Secure Architecture Principles

- Basic notions
- Access Control
- Isolation & Least Privilege

Secure Architecture Principles

Basic notions

PRIMER: BASIC NOTIONS AND TERMINOLOGY

CISPA Center for IT-Security, Privacy and Accountability

- Functionality
  - If user does ⟨**some expected input**⟩
    Then system does ⟨**some expected action**⟩
- Security
  - If a user or outsider does ⟨**some unexpected thing**⟩
    Then system does **not** do ⟨**any really bad action**⟩
- Why is security difficult?
  - What are **all** possible **unexpected things**?
  - How do we know that all of them are protected?
  - At what level of system abstraction?
    - Physical, hardware, operating system, software?

Security Objectives

- What do we want to achieve in computer security?
  - **Confidentiality**: Ensuring that information is not accessed by unauthorized persons (e.g., access control or encryption)
  - **Integrity**: Ensuring that information is not altered by unauthorized persons in a way that is not detectable by authorized users (e.g., access control or checksums)
  - **Availability**: Ensuring timely and reliable access to and use of information and preventing unauthorized withholding of information.
  - **Authenticity**: Ensuring that users are the persons they claim to be.
  - **Authorization**: What information is an authenticated user allowed to access or which operations allowed to perform
  - Different other objectives:
    - Non-repudiation, accountability, privacy, anonymity, unlinkability,…

Security objectives in Android's context

- Privacy
  - Confidentiality and integrity of personal data
    - Easy? User wants to share some data, while protecting other data
    - "If the product (=app) is free, you are the product"
      See example of advertisement libs collecting data
- Integrity of system
  - "Trusted computing base": Software (and hardware) of a system responsible for enforcing security policies.
- Integrity of applications' code and data
  - Applications and their data should not be compromised by other applications
- Confidentiality of applications' data
- Authenticity of applications
- Others: Network,…

Realizing Security in Practice

CISPA Center for IT-Security, Privacy and Accountability

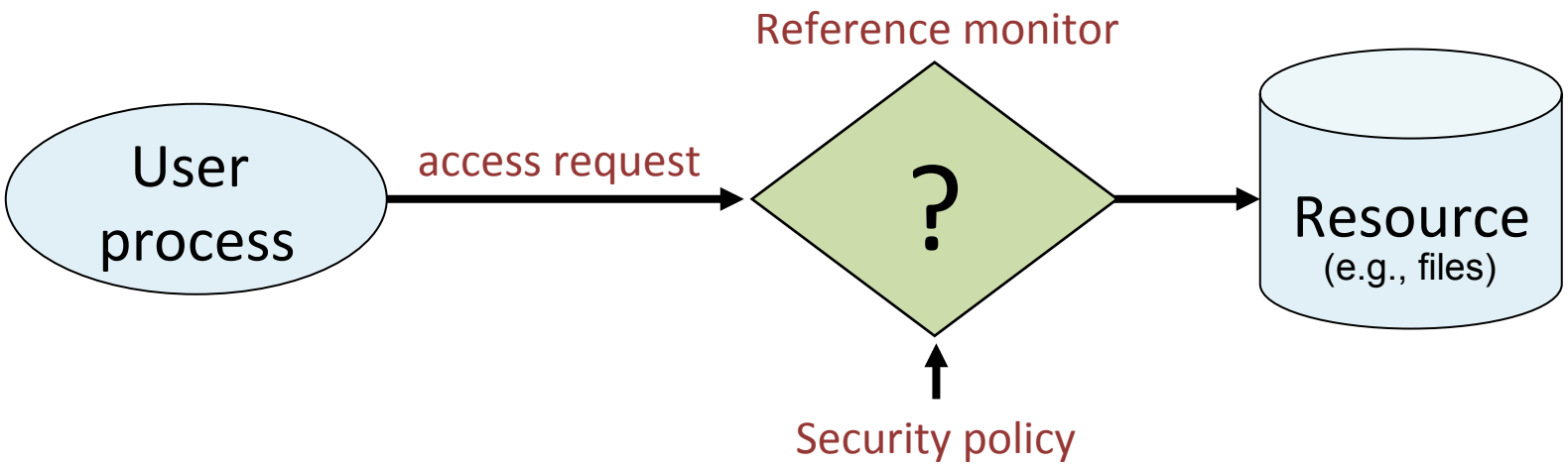
- Design can be good
- But implementation can be insecure
  - If implementation allows more actions than design, then attack can succeed as a result of implementation error
- Defining policies in natural language mostly easy:
  "Sven is the only person allowed to read the classified student files"
  - Securely realizing such policies is much harder
    - Semantic gap: Systems deal with processes and files. What does this high-level policy mean exactly in terms of the access control supported by the system?
    - Can an attacker escalate his privileges to the one of Alice to access patient files?
      - Logical errors
      - Configuration errors
      - Implementation errors

Secure Architecture Principles

Access Control

Assumptions

- System knows who the user is
  - Authentication via name and password, other credential
  - User usually interacts with system through his user processes
- Access requests pass through gatekeeper (*reference monitor*)
  - System must not allow monitor to be bypassed

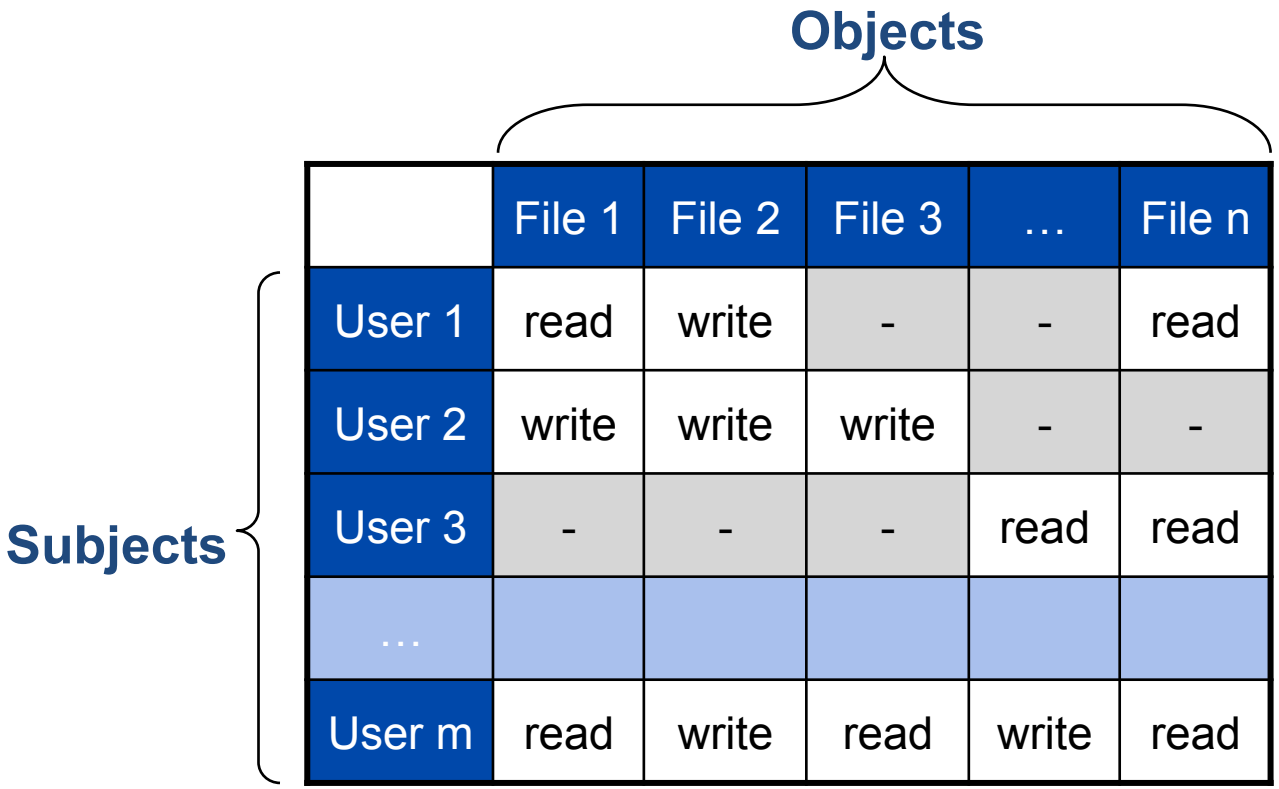


BASIC NOTIONS

- *Objects **O***
  - Passive entities that need to be protected (e.g., resources such as files, memory, ports,…)
- *Subjects **S***
  - Active principals that access objects (e.g., users, processes, group, role,…)
    - Mapping from user ➔ process:
      User's interaction with a system is done through the user's processes
- *Access rights **R***
  - Access right $r \in \boldsymbol{R}$ describes how a subject $s \in \boldsymbol{S}$ can access an object $o \in \boldsymbol{O}$
    - Read, write, execute, delete,…
- Access control function **f(s, o, r)**
  - Lookup access right *r* from ***R*** for the combination (s, o)
  - If it exists, grant access, otherwise not (*"Whitelisting"*)
    - "Blacklisting": Negate return access control decision ('everything except ***R*** is allowed)

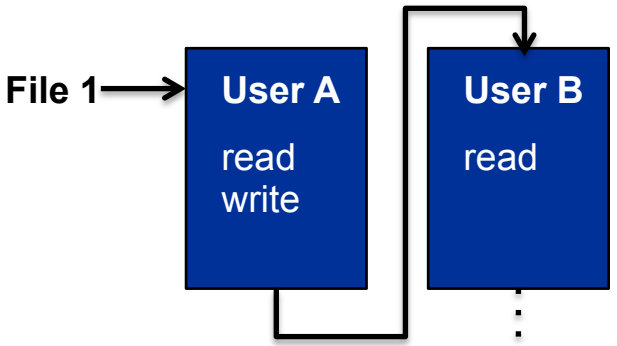
Access control matrix [Lampson]

- Store column of matrix with the resource (*objects*)
  - Example: Files
- Store row of matrix with active principals accessing resources (*subjects*)
  - Example: Users, Processes

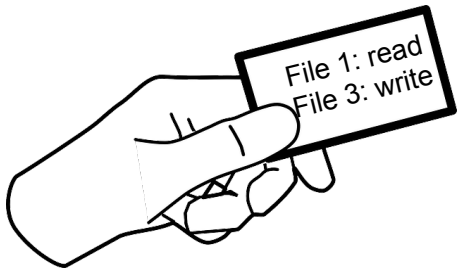


ACL vs Capabilities

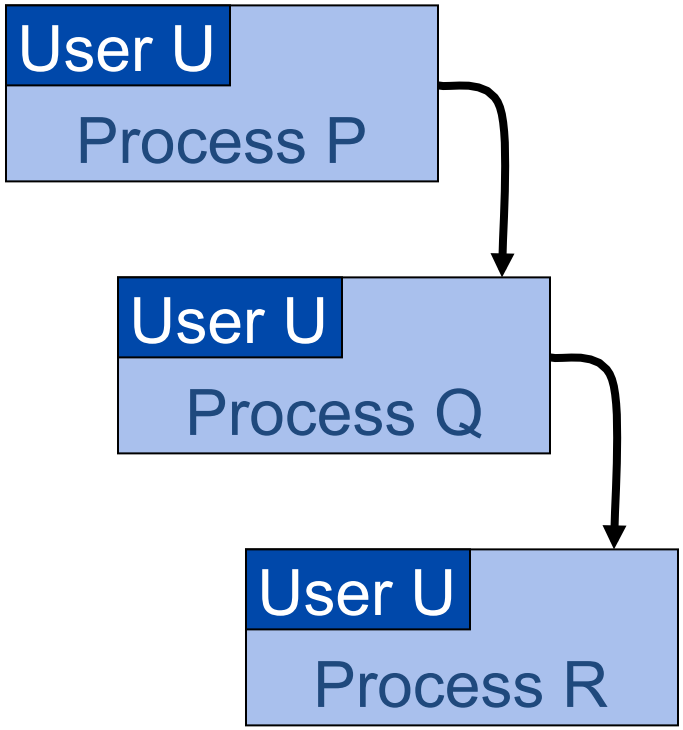
- Access control list
  - Associate list with each object
  - Check user/group against list
  - Relies on authentication: need to know user

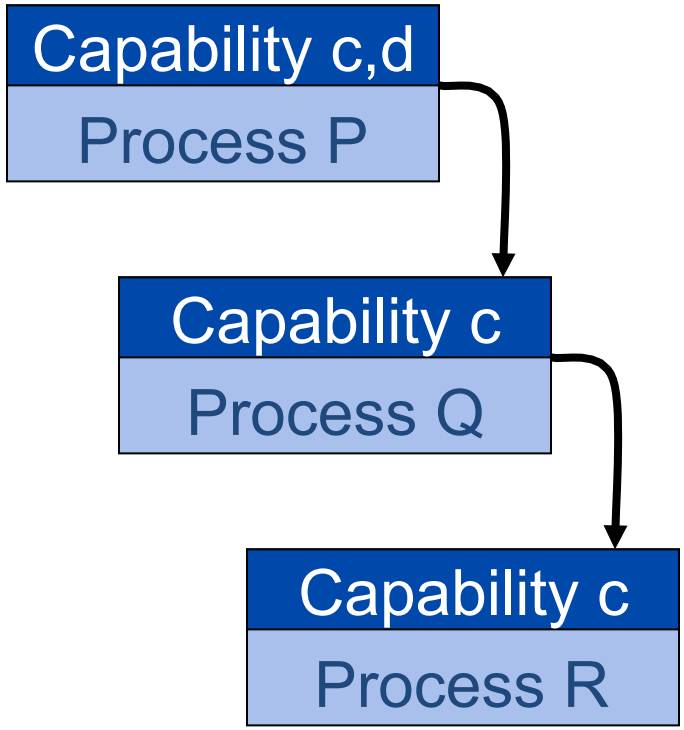


- Capabilities
  - Capability is unforgeable ticket that defines the privilege of its holder
    - Usual implementation: random bit sequence (reference), or managed by OS
    - Can be passed from one process to another
  - Reference monitor checks ticket
    - Does not need to know identity of user/process



Inheriting Access Rights and Capabilities





ACL vs Capabilities

	ACL	Capability
Delegation	Try to get owner to add permission to list? More common: let other process act under current user	Process can pass on capability at run time
Revocation	Remove/modify cell in ACL for revoked resource(s) or remove row of user from ACL	Try to get capability back from process? Possible in some systems if appropriate bookkeeping • OS knows which data is capability • If capability is used for multiple resources, have to revoke all or none … • Other details …

