SYSTEMS SECURITY RESEARCH LAB

ROYAL HOLLOWAY UNIVERSITY OF LONDON

COPPERDROID

On the Reconstruction of Android Malware Behaviors

Oct 11, 2014
HackInBo 2014

Lorenzo Cavallaro

Systems Security Research Lab (S^2Lab)
s2lab.isg.rhul.ac.uk

Information Security Group
Royal Holloway University of London

WHO AM I?

- Post-doc researcher, VU Amsterdam, working with:
  (Jan 2010—Dec 2011)
  - → Prof. Andy Tanenbaum
    (OS dependability)
  - → Prof. Herbert Bos
    (memory errors, malware analysis, and taint analysis)
- Post-doc researcher, UC at Santa Barbara, working with:
  (Apr 2008—Jan 2010)
  - → Prof. Giovanni Vigna and Prof. Christopher Kruegel
    (malware analysis and detection)
- Visiting PhD student, Stony Brook University, working with:
  (Sep 2006—Feb 2008)
  - → Prof. R. Sekar
    (memory errors protections, taint analysis, malware analysis)

HACKINBO

SYSTEMS SECURITY RESEARCH LAB

ROYAL HOLLOWAY UNIVERSITY OF LONDON

WHO AM I?

- Post-doc researcher, VU Amsterdam, working with:
  (Jan 2010—Dec 2011)
  - → Prof. Andy Tanenbaum
    (OS dependability)
  - → Prof. Herbert Bos
    (memory errors, malware analysis, and taint analysis)
- Post-doc researcher, UC Santa Barbara, working with:
  (Apr 2008—Jan 2010)
  - → Prof. Giovanni Vigna and Prof. Christopher Kruegel
    (malware analysis and detection)
- Visiting PhD student, Stony Brook University, working with:
  (Sep 2006—Feb 2008)
  - → Prof. R. Sekar
    (memory errors protections, taint analysis, malware analysis)

Jan 2012 Lecturer (∼Assistant Professor) in the ISG
Jan 2014 Senior Lecturer (∼Associate Professor) in the ISG

Systems Security Research Lab (S^{2}Lab) — http://s2lab.isg.rhul.ac.uk

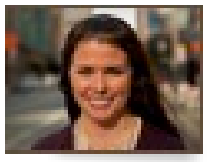
Information Security Group, Royal Holloway University of London
<lorenzo.cavallaro@rhul.ac.uk> — http://www.isg.rhul.ac.uk/sullivan

Royal Holloway
University of London

ANDROID

Google readies Android 'KitKat' amid 1 billion device activations milestone

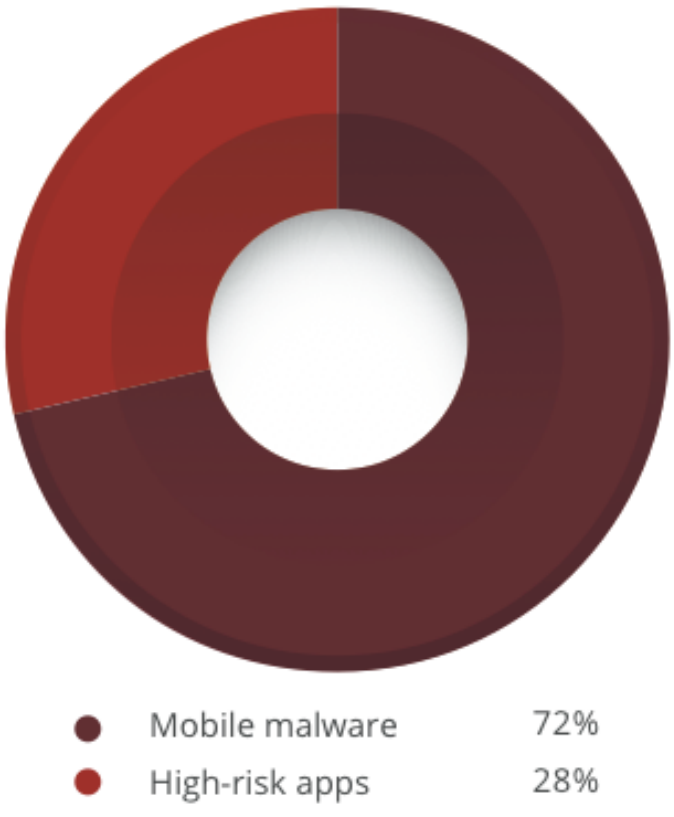
Summary: *Chocolate is nice and all, we all want to know more about how Google will have mobile users salivating for the next installment.*



By Rachel King for Between the Lines | September 3, 2013 -- 17:36 GMT (18:36 BST)

