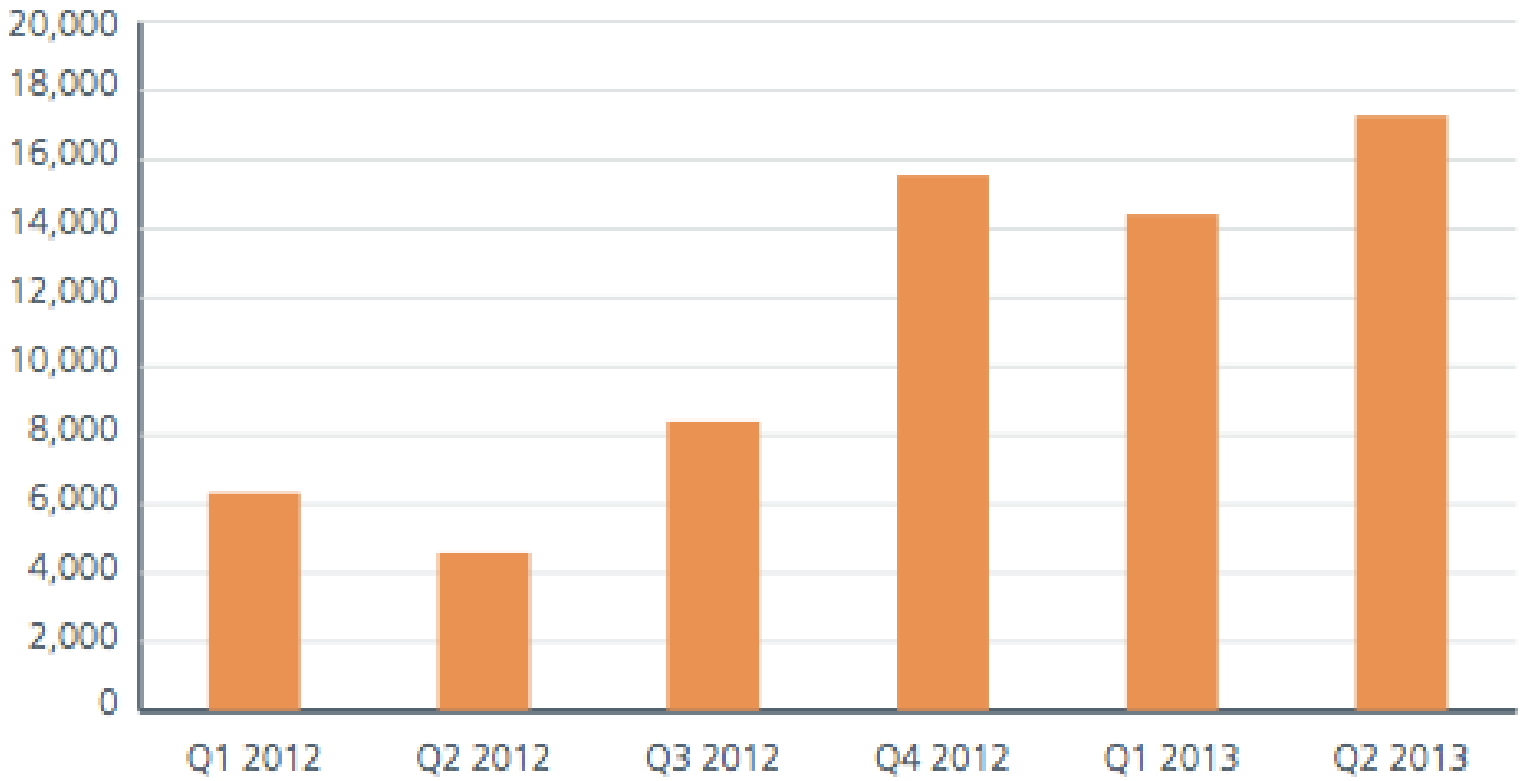
Università degli Studi di Milano
Facoltà di Scienze e Tecnologie
Dipartimento di Informatica

Mobile Security

Srdjan Matic <srdjan@security.di.unimi.it>
Aristide Fattori <aristide@security.di.unimi.it>

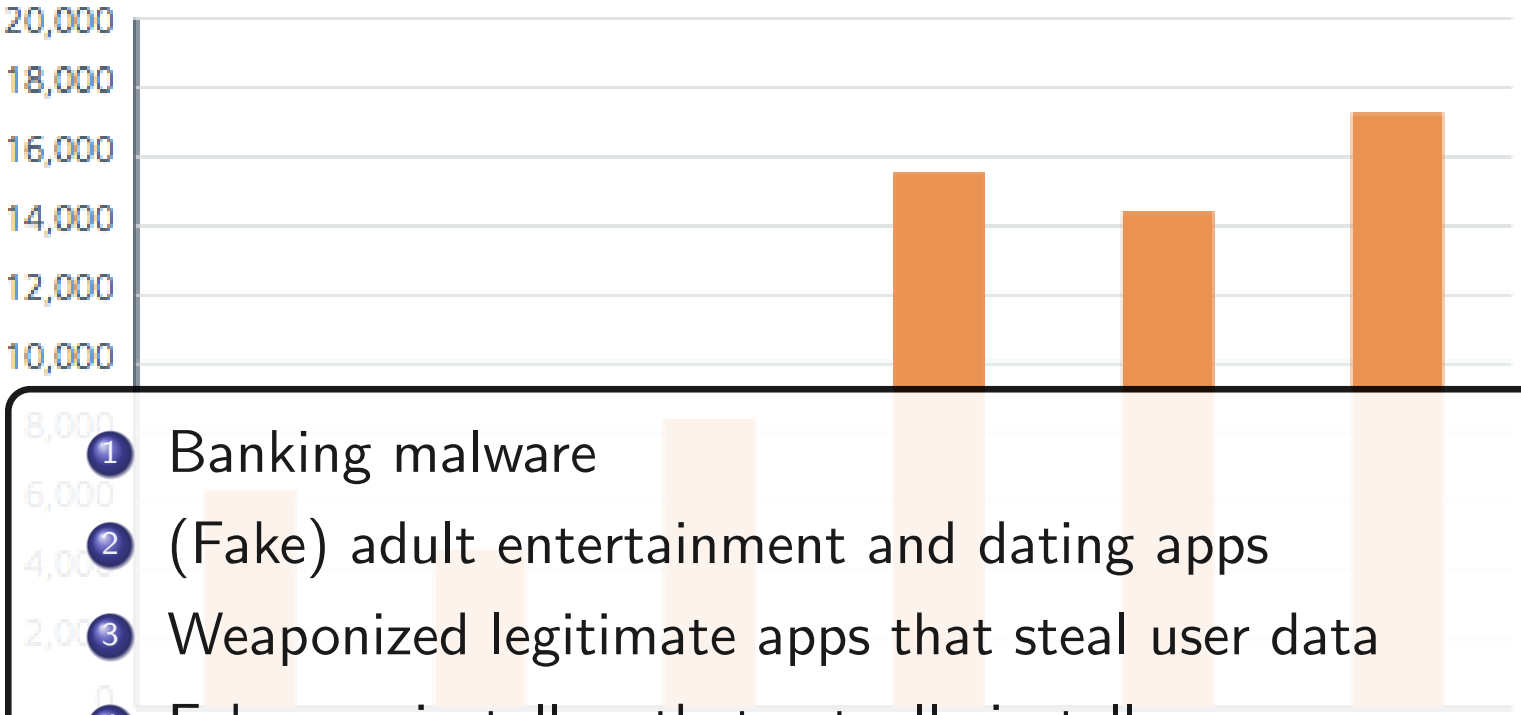
A.A. 2013–2014

McAfee Q2 2013
Threat Report for Mobile Malware Threats



Source: McAfee Threats Report: Second Quarter 2013

McAfee Q2 2013
Threat Report for Mobile Malware Threats



1. Banking malware
2. (Fake) adult entertainment and dating apps
3. Weaponized legitimate apps that steal user data
4. Fake app installers that actually install spyware

Source: McAfee Threats Report: Second Quarter 2013

Android Malware: the Rise

Why?

