Mobile Security

Dr. Eleonora Losiouk
Department of Mathematics
University of Padua
elosiouk@math.unipd.it
https://www.math.unipd.it/~elosiouk/

Università degli Studi di Padova

SPRITZ Security & Privacy Research Group

Dipartimento Matematica

From the eyes of an app

- Android is based on Linux
- Each app has its own Linux user ID*
- Each app lives in its own security *sandbox*
  - Standard Linux process isolation
  - Restricted file system permissions

App Installation

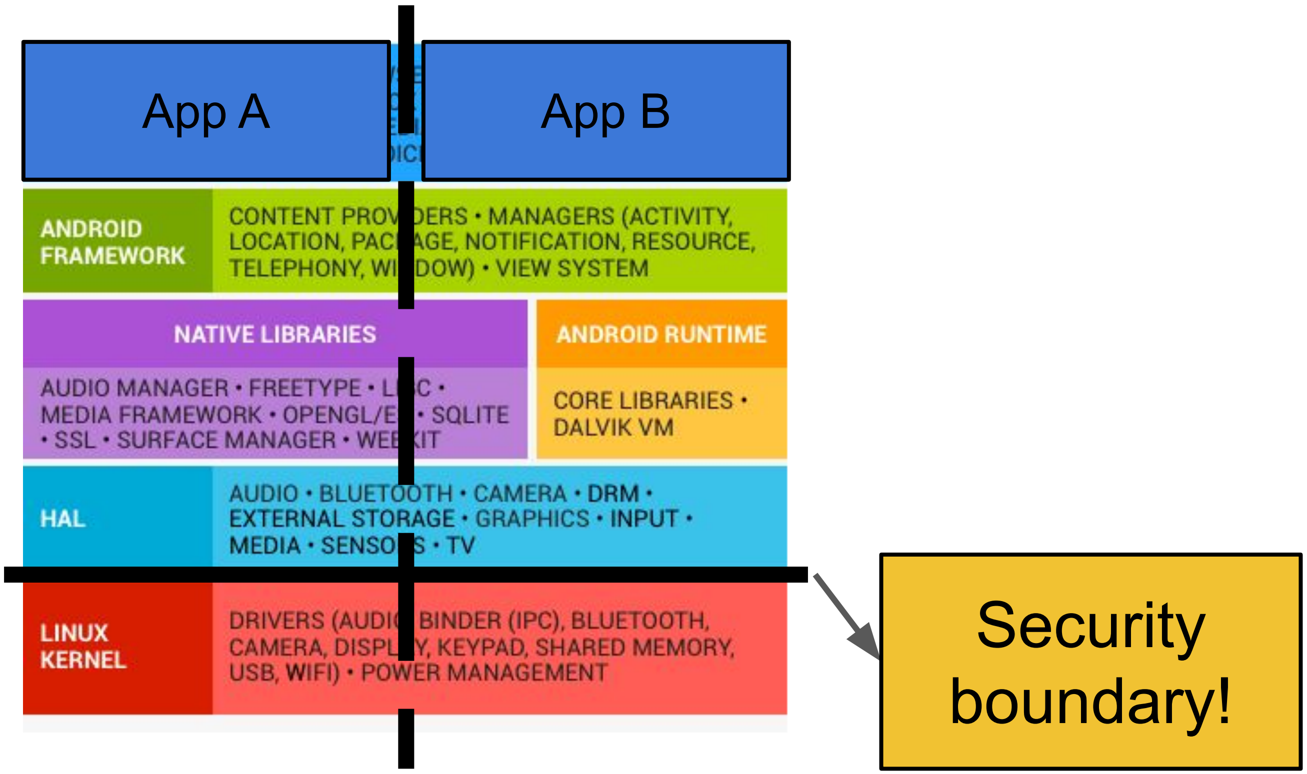
- The Android framework creates a new Linux user
- Each app is given a private directory
  - Also called "Internal Storage"
  - No other app can access it*

* There are ways to setup apps so that they share the user ID. See "sharedUserId".