Follow @rachelking

ANDROID CUMULATIVE THREAT VOLUME



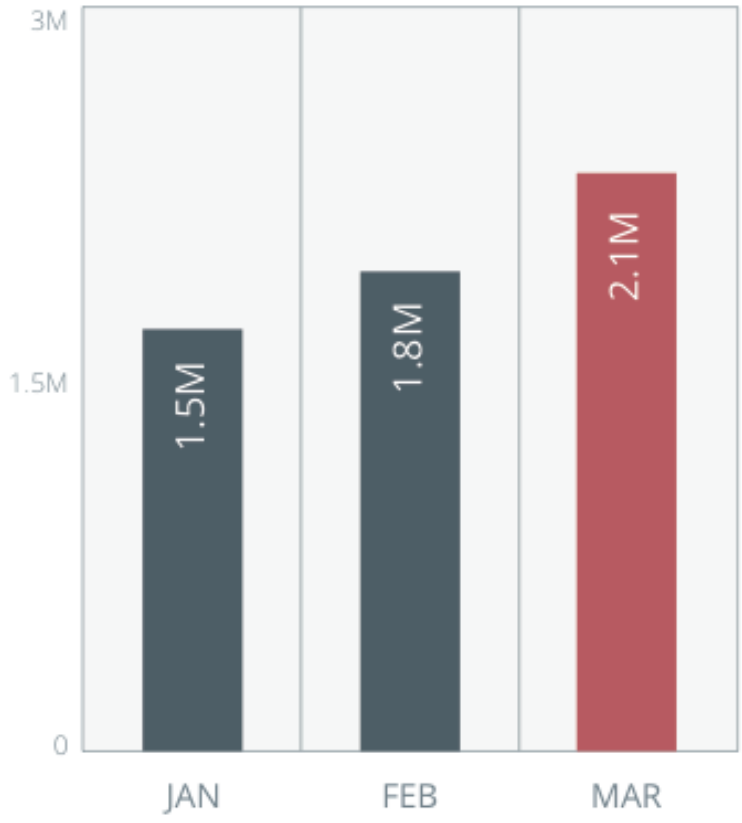


Figure: Source: TrendLabs 1Q 2014 Security Roundup

ANDROID CUMULATIVE THREAT VOLUME

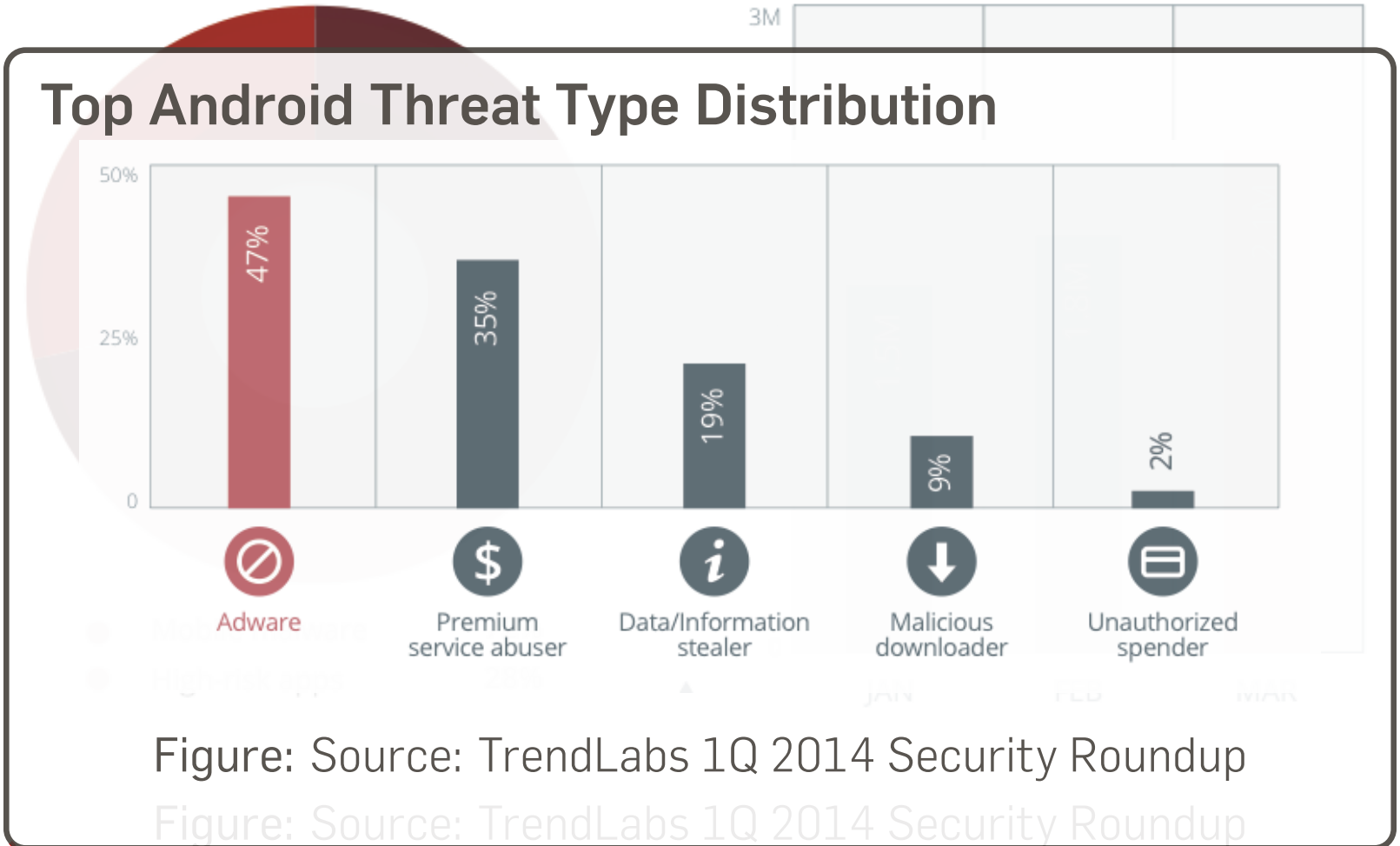


Figure: Source: TrendLabs 1Q 2014 Security Roundup

Figure: Source: TrendLabs 1Q 2014 Security Roundup

THE (NOT SO SHORT) INTRODUCTION TO ANDROID

- Modified Linux kernel
- Android apps written (mostly) in Java and run in a Java-like (Dalvik) VM as userspace processes
- Native code may be executed through JNI or native (NDK)
- Apps logically divided in components
  - → Activity, e.g., GUI components
  - → Services, similar to UNIX daemons
  - → Broadcast Receivers, to act upon the receipt of specific events, e.g., phone call, SMS
  - → Content Providers, storage-agnostic ACL-controlled abstractions to access data

ANDROID SECURITY MODEL

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

ANDROID 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 to enforce system-wide DAC

Permissions

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

ANDROID 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 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