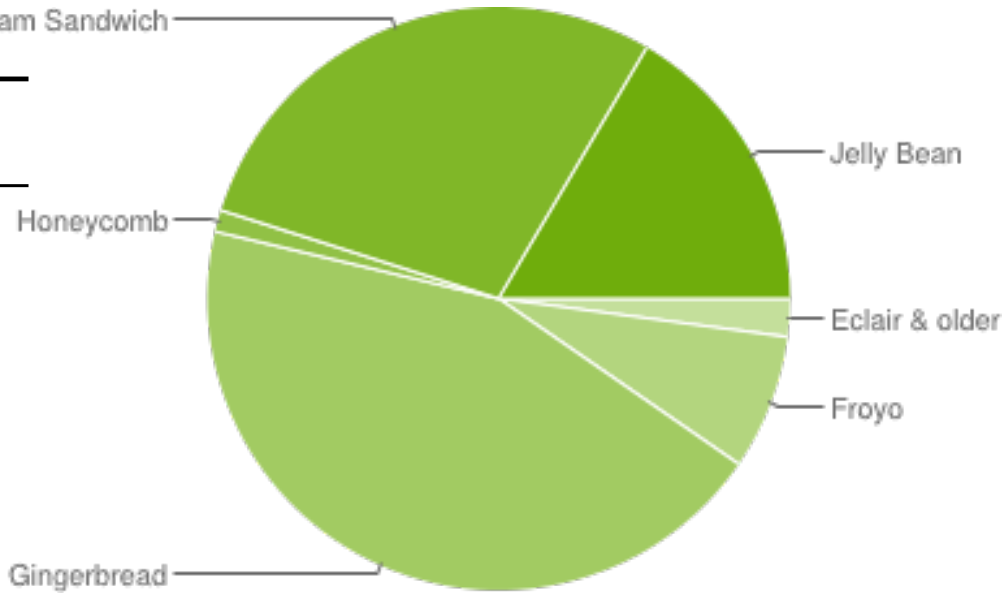
Android Malware: the Rise

The rise of Android malware is due to many factors

- Widely adopted on heterogeneous devices
- Producers push patches/updates slowly
- Operators' and Producers' customizations (often closed-source)
- Rooted devices, jailbreaks
- Several custom ROMS: CyanogenMod, MIUI,
- Custom kernels, modems
- A number of interesting information on a phone
- Few (or none) barriers in official markets
- Unofficial markets without control

Android Malware: the Rise

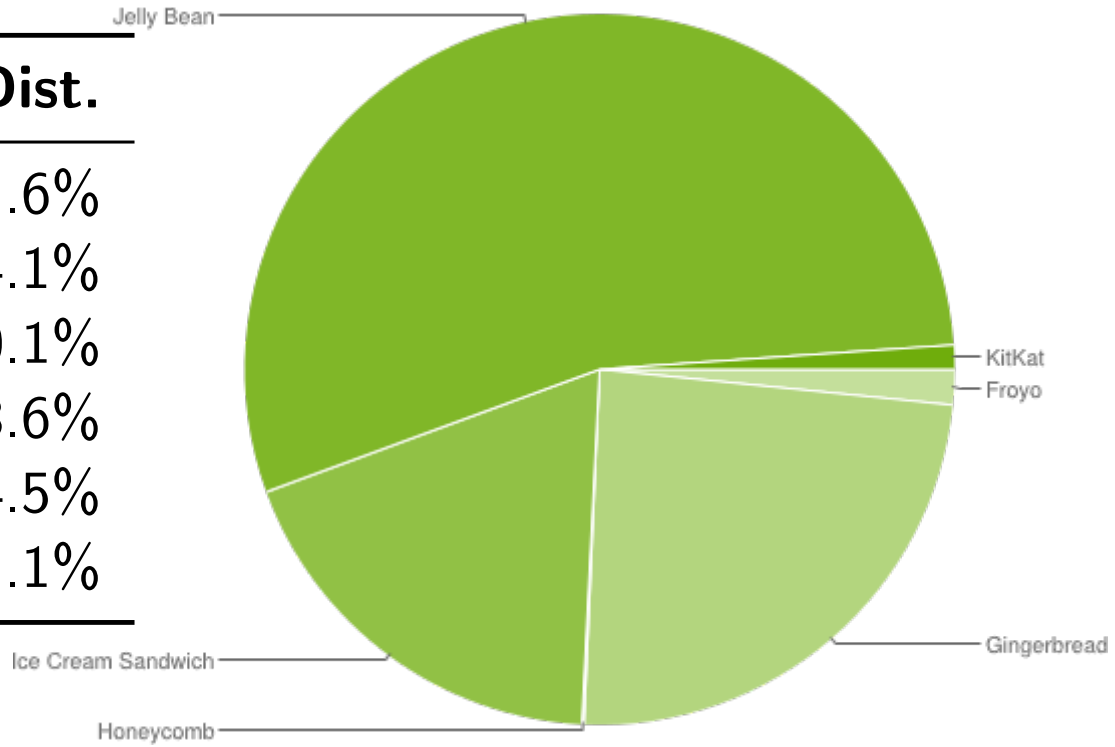
Version	Codename	Dist.
1.6	Donut	0.2%
2.1	Eclair	1.9%
2.2	Froyo	7.5%
2.3 - 2.3.7	Gingerbread	44.1%
3.1 - 3.2	Honeycomb	1.2%
4.0.3 - 4.0.4	ICS	28.6%
4.1	Jelly Bean	16.5%



Source: Android Developers (Mar. '13)

Android Malware: the Rise

Version	Codename	Dist.
2.2	Froyo	1.6%
2.3 - 2.3.7	Gingerbread	24.1%
3.2	Honeycomb	0.1%
4.0.3 - 4.0.4	ICS	18.6%
4.1 - 4.3	Jelly Bean	54.5%
4.4	KitKat	1.1%



Source: Android Developers (Dec. '13)

Information and Resources on a Phone

- **Personal information**: SMS, contacts, mails, ...
- **Espionage**: intercepting calls, SMSs, location, ...
- **Access** to enterprise networks
- **Money**: sending SMS to premium-rate numbers
- **Money**: many phones have direct access to CC
- **SPAM**: phone $\Rightarrow$ perfect spambot

Background

Dalvik and Zygote

Dalvik

Android Applications are coded in Java and interpreted by a custom VM, the DVM

Zygote

Every App has its own DVM instance, spawned by a Zygote process from where it inherits resources

Native Code

Native (ARM) code may be executed by an App through JNI or natively (NDK)

Dalvik and Zygote

Dalvik

Android Applications are coded in Java and interpreted by a custom VM, the DVM

Zygote

Every App ...gote process fro...

All running on top of a Linux OS

Native Co...

Native (ARM) code may be executed by an App through JNI or natively (NDK)

Security Model

No application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user

Security Model

No application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user

Sandboxing

Every App has its own UID/GID, used to enforce system-wide DAC

Permissions

To be granted a permission, App must explicitly request it (e.g., send an SMS, place a call)

Security Model

No application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user

Sandboxing

Every App has its own UID/GID, used to enforce system-wide DAC

Permissions

To be granted a permission, App must explicitly request it (e.g., send an SMS, place a call)

All types of applications—Java, native, and hybrid—are sandboxed in the same way and have the same degree of security from each other.

Activity

- Provides a screen with which users can interact;
- e.g., write a text, view a map, send an email.

Service

- Runs in the background like Unix Dæmons;
- e.g., async network operations, receiving data;

Receiver

- Listen (and respond) to broadcast events from the system;
- e.g., receive a SMS, incoming calls;

Content Provider

- define a storage-agnostic abstraction to transparently access data;
- enforce access control;

Intents

"An abstract representation of an operation to be performed"

Three main components are activated by intents.
Intents have different meaning/purpose depending on the recipient.

Intent Meaning per Recipient

- Activity: an action that must be performed (e.g., to send an e-mail, an App will broadcast the corresponding intent; the email activity will therefore be executed)
- Service: similar to activity
- Receiver: a container for received data.

Manifest File

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/[...]"
    package="test.AndroidSMS"
    android:versionCode="1"
    android:versionName="1.0">

<uses-permission android:name="[...].RECEIVE_SMS" />
<uses-permission android:name="[...].SEND_SMS" />
<uses-permission android:name="[...].INTERNET" />

<application android:label="@string/app_name" >
  <receiver android:name=".SMSReceiver">
    <intent-filter>
      <action android:name="[...].Telephony.SMS_RECEIVED" />
    </intent-filter>
  </receiver>
</application>
```

Manifest File

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/[...]"
    package="test.AndroidSMS"
    android:versionCode="1"
    android:versionName="1.0">

<uses-permission android:name="[...].RECEIVE_SMS" />
<uses-permission android:name="[...].SEND_SMS" />
<uses-permission android:name="[...].INTERNET" />

<application android:label="@string/app_name" >
  <receiver android:name=".SMSReceiver">
    <intent-filter>
      <action android:name="[...].Telephony.SMS_RECEIVED" />
    </intent-filter>
  </receiver>
</application>
```

Manifest File

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/[...]"
    package="test.AndroidSMS"
    android:versionCode="1"
    android:versionName="1.0">

<uses-permission android:name="[...].RECEIVE_SMS" />
<uses-permission android:name="[...].SEND_SMS" />
<uses-permission android:name="[...].INTERNET" />