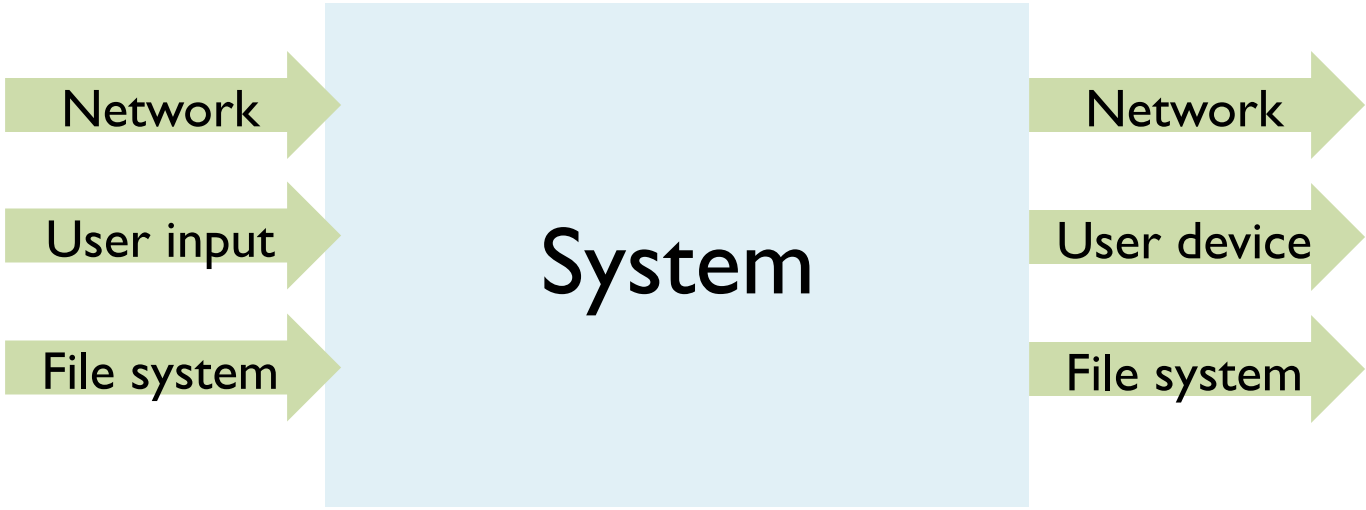
Secure Architecture Principles

Isolation & Least Privilege

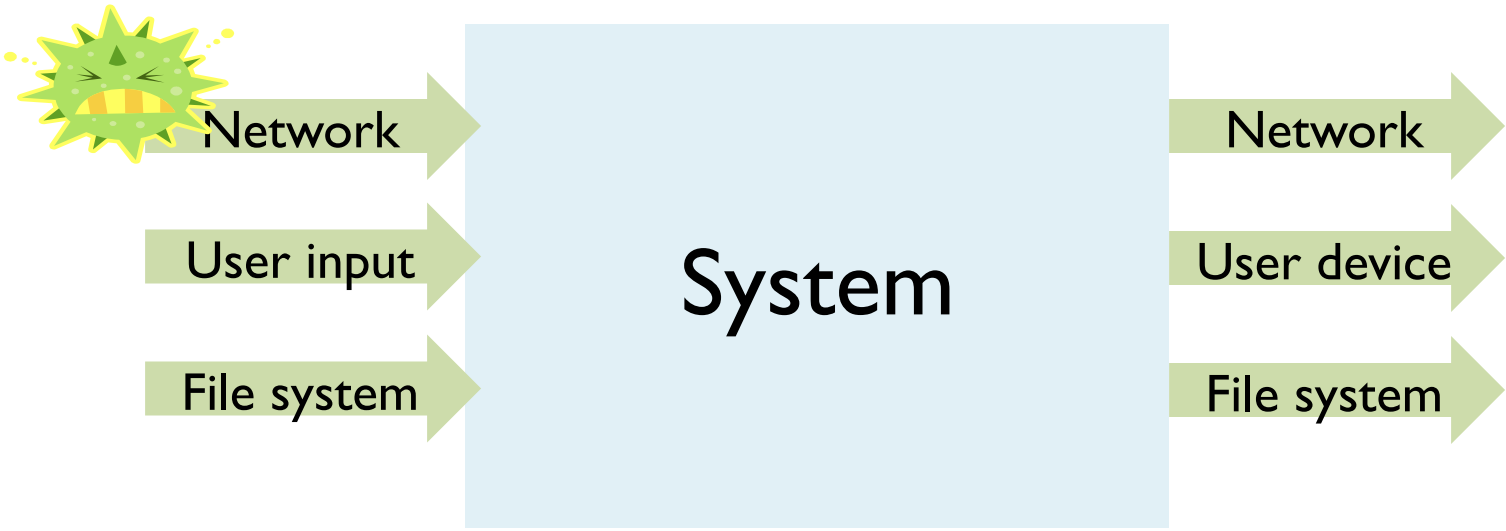
Principles of Secure Design

- Compartmentalization
  - Isolation
  - Principle of least privilege
- Defense in depth
  - Use more than one security mechanism
  - Secure the weakest link
  - Fail securely
- Keep it simple

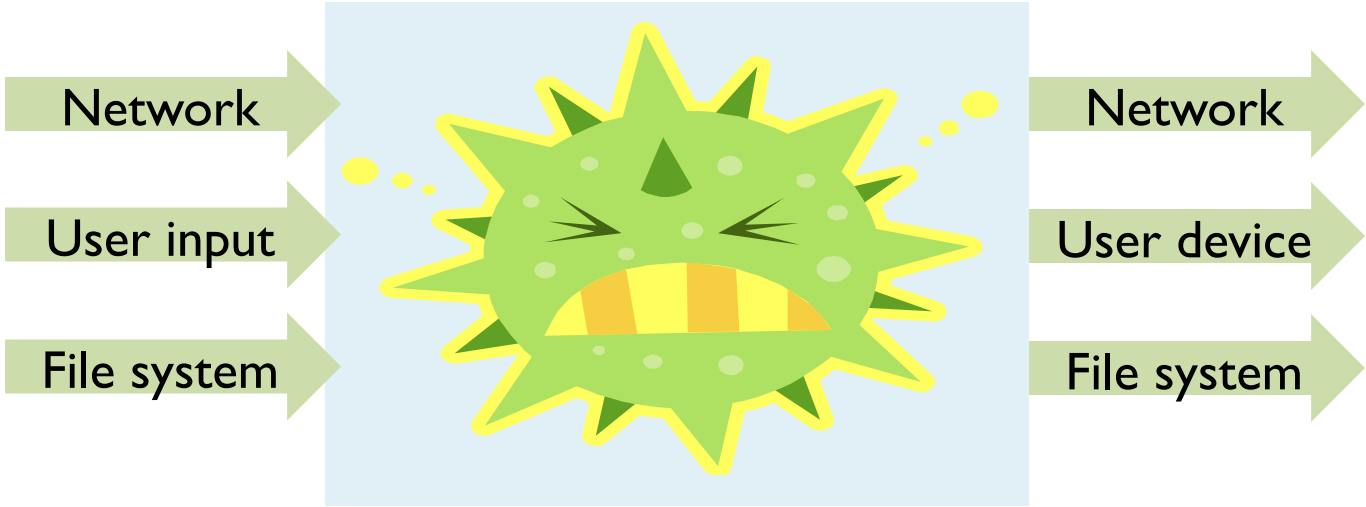
Monolithic design



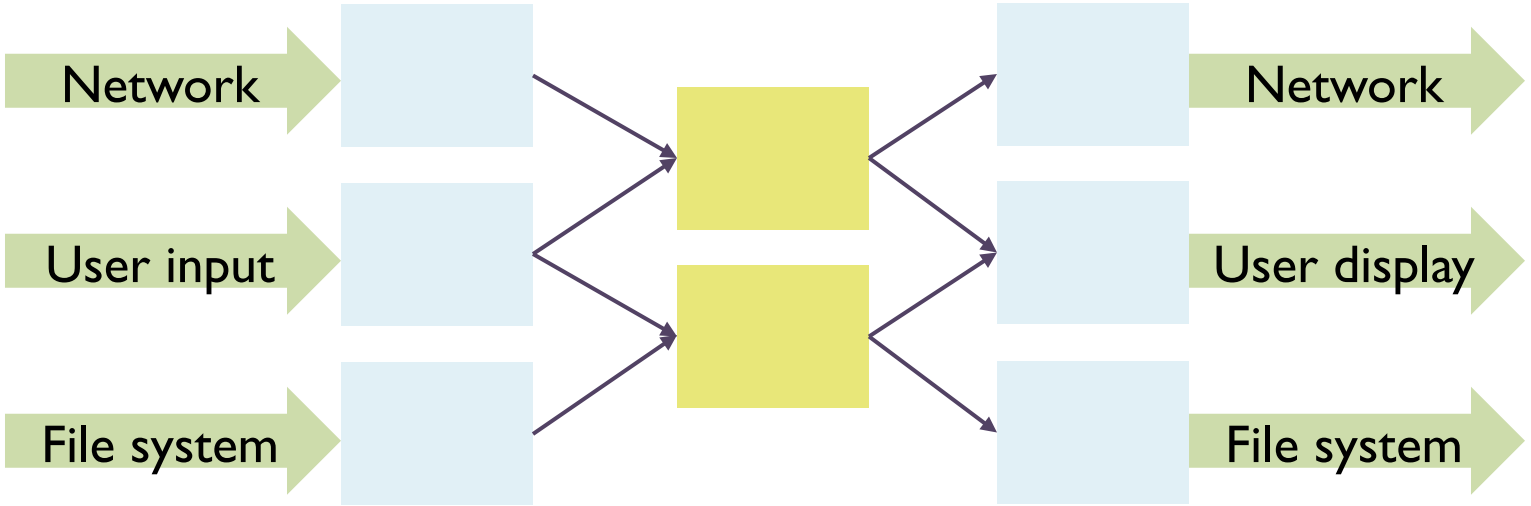
Monolithic design



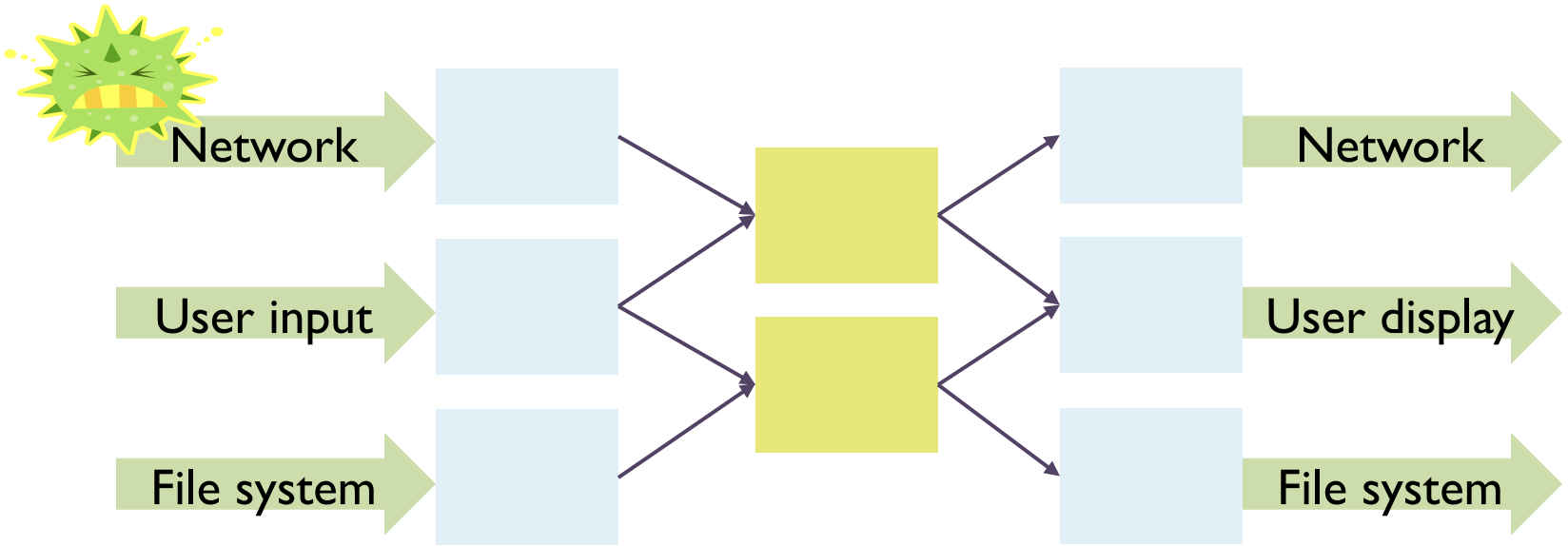
Monolithic design



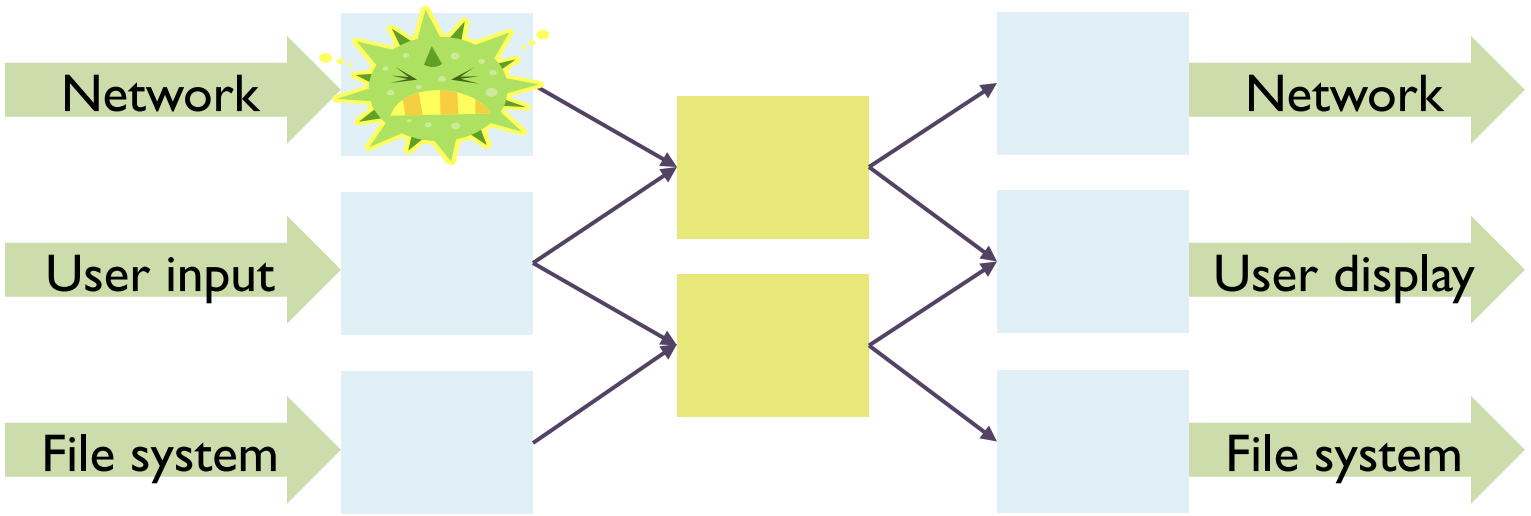
Component design



Component design



Component design



Compartmentalization possible at different levels:

- Split software into modules (e.g. SSH daemon)
- OS mechanisms (chroot, UIDs, containers, …)
- Micro-Kernel
- Virtual machines

Principle of Least Privilege

CISPA Center for IT-Security, Privacy and Accountability

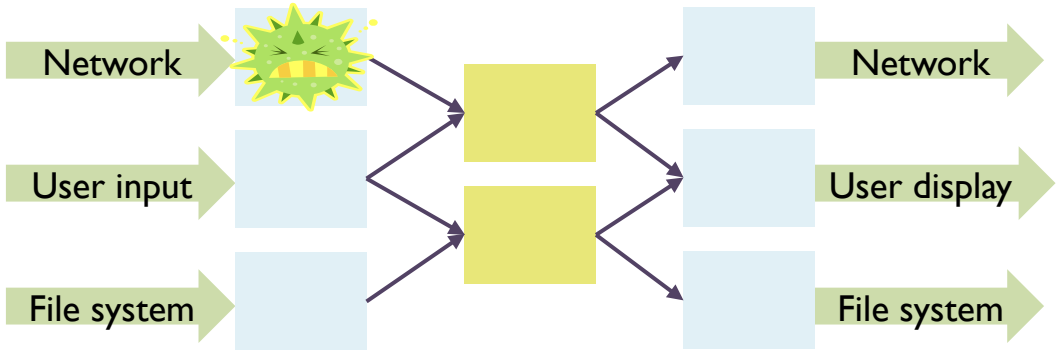
- What is a 'privilege'?
  - Ability to access or modify a resource (e.g., file, hardware, networking)
  - Privileged process: Process that has access to some resource not generally available
  - More secure systems have many types of privilege
- Assume compartmentalization and isolation
  - Separate the system into independent modules
  - Limit interaction between modules

Principle of least privilege:
A system module should only have the minimal privileges required for its intended purposes

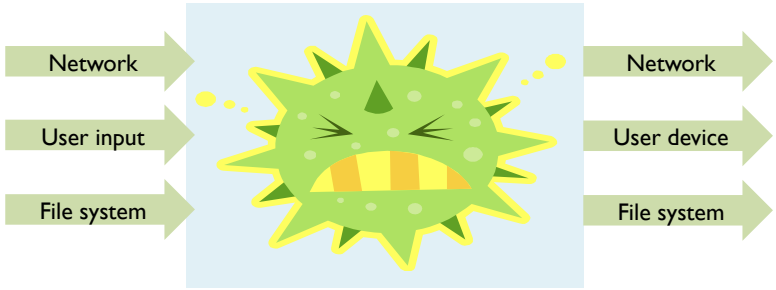
- Obvious reason: A malicious or compromised process cannot misuse privileges that it does not have!