INTENTS

An abstract representation of an operation to be performed

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="test.AndroidSMS.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="test.AndroidSMS.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="test.AndroidSMS.SMS_RECEIVED" />
    </intent-filter>
  </receiver>
</application>
```

THE BINDER PROTOCOL

IPC/RPC

The Binder protocol enables 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

THE BINDER PROTOCOL

Binder Driver

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

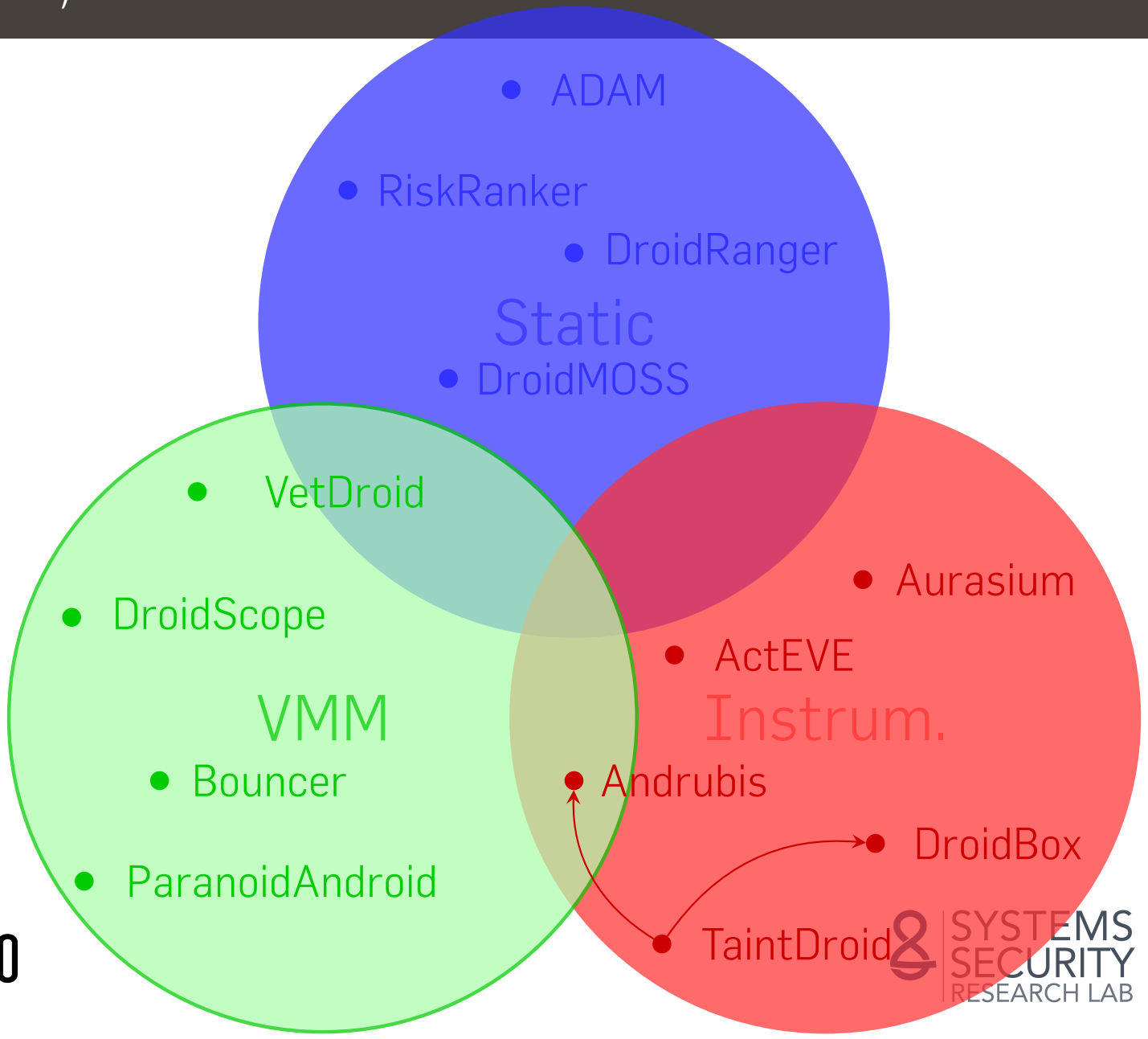
`ioctl`

`ioctls` are used to by Apps to interact with 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 ANALYSIS

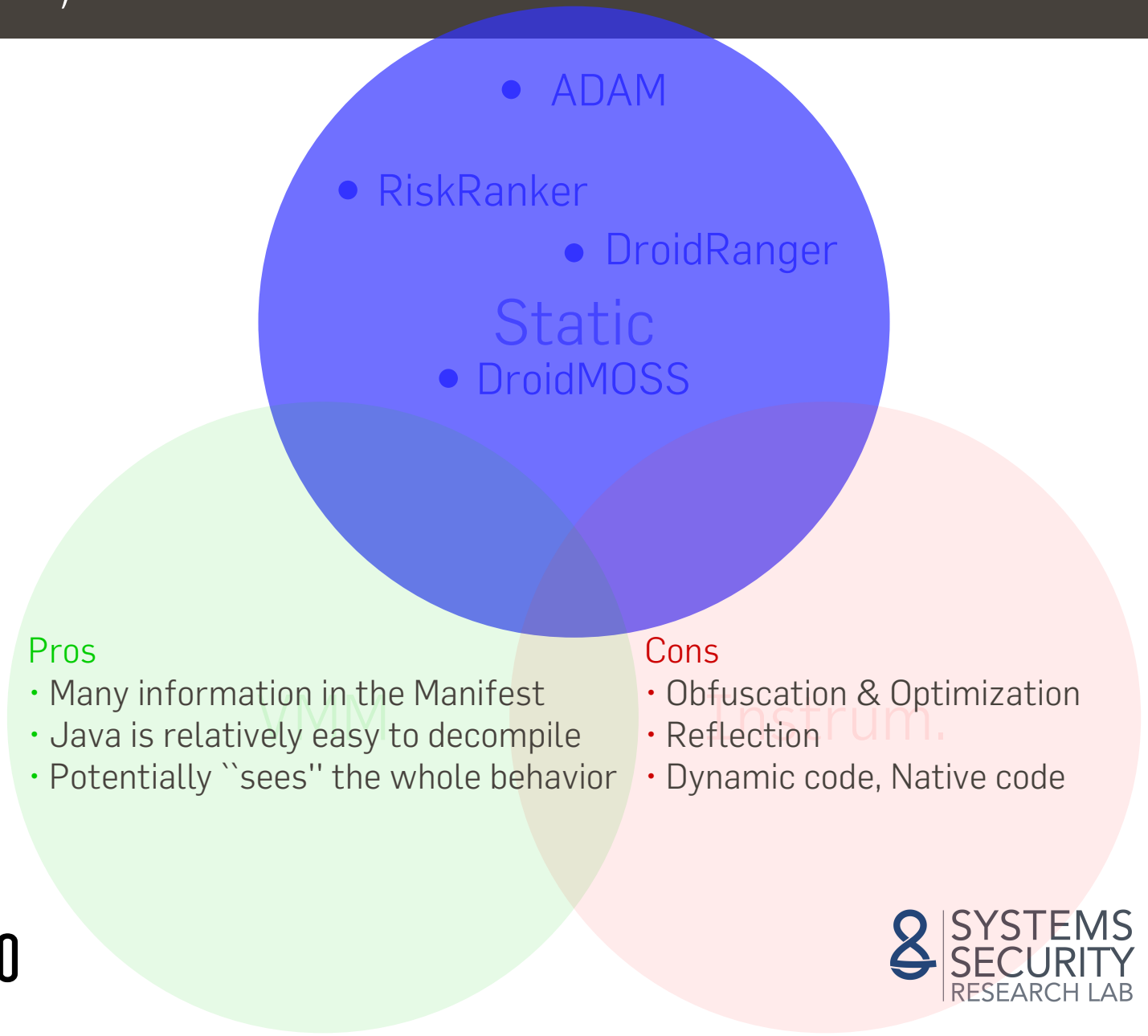


HACKINBO

SYSTEMS SECURITY RESEARCH LAB

ROYAL HOLLOWAY UNIVERSITY OF LONDON

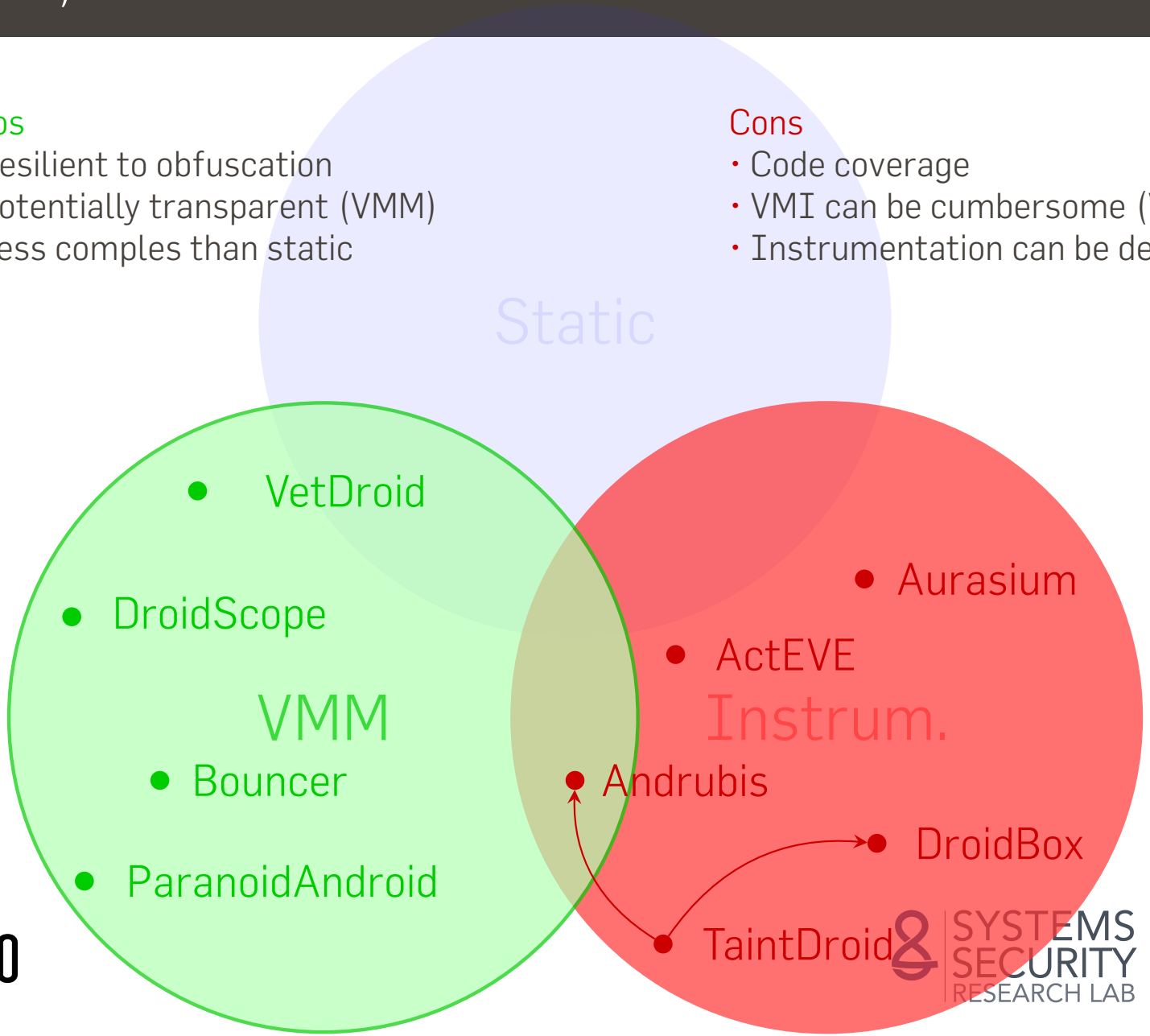
(ANDROID) MALWARE ANALYSIS: STATIC



(ANDROID) 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



HackInBo

SYSTEMS SECURITY RESEARCH LAB

ROYAL HOLLOWAY UNIVERSITY OF LONDON

SYSTEM-CALL CENTRIC ANALYSIS OF ANDROID MALWARE?

Traditional Roots

A well-established technique to characterize process behaviours

Can it be applied to Android?

- Android architecture is different than traditional devices
- Are all the interesting behaviours achieved through system calls?
  - → Dalvic VM
    (Android-specific behaviours, e.g., SMS, phone calls)
  - → OS interactions
    (e.g., creating a file, network communication)

COPPERDROID

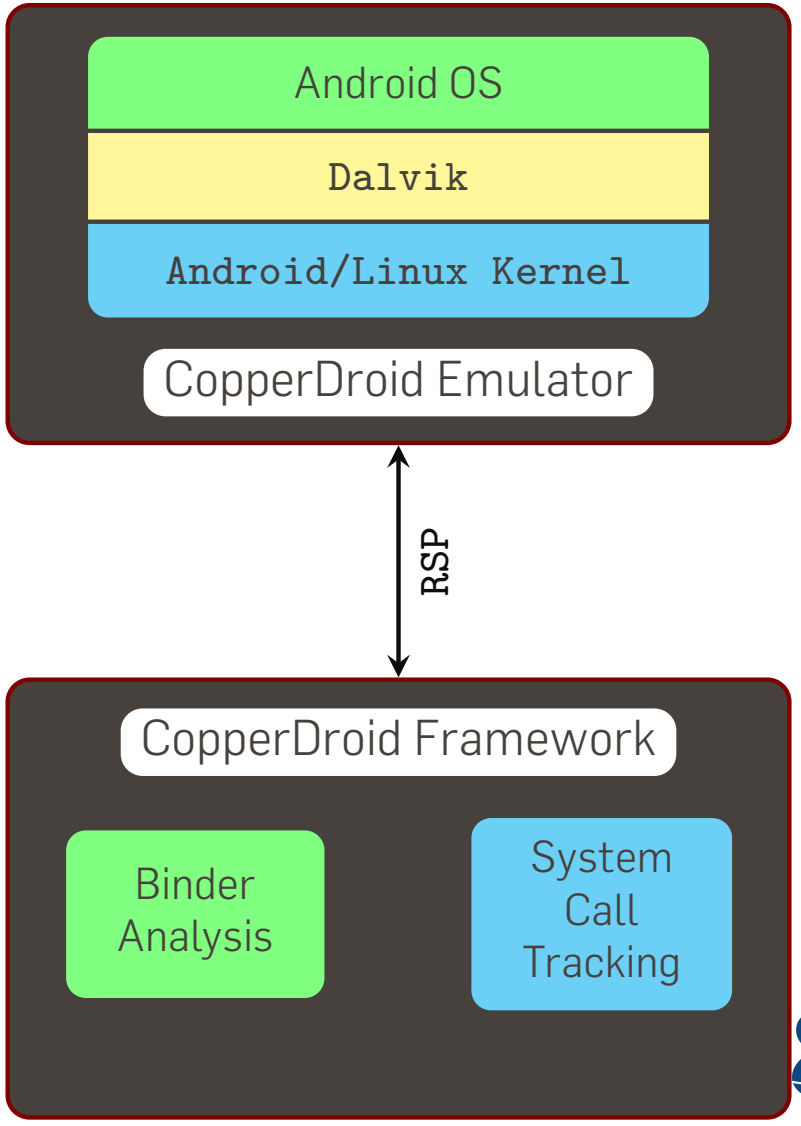
COPPERDROID

Analysis Goal

Automatically reconstructs the behaviors of Android (malicious) apps

- Unified system call-centric analysis
  - → Obs: behaviors are eventually achieved via system interactions
- Avoids 2-level (complex) VMIs
- Avoids invasive modification of the Android system (in fact, none)
- Android version-independent
- Dynamically stimulates Apps to disclose additional behaviors
- Extensive evaluation on 2,900+ Android malware

ARCHITECTURE (LEGACY)



SYSTEM CALLS ON LINUX ARM

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 (SoftWare Interrupt)
- `r7` contains the number of the invoked system call
- `r0-r5` contain parameters
- `lr` contains the return address

TRACKING SYSTEM CALLS

System Call Analysis

- Intercept when a system call is invoked
- We need to intercept return to user-space too!
- There is no `SYSEXIT/SYSRET` to intercept
- Not every system call 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)

TRACKING SYSTEM CALLS

System Call Analysis

[c5b02000 - 35 - 35 - zygote] fork( ) = 0x125
[c5b02000 - 35 - 35 - zygote] getpgid(0x41) = 0x23
[c5b02000 - 35 - 35 - zygote] setpgid(0x125, 0x23) = 0x0
[c1c18000 - 293 - 293 - zygote] getuid32( ) = 0x0
[c1c18000 - 293 - 293 - zygote] open(/acct/uid/0/tasks, ...
[c1c18000 - 293 - 293 - zygote] fstat64(0x13, 0xbef7f910) = 0x0
[c1c18000 - 293 - 293 - zygote] mprotect(0x40008000, 0x1000, 0x3) = 0x0
[c1c18000 - 293 - 293 - zygote] mprotect(0x40008000, 0x1000, 0x1) = 0x0
[c1c18000 - 293 - 293 - zygote] write(0x13 - /acct/uid/0/tasks, 0xa24c0 "'0'", 0x1) = 0x1

BRIDGING THE SEMANTIC GAP

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
- ...

BRIDGING THE SEMANTIC GAP

Observation: when executing kernel code, the base of the stack points to the current executing thread.

BRIDGING THE SEMANTIC GAP

Observation: when executing kernel code, the base of the stack points to the current executing thread.

arch/arm/include/asm/thread_info.h