<application android:label="@string/app_name" >
  <receiver android:name=".SMSReceiver">
    <intent-filter>
      <action android:name="[...].Telephony.SMS_RECEIVED" />
    </intent-filter>
  </receiver>
</application>
```

Binder

IPC/RPC

The **Binder** protocol allows fast inter-process communication between Apps or between Apps and the system. It also allows Apps to invoke other components' functions (e.g., to place a call or to send a SMS).

AIDL

The Android Interface Definition Language is used to define which methods of a service can be invoked remotely, among with their parameters. AIDL specifications for Android's core services are available online.

Binder Driver

The **Binder** core is implemented as a device driver. User-space processes (Apps) can interact with the driver through the `/dev/binder` virtual device.

`ioctl`

`ioctls` are used to by Apps to interact with the **Binder**. Each `ioctl` takes as argument a command and a data buffer.

`BINDER_WRITE_READ`

Allows data to be sent/received among Apps.

Android Malware

Dissecting Android Malware: Characterization and Evolution

http://www.csc.ncsu.edu/faculty/jiang/pubs/OAKLAND12.pdf

Infection vectors

Repackaged apps

Most used infection vector so far

- locate and download popular apps
- disassemble and embed malicious payload
- reassemble and resubmit

Often mixed with *update attacks*

Other means

- Drive by
- Spyware
- Trojans
- ...

Activation

Android apps have many different entry points that can be leveraged by malware to start their malicious activities

Most used events

- `BOOT_COMPLETED`
- `SMS_RECEIVED`
- `PACKAGE_*`
  (Actions on apps)
- `ACTION_MAIN`
  (repackaged apps that replace main activity of original app)
- ...

Payloads

Common payloads

- Privilege escalation: exploid, RATC, Zimperlich
- Remote control: mainly through HTTP C&C
- Financial charges: SMS to premium numbers, automatically subscribing to premium services
- Information stealing

One of the first really sophisticated malware found in the wild

Main Features

- Root exploits (encrypted payload)
- Remote control through C&C channels (encrypted)
- Stealthy installation of embedded apps
- Obfuscation: strings, servers, payloads (also through JNI)
- Latest version use update attacks

Distributed through third-party Android Markets in China

Main Features

- Anti-analysis: repackaging, integrity checks, obfuscation
- Anti-analysis(2): update attacks, dynamic code loading
- Anti-analysis(3): detection of anti-virus software
- 2 C&C channels: one classic and one based on blogs on baidu (encrypted)

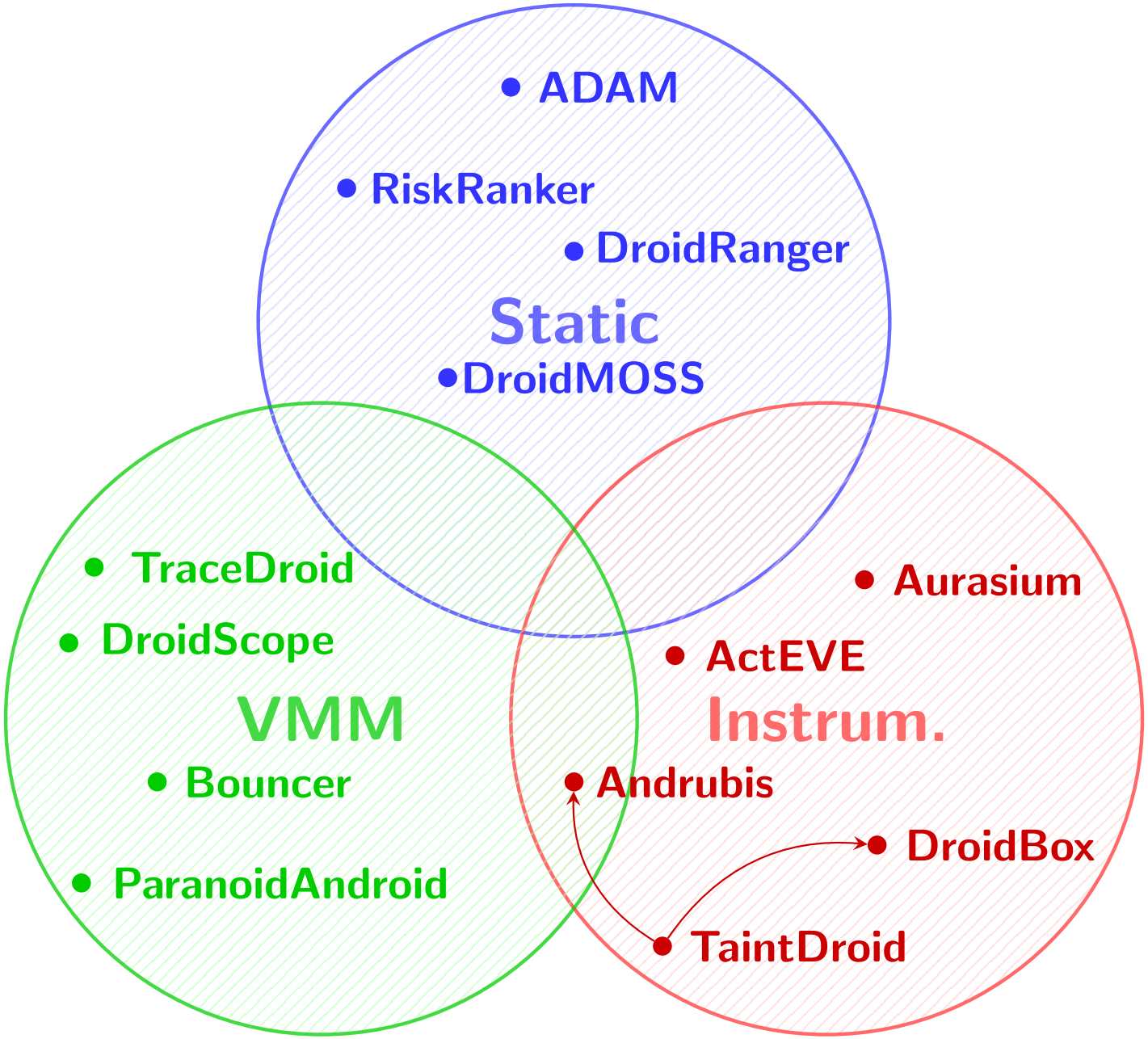
Hands on!

`https://code.google.com/p/androguard/`
`https://code.google.com/p/android-apktool/`
`http://security.di.unimi.it/sicurezza1314/demo.tar.gz`

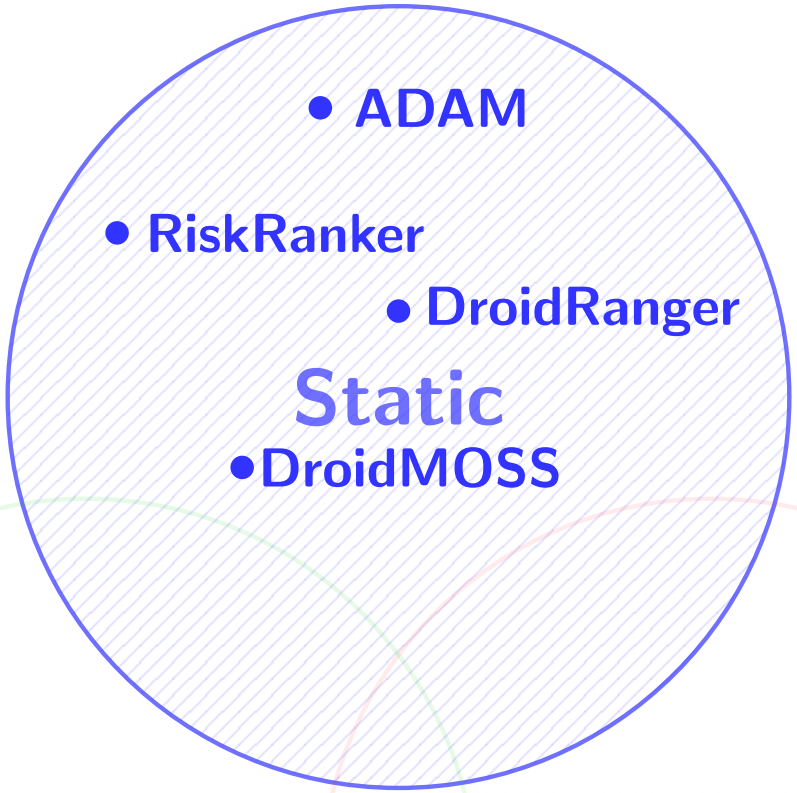
- Completare `demo_decompiler.py`
- Usarlo per reversare `demo/Demosicurezza/bin/Demosicurezza-debug.apk`
- Repackare `Demosicurezza_nolog.apk` in modo che stampi su logcat il contenuto del SMS ricevuto. Avete i sorgenti in `Demosicurezza/` per comodità, ma il repackaging va fatto con apktool.

