked
- SELinux policy is checked if DAC allows and action

Evaluation Concerns

- ✓ **Check Apps information**
  - ✓ UID, GID, Shared GID,
  - ✓ Data directories
  - ✓ Permissions
- ✓ **Check processes information**
  - ✓ Owners
  - ✓ UID match
- ✓ **Check SELinux configuration**
  - ✓ Modes
  - ✓ Contexts (ps –Z)
  - ✓ Policy statements

PCI PTS security concerns

DTR B17:

The evidence shall

- show if POI allows for non-firmware code execution
- show that device enforces separation FW, SW with/without security functionality
- explain the communication mechanism between separated components (applications, firmware, CPUs)
- detail mechanism for execution of ROM based config or program data
- Note the configuration and its mechanism for domain separation (e.g. SELinux configuration and rules)
- show how FW prevents execution of memory holding data object
- provide clear security guidance present in the Security Policy on application development

Questions?