```
#define THREAD_SIZE 8192
static inline struct thread_info *current_thread_info(void)
{
    register unsigned long sp asm ("sp");
    return (struct thread_info *) (sp & ~(THREAD_SIZE – 1));
}
```

BRIDGING THE SEMANTIC GAP

Observation: when executing kernel code, the base of the stack points to the current executing thread.

struct thread_info

```
struct thread_info {
        unsigned long flags;
        int preempt_count;
        mm_segment_t addr_limit;
        struct task_struct *task; /* main task structure */
        ...
}
```

BRIDGING THE SEMANTIC GAP

Observation: when executing kernel code, the base of the stack points to the current executing thread.

struct task_struct

```
struct task_struct {
        volatile long state;
        void *stack;
        ...
        pid_t pid;
        pid_t tgid;
        ...
        char comm[TASK_COMM_LEN];
        ...
}
```

BINDER

The Binder protocol is the core of Android IPC/RPC

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

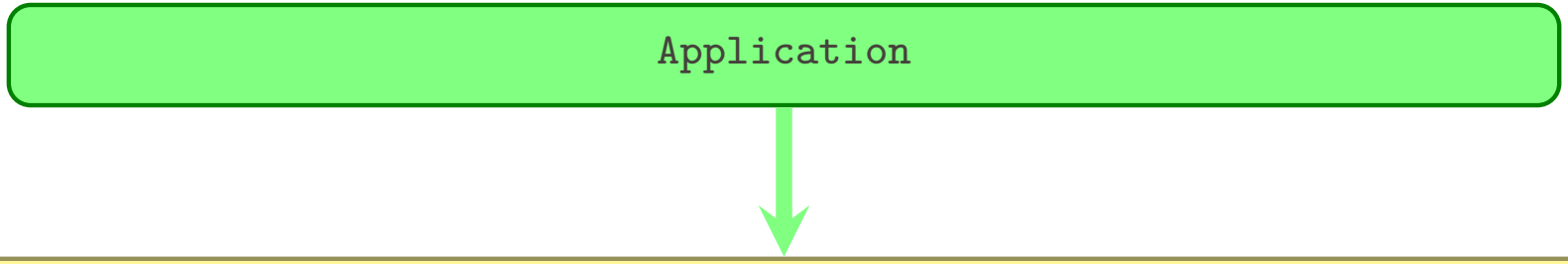
For example, applications cannot send SMSs on their own, but must invoke (RPC) the proper system service to do that.

BINDER

Application

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

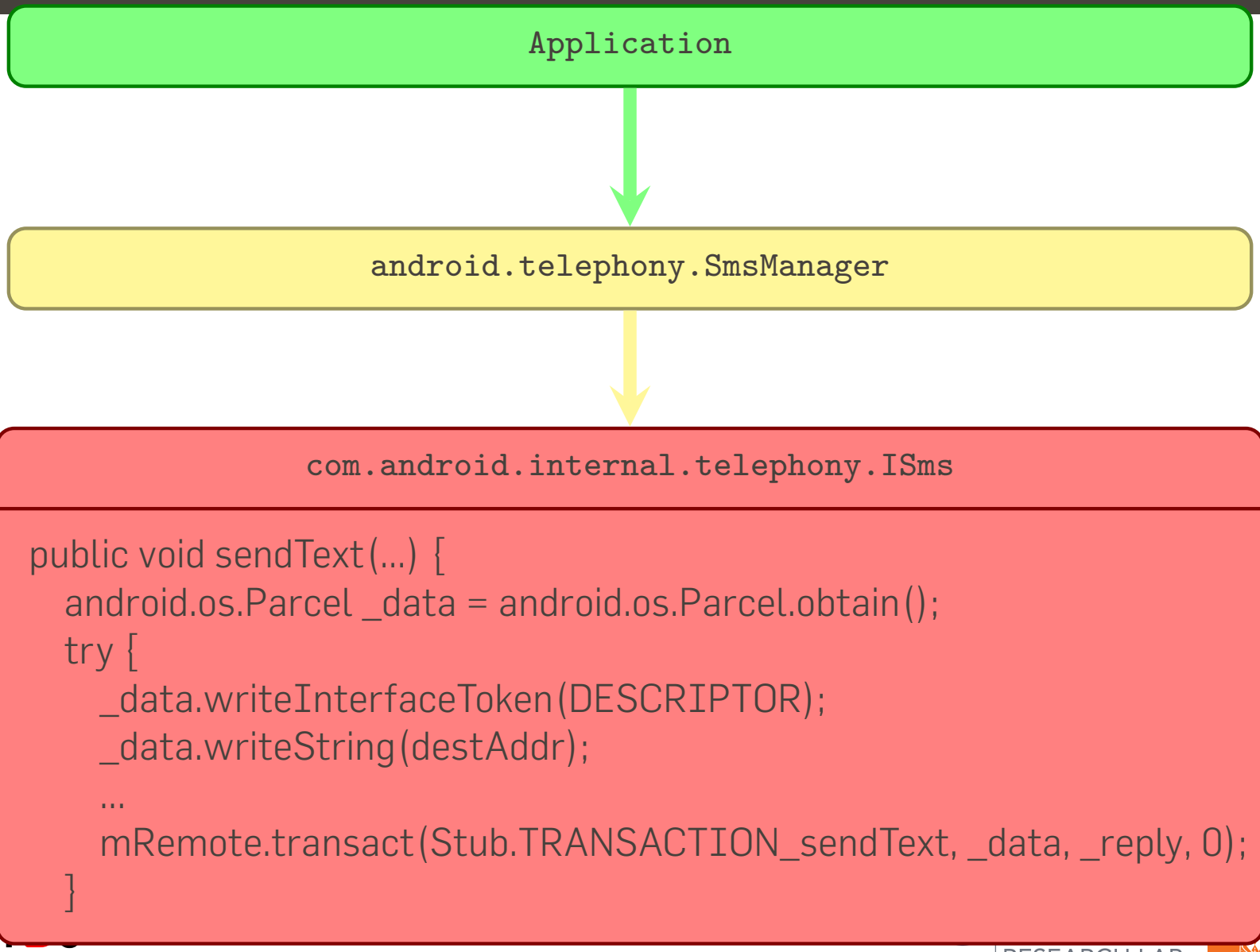
BINDER



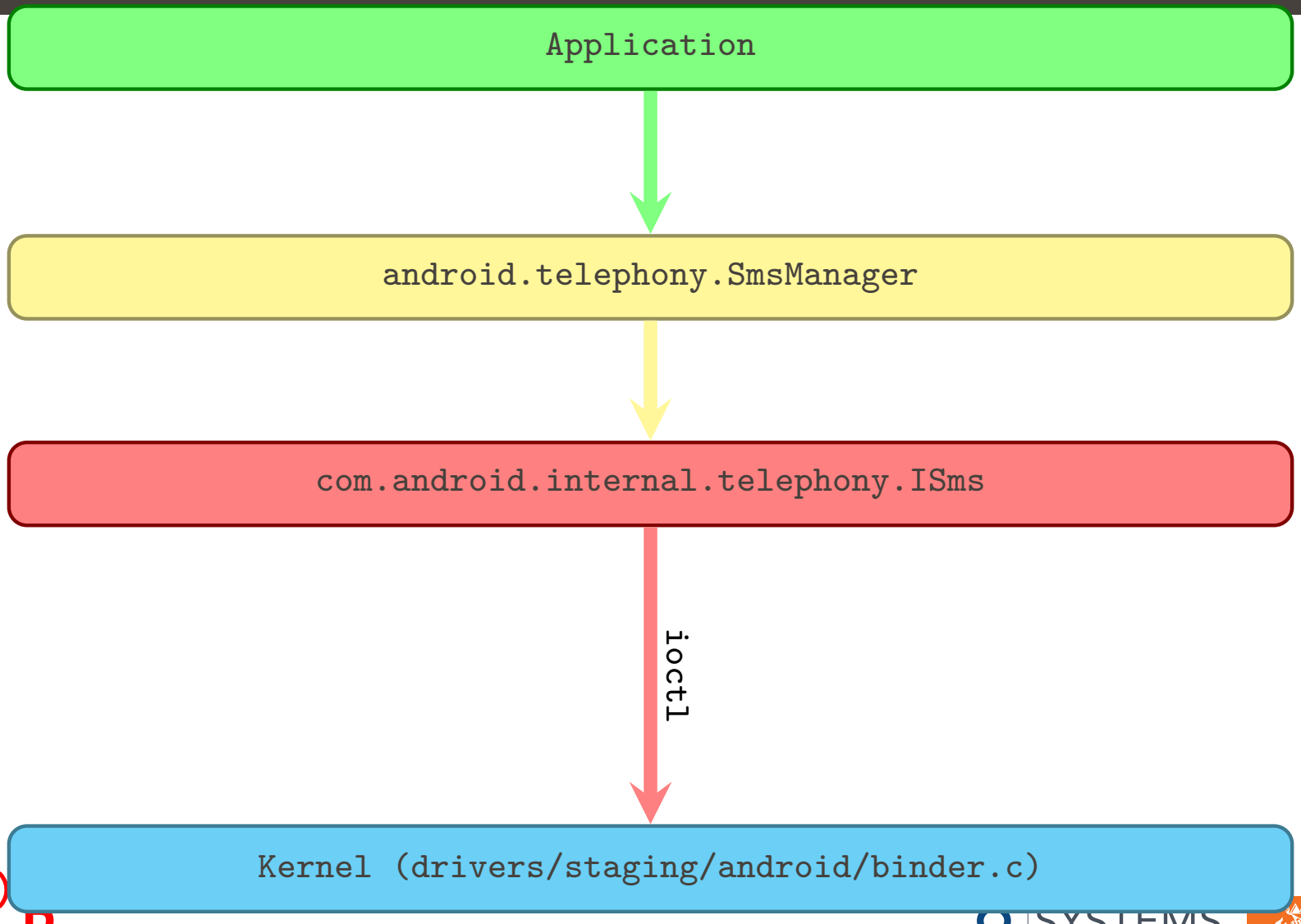
android.telephony.SmsManager

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

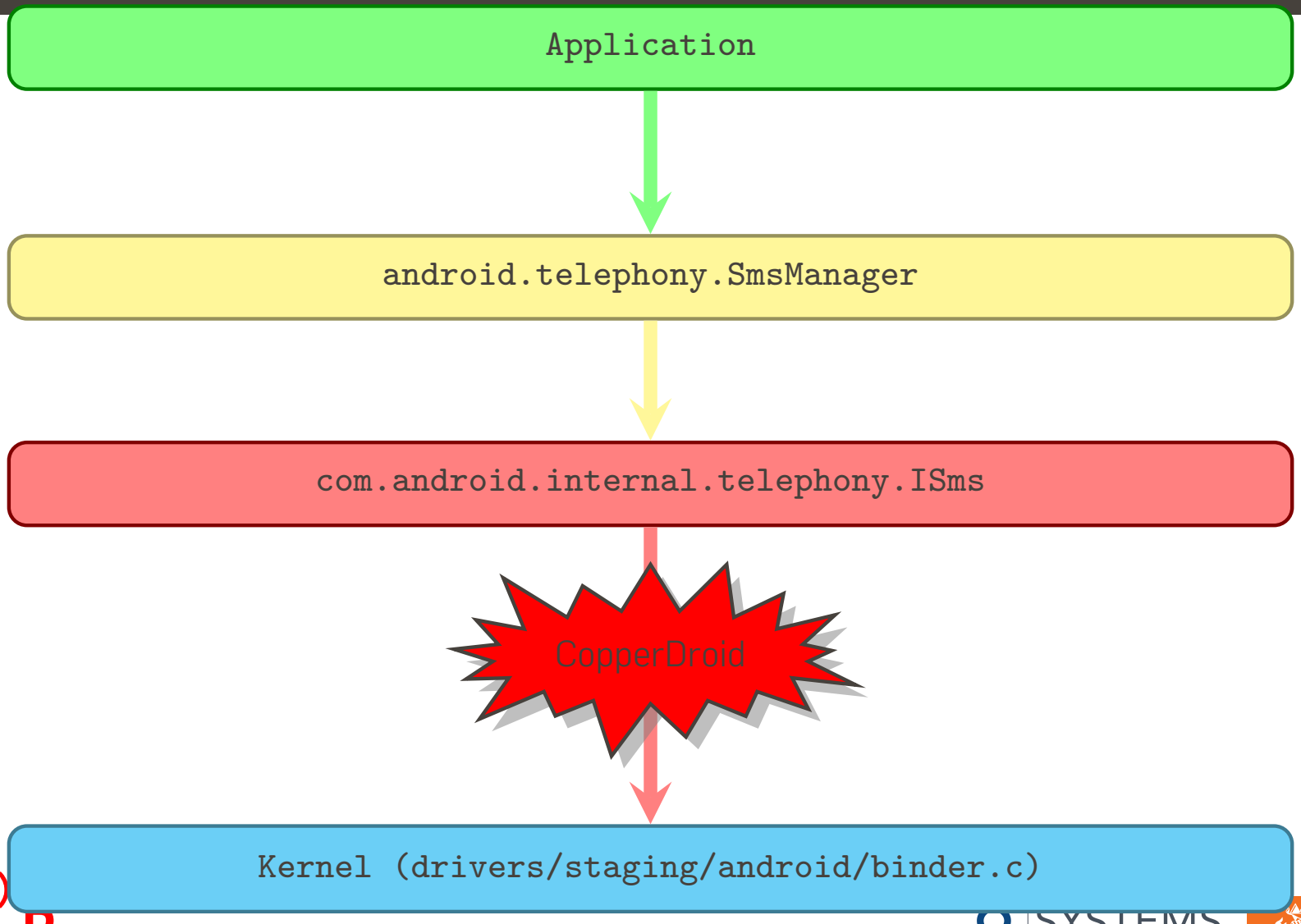
BINDER


Application
android.telephony.SmsManager
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);
}

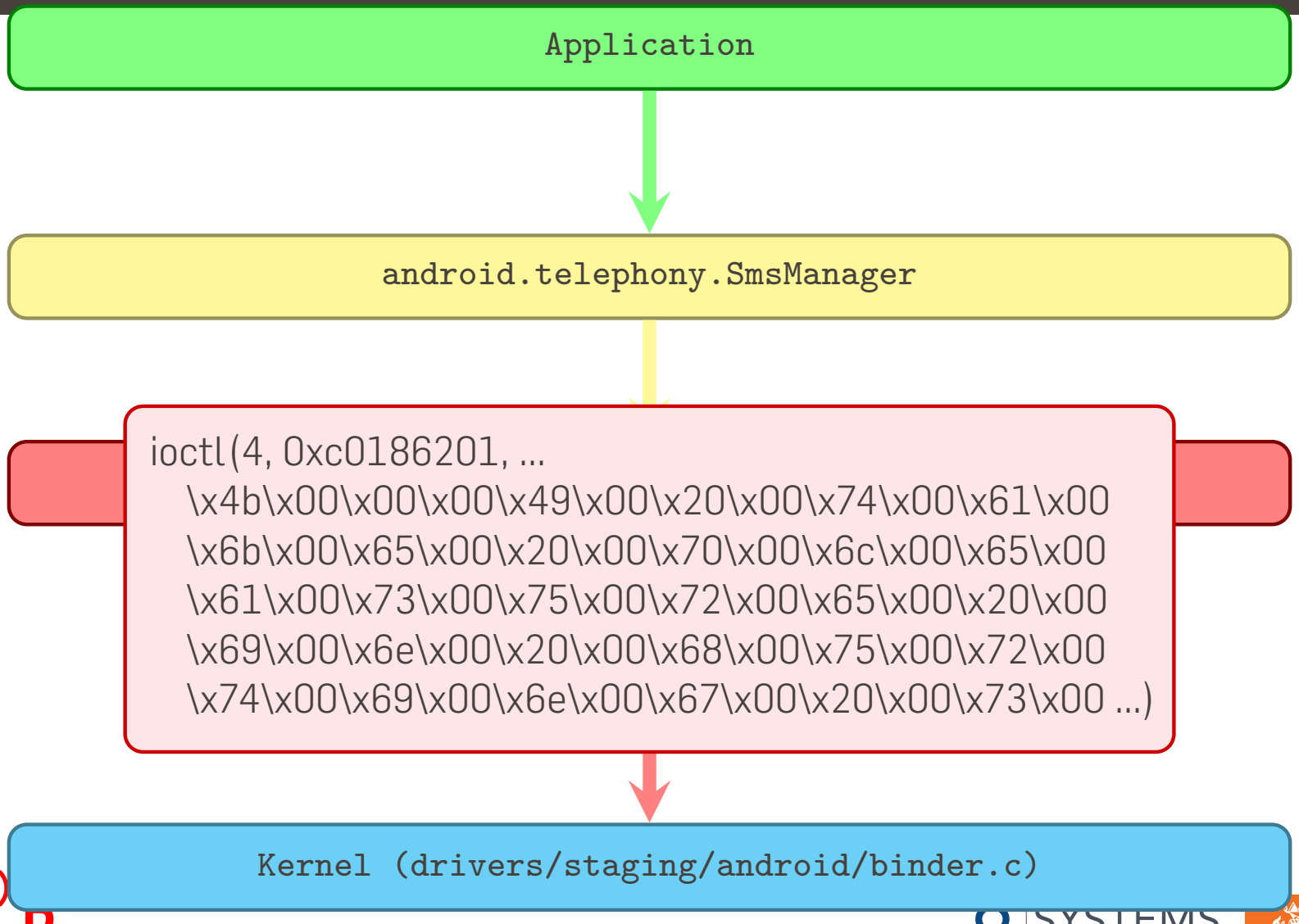
BINDER


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

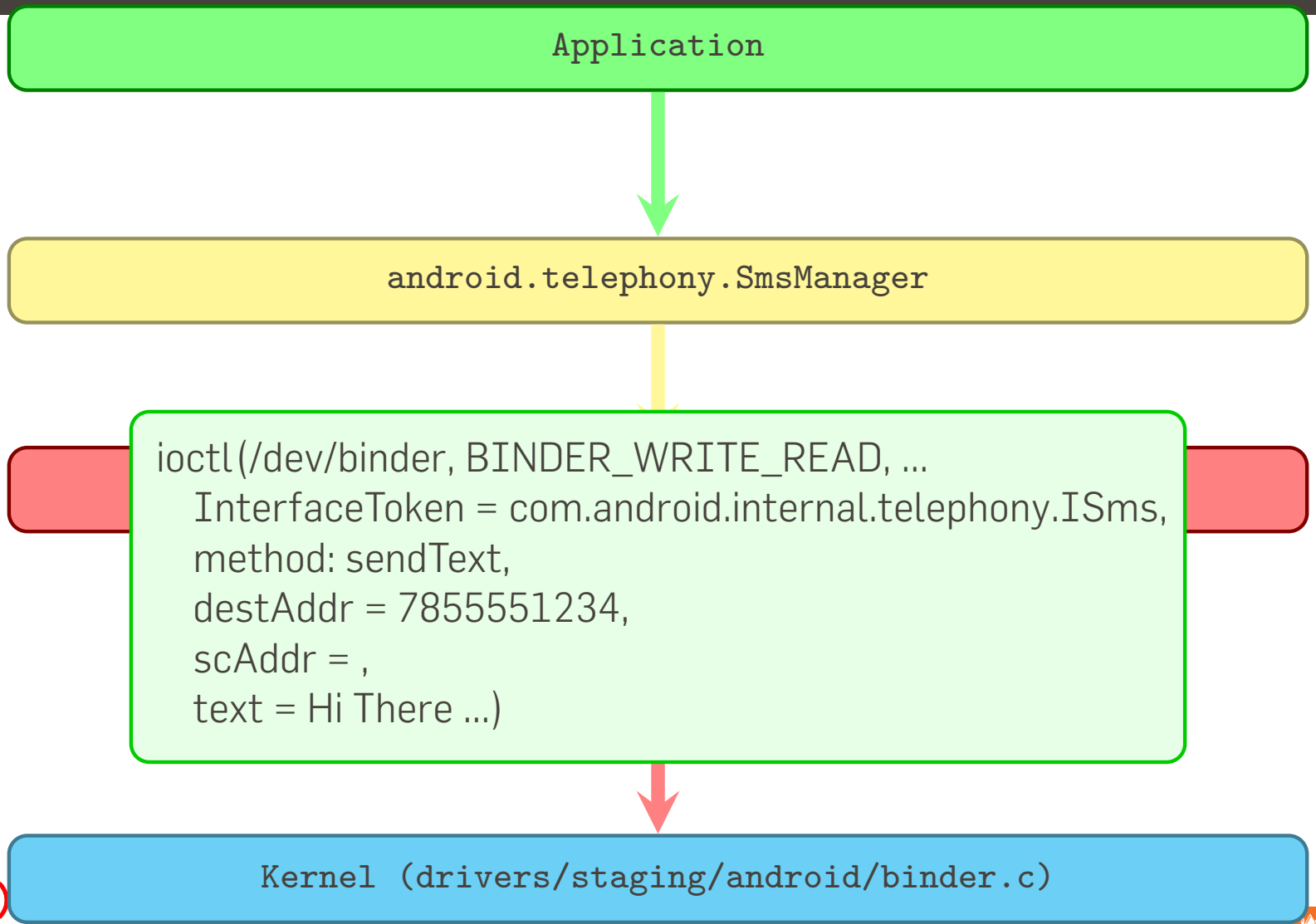
BINDER


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

BINDER

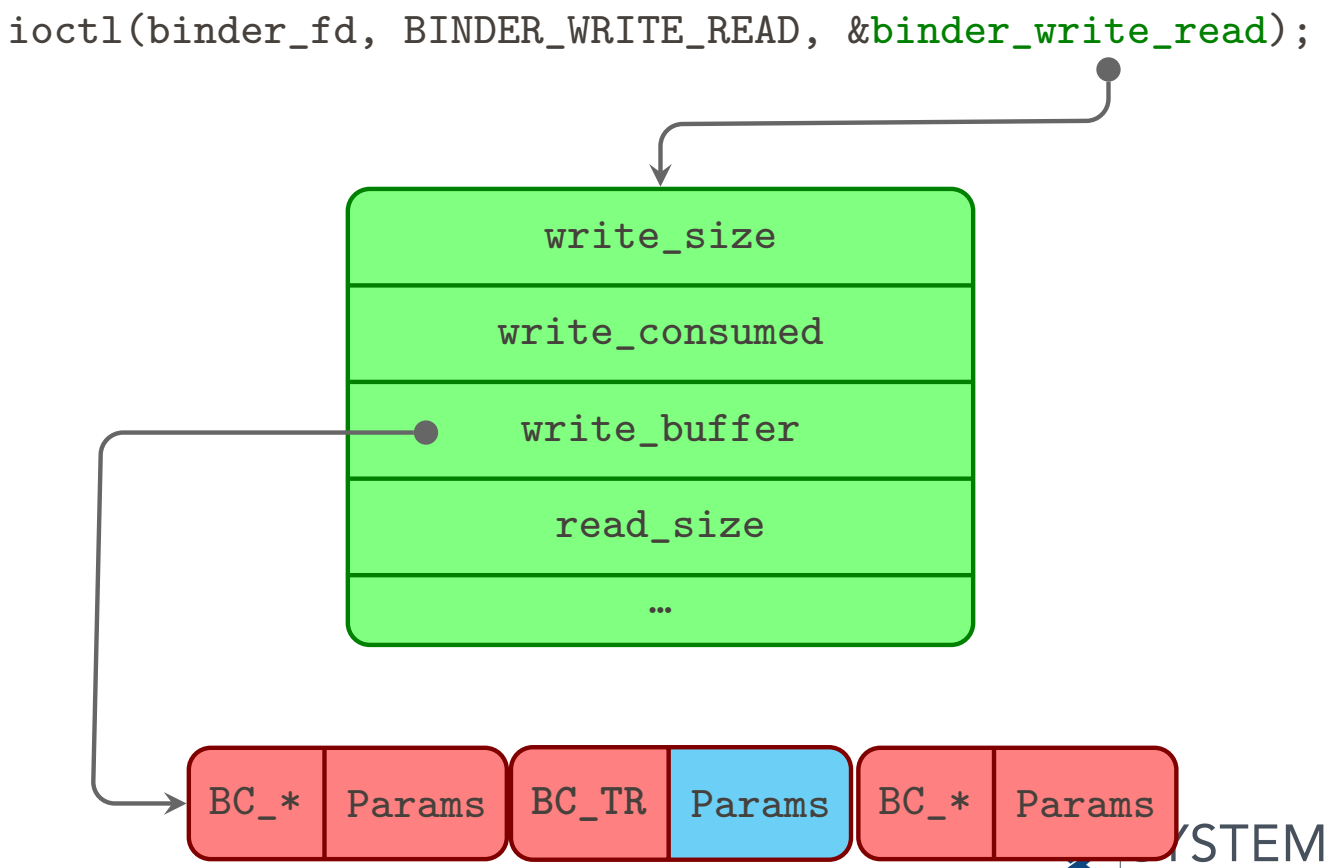

Application
android.telephony.SmsManager