Android Malware Analysis

Malware Analysis



Malware Analysis: Static



Pros

- Many information in the Manifest
- Java is relatively easy to decompile
- Potentially "sees" the whole behavior

Cons

- Obfuscation & Optimization
- Reflection
- Dynamic code, Native code

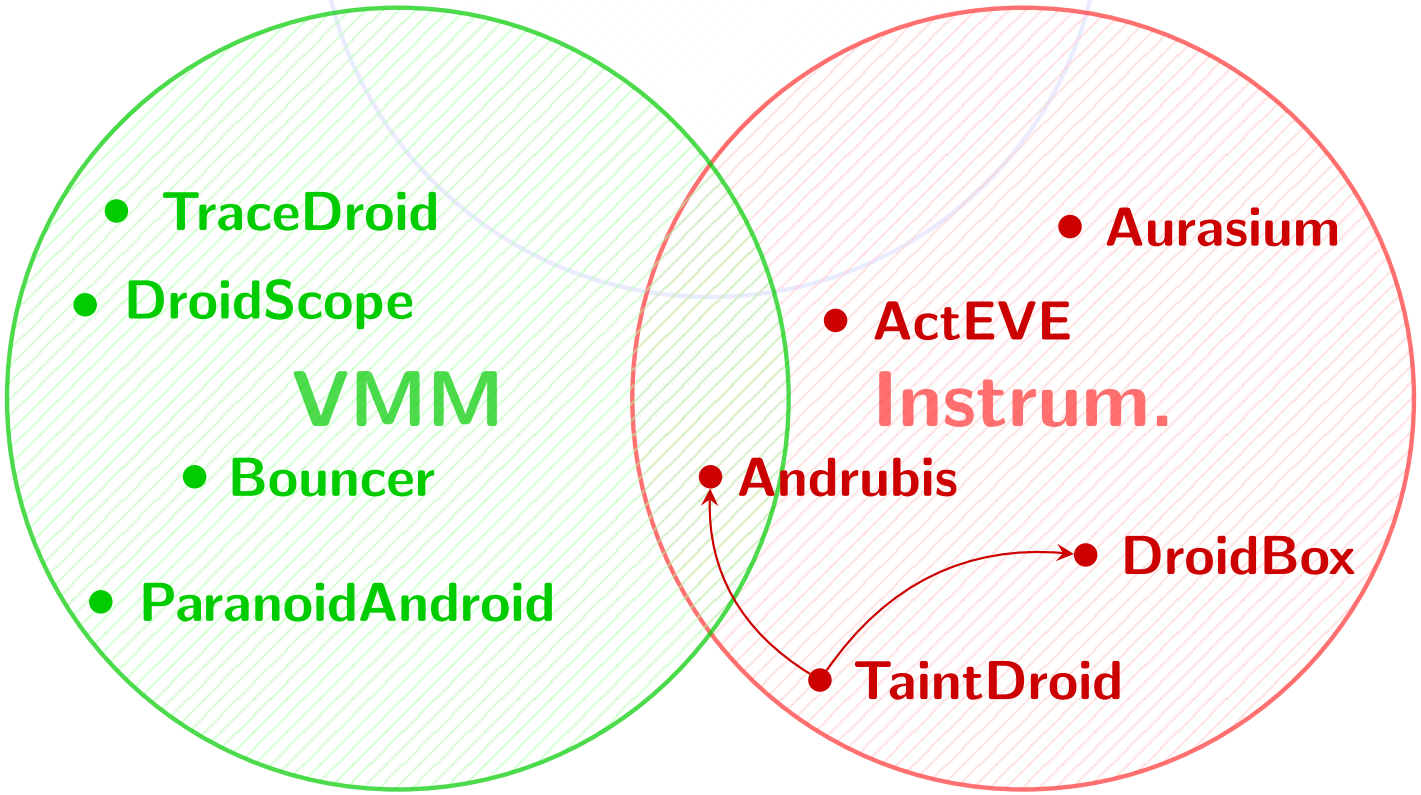
Malware Analysis: Dynamic

Pros

- Resilient to obfuscation
- Potentially transparent (VMM)
- Less comples than static

Cons

- Code coverage
- VMI can be cumbersome (VMM)
- Instrumentation can be detected



Traditional Roots

A well-established technique to characterize behaviour of a proccess through system calls

(e.g. Anubis—`http://anubis.iseclab.org/`)

Can it be applied to Android?

- Android architecture is different than traditional devices
- Are all the *interesting* behaviours achieved through system calls?
  - Dalvik VM
    (Android behaviour such as sending an SMS or receiving a call)
  - OS interactions
    (i.e. reading/writing a file)

CopperDroid

A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors

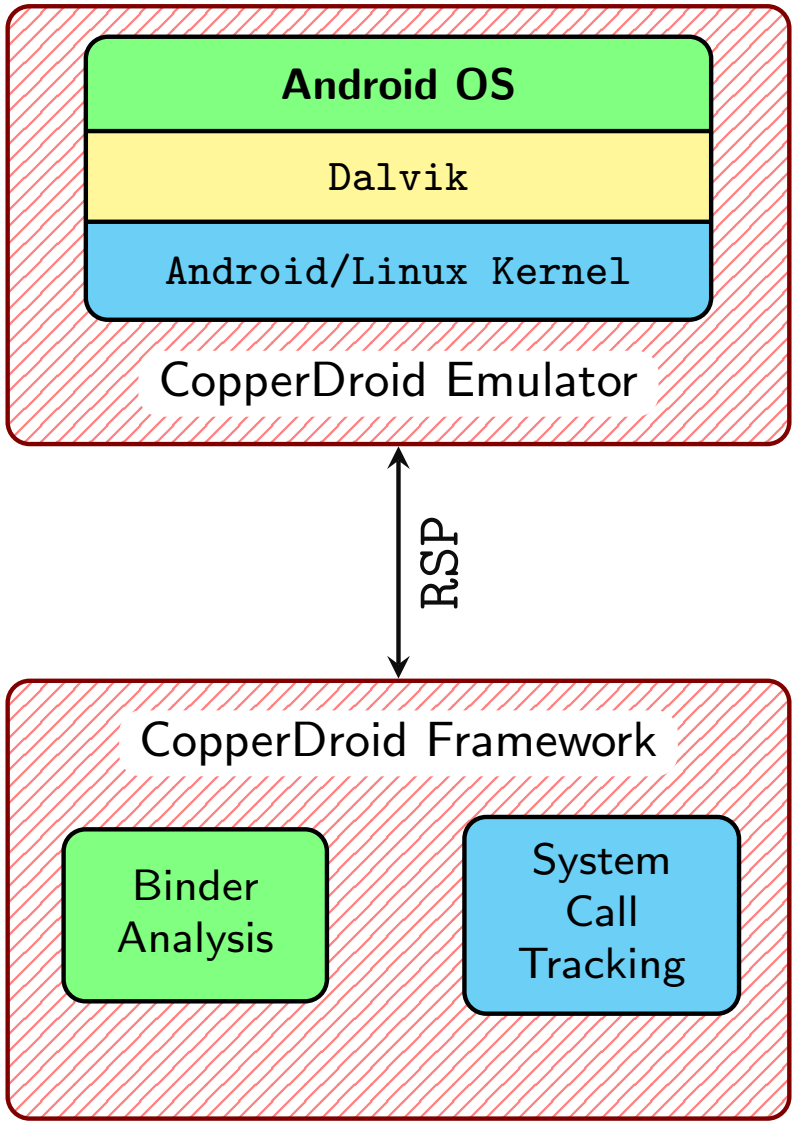
http://www.artificialstudios.org/alessandro.reina/pubs/eurosec13.pdf

Analysis Framework

Automatically reconstructs the behaviors of Android malware

- Unified analysis that avoid 2-level VMIs
  - All the behaviors are eventually achieved via system interactions
  - Dalvik VMI optionally enabled to gather further insights
- Dynamically stimulates Apps to disclose additional behaviors
- Extensive evaluation on 2,900 Android malware

Architecture



Invoking Syscalls

Like on Intel, on ARM architecture invoking a system call induces a user-to-kernel transiction.
(current `CPL` is stored in the `cpsr` register)

System calls on Linux ARM

- On ARM invoked through the `swi` instruction (**S**oft**W**are **I**nterrupt)
- `r7` contains the number of the invoked syscall
- `r0`-`r5` contain parameters
- `lr` contains the return address

Tracking System calls

System call Analysis

- Intercept when a syscall is invoked
- We need to intercept return to user-space too!
- There is no `SYSEXIT/SYSRET` to intercept
- Not every syscall actually *returns* to `lr`
  (e.g., `exit`, `execve`)

CopperDroid's Approach

- instruments QEMU's emulation of the `swi` instruction
- instruments QEMU to intercept every `cpsr_write`
  (Kernel → User)

When dealing with out-of-the-box analyses it is essential to retrieve information about the analyzed system

CopperDroid VMI

CopperDroid inspects the Android Kernel to retrieve the following:

- Process Names
- PIDs & TIDs
- Process resources
- ...

Binder

The Binder protocol is the core of Android IPC/RPC.

- Intents are carried through binder
- Interactions with the system (SMS/Calls) go through binder
- Binder driver enforces (some) permission policies

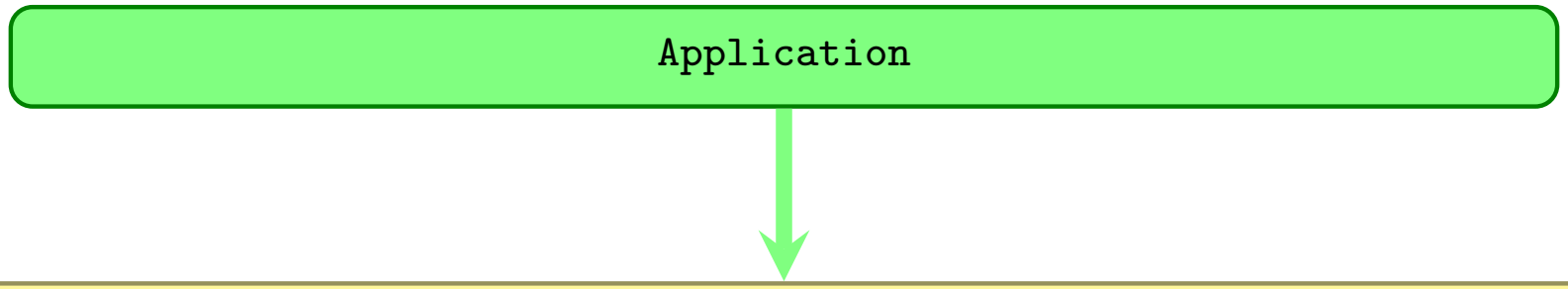
For example, applications cannot send SMSs on their own, but must invoke (RPC) the proper system service to do that.

Binder

Application