Abstract example

- Compartmentalization helps with realizing least privilege
  - Attacker gains only “network” privilege in the compromised module



- In comparison: Monolithic design complicates least privilege
  - Attacker gains all privileges



Android Security Architecture

- Package Integrity
- Sandboxing
- Permissions & Least Privilege
- Type Enforcement

Android Security Architecture

Package Integrity

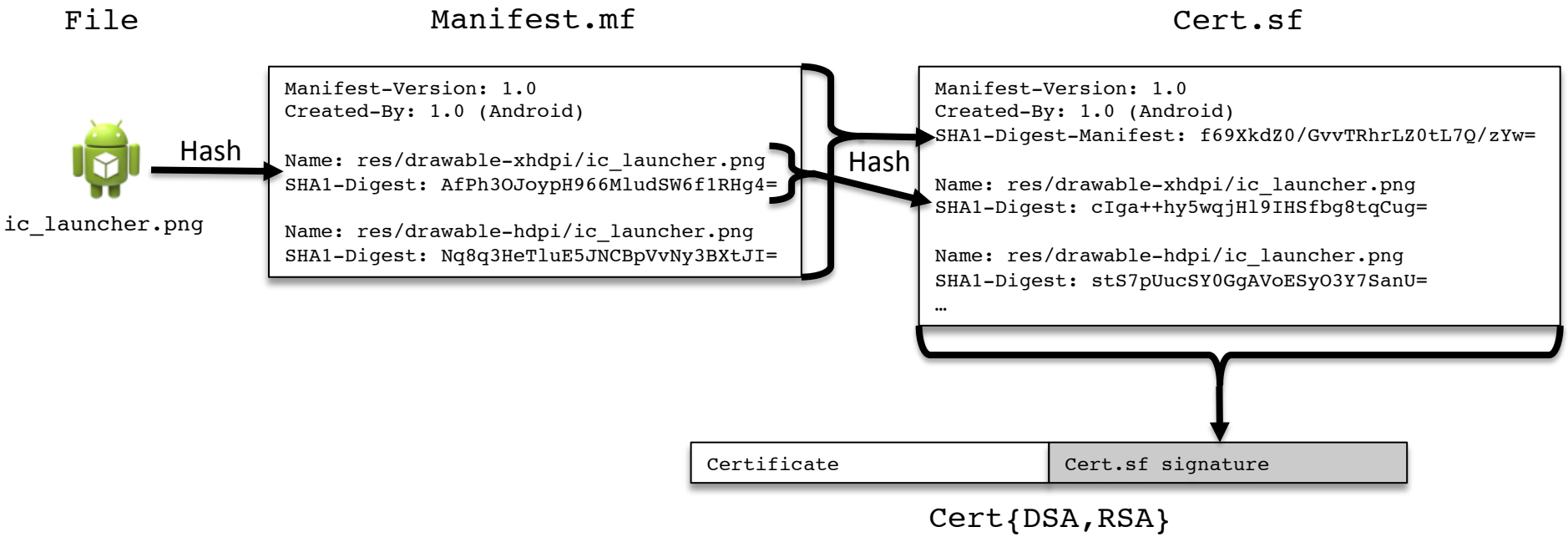
Recap: Application packages (APK)

- APK is simply a packaging format like **JAR**, ZIP, or TAR
- Components of applications
  - Activity: User interface
  - Service: Background service
  - Content Provider: SQL-like database
  - Broadcast receiver: Mailbox for broadcasted messages

- Applications can contain native code (C/C++ shared libraries) and resources (e.g., images)
  - Native code provided as shared library files that can be dynamically linked into the process
  - Resources and assets: String values, layout definitions, drawables (pictures), raw data
- META-INF contains the application certificate and package manifest
  - Package manifest not to be mistaken with the application manifest !

Package manifest

- Created with `jarsigner`
- `META-INF/`
  - `Manifest.mf`: List of files in the packages and their hash value
  - `Cert.{RSA,DSA}`: Certificate of the application (PKCS#7) and signature of the `CERT.SF` file
  - `Cert.SF`: List of resources and their signature value



http://www.manpagez.com/man/1/jarsigner/

VERIFYING OF PACKAGE MANIFEST

CISPA Center for IT-Security, Privacy and Accountability

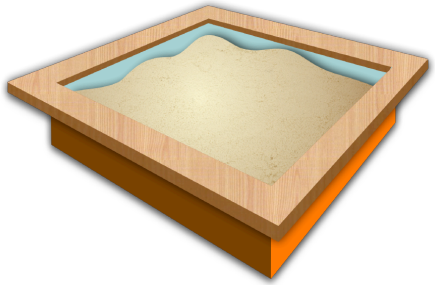
1. Verify the certificate
2. Verify signature of Cert.sf using the public key from the certificate
3. Verify entries in Cert.sf match the corresponding sections in the Mainfest.mf
4. For each file listed in the Mainfest.mf check that its computed hash corresponds to the hash value of the entry

Chain-of-Trust:
(PKI ➔) Package Certificate in Cert.{DSA,RSA} ➔ Cert.sf ➔ Manifest.mf ➔ Files

- Integrity check
  - Chain-of-trust ensures that the APK content has not been modified since the developer created the APK
- Same origin policy
  - Updates of applications only allowed when update is signed with the same developer key
  - But: Google encourages *self-signed* certificates
    - Authenticity of developer not ensured!
      - Re-signing possible
    - Trust on first install

Android Security Architecture

Sandboxing

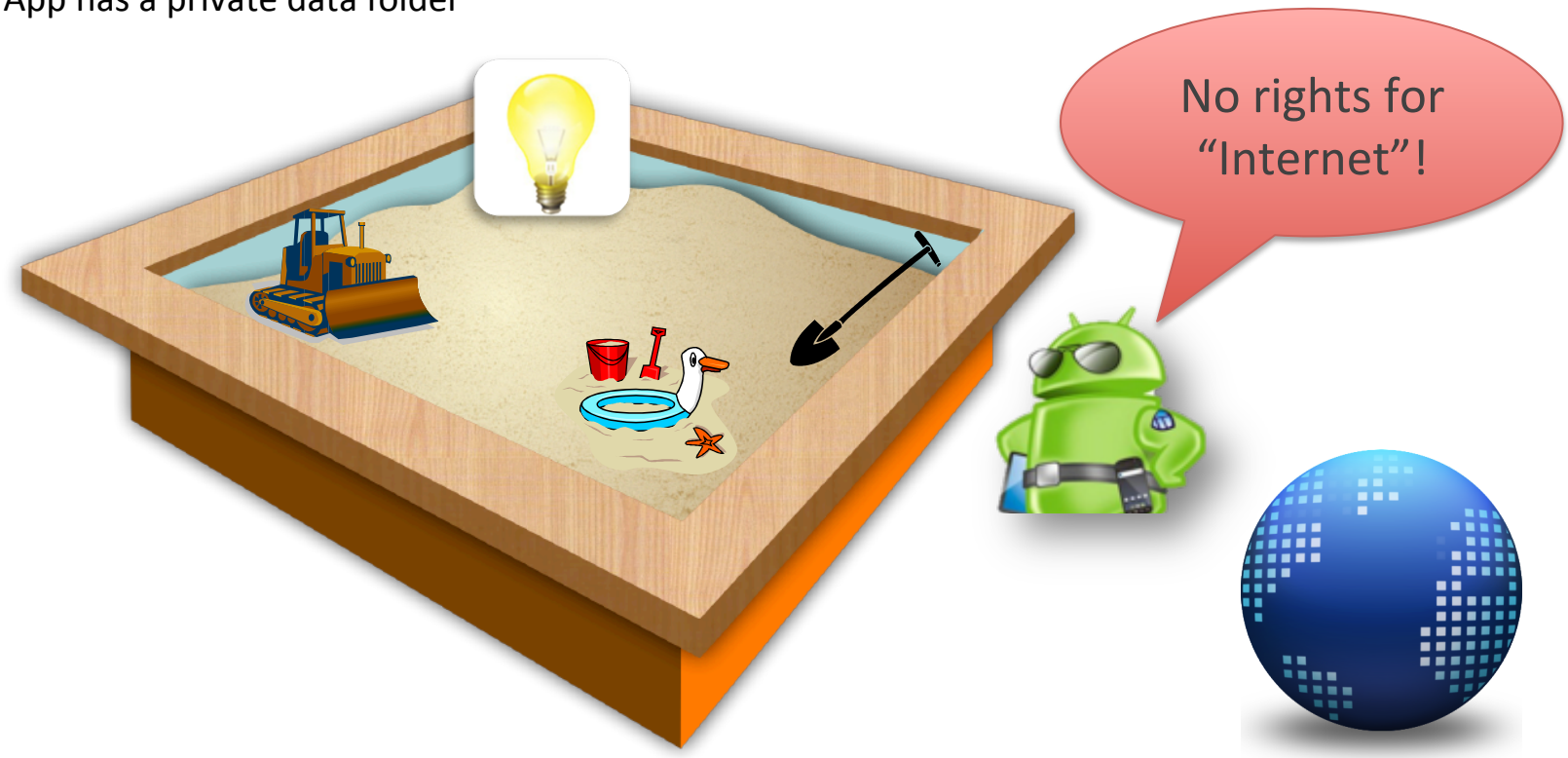


General Idea

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

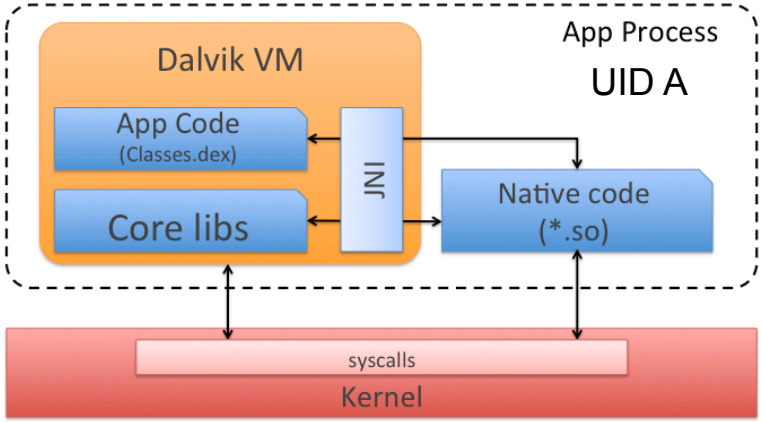
Application Isolation by Sandboxing

- Each application is isolated in its own sandbox
  - Applications can access only its own resources
  - Access to sensitive resources depends on the application's rights
- Sandboxing is enforced by Linux
  - Each App is assigned a unique UserID during installaion and runs in separate process
  - Each App has a private data folder

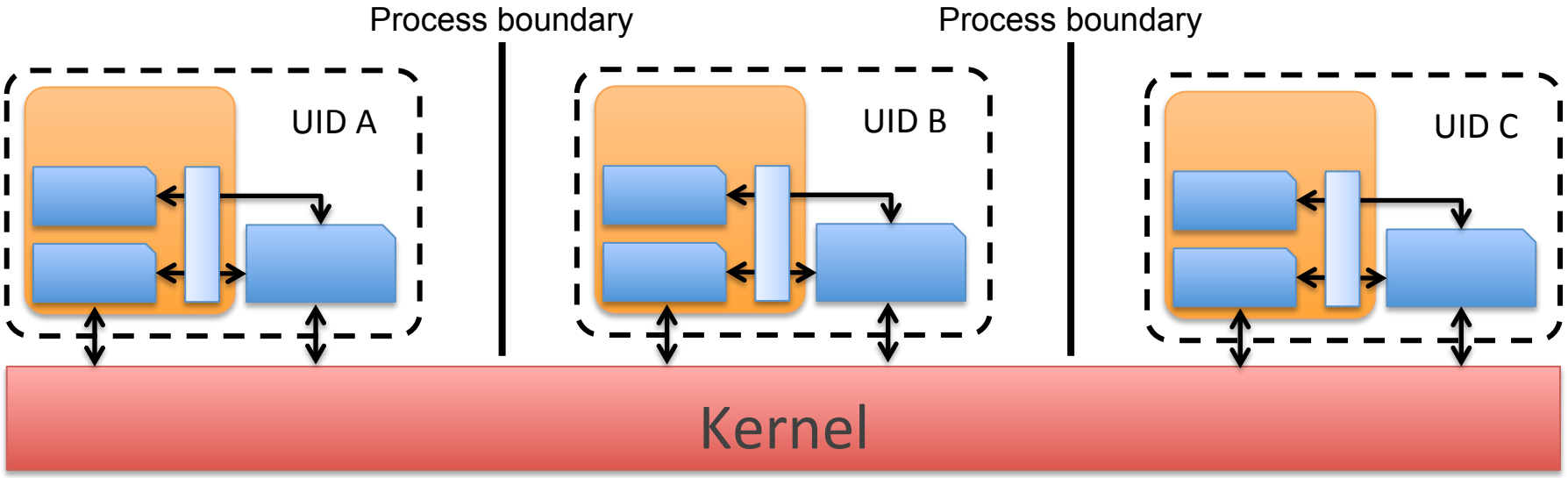


Application Sandbox

- Isolation: Android enforces isolation at the process level and relies on the underlying Linux kernel:
  Every installed app has a separate *user ID* (UID) and *group ID* (GID)
  - Each app lives in its own sandbox