ioctl(4, 0xc0186201, ...
\x4b\x00\x00\x00\x49\x00\x20\x00\x74\x00\x61\x00
\x6b\x00\x65\x00\x20\x00\x70\x00\x6c\x00\x65\x00
\x61\x00\x73\x00\x75\x00\x72\x00\x65\x00\x20\x00
\x69\x00\x6e\x00\x20\x00\x68\x00\x75\x00\x72\x00
\x74\x00\x69\x00\x6e\x00\x67\x00\x20\x00\x73\x00 ...)
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)

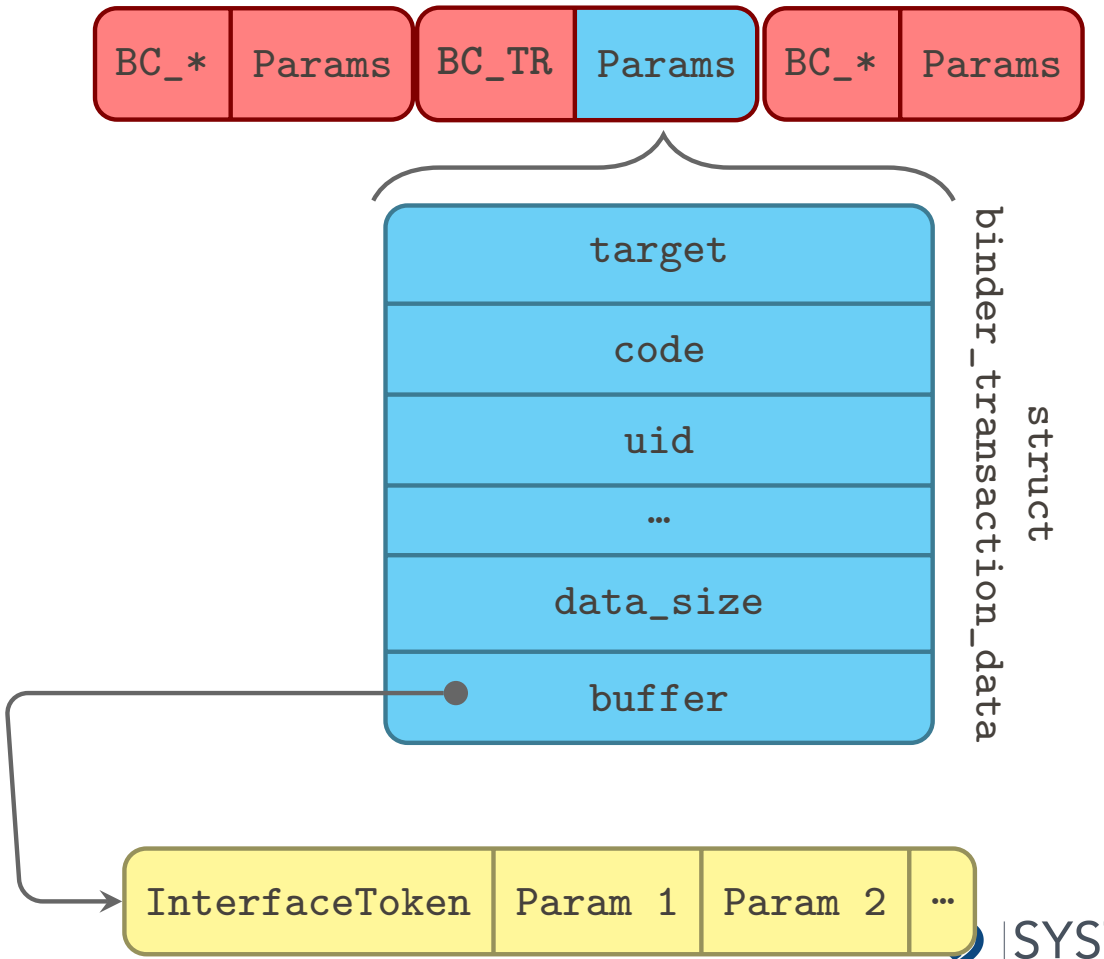
BINDER

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



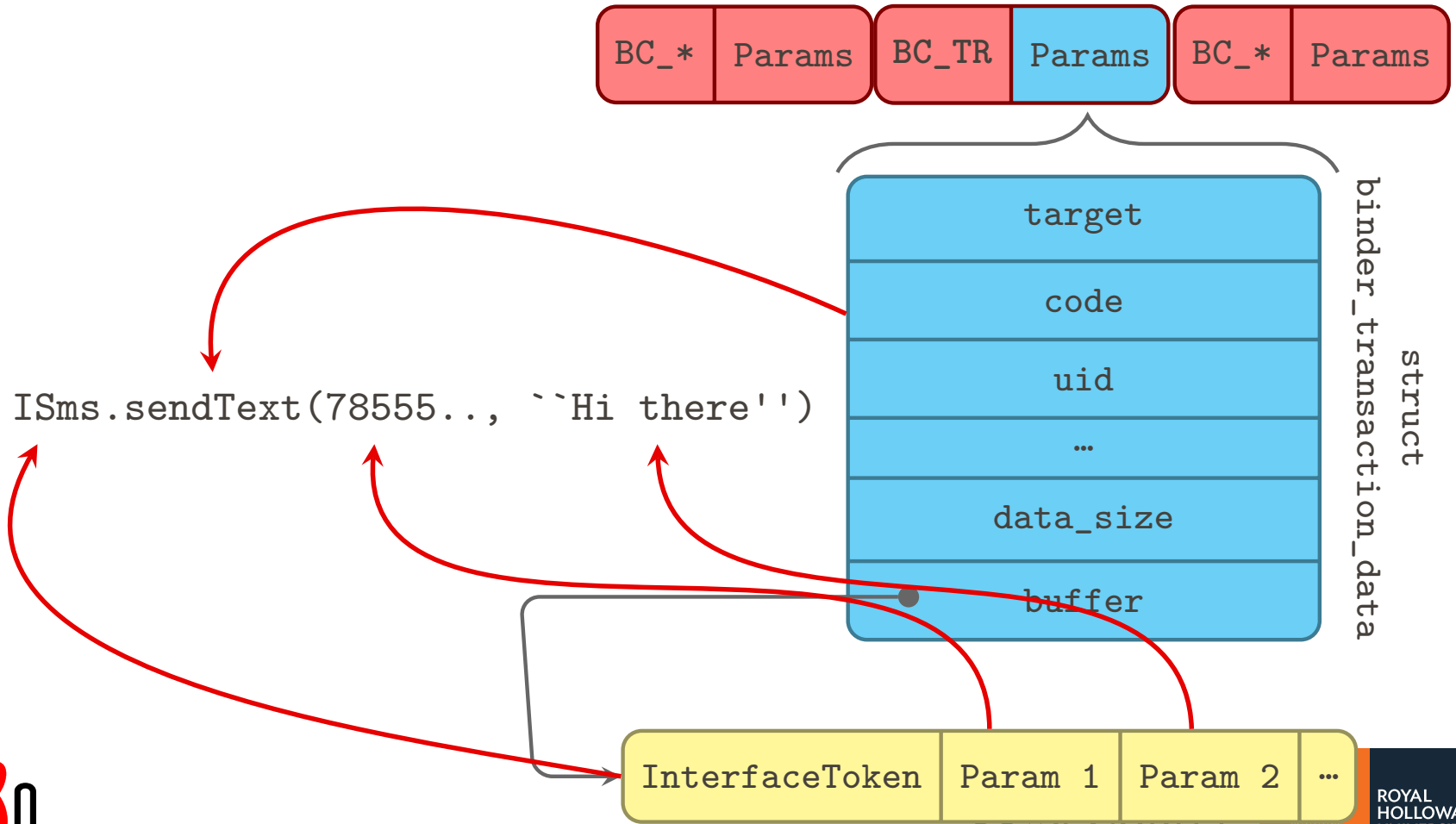
BINDER

CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs



BINDER

CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs

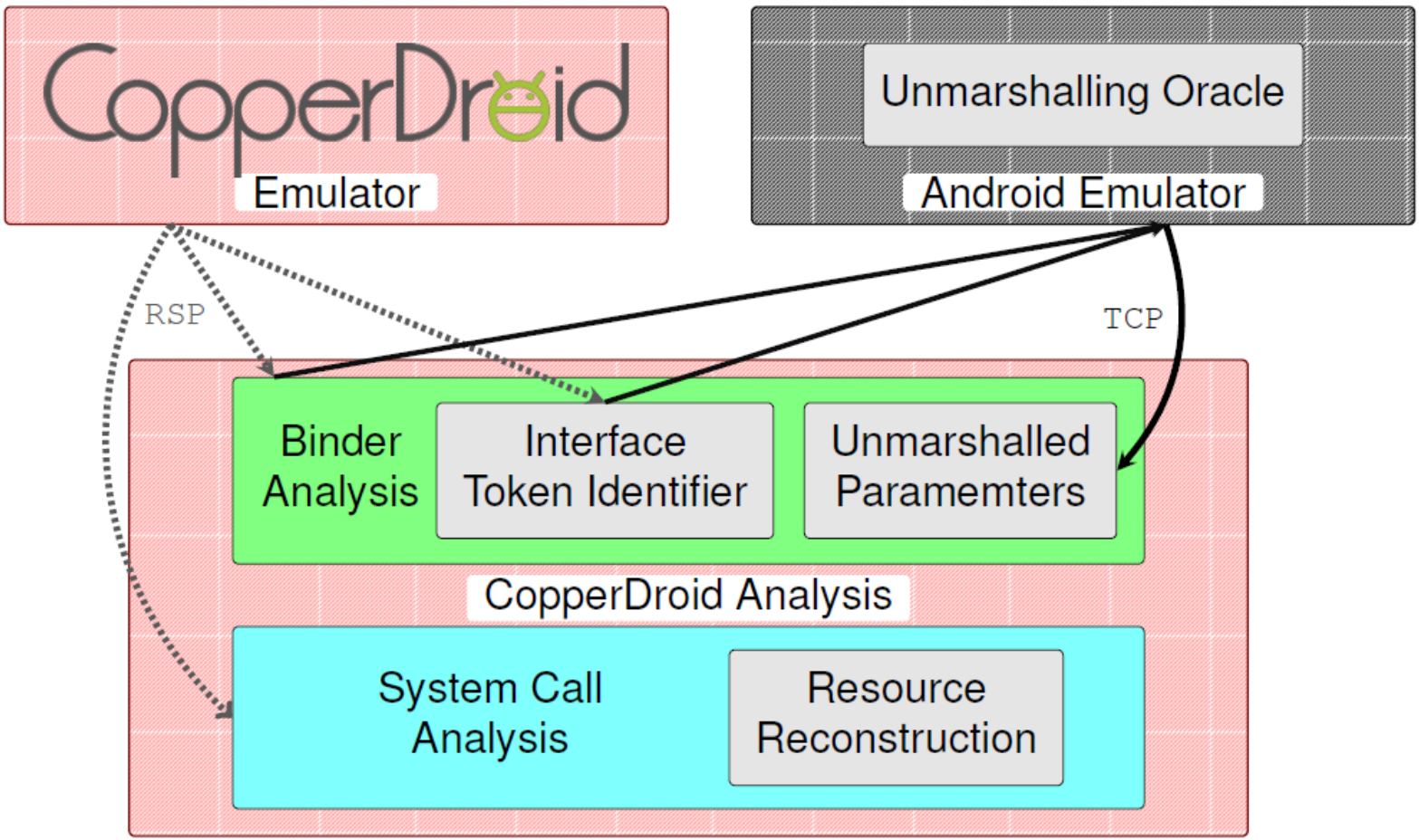


AUTOMATIC UNMARSHALLING OF ANDROID OBJECTS

AUTOMATIC UNMARSHALLING OF ANDROID OBJECTS

- Primitive types (and `Long`, `String`, `Integer`, and `Boolean`) are easy to unmarshall
  - → Limited number of manually-written procedures
- A manual-driven approach for complex Android objects is cumbersome
  - → 300+ Android objects (increasing from version to version)
  - → Manual-driven approach is error-prone and not scientifically exciting
- We ask to an unmarshalling Oracle!

UNMARSHALLING ORACLE FRAMEWORK



UNMARSHALLING DATA

- When sent from one process (application) to another, data is sent in serialized Parcels via IPC
- There are three methods for serializing data into Parcels:
  - → Primitives
    - → e.g., Integers, Strings, Boolean Arrays
  - → Parcelable Objects
  - → Binder Objects

UNMARSHALLING DATA

- When sent from one process (application) to another, data is sent in serialized Parcels via IPC
- There are three methods for serializing data into Parcels:
  - → Primitives
  - → Parcelable Objects
    - → Objects must implement the interface Parcelable to be written in Parcels (e.g., Intent)
    - → Make use of the AIDL
  - → Binder Objects

UNMARSHALLING DATA