App Isolation

- Apps are run in separate processes
- Apps being in sandbox means that they can't
  - talk to each other
  - do anything security-sensitive
- Q: how can apps do anything interesting?
- This is when architecture & security get mixed up

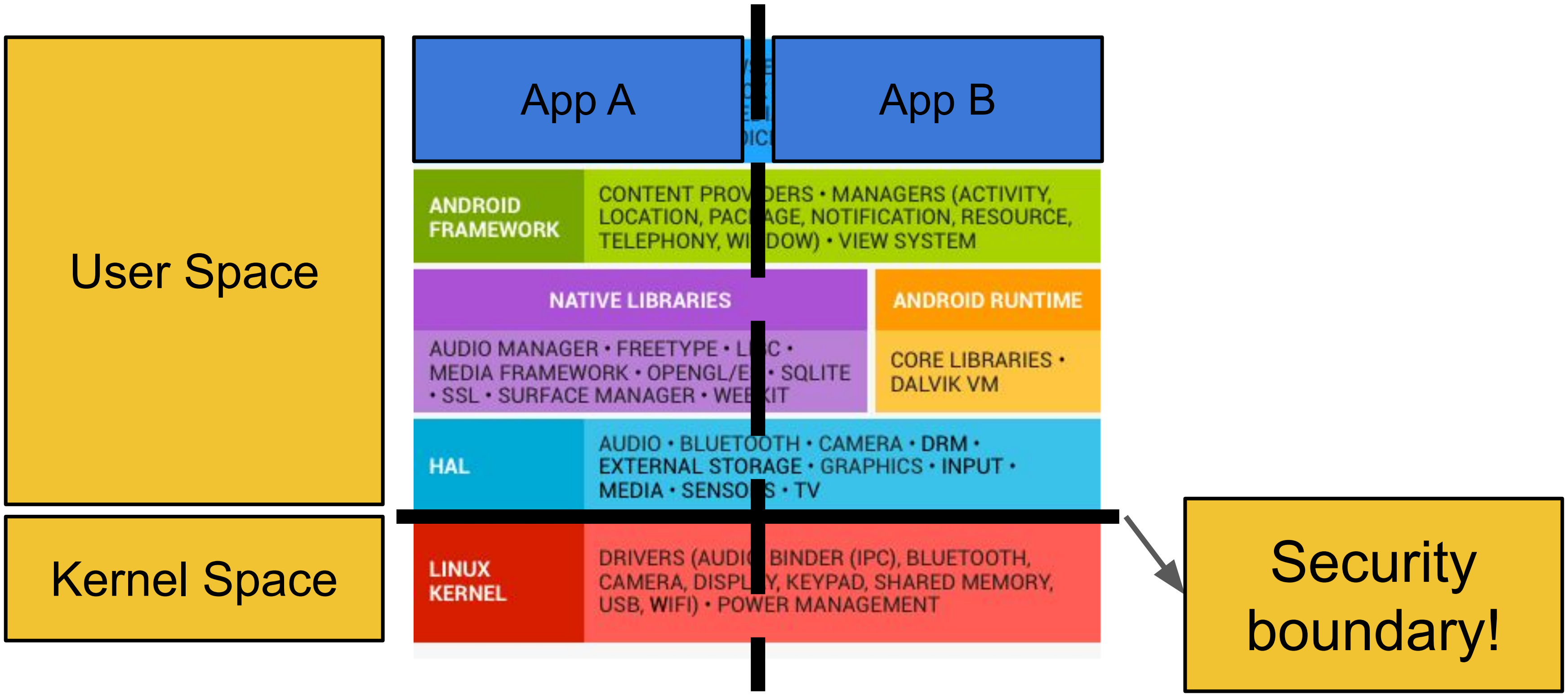
Android Framework Architecture



Asking favors to the OS, aka "syscalls"

- Traditional OSes (like Windows, Linux, Android) have two worlds: user-space vs. kernel-space
- User-space is where user processes and apps live
  - They can't do much by themselves
- Kernel-space is where the actual OS lives
  - The OS is the God on your machine & information

Android Framework Architecture



Example: Storing a File

- Let's say a process wants to save a file on the hard drive

- The process has no access to the physical hard drive
  - It would be too dangerous!

- The process needs to ask the OS
  - Would you mind saving file X with content ABC?

Example: Storing a File

- The developer uses high-level APIs

```
{
    ...
    OutputStreamWriter writer = new OutputStreamWriter(...)
    writer.write(data);
    writer.close();
    ...
}
```

- Under the hood, the process needs to ask the OS
  - Would you mind writing "data" in file XYZ?