```
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage("7855551234", null, "Hi There", null, null);
```

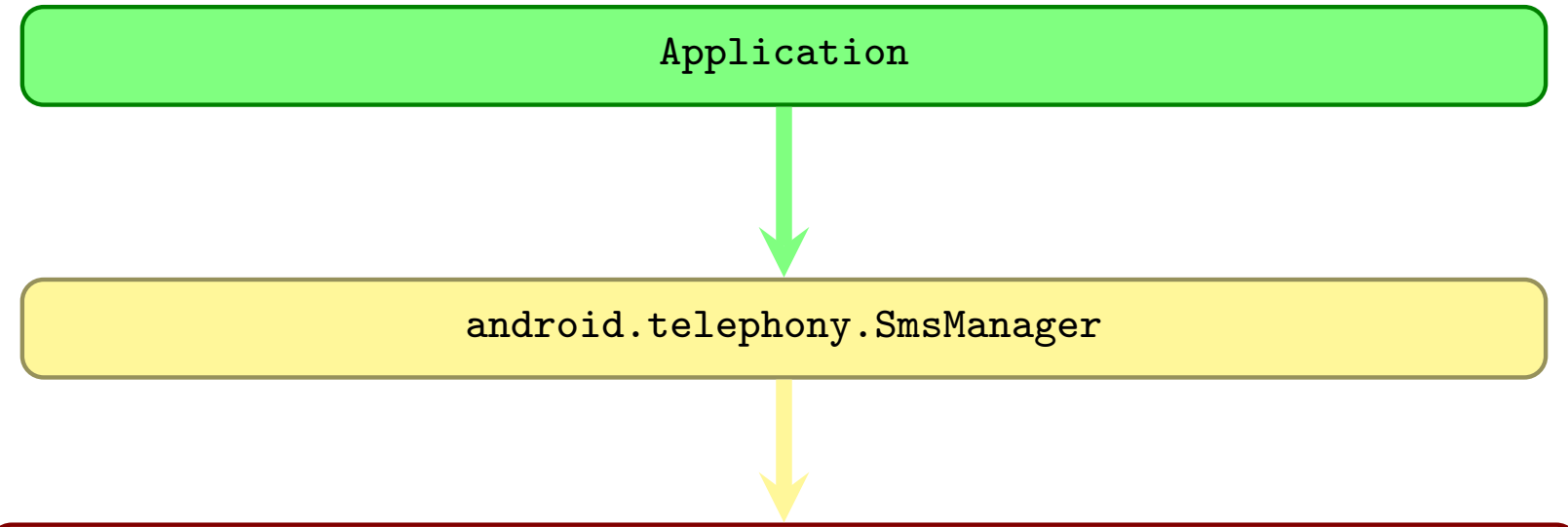
Binder



`android.telephony.SmsManager`