- When sent from one process (application) to another, data is sent in serialized Parcels via IPC
- There are three methods for serializing data into Parcels:
  - → Primitives
  - → Parcelable Objects
  - → Binder Objects
    - → Interfaces (e.g., IAccountManagerResponse)
    - → References to the object (e.g., PendingIntent)

Value-based (Coarse-grained) Data Flow Analysis

VALUE-BASED (COARSE-GRAINED) DATA FLOW ANALYSIS

- Useful to abstract a stream of low-level events into high-level behaviors
- We build a data dependence graph (DPD)
  - → Nodes are system calls
  - → Edges represent value-based data dependency
- We then identify def-use chains to cluster related system calls together
  - → File system accesses (e.g., `open`, `read`, `write`, `dup[2]?`)
  - → Network communications (e.g., `socket`, `connect`, `send`, `recv`)
  - → Binder reference-based object passing (via `ashmem`)

SAMPLE COPPERDROID OUTPUT

[c5b02000 - 35 - 35 - zygote] fork( ) = 0x125
[c5b02000 - 35 - 35 - zygote] getpgid(0x41) = 0x23
[c5b02000 - 35 - 35 - zygote] setpgid(0x125, 0x23) = 0x0
[c1c18000 - 293 - 293 - zygote] getuid32( ) = 0x0
[c1c18000 - 293 - 293 - zygote] open(/acct/uid/0/tasks, ...) = 0x13
[c1c18000 - 293 - 293 - zygote] fstat64(0x13, 0xbef7f910) = 0x0
[c1c18000 - 293 - 293 - zygote] mprotect(0x40008000, 0x1000, 0x3) = 0x0
[c1c18000 - 293 - 293 - zygote] mprotect(0x40008000, 0x1000, 0x1) = 0x0
[c1c18000 - 293 - 293 - zygote] write(0x13 - /acct/uid/0/tasks, 0xa24c0 "'0'", 0x1) = 0x1
[c1c18000 - 293 - 293 - zygote] close(0x13) = 0x0
[c1c18000 - 293 - 293 - zygote] prctl(0x8, 0x1, 0x0, 0x0, 0x0) = 0x0
[c1c18000 - 293 - 293 - zygote] setgroups32(0x2, 0xbef7fa20) = 0x0
[c1c18000 - 293 - 293 - zygote] setgid32(0x2722) = 0x0
[c1c18000 - 293 - 293 - zygote] open(/acct/uid/10018/tasks, 0x2442, 0x1b6) = 0xfffffffe

SAMPLE COPPERDROID OUTPUT