Example: Saving a file

- Going down: Java -> libc -> syscalls
- fd = open(const char *filename, int flags, umode_t mode)
- n = write(unsigned int fd, char *buf, size_t count)
- close(unsigned int fd);

How are syscalls actually invoked?

- Each architecture has its own convention
- x86 (ref)
  - syscall number in "eax", arguments in "ebx", "ecx", "edx", "esi", "edi", ...
  - execute instruction "int 0x80"
  - return value in "eax"
- x86-64 (ref)
  - syscall number in "rax", args in "rdi", "rsi", "rdx", "rcx", "r8", "r9", ...
  - execute instruction "int 0x80" or "syscall"
  - return value in "rax"

How are syscalls actually invoked?

- ARM (ref)
  - execute instruction "swi" or "svc"
  - syscall number in "r7", args in "r0", "r1", "r2", ...
  - return value in "r0"

- More architectures:
  - ref
  - "man syscall"

man syscall (arguments)

arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes
arm/OABI	a1	a2	a3	a4	v1	v2	v3	
arm/EABI	r0	r1	r2	r3	r4	r5	r6	
arm64	x0	x1	x2	x3	x4	x5	-	
blackfin	R0	R1	R2	R3	R4	R5	-	
i386	ebx	ecx	edx	esi	edi	ebp	-	
ia64	out0	out1	out2	out3	out4	out5	-	
mips/o32	a0	a1	a2	a3	-	-	-	See below
mips/n32,64	a0	a1	a2	a3	a4	a5	-	
parisc	r26	r25	r24	r23	r22	r21	-	
s390	r2	r3	r4	r5	r6	r7	-	
s390x	r2	r3	r4	r5	r6	r7	-	
sparc/32	o0	o1	o2	o3	o4	o5	-	
sparc/64	o0	o1	o2	o3	o4	o5	-	
x86_64	rdi	rsi	rdx	r10	r8	r9	-	
x32	rdi	rsi	rdx	r10	r8	r9	-	

man syscall (return value)

arch/ABI	instruction	syscall #	retval	Notes
arm/OABI	swi NR	-	a1	NR is syscall #
arm/EABI	swi 0x0	r7	r0	
arm64	svc #0	x8	x0	
blackfin	excpt 0x0	P0	R0	
i386	int $0x80	eax	eax	
ia64	break 0x100000	r15	r8	See below
mips	syscall	v0	v0	See below
parisc	ble 0x100(%sr2, %r0)	r20	r28	
s390	svc 0	r1	r2	See below
s390x	svc 0	r1	r2	See below
sparc/32	t 0x10	g1	o0	
sparc/64	t 0x6d	g1	o0	
x86_64	syscall	rax	rax	See below
x32	syscall	rax	rax	See below

Not all requests are as easy as opening a file...

- Get current location?
- Send an SMS?
- Display something to the UI?
- Play a sound?
- Talk to other apps!?