```
public void sendTextMessage(...) {
  ...
  ISms iccISms = ISms.Stub.asInterface(ServiceManager.getService("isms"));
  if (iccISms != null)
    iccISms.sendText(destinationAddress, scAddress, text, sentIntent, deliveryIntent);
  ...
```

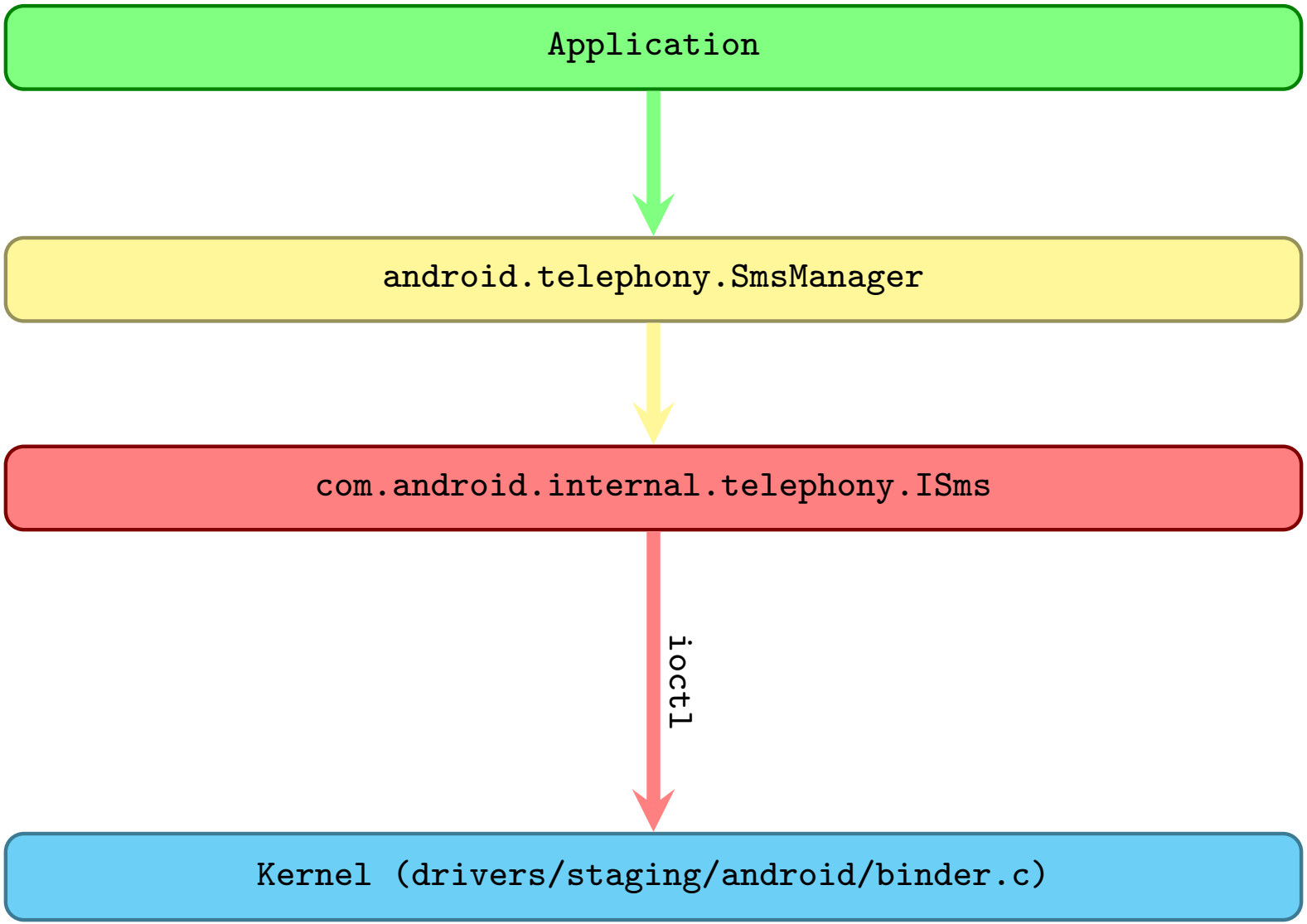
Binder



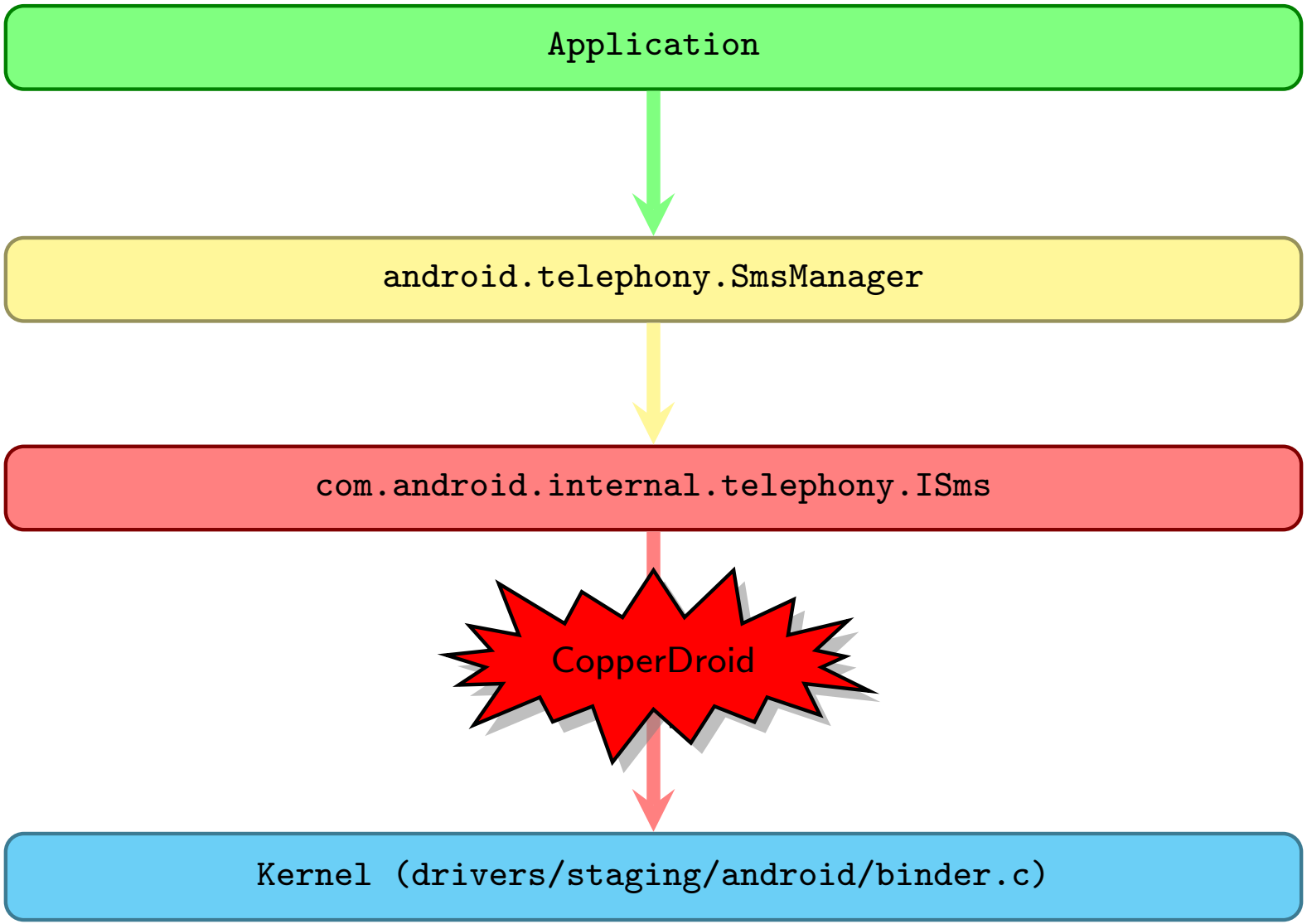
com.android.internal.telephony.ISms