JNI = Java Native Interface



Recap: Android process list

```
130|shell@grouper:/ $ ps
USER      PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      120   1     9748   1016    ffffffff 00000000 S /system/bin/netd
root      121   1     1032   236     ffffffff 00000000 S /system/bin/debuggerd
system    122   1     22776  7148    ffffffff 00000000 S /system/bin/surfaceflinger
root      123   1     678556 46020   ffffffff 00000000 S zygote
drm       124   1     12588  5112    ffffffff 00000000 S /system/bin/drmserver
media     125   1     24876  7824    ffffffff 00000000 S /system/bin/mediaserver
install   126   1     996    216     ffffffff 00000000 S /system/bin/installd
keystore  128   1     3388   1236    ffffffff 00000000 S /system/bin/keystore
root      130   1     988    416     ffffffff 00000000 S /system/bin/tf_daemon
media_rw  132   1     3544   188     ffffffff 00000000 S /system/bin/sdcard
gps       133   1     9760   2380    ffffffff 00000000 S /system/bin/glgps
shell     134   1     5616   184     ffffffff 00000000 S /sbin/adbd
root      189   2     0      0       ffffffff 00000000 S flush-179:0
root      401   2     0      0       ffffffff 00000000 S irq/210-host_sp
root      403   2     0      0       ffffffff 00000000 S irq/214-host_sp
root      410   2     0      0       ffffffff 00000000 S irq/218-host_sp
shell     483   134   1288   688     c010d328 40193790 S logcat
system    498   123   781020 58348   ffffffff 00000000 S system_server
u0_a31    580   123   724900 50892   ffffffff 00000000 S com.android.systemui
wifi      597   1     3444   2380    ffffffff 00000000 S /system/bin/wpa_supplicant
u0_a32    704   123   709992 38552   ffffffff 00000000 S com.google.android.inputmethod.latin
u0_a60    720   123   687532 21160   ffffffff 00000000 S com.nuance.xt9.input
radio     733   123   703012 29344   ffffffff 00000000 S com.android.phone
nfc       746   123   703812 25736   ffffffff 00000000 S com.android.nfc
u0_a33    760   123   720624 46996   ffffffff 00000000 S com.android.launcher
u0_a21    800   123   846560 41256   ffffffff 00000000 S com.google.android.gms
u0_a21    814   123   728632 39768   ffffffff 00000000 S com.google.process.gapps
u0_a14    882   123   695896 31168   ffffffff 00000000 S android.process.media
u0_a21    912   123   731184 37948   ffffffff 00000000 S com.google.process.location
u0_a13    1177  123   692432 26184   ffffffff 00000000 S com.google.android.deskclock
u0_a45    1262  123   719612 37348   ffffffff 00000000 S com.android.vending
u0_a20    1307  123   707500 31036   ffffffff 00000000 S com.google.android.gm
u0_a9     1407  123   695448 25968   ffffffff 00000000 S com.android.chrome
u0_a53    1426  123   717764 34820   ffffffff 00000000 S com.google.android.talk
u0_a38    1553  123   705228 32540   ffffffff 00000000 S com.google.android.music:main
u0_a7     1591  123   688408 24784   ffffffff 00000000 S com.android.providers.calendar
dhcp      1666  1     1024   468     ffffffff 00000000 S /system/bin/dhcpcd
root      1704  2     0      0       ffffffff 00000000 S irq/200-host_sp
u0_a48    1757  123   689760 22900   ffffffff 00000000 S com.google.android.setupwizard
u0_a47    1776  123   715156 52048   ffffffff 00000000 S com.google.android.apps.plus
u0_a35    1798  123   730144 34396   ffffffff 00000000 S com.google.android.apps.magazines
```

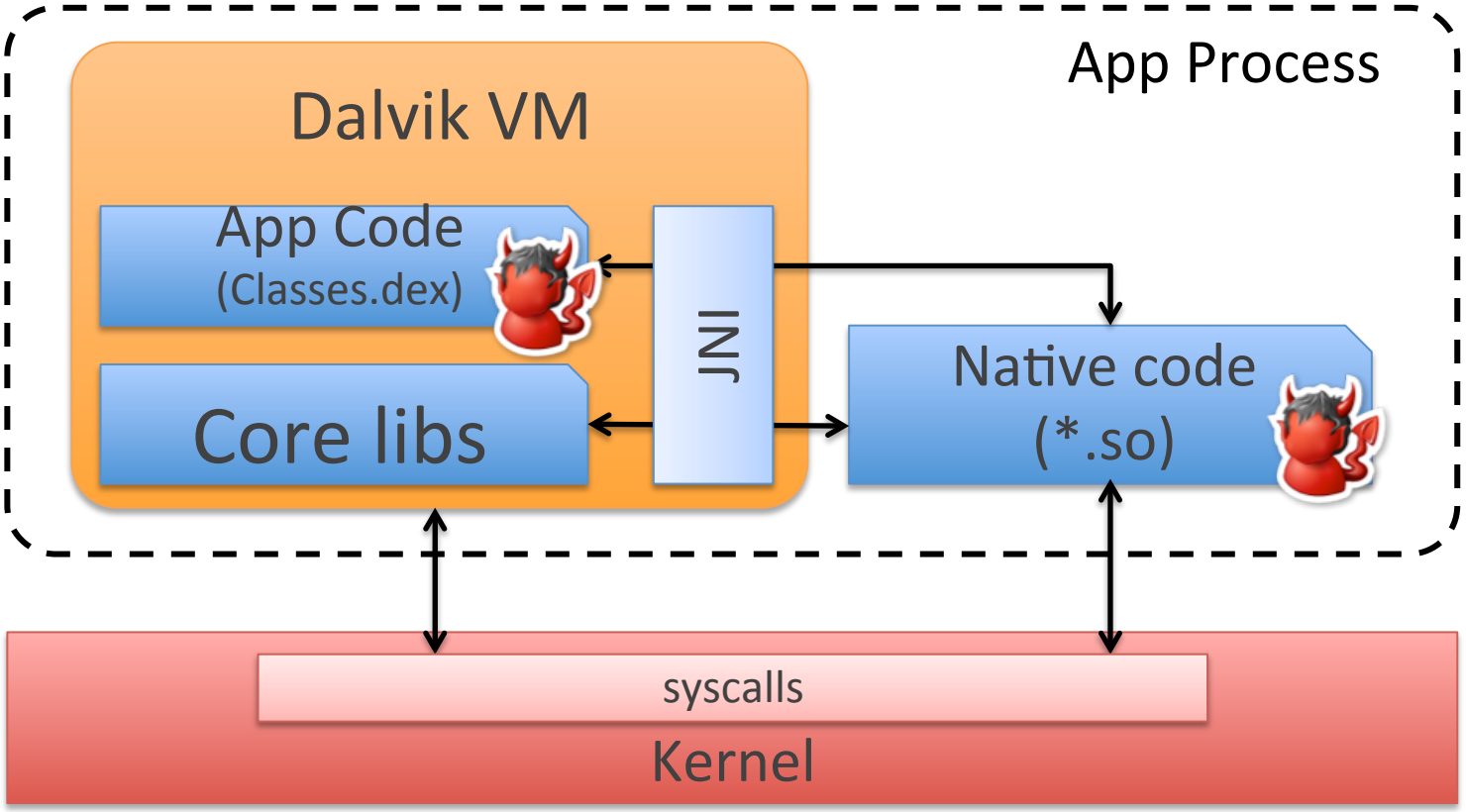
Predefined UIDs

- Application UIDs > 10000
- Predefined UIDs for core Android apps and services
  - `$(Android_root_folder)/system/core/include/private/android_filesystem_config.h`

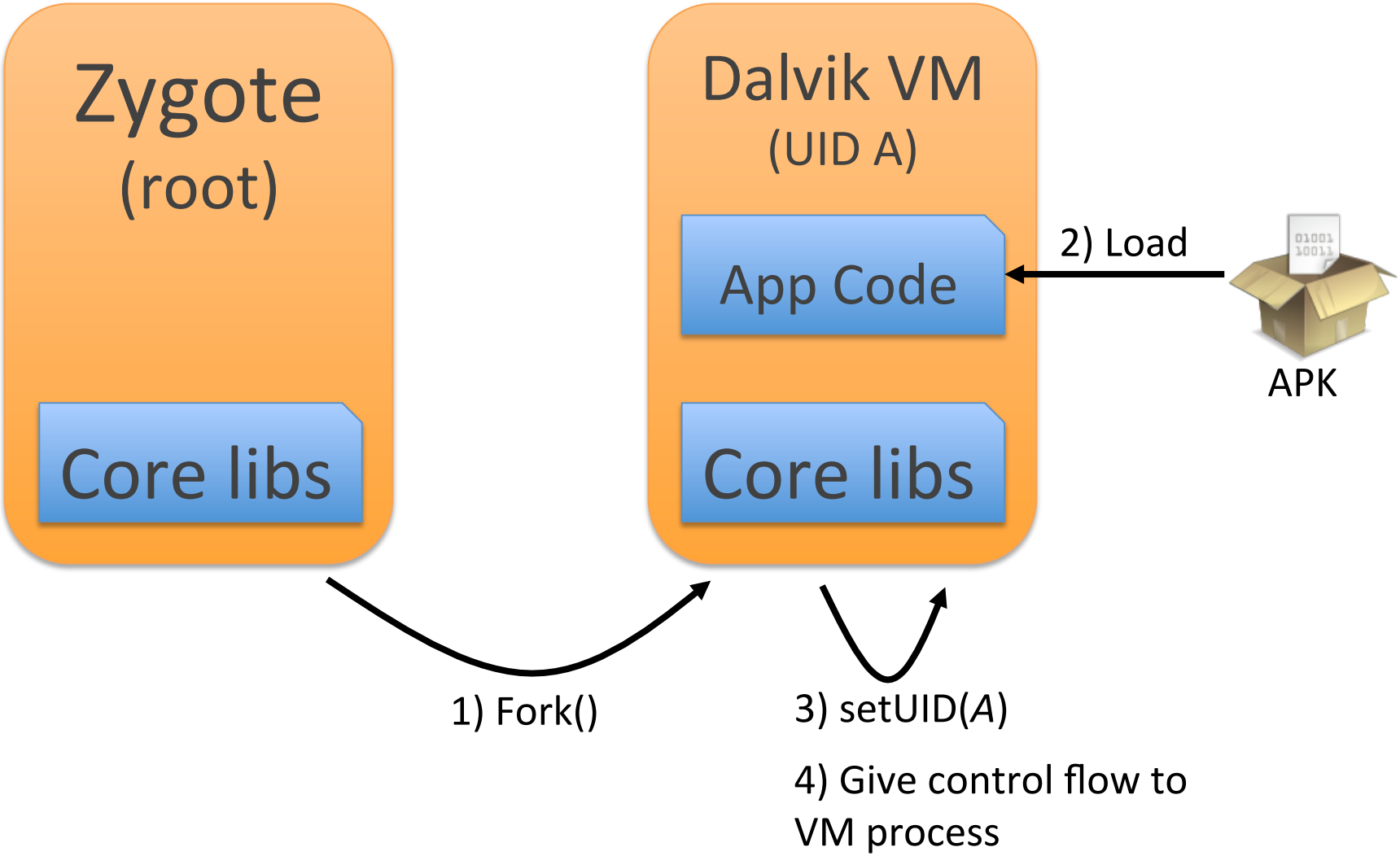
```
/* This is the master Users and Groups config for the platform.
 * DO NOT EVER RENUMBER
 */
#define AID_ROOT             0  /* traditional unix root user */
#define AID_SYSTEM        1000  /* system server */
#define AID_RADIO         1001  /* telephony subsystem, RIL */
#define AID_BLUETOOTH     1002  /* bluetooth subsystem */
#define AID_GRAPHICS      1003  /* graphics devices */
#define AID_INPUT         1004  /* input devices */
#define AID_AUDIO         1005  /* audio devices */
...
#define AID_SHELL         2000  /* adb and debug shell user */
#define AID_CACHE         2001  /* cache access */
#define AID_DIAG          2002  /* access to diagnostic resources */
/* The 3000 series are intended for use as supplemental group id's only.
 * They indicate special Android capabilities that the kernel is aware of. */
#define AID_NET_BT_ADMIN  3001  /* bluetooth: create any socket */
#define AID_NET_BT        3002  /* bluetooth: create sco, rfcomm or l2cap sockets */
#define AID_INET          3003  /* can create AF_INET and AF_INET6 sockets */
...
#define AID_EVERYBODY     9997  /* shared between all apps in the same profile */
#define AID_MISC          9998  /* access to misc storage */
#define AID_NOBODY        9999
#define AID_APP          10000  /* first app user */
```

Application sandbox

- DVM is **not** a security boundary!
  - Could be easily circumvented with native code

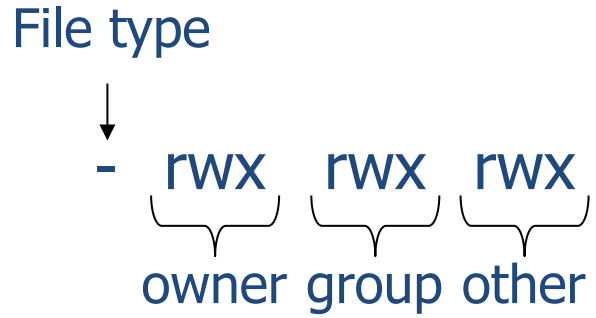


Recap: Starting new application process



Private app directory

- Implemented based on Linux' discretionary file access control
- Permissions pre-set by system
  - **R**ead, **W**rite, **E**xecute
  - Represented by vector of four octal values
    - 0o4: Read, 0o2: Write, 0o1: Execute
    - E.g.: 754 = (RWX, RX, R)



- Discretionary: owner can change permissions or even the ownership
- Only owner and root can change permissions
  - This privilege cannot be delegated or shared
- Since v4.3: SELinux mandatory access control in addition to discretionary access control

PRIVATE APP DIRECTORY

CISPA Center for IT-Security, Privacy and Accountability

```
root@android:/data/data # ls -l
drwxr-x--x u0_a44    u0_a44            2014-10-03 18:33 android.deviceadmin.cts
drwxr-x--x u0_a37    u0_a37            2014-09-27 13:03 com.android.apps.tag
drwxr-x--x u0_a1     u0_a1             2014-09-27 13:03 com.android.backupconfirm
drwxr-x--x bluetooth bluetooth           2014-09-27 13:05 com.android.bluetooth
drwxr-x--x u0_a3     u0_a3             2014-09-27 17:24 com.android.browser
drwxr-x--x u0_a4     u0_a4             2014-09-27 13:03 com.android.calculator2
drwxr-x--x u0_a5     u0_a5             2014-09-27 13:03 com.android.calendar
drwxr-x--x u0_a7     u0_a7             2014-09-27 13:03 com.android.certinstaller
drwxr-x--x u0_a0     u0_a0             2014-09-27 13:03 com.android.contacts
drwxr-x--x u0_a8     u0_a8             2014-09-27 13:11 com.android.defcontainer
drwxr-x--x u0_a9     u0_a9             2014-09-27 13:03 com.android.deskclock
drwxr-x--x u0_a2     u0_a2             2014-09-27 13:03 com.android.dreams.basic
drwxr-x--x u0_a29    u0_a29            2014-09-27 13:03 com.android.dreams.phototable

root@android:/data/data # ls -l com.android.providers.contacts/
drwxrwx--x u0_a0     u0_a0             2014-09-27 17:23 app_voicemail-data
drwxrwx--x u0_a0     u0_a0             2014-09-27 13:03 cache
drwxrwx--x u0_a0     u0_a0             2014-09-27 17:59 databases
drwxrwx--x u0_a0     u0_a0             2014-09-27 13:03 files
lrwxrwxrwx install   install            2014-09-27 13:03 lib -> /data/app-lib/com.android.providers.contacts
drwxrwx--x u0_a0     u0_a0             2014-09-27 13:04 shared_prefs

root@android:/data/data # ls -l com.android.providers.contacts/files/
drwx------ u0_a0     u0_a0               2014-09-27 17:34 photos
drwx------ u0_a0     u0_a0               2014-09-27 13:03 profile
```