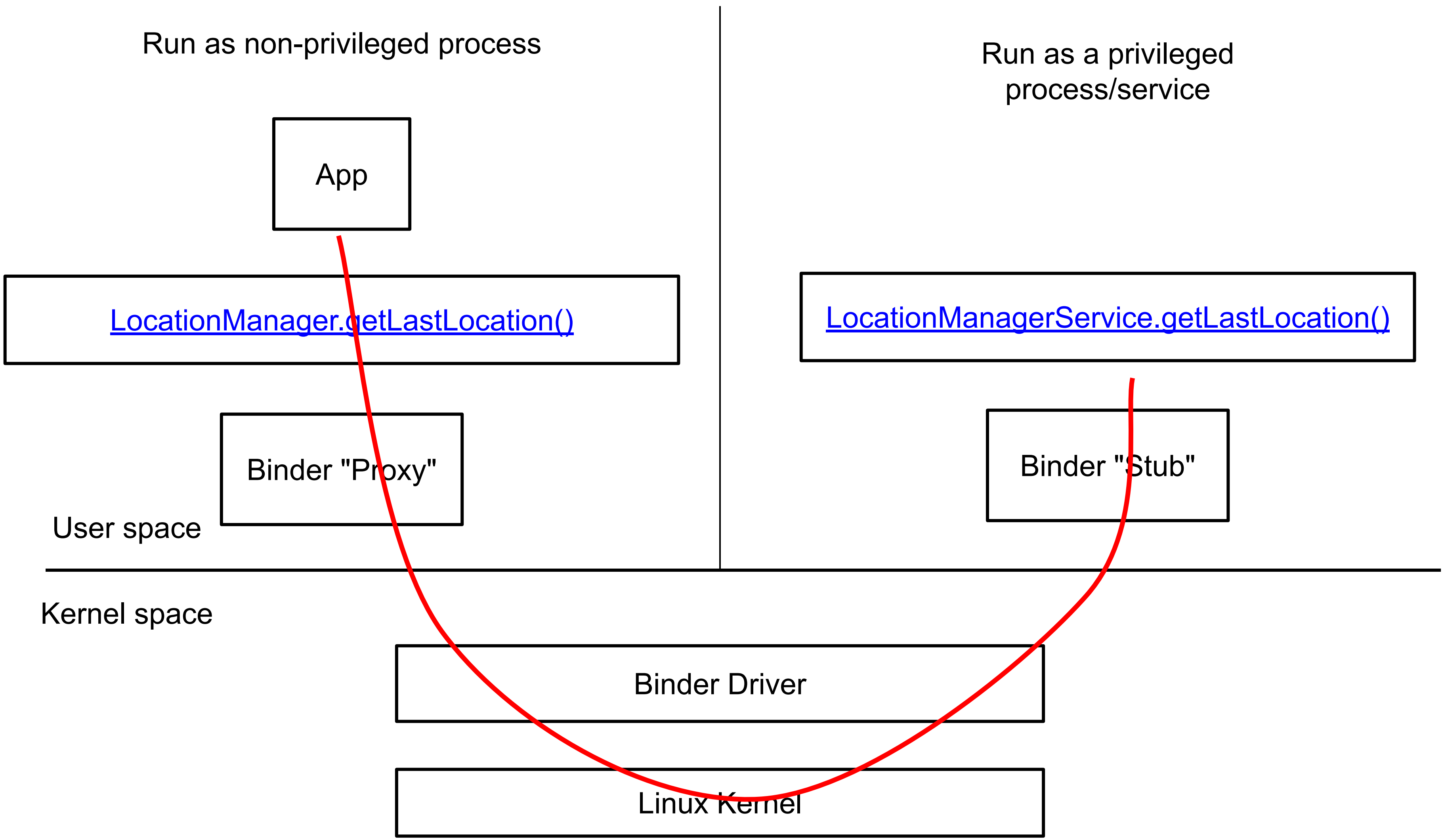
Example: getLastLocation()

- App invokes Android API
  - LocationManager.getLastLocation() (ref)
  - We are still within the app's sandbox!
- Actual implementation of the privileged API
  - LocationManagerService.getLastLocation() (ref)
  - We are in a "privileged" service
- How do we go from one side to the other one?

Crossing the bridge

- Binder!
- Binder: one of the main Android's "extensions" over Linux
- It allows for
  - Remote Procedure Call (RPC)
  - Inter-Process Communication (IPC)

Binder RPC



Run as non-privileged process
Run as a privileged process/service
App
LocationManager.getLastLocation()
LocationManagerService.getLastLocation()
Binder "Proxy"
Binder "Stub"
User space
Kernel space
Binder Driver
Linux Kernel

Binder details

- Proxy and Stub are automatically generated starting from AIDL
- Binder internals
  - /dev/binder
  - ioctl syscall
    - Multi-purpose syscall, to talk to drivers
    - The Binder kernel driver takes care of it, dispatches messages and returns replies

Many "Managers"