```
public void sendText(...) {
  android.os.Parcel _data = android.os.Parcel.obtain();
  try {
    _data.writeInterfaceToken(DESCRIPTOR);
    _data.writeString(destAddr);
    ...
    mRemote.transact(Stub.TRANSACTION_sendText, _data, _reply, 0);
  }
```

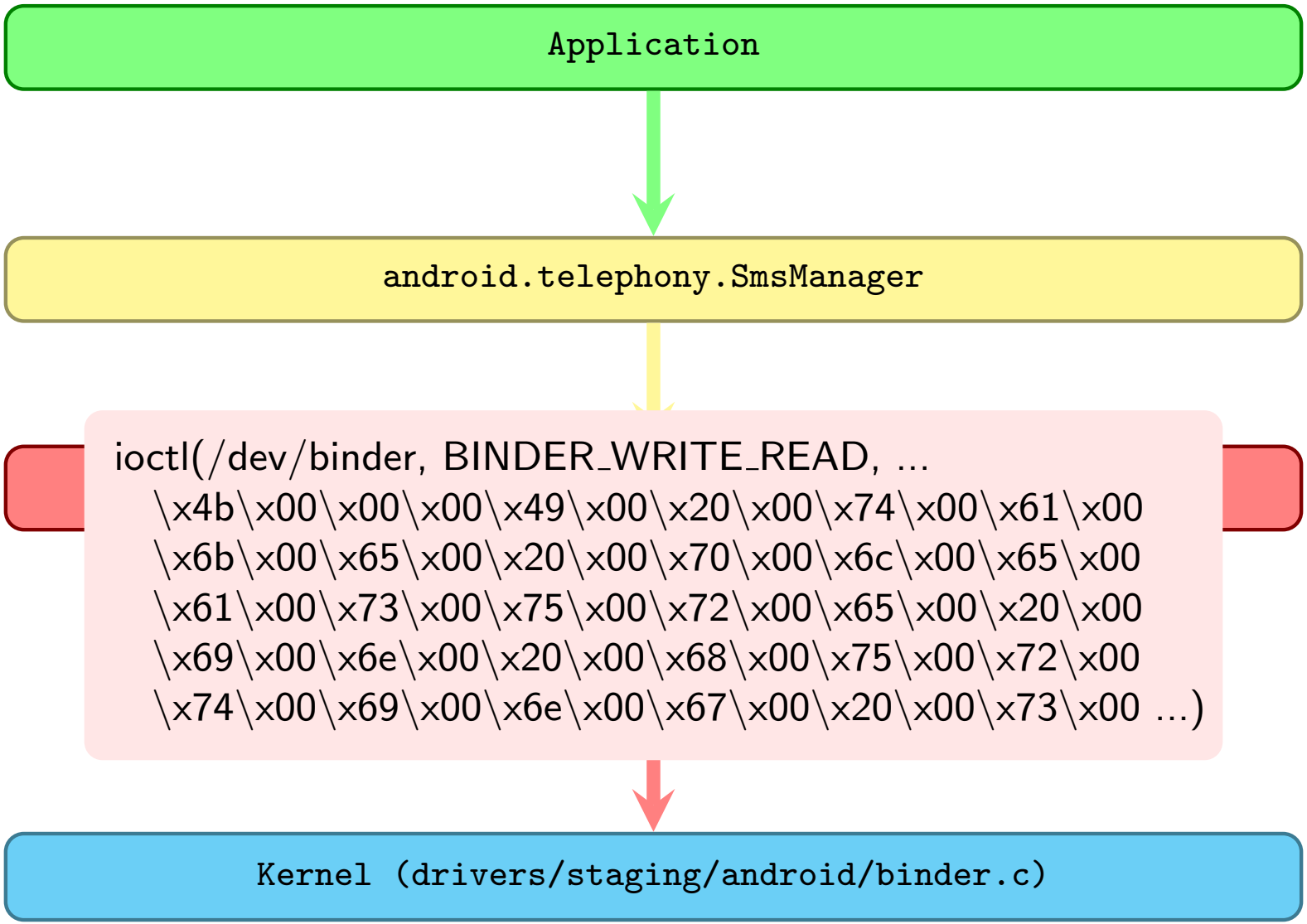
Binder


Application
android.telephony.SmsManager
com.android.internal.telephony.ISms
ioctl
Kernel (drivers/staging/android/binder.c)

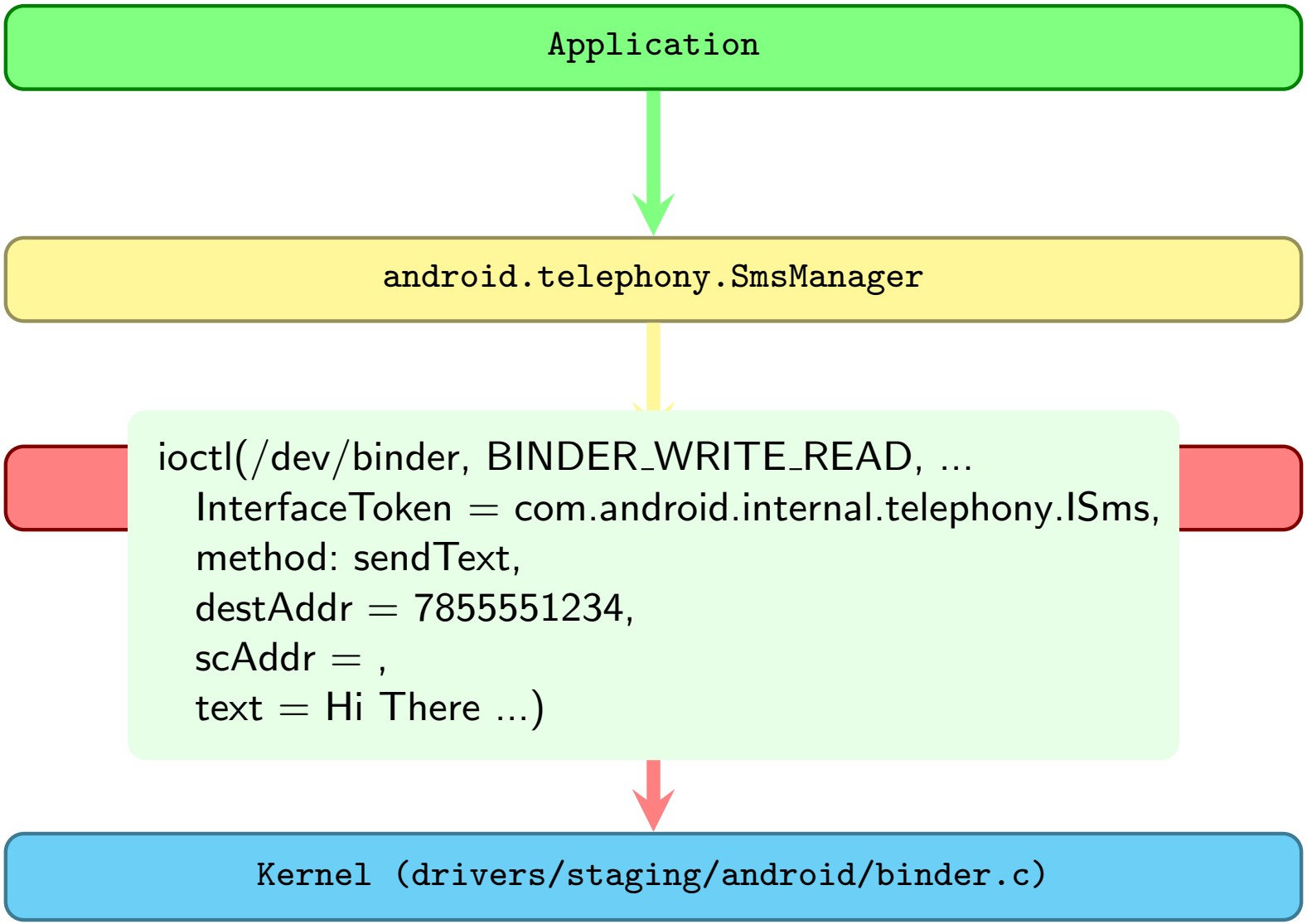
Binder


Application
android.telephony.SmsManager
com.android.internal.telephony.ISms
CopperDroid
Kernel (drivers/staging/android/binder.c)

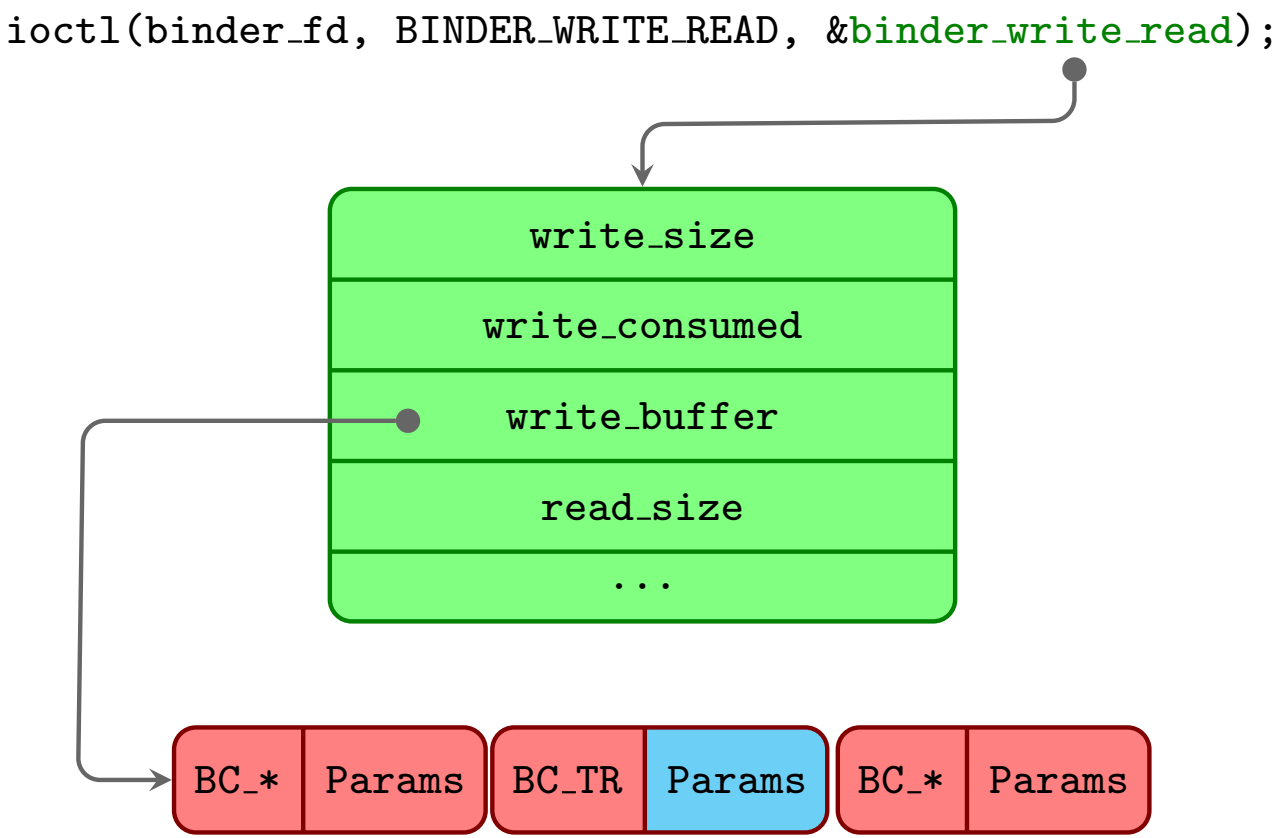
Binder


Application
android.telephony.SmsManager
ioctl(/dev/binder, BINDER_WRITE_READ, ...
\x4b\x00\x00\x00\x49\x00\x20\x00\x74\x00\x61\x00
\x6b\x00\x65\x00\x20\x00\x70\x00\x6c\x00\x65\x00
\x61\x00\x73\x00\x75\x00\x72\x00\x65\x00\x20\x00
\x69\x00\x6e\x00\x20\x00\x68\x00\x75\x00\x72\x00
\x74\x00\x69\x00\x6e\x00\x67\x00\x20\x00\x73\x00 ...)
Kernel (drivers/staging/android/binder.c)

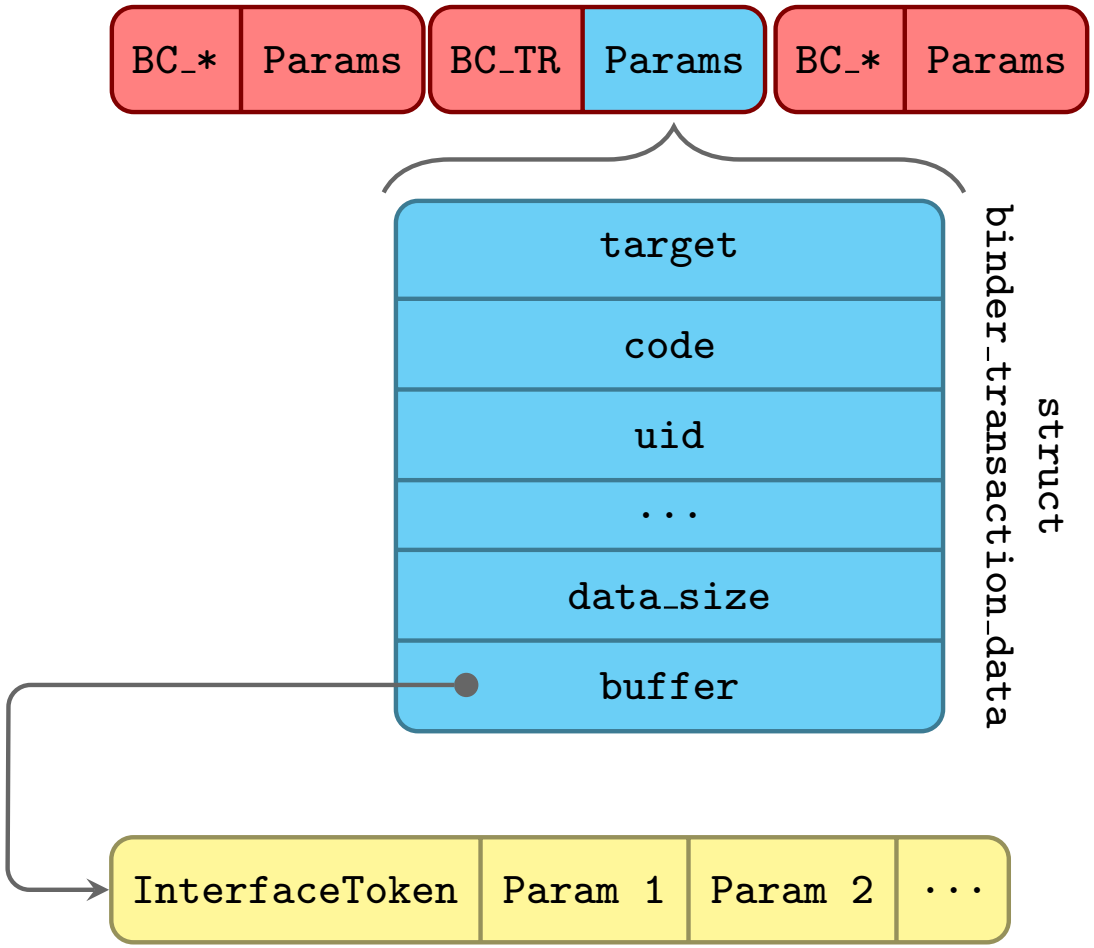
Binder


Application
android.telephony.SmsManager
ioctl(/dev/binder, BINDER_WRITE_READ, ...
InterfaceToken = com.android.internal.telephony.ISms,
method: sendText,
destAddr = 7855551234,
scAddr = ,
text = Hi There ...)
Kernel (drivers/staging/android/binder.c)

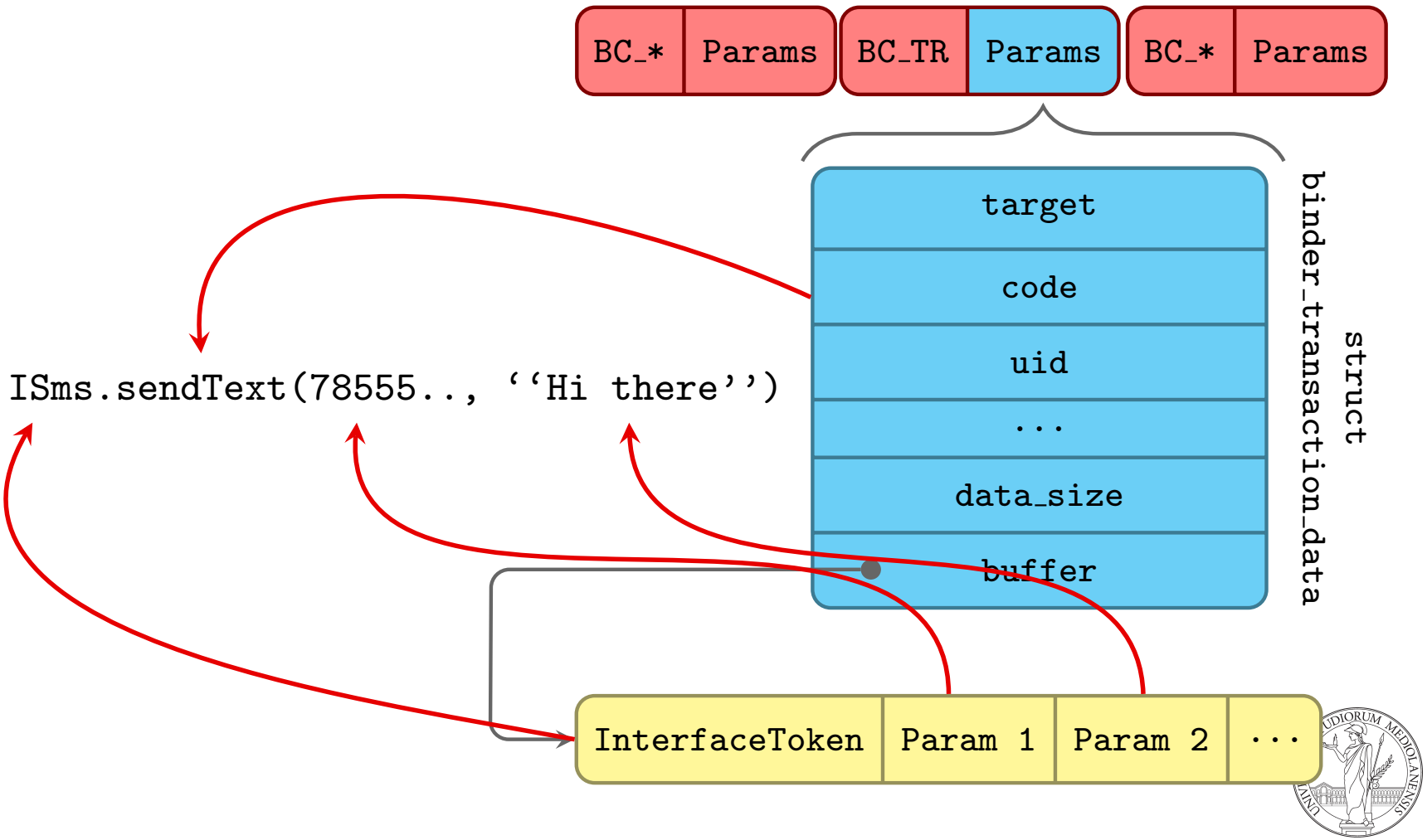
CopperDroid *deeply* inspects the Binder protocol intercepting a subset of the `ioctls` issued by userspace Apps.



CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs



CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs



Interface	Method
IPhoneSubInfo	getDeviceId
	getDeviceSvn
	getSubscriberId
	getIccSerialNumber