UNIVERSITÄT DES SAARLANDES

Predefined file access rights

- $(Android_root_folder)/system/core/include/private/android_filesystem_config.h

```
/* Rules for directories.
** These rules are applied based on "first match", so they
** should start with the most specific path and work their
** way up to the root.
*/
static const struct fs_path_config android_dirs[] = {
    { 00770, AID_SYSTEM, AID_CACHE,  0, "cache" },
    { 00771, AID_SYSTEM, AID_SYSTEM, 0, "data/app" },
    { 00771, AID_SYSTEM, AID_SYSTEM, 0, "data/app-private" },
    { 00771, AID_ROOT,   AID_ROOT,   0, "data/dalvik-cache" },
    { 00771, AID_SYSTEM, AID_SYSTEM, 0, "data/data" },
    { 00771, AID_SHELL,  AID_SHELL,  0, "data/local/tmp" },
    { 00771, AID_SHELL,  AID_SHELL,  0, "data/local" },
    { 01771, AID_SYSTEM, AID_MISC,   0, "data/misc" },
    { 00770, AID_DHCP,   AID_DHCP,   0, "data/misc/dhcp" },
    { 00771, AID_SHARED_RELRO, AID_SHARED_RELRO, 0, "data/misc/shared_relro" },
    { 00775, AID_MEDIA_RW, AID_MEDIA_RW, 0, "data/media" },
    { 00775, AID_MEDIA_RW, AID_MEDIA_RW, 0, "data/media/Music" },
    { 00771, AID_SYSTEM, AID_SYSTEM, 0, "data" },
    { 00750, AID_ROOT,   AID_SHELL,  0, "sbin" },
...
/* Rules for files.
** These rules are applied based on "first match", so they
** should start with the most specific path and work their
** way up to the root. Prefixes ending in * denotes wildcard
** and will allow partial matches.
*/
static const struct fs_path_config android_files[] = {
    { 00440, AID_ROOT,      AID_SHELL,     0, "system/etc/init.goldfish.rc" },
    { 00550, AID_ROOT,      AID_SHELL,     0, "system/etc/init.goldfish.sh" },
    { 00440, AID_ROOT,      AID_SHELL,     0, "system/etc/init.trout.rc" },
    { 00550, AID_ROOT,      AID_SHELL,     0, "system/etc/init.ril" },
```

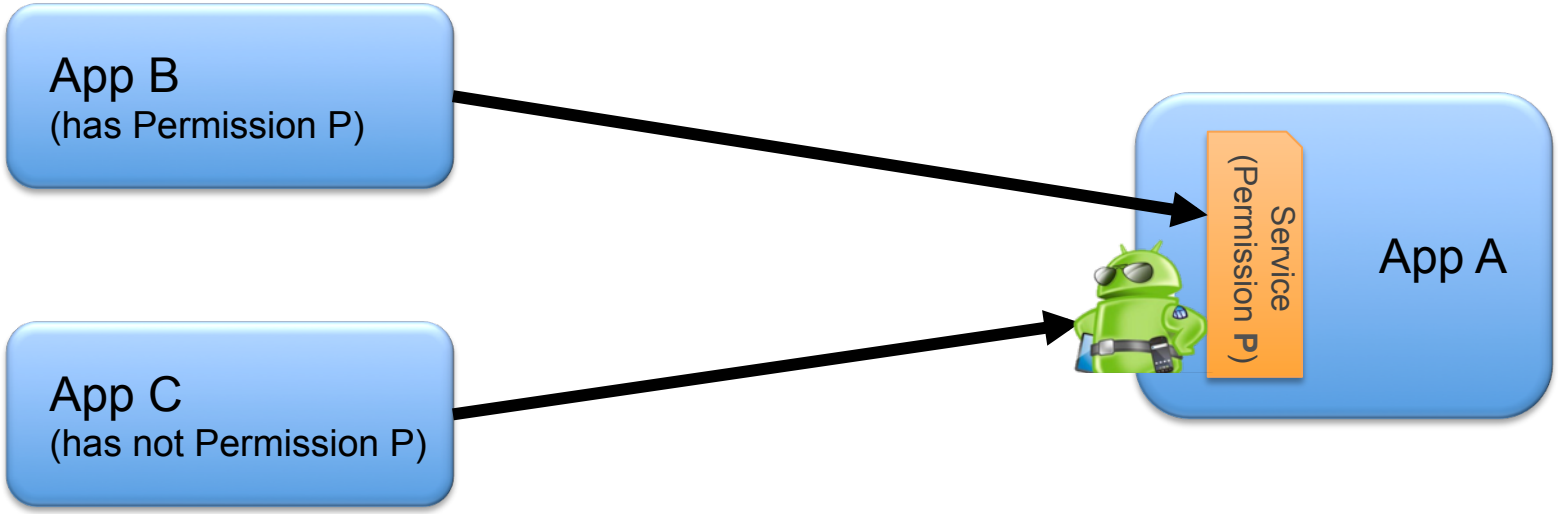
Android Security Architecture

Permissions & Least Privilege

Android Permission System

- 'Access rights' in Android's application framework
  - Permissions are required to gain access to
    - System resources (logs, battery, etc.)
    - Sensitive data (SMS, contacts, e-mails, etc.)
    - System interfaces (Internet, send SMS, etc.)
  - Currently more than 140 default permissions defined in Android
- Permissions are assigned to sandboxes, i.e., UIDs
- Application (developers) can also define their own permissions to protect application interfaces

Android Permission: Example



- Application A contains an exported component (e.g., Service), which is protected by Permission **P**
- Application B holds Permission **P** and thus can access the exported component of Application A
- Application C does **not** hold Permission **P** and hence cannot access the exported component

Protection Levels

Permissions categorised in four different proctection levels

- **Normal:**
  - Low-risk permissions that form no danger to the system integrity or end-user
  - Default value
  - No end-user approval is required to grant this permission to an application
- **Dangerous:**
  - Higher-risk permissions that give access to functions or data that can compromise the system integrity or the end-user privacy
  - User has to explicitly approve such permissions during app installation
- **Signature:**
  - Permission is only granted, if the requesting application has been signed with the same developer certificate as the application that defined the permission
  - Are automatically granted without end-user approval required.
- **SignatureOrSystem:**
  - Like Signature, but additionally is also granted if the requesting application is a system application. System applications means that the app was pre-installed during system build or that is was signed with the platform key (i.e., the key the OS vendor maintains)

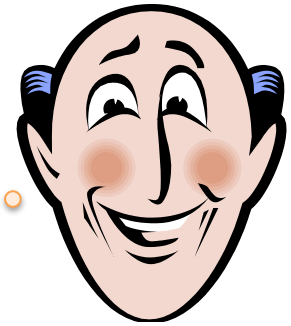
Android Installation Process: Benign App

Color Flashlight

Download App

Permissions

Install



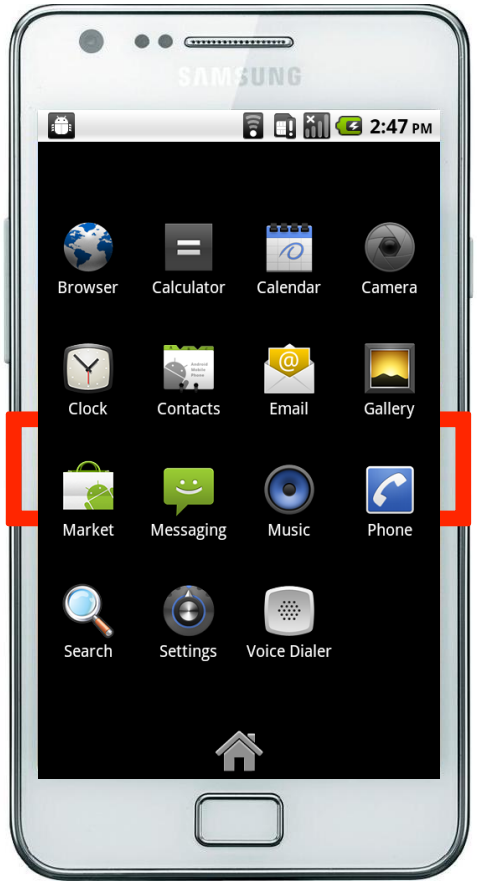
User

Requested permissions are reasonable

Android Installation Process: Security-Critical App

Color Flashlight

Download app



Permissions

Deny install

User

Why does this app request permission to send SMS?

Application Manifest File

CISPA Center for IT-Security, Privacy and Accountability

```
<uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Permissions requested

```
<permission
        android:name="de.unisaarland.cs.CUSTOM_PERMISSION"
        android:description="@string/customPermDesc"
        android:label="Access our app"
        android:protectionLevel="signature" >
</permission>
```

Permissions defined by app (developer)

…

```
<application android:icon="@drawable/icon" android:label="@string/app_name”>
```

…

```
<service
            android:name="de.unisaarland.cs.LocalService"
            android:permission="de.unisaarland.cs.CUSTOM_PERMISSION"
            android:exported=”true" >
</service>
```

Service component protected by Permission

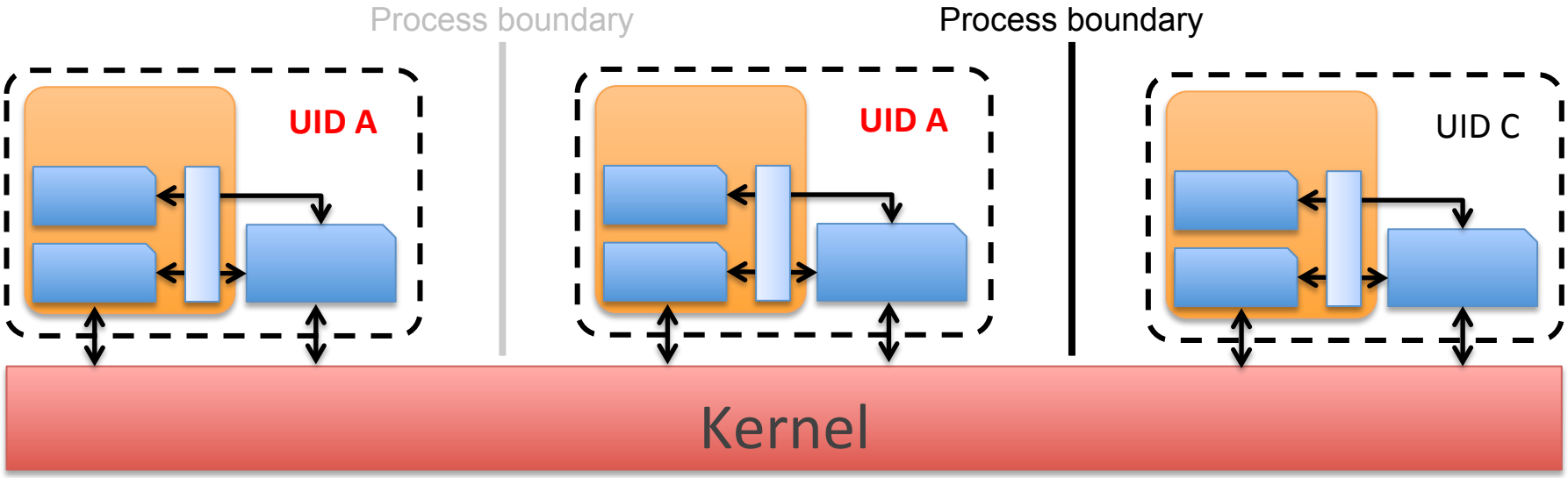
```
<provider
            android:name="de.unisaarland.cs.LocalProvider"
            android:authorities=”infsec”
            android:exported=”true"
            android:readPermission="de.unisaarland.cs.CUSTOM_READ_PERMISSION"
            android:writePermission="de.unisaarland.cs.CUSTOM_WRITE_PERMISSION" >
</provider>
```

ContentProvider component protected by Permission