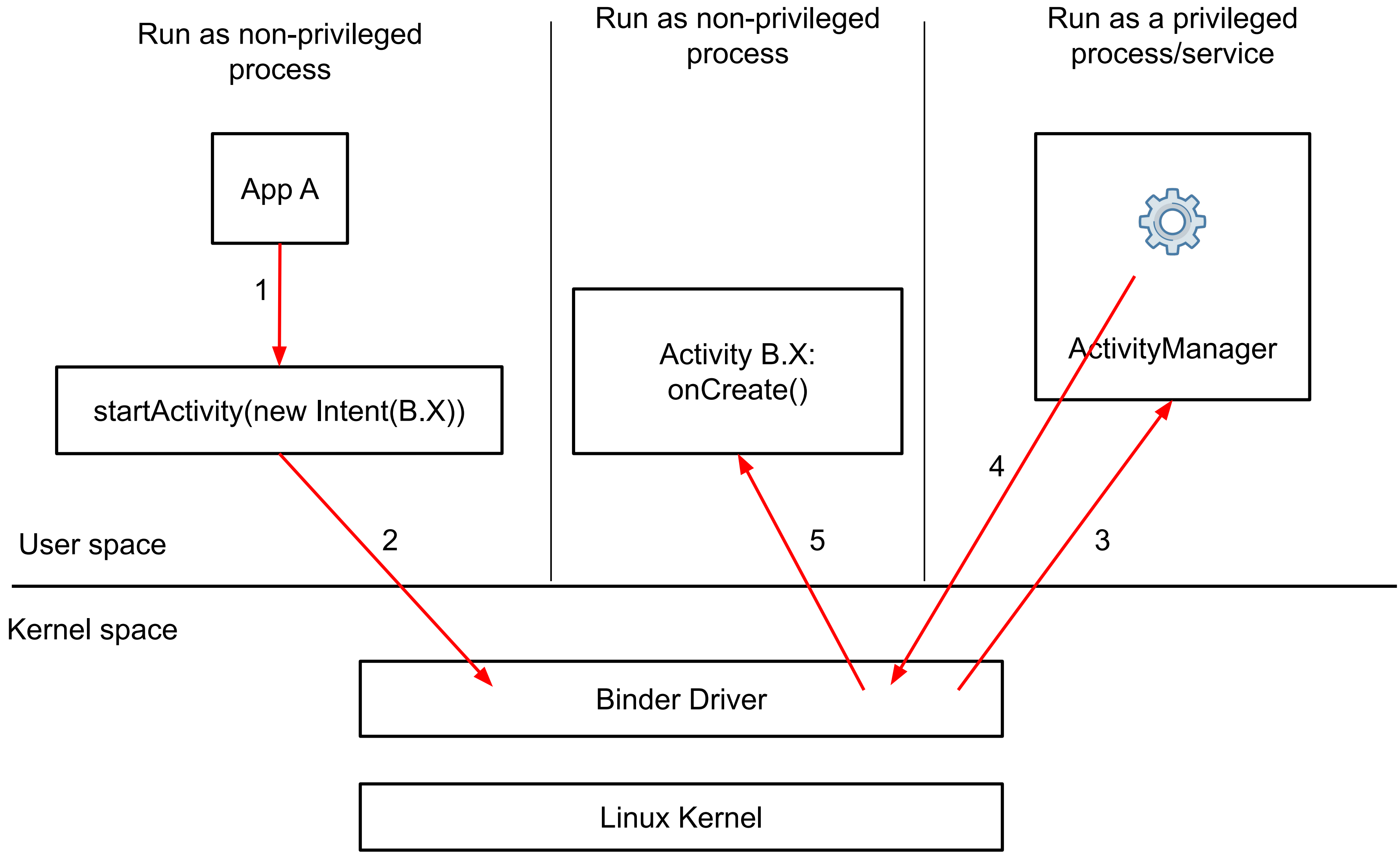
- Activity Manager
- Package Manager
- Telephony Manager
- Resource Manager
- Location Manager
- Notification Manager
- Resource Manager

```
$ adb shell service list
```

Binder as IPC mechanism

- How do apps talk to each other?
- High-level API: Intents
- Under the hood: Binder calls!

Binder IPC: A → B.X



What about security?

- Can an app always do all these things? Nope.
- It has a private folder... that's it?
  - It can start other apps (the main activity is always "exported")
  - It can show things on the screen (when the app is in foreground)
- It can't
  - Open internet connection
  - Get current location
  - Write on the external storage
  - ...

Android Permission System (overview, ref)

- Android framework defines a long list of permissions
- Each of these "protects" security-sensitive capabilities
  - The ability to "do" something sensitive
    - Open Internet connection, send SMS
  - The ability to "access" sensitive information
    - Location, user contacts, ...