	getLine1Number
	getLine1AlphaTag
	getVoiceMailNumber
ISms	getAllMessagesFromIccEf
	updateMessageOnIccEf
	copyMessageToIccEf
	sendData
	sendText
	sendMultipartText

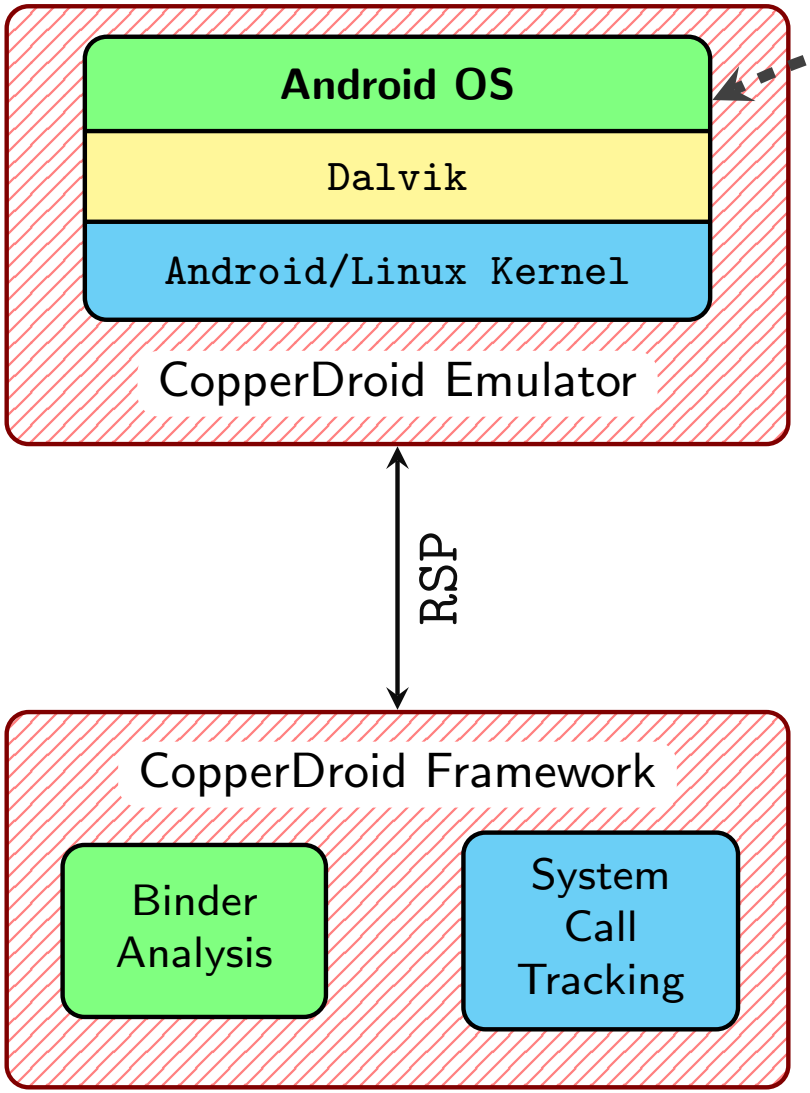
Stimulation

Android malware needs to be properly stimulated to trigger more malicious behaviors and increase coverage of dynamic analysis.

CopperDroid Ad-Hoc Stimuli

1. Identifies events the target reacts to
   (mostly contained in the Manifest file)
2. During the analysis, injects custom events
   (of those identified as useful)

Stimulation



To inject events CopperDroid leverages MonkeyRunner

Evaluation

CopperDroid analyzed 1,226, 395, and 1,365 malware respectively from: Android Malware Genome Project, Contagio repository, and McAfee UK.

28% additional behaviors on 60% of Genome samples!
22% additional behaviors on 73% of Contagio samples!
28% additional behaviors on 61% of McAfee samples!

#	Malware Family	Stim.	Samples w/ Add. Behav.	Behavior w/o Stim.	Incr. Behavior w/ Stimuli	
1	ADRD	3.9	17/21	7.24	4.5	(63%)
2	AnserverBot	3.9	186/187	31.52	8.2	(27%)
3	BaseBridge	2.9	70/122	16.44	5.2	(32%)
4	BeanBot	3.1	4/8	0.12	3.8	(3000%)
5	CruseWin	4.0	2/2	1.00	2.0	(200%)
6	GamblerSMS	4.0	1/1	1.00	3.0	(300%)
7	SMSReplicator	4.0	1/1	0.00	6.0	($\perp$)
8	Zsone	5.0	12/12	16.67	3.8	(23%)

Future and ongoing research directions

Stimulation

So far, low hanging fruit approach:

- Need to find more complex, yet *time-efficient* solutions (Tainting/concolic exec run in the order of hours)
- Human-driven stimulation (scalable?)
- Application similarity

Detection

Leverage CopperDroid base of knowledge to create new detection approaches:

- lightweight
- non-root (?)
- detection of repackaged apps (?)