```
</application>
```

Shared UIDs

- Applications with shared UID also share their permissions
  - Mutual access to data directory allowed as well
- Shared UIDs only allowed for applications signed with same developer key



Isolated Processes

- Security-aware application developer can declare in application manifest that a **Service** component should be executed as an *isolated process*
  - Component executed on separate process with UID *nobody*
  - *Nobody* is a UID with no privileges
    - All permission checks will return deny
    - No file system access
  - only communication with it is through the Service API
- Code that can be potentially compromised can thus be securely isolated from the remainder of the application
  - Allows compartmentalization of the app

Enforcing permissions

- Permissions are either
  1. Simply associated strings (most permissions)
     - Enforced in Android's middleware
  2. Mapped to Linux GIDs (few: Internet, Bluetooth, ext. storage,…)
     - Enforced by the Linux kernel

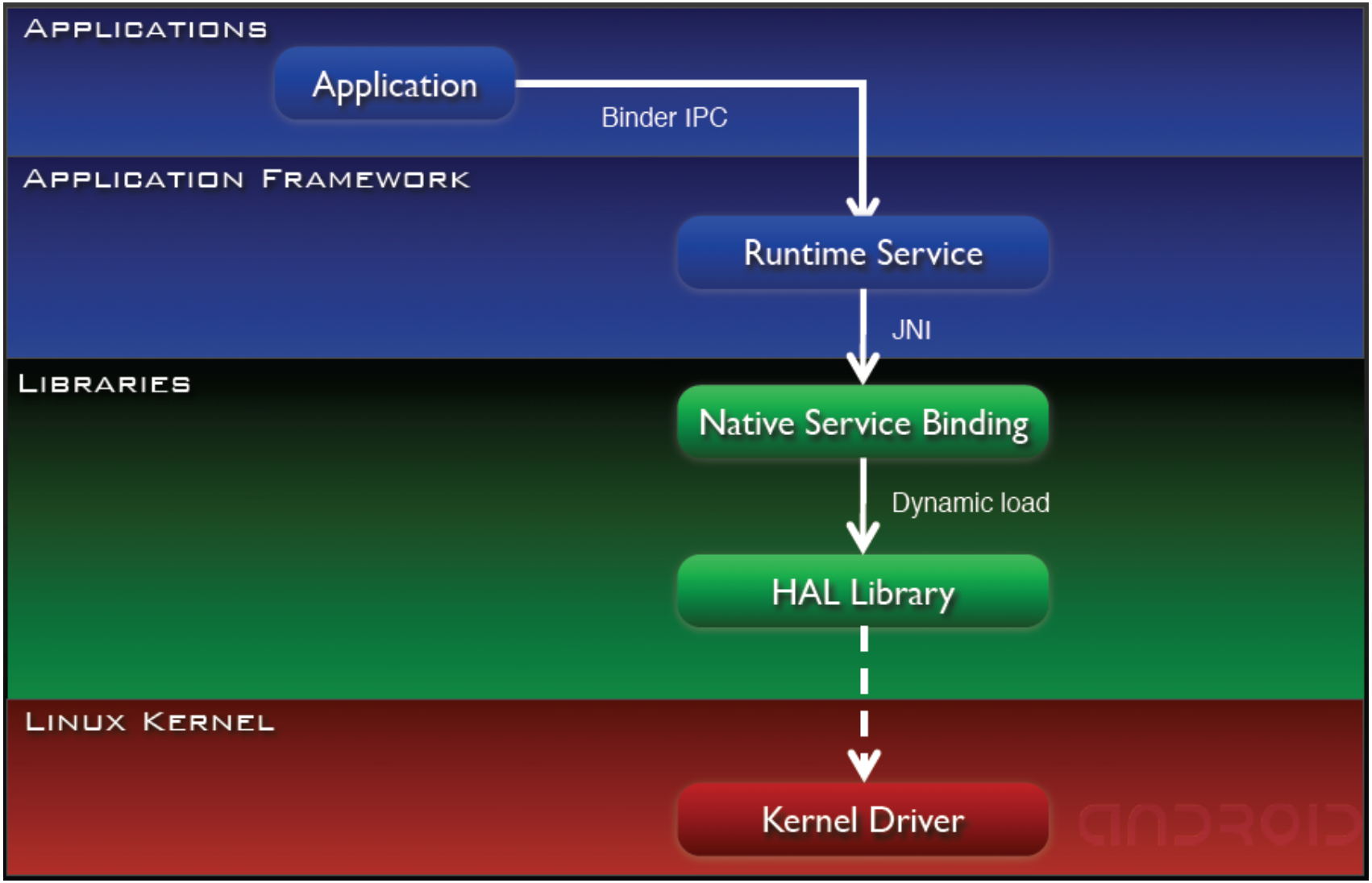
Enforcing permissions in the middleware

- Applications should not have direct access to certain highly sensitive resources (radio, etc.)
  - Those resources are “encapsulated” in core Android services (“Reference monitors”)
  - Those services provide an API to applications to allow controlled access access to resources via the services
  - Permissions define whether a core service performs an operation on behalf of an application or not

Application Framework

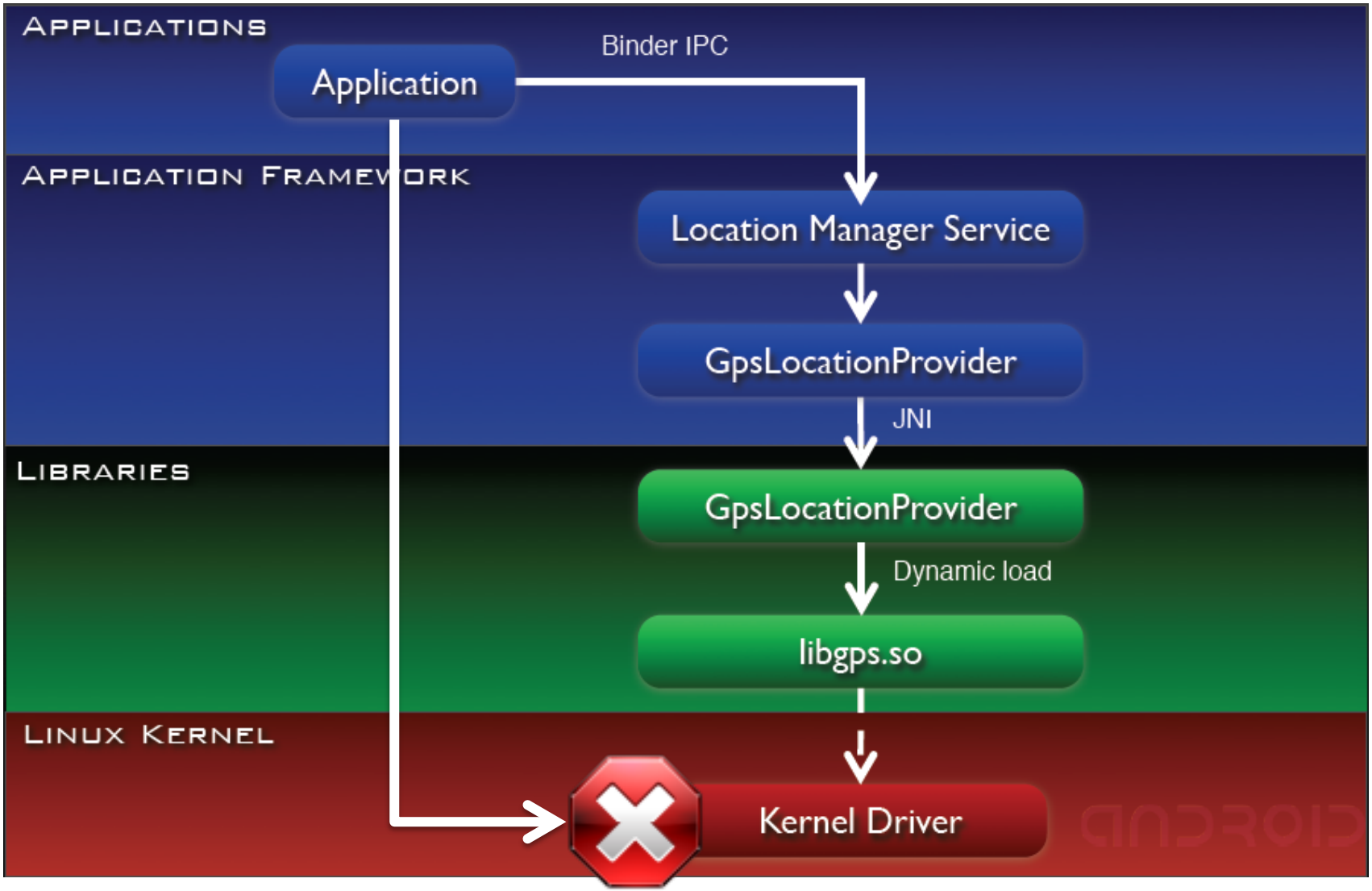
- Provides developers API to basic functionalities and services (e.g., set alarms, access location information, take advantage of device HW,…)
- APIs are the same as for the core applications (e.g., Phone, Contacts,…)
- Activity Manager
  - Permission decision point
  - Responsible for starting applications
- Package Manager
  - Management of Permissions and Applications

App → Runtime Service → lib



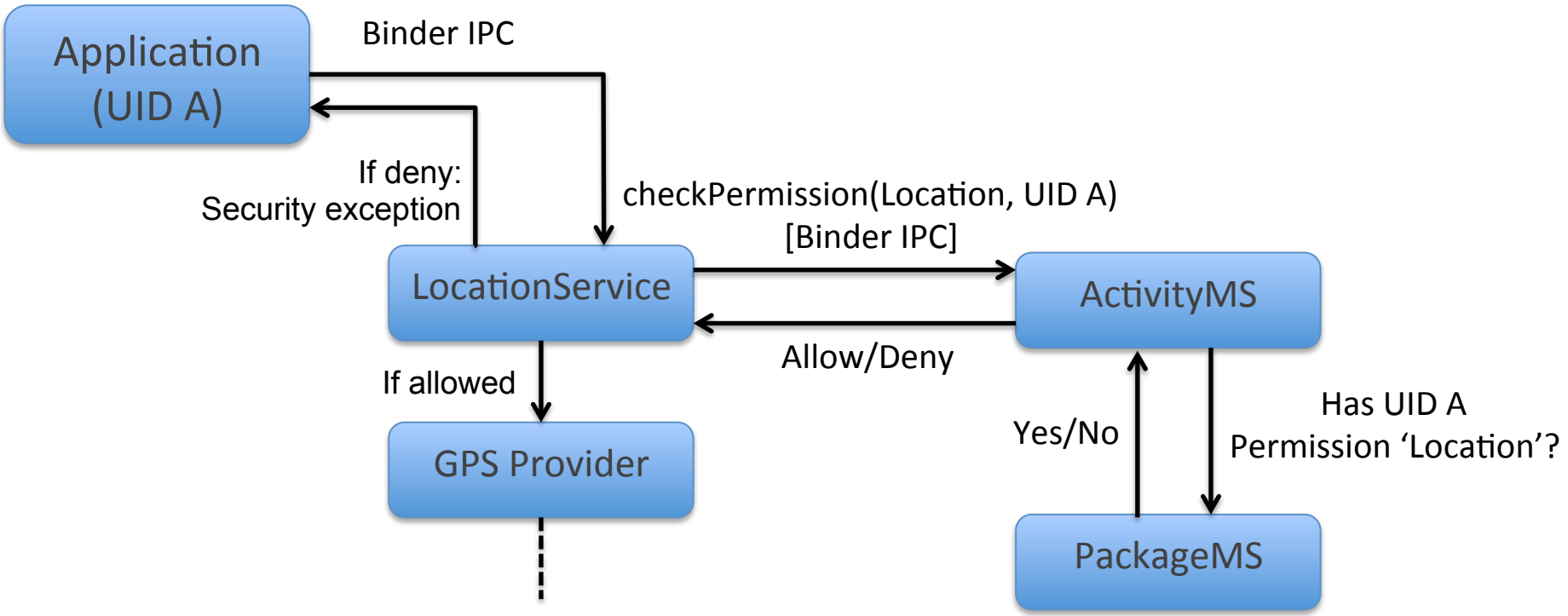
(Android Anatomy and Physiology, Patrick Brady)

App → Runtime Service → lib



(Android Anatomy and Physiology, Patrick Brady)

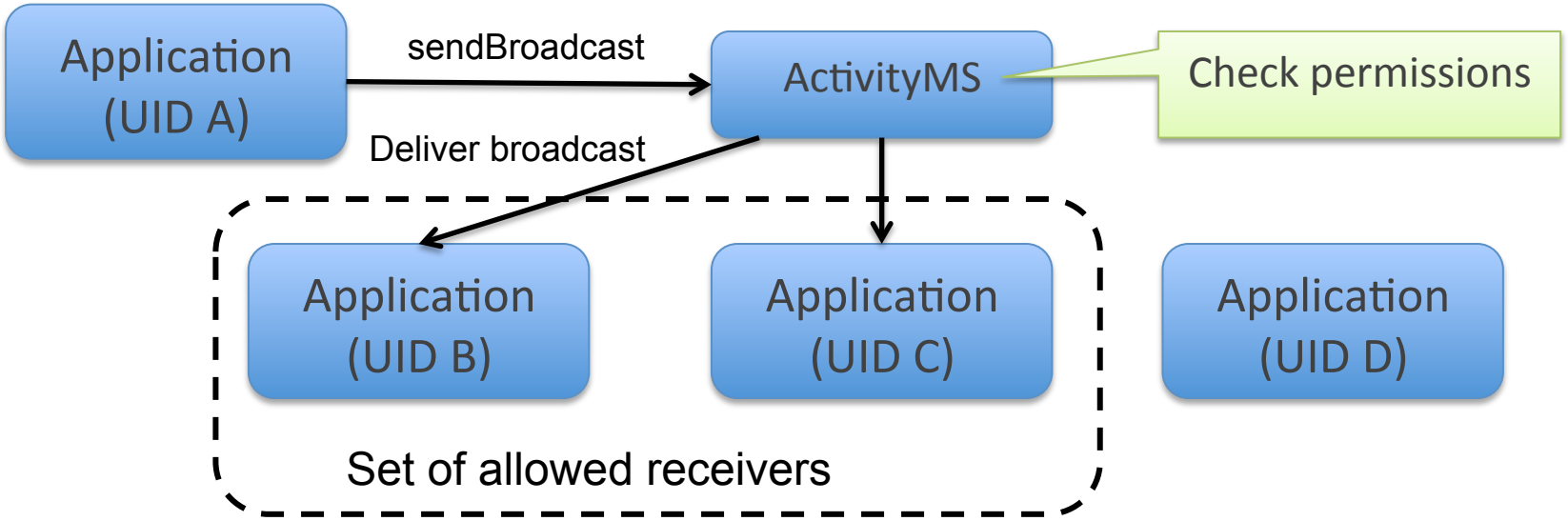
The role of the ActivityManagerService



- 'Location' is a defined permission, e.g., one of the >140 predefined
- UID of the application is provided to LocationService by Binder IPC

The role of the ActivityManagerService

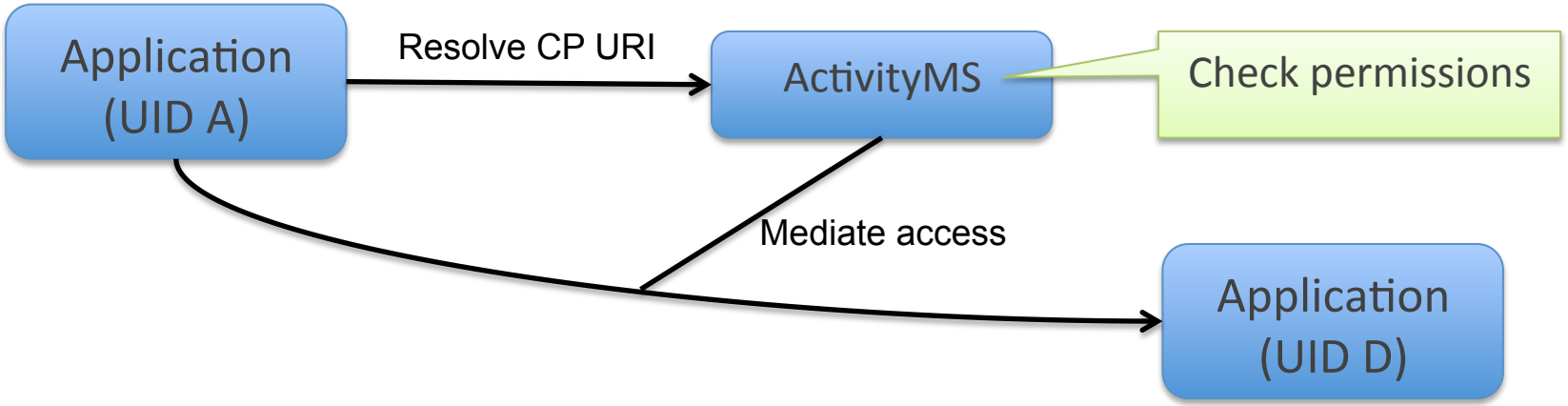
- Checking if broadcast should be delivered to receiver



- Delivery depends on
  - Permission required by sender:
    `sendBroadcast(Intent intent, String receiverPermission)`
- Permission required by receiver:
  - `<receiver android:name=".MyReceiver" android:permission="com.exmaple.PERMISSION">`
  - `registerReceiver(BroadcastReceiver receiver, IntentFilter filter, String broadcastPermission, Handler scheduler)`

The role of the ActivityManagerService

- Mediating ContentProvider access



- Separate permissions for read and write access