Examples of Permissions

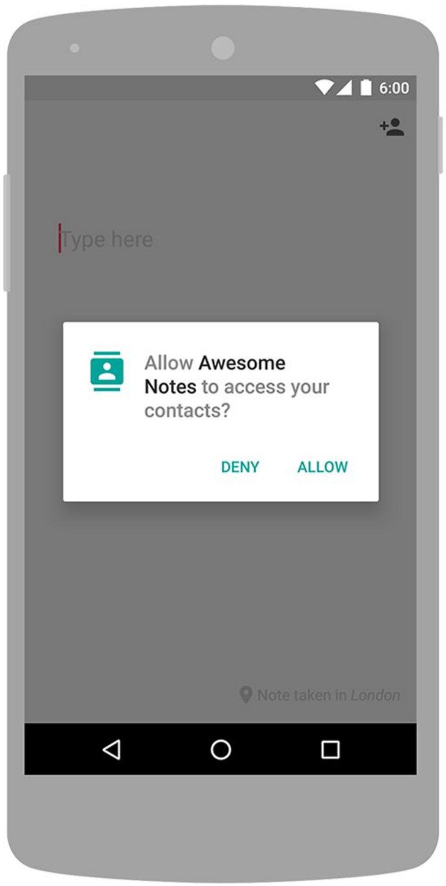
- INTERNET (string: "android.permission.INTERNET")
- ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, CHANGE_NETWORK_STATE, READ_PHONE_STATE
- ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION
- READ_SMS, RECEIVE_SMS, SEND_SMS
- ANSWER_PHONE_CALLS, CALL_PHONE, READ_CALL_LOG, WRITE_CALL_LOG
- READ_CONTACTS, WRITE_CONTACTS
- READ_CALENDAR, WRITE_CALENDAR
- RECORD_AUDIO, CAMERA
- BLUETOOTH, NFC
- RECEIVE_BOOT_COMPLETED
- SYSTEM_ALERT_WINDOW
- SET_WALLPAPER

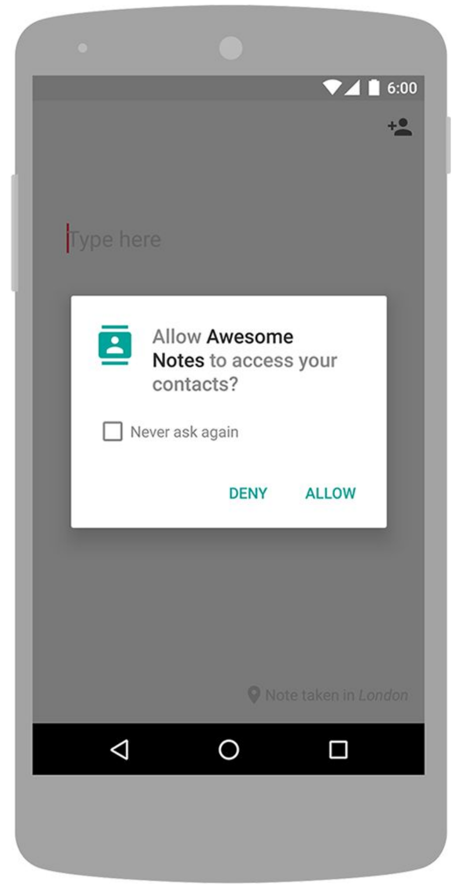
Permissions Protection Levels (doc)

- Normal
- Dangerous
- Signature
- SignatureOrSystem

Granting Dangerous Permissions (doc)

- Runtime requests
    - If device's API level >=23 (Android 6) AND app's targetSdkVersion >= 23
- Facts
    - The user is not notified at install time
    - The app initially doesn't have the permission, but it can be run
    - App needs to ask at runtime ("runtime prompt")
- Users have the option to disable them


6:00
Type here
Allow Awesome Notes to access your contacts?
DENY
ALLOW
Note taken in London