```
[c5b02000 - 35 - 35 - zygote] fork( ) = 0x125
[c5b02000 - 35 - 35 - zygote] getpgid( 0x41 ) = 0x23
[c5b02000 - 35 - 35 - zygote] setpgid( 0x125, 0x23 ) = 0x0
[c1c18000 - 293 - 293 - zygote] getuid32( ) = 0x0
[c1c18000 - 293 - 293 - zygote] open (/acct/uid/0/tasks, 0x20242, 0x1b6) = 0x
[c1c18000 - 293 - 293 - zygote] fstat64 ( 0x13 , 0xbef7f910 ) = 0x0
[c1c18000 - 293 - 293 - zygote] mprotect( 0x40008000, 0x1000, 0x3 )
= 0x0
[c1c18000 - 293 - 293 - zygote] mprotect( 0x40008000, 0x1000, 0x1 )
= 0x0
[c1c18000 - 293 - 293 - zygote]  write ( 0x13 - /acct/uid/0/tasks,
0xa24c0 "'0'", 0x1 ) = 0x1
[c1c18000 - 293 - 293 - zygote]  close ( 0x13 ) = 0x0
[c1c18000 - 293 - 293 - zygote] prctl( 0x8, 0x1, 0x0, 0x0, 0x0 ) = 0x0
[c1c18000 - 293 - 293 - zygote] setgroups32( 0x2, 0xbef7fa20 ) = 0x0
[c1c18000 - 293 - 293 - zygote] setgid32( 0x2722 ) = 0x0
[c1c18000 - 293 - 293 - zygote] open( /acct/uid/10018/tasks, 0x20242,
0x1b6 ) = 0xfffffffe
```

SAMPLE COPPERDROID OUTPUT