```
<provider
      android:name="com.example.demo.mycontentprovider"
      android:authorities="com.example.demo.provider"
      android:readPermission="com.example.READ_PERM"
      android:writePermission="com.example.WRITE_PERMISSION">
</provider>
```

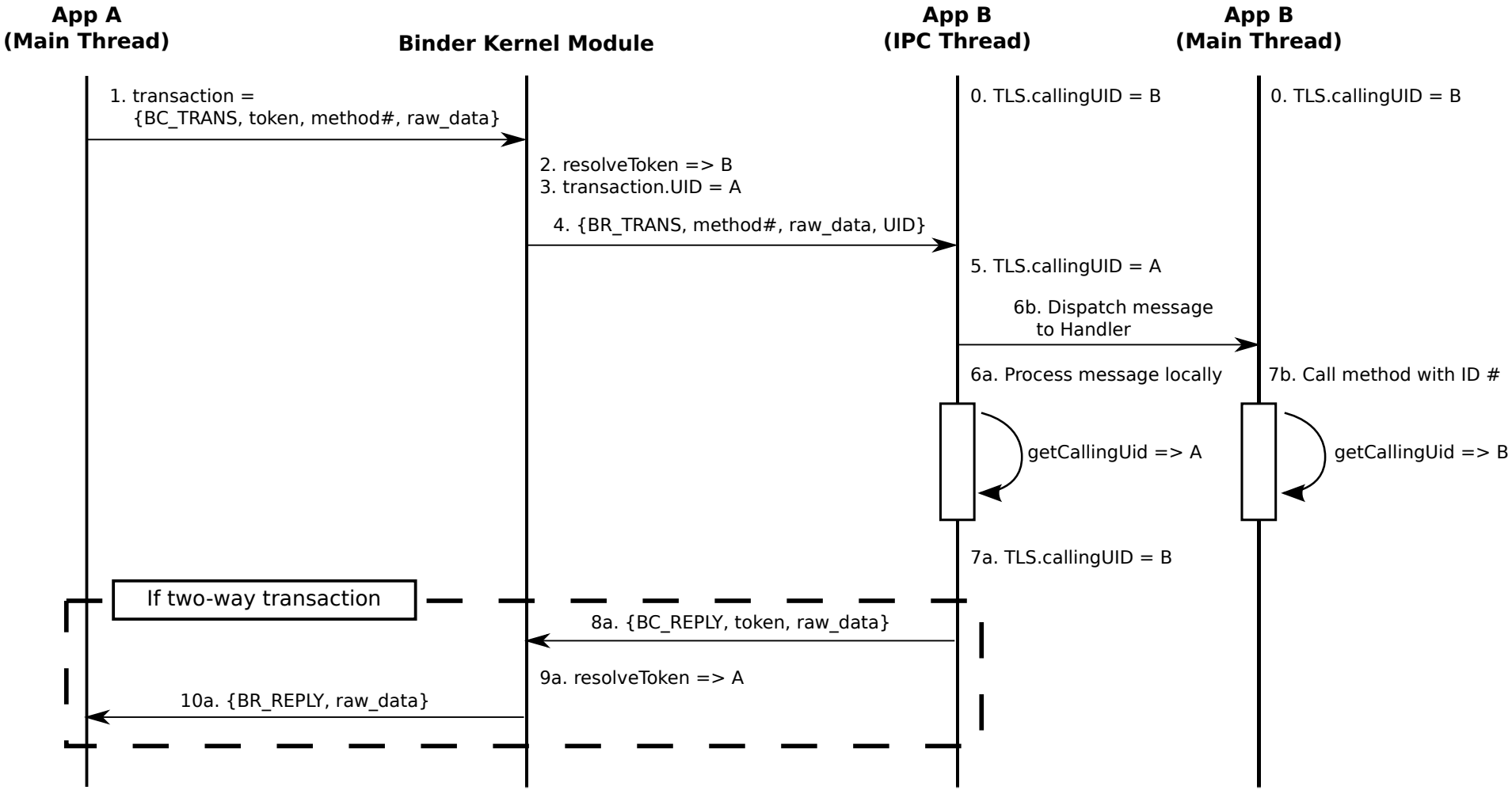
Permission Enforcement in Middleware/Applications

- Binder IPC provides certain information to the callee of IPC
  - `getCallingUID()`: returns caller's UID
  - `getCallingPID()`: returns caller's PID
- ActivityManager can be queried for a permission check from every application (system or 3rd party):
  - `checkPermission(String Perm)`: checks if caller has been granted the permission "Perm"
  - Can also be called by applications themselves: `checkPermission(int UID, String Perm)`: checks if application with "UID" has been granted the permission "Perm", where UID can be retrieved with `getUID()`

BINDER-IPC AS BUILDING BLOCK FOR PERMISSION ENFORCEMENT

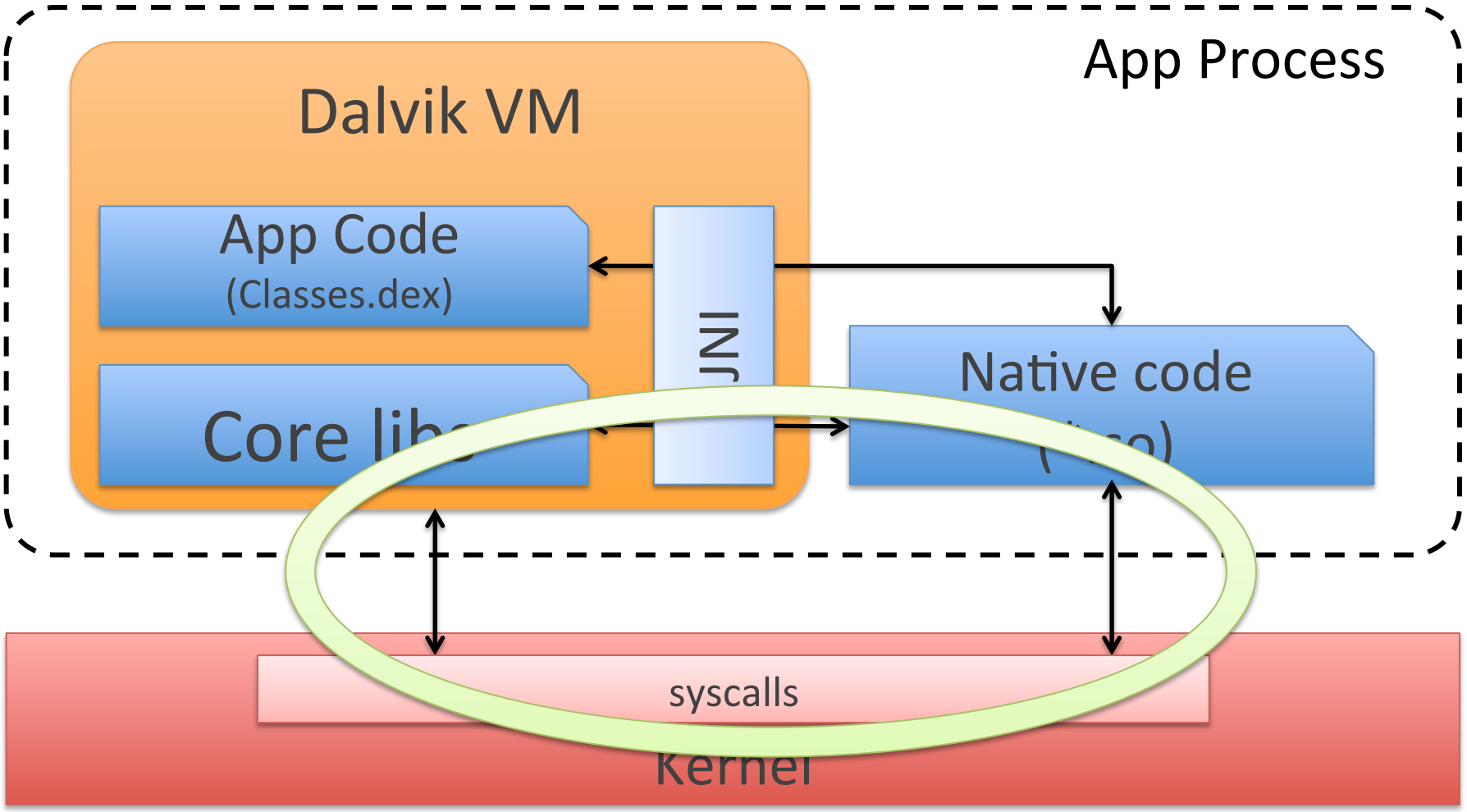
- Standard Linux kernel
- Patches for Android (e.g., aggressive Power Management, Logger, Binder)
- Binder:
  - High-performance, shared memory based IPC
  - Synchronous calls between processes
  - Per-process thread pool for processing requests

Binder protocol (high-level)



Enforcing permissions by the Linux kernel

- For performance and management reasons certain operations not performed through an application framework service but app process interacts directly with the kernel
  - Networking, file system access



JNI = Java Native Interface

Enforcing permissions by the Linux kernel

CISPA Center for IT-Security, Privacy and Accountability

- For performance and management reasons certain operations not performed through an application framework service but app process interacts directly with the kernel
  - Networking, file system access
- Requires a mapping from Permission string to something the kernel understands
  - Solution: Using Linux group IDs ("paranoid networking")
  - App UID must be member of a Linux group to have access to sockets, etc.
    - UID of an app with corresponding permission is added to group during install
  - Kernel access errors translated into Java security exceptions by core libraries

Permission-to-GID mapping

- $(Android_root_folder)/frameworks/base/data/etc/platform.xml

```
<!-- The following tags are associating low-level group IDs with
     permission names.  By specifying such a mapping, you are saying
     that any application process granted the given permission will
     also be running with the given group ID attached to its process,
     so it can perform any filesystem (read, write, execute) operations
     allowed for that group. -->
    <permission name="android.permission.BLUETOOTH_ADMIN" >
        <group gid="net_bt_admin" />
    </permission>
    <permission name="android.permission.BLUETOOTH" >
        <group gid="net_bt" />
    </permission>
    <permission name="android.permission.BLUETOOTH_STACK" >
        <group gid="net_bt_stack" />
    </permission>
    <permission name="android.permission.NET_TUNNELING" >
        <group gid="vpn" />
    </permission>
    <permission name="android.permission.INTERNET" >
        <group gid="inet" />
    </permission>
...
```

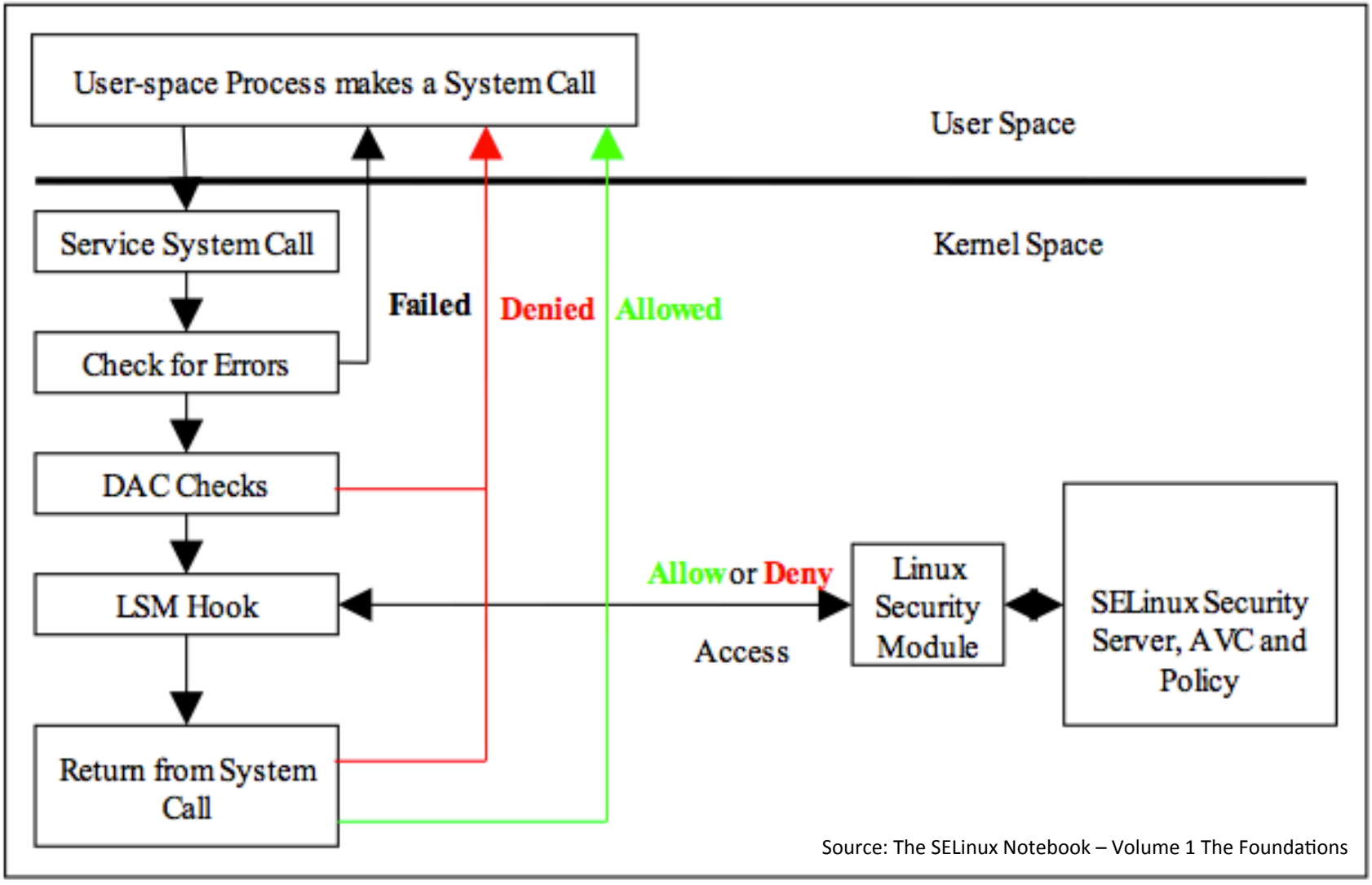
Android Security Architecture

Type Enforcement

SELinux overview

- Mandatory Access Control with SELinux supported since v4.3
- SELinux (Security Enhanced Linux)
  - Primarily developed by the NSA, part of Linux kernel since 2008
  - Implemented as a kernel module for the *Linux Security Modules* framework (LSM)
  - Ported to Android by the SEAndroid project (also NSA)
  - Core parts of SEAndroid integrated upstream into the Android source code
    - Integration into Zygote, PackageManagerService, and system boot
- Security policies and enforcement based on *type enforcement*
  - Supports further access control models including *role-based access control* and *multi-level access control* (not covered in this course)

Linux Security Modules



Source: The SELinux Notebook – Volume 1 The Foundations

Type Enforcement

- Every subject and object is *labeled* with a security context:
  Consistent list of attributes
  - Security context stored with the object (e.g., extended file system attributes)
- Primary attribute: *type*
  - Simply a string, policy gives a meaning to this attribute
- Type enforcement rule stated as:
  `subject_type object_type: object_class operation`
  - Object class: “file”, “socket”, …
  - Operation: “read”, “write”, “open”
  - Only if a rule is defined, access is allowed, otherwise denied (“whitelisting”)
- Much more fine-grained access control polices than with Linux discretionary access control
  - SELinux MAC outranks the Linux DAC: Root processes only as powerful as their SELinux type (effectively removes the ‘root’ user)

SHORT EXAMPLE RULE

CISPA Center for IT-Security, Privacy and Accountability