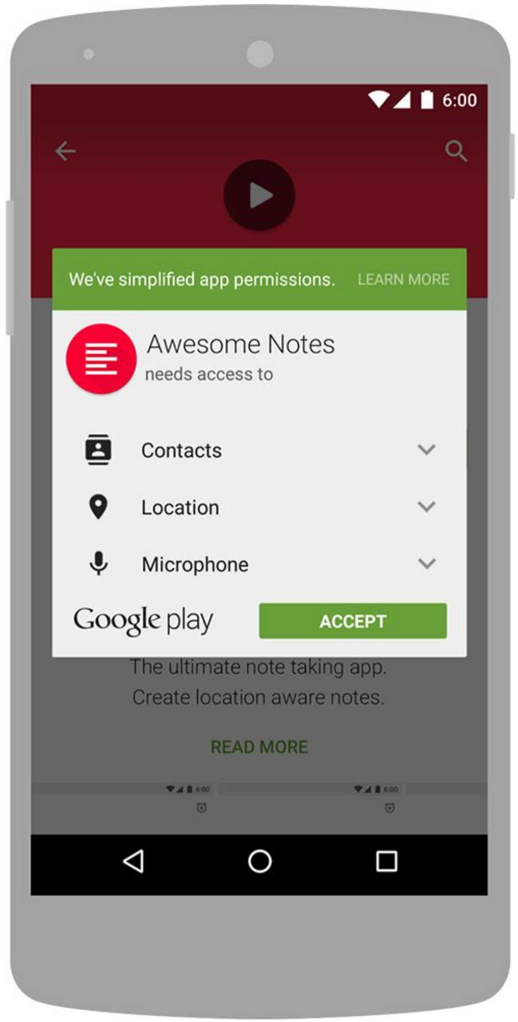

6:00
Type here
Allow Awesome Notes to access your contacts?
Never ask again
DENY
ALLOW
Note taken in London

Granting Dangerous Permissions (doc)

- Install-time requests
  - If device's API level <23 OR app's targetSdkVersion < 23

- Facts
  - The user is asked about all permissions at installation time
  - If user accepts: *all* permissions are granted
  - If user does not accept: app installation is aborted

- Users do *not* have the option to disable them


6:00
We've simplified app permissions.
LEARN MORE
Awesome Notes
needs access to
Contacts
Location
Microphone
Google play
ACCEPT
The ultimate note taking app.
Create location aware notes.
READ MORE

Runtime vs Install-time Prompts

- Runtime
  - Pros: Users can install apps without giving all permissions
  - Pros: Users have contextual information to decide accept/reject
  - Pros: Permissions can be selectively enabled/disabled
  - Cons: Multiple prompts can be annoying
- Install time
  - Pros: no annoying prompts after installation
  - Cons: "all-or-nothing", grant all permissions or app can't be installed
  - Cons: No contextual info to take informed decisions

Permissions Groups

- Permissions are organized in groups
- Permissions requests are handled at a group level
  - User grants X -> all permissions in X's group are automatically granted if an app's update asks for them
- Security implications!

Permissions Groups: An Example

- SMS permission group
  - RECEIVE_SMS, READ_SMS, **SEND_SMS**
- PHONE permission group
  - READ_PHONE_STATE, READ_PHONE_NUMBERS, **CALL_PHONE, ANSWER_PHONE_CALLS**
- Group/permission mappings: link

Permissions from an app's perspective

Permission Request


```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.awesomeapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application ...>
        ...
    </application>
</manifest>
```

Linux groups

- INTERNET permission -> app's user is added to "inet" Linux group
- BLUETOOTH permission -> app's user is added to "bt_net" Linux group
- Declaration in AOSP: code

Custom Permissions (doc)

- Apps can define "custom" permissions!

```
<permission
  android:name="com.example.myapp.permission.DEADLY_STUFF"
  android:label="@string/permlab_deadlyStuff"
  android:description="@string/permdesc_deadlyStuff"
  android:permissionGroup="android.permission-group.DEADLY"
  android:protectionLevel="signature" />
```

- The "system" permissions are defined in the same way
    - AndroidManifest.xml
- By default, android:exported is "false"
- BUT: if the component defines an "intent filter", then the default value is "true"

Components Permission Enforcement

- Apps' components can specify which permissions are required to use them

```
<receiver
   android:name="com.example.myapp.DeadlyReceiver"
   android:permission="com.example.myapp.permission.DEADLY_STUFF">
   <intent-filter>
     <action android:name="com.example.myapp.action.SHOOT"/>
   </intent-filter>
</receiver>
```

Custom Permission Use Cases

- protectionLevel = "signature"
  - Only apps signed by the same developer / company can get it
  - Example: big company with many apps
    - Facebook wants all its apps to have access to users' posts
    - Facebook does not want any other app to have access to this information
- protectionLevel = "dangerous"
  - App controls security-related things / information (which are not strictly related to Android)
  - App wants to provide this capability to other apps, but it wants to warn the user first