```
allow untrusted_app sdcard_external:file { create open read }
```

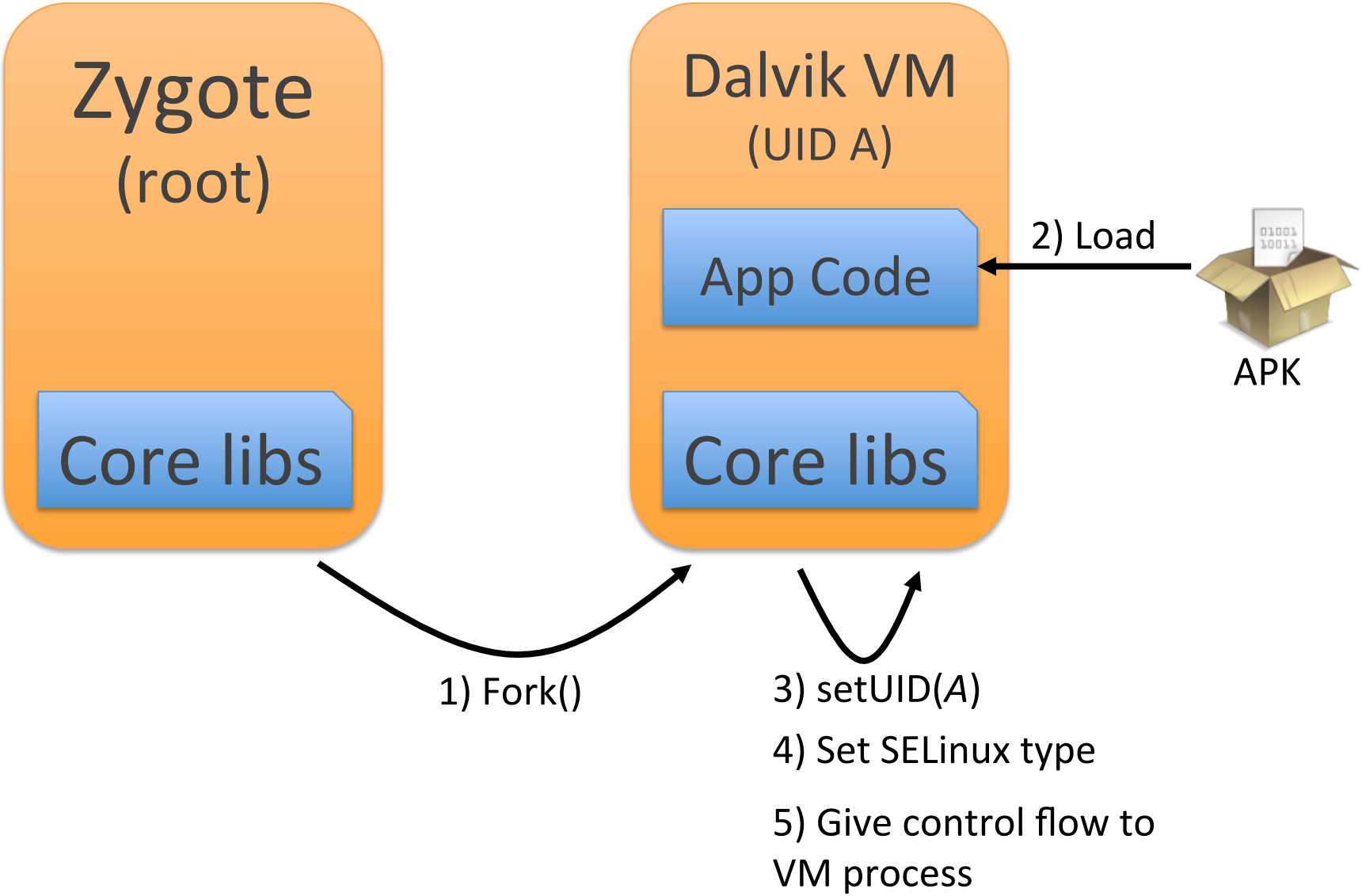
- `untrusted_app:` Subject type, i.e., process of the app
- `sdcard_external:` Object type
- `file:` Object class, i.e., a file object
- `{create open read}:` Operations, i.e., “create a file”, …

- The rule states, that an application whose process is has the type “untrusted_app” can create, read, write file objects with type “sdcard_external”
  - If the system ensures that all files stored on the sdcard get the type “sdcard_external” and that all 3rd party apps get the type “untrusted_app”, than this rule governs how 3rd party apps can operated on files on the SDcard

Labeling Android applications

- Integrated into the PackageManagerService and Zygote
  - PackageManagerService requests a type for a newly installed app from the SELinux policies, stores the type in the package management
  - When starting a new app, ActivityManagerService requests the app's type from PackageManagerService and passes the type as argument to Zygote. Zygote labels the newly created application process.
    - Policy must give Zygote higher privileges to be able to label other processes
    - Zygote must check which process wants to set the type and whether the process is allowed to do so (Only system server process running ActivityManagerService should be allowed)

SELinux integration into Zygote



Determining Application type

- Based on package information
  - developer signature
  - package name
  - permissions
  - ...
- SE Android introduced new install policy definitions to define the mapping from package information to security type

Determining Application type

- $(ANDROID_ROOT_DIR)/frameworks/base/services/java/com/android/server/pm/SELinuxMMAC.java

```
/**
     * Labels a package based on an seinfo tag from install policy.
     * The label is attached to the ApplicationInfo instance of the package.
     * @param pkg object representing the package to be labeled.
     * @return boolean which determines whether a non null seinfo label
     *         was assigned to the package. A null value simply meaning that
     *         no policy matched.
     */
    public static boolean assignSeinfoValue(PackageParser.Package pkg) {

        // We just want one of the signatures to match.
        for (Signature s : pkg.mSignatures) {
            if (s == null)
                continue;

            Policy policy = sSigSeinfo.get(s);
            if (policy != null) {
                String seinfo = policy.checkPolicy(pkg.packageName);
                if (seinfo != null) {
                    pkg.applicationInfo.seinfo = seinfo;
                    if (DEBUG_POLICY_INSTALL)
                        Slog.i(TAG, "package (" + pkg.packageName +
                                "") labeled with seinfo=" + seinfo);

                    return true;
                }
            }
        }
       ...
```

Example Security Contexts

- Security context shown in the form:
  `user:role:type[:level]`
  - User: SELinux user ID (not covered in this course)
  - Role: Role for RBAC (not covered)
  - Type: Security type
  - Level: Multi-level security (not covered)
    - Android does not (yet) use roles and MLS => generic user *u*, role *r* and level *s0*
- Security contexts of the shell:

```
shell@hammerhead:/ $ id -z
uid=2000(shell) gid=2000(shell) groups=1003(graphics),
1004(input),1007(log),1011(adb),1015(sdcard_rw),
1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),
3006(net_bw_stats) context=u:r:shell:s0
```

Example Security Contexts

- Process list showing security contexts:

```
shell@hammerhead:/ $ ps –Z
LABEL                          USER      PID   PPID  NAME
u:r:init:s0                    root      1     0     /init
u:r:kernel:s0                  root      2     0     kthreadd
u:r:kernel:s0                  root      3     2     ksoftirqd/0
u:r:kernel:s0                  root      4     2     kworker/0:0
u:r:kernel:s0                  root      5     2     kworker/0:0H
u:r:kernel:s0                  root      7     2     kworker/u:0H
u:r:kernel:s0                  root      8     2     migration/0
…
u:r:system_server:s0           system    783   206   system_server
u:r:wpa:s0                     wifi      882   1     /system/bin/wpa_supplicant
u:r:platform_app:s0            u0_a20    884   206   com.android.systemui
u:r:untrusted_app:s0           u0_a7     913   206   android.process.media
u:r:untrusted_app:s0           u0_a9     1037  206   com.google.android.gms
u:r:radio:s0                   radio     1061  206   com.redbend.vdmc
u:r:nfc:s0                     nfc       1080  206   com.android.nfc
u:r:radio:s0                   radio     1103  206   com.android.phone
u:r:radio:s0                   radio     1129  206   com.android.telecomm
u:r:untrusted_app:s0           u0_a3     1178  206   android.process.acore
```

Example Security Contexts

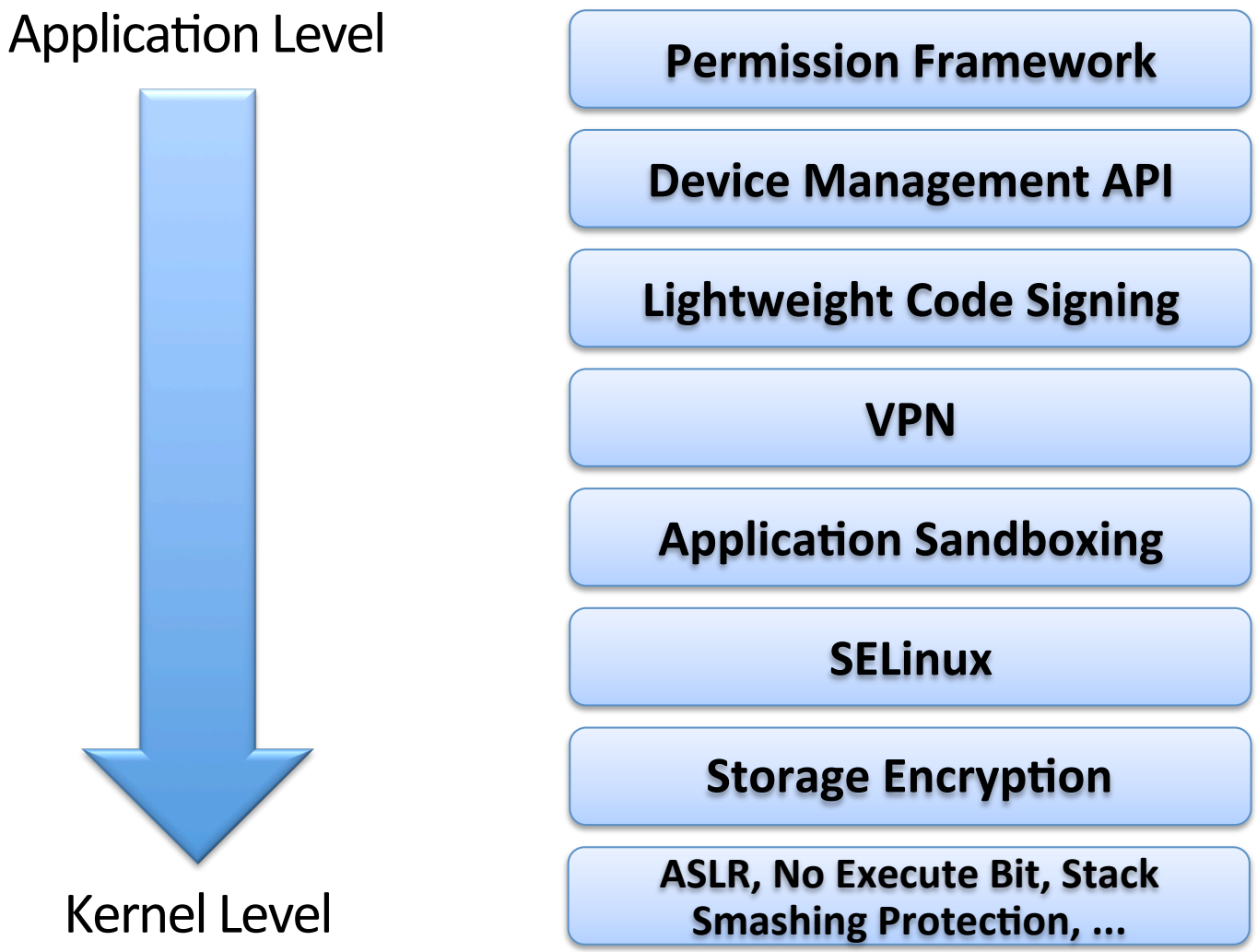
- Defined in `$(Android_root)/external/sepolicy/file_contexts`
- Files labeled during system build
- File system SELinux security contexts:

```
shell@hammerhead:/ $ ls –lZ
drwxr-xr-x root     root              u:object_r:cgroup:s0 acct
drwxrwx--- system   cache             u:object_r:cache_file:s0 cache
lrwxrwxrwx root     root              u:object_r:rootfs:s0 charger -> /sbin/healthd
dr-x------ root     root              u:object_r:rootfs:s0 config
lrwxrwxrwx root     root              u:object_r:rootfs:s0 d -> /sys/kernel/debug
drwxrwx--x system   system            u:object_r:system_data_file:s0 data
-rw-r--r-- root     root              u:object_r:rootfs:s0 default.prop
drwxr-xr-x root     root              u:object_r:device:s0 dev
lrwxrwxrwx root     root              u:object_r:rootfs:s0 etc -> /system/etc
-rw-r--r-- root     root              u:object_r:rootfs:s0 file_contexts
dr-xr-x--- system   system            u:object_r:firmware_file:s0 firmware
-rw-r----- root     root              u:object_r:rootfs:s0 fstab.hammerhead
-rwxr-x--- root     root              u:object_r:rootfs:s0 init
```

Android Security Architecture

Security Evolution

Security in the Stock Software Stack



Stock Security Feature Evolution

Android 2.2 “Froyo”

- User log-in authentication
  (PIN, password, pattern)
- Device Administration API
  (Remote wipe, Exchange server)
- Same origin policy for updates
  (actually available from the beginning)

Android 2.3 “Gingerbread”

- Memory Management Security Enhancements
  (Format string vulnerability protections, Hardware-based No eXecute, Linux mmap_min_addr to mitigate null pointer dereference privilege escalation)

Stock Security Feature Evolution (cont.)

Android 3.x “Honeycomb”

- Full Filesystem Encryption
  (dmcrypt implementation of AES128 with CBC and ESSIV:SHA256 with PBKDF2 protection for key generation)
- New remote management features
  (policies for encrypted storage and passwords)
- Pluggable DRM Framework

Android 4.0 “Ice Cream Sandwich”

- Address Space Layout Randomization (ASLR)
- Secure management of credentials
  (Exposed keychain API)
- VPN client API
- Device policy management for camera
- Extended MS Exchange support

Stock Security Feature Evolution (cont.)

Android 4.1 / 4.2 / 4.3 “Jelly Beans”

- ContentProvider default access now “deny”
- App Encryption
- Application verification
- Secure USB debugging
- More control of premium SMS
- More memory management features
  (Position Independent Executable support, Read-only relocations / immediate binding, prevent leaking of kernel addresses)
- Hardening of crucial system daemons and libraries
- Source code gives first indications of upcoming multi-user support
- Started integrating SELinux
- Capability bounding
- KeyChain support
- “App Ops” / “Intent Firewall” (non-enforcing)

Stock Security Feature Evolution (cont.)

Android 4.4 “KitKat”

- SELinux in enforcing mode
- Elliptic Curve Digital Signature Algorithm (ECDSA) support
- Per-User VPN support (on multi-user devices)
- FORTIFY_SOURCE level 2
- Certificate pinning