```
[c5b02000 - 35 - 35 - zygote] fork( ) = 0x125
[c5b02000 - 35 - 35 - zygote] getpgid( 0x41 ) = 0x23
[c5b02000 - 35 - 35 - zygote] setpgid( 0x125, 0x23 ) = 0x0
[c1c18000 - 293 - 293 - zygote] getuid32( ) = 0x0
[c1c18000 - 293 - 293 - zygote] open (/acct/uid/0/tasks, 0x20242, 0x1b6) = 0x
[c1c18000 - 293 - 293 - zygote] fstat64 ( 0x13 , 0xbef7f910 ) = 0x0
[c1c18000 - 293 - 293 - zygote] mprotect( 0x40008000, 0x1000, 0x3 )
= 0x0
[c1c18000 - 293 - 293 - zygote] mprotect( 0x40008000, 0x1000, 0x1 )
= 0x0
[c1c18000 - 293 - 293 - zygote] write( 0x13 , /acct/uid/0/tasks,
0xa24c0 "'0'", 0x1 ) = 0x1
[c1c18000 - 293 - 293 - zygote]  close ( 0x13 ) = 0x0
[c1c18000 - 293 - 293 - zygote] prctl( 0x8, 0x1, 0x0, 0x0, 0x0 ) = 0x0
[c1c18000 - 293 - 293 - zygote] setgroups32( 0x2, 0xbef7fa20 ) = 0x0
[c1c18000 - 293 - 293 - zygote] setgid32( 0x2722 ) = 0x0
[c1c18000 - 293 - 293 - zygote] open( /acct/uid/10018/tasks, 0x20242,
0x1b6 ) = 0xfffffffe
```

Group as one action:
File Access

SAMPLE COPPERDROID OUTPUT

```
[c5b02000 - 35 - 35 - zygote] fork( ) = 0x125
[c5b02000 - 35 - 35 - zygote] getpgid( 0x41 ) = 0x23
[c5b02000 - 35 - 35 - zygote] setpgid( 0x125, 0x23 ) = 0x0
[c1c18000 - 293 - 293 - zygote] getuid32( ) = 0x0
[c1c18000 - 293 - 293 - zygote] open (/acct/uid/0/tasks, 0x20242, 0x1b6) = 0x
[c1c18000 - 293 - 293 - zygote] fstat64 ( 0x13 , 0xbef7f910 ) = 0x0
[c1c18000 - 293 - 293 - zygote] mprotect( 0x40008000, 0x1000, 0x3 )
= 0x0
[c1c18000 - 293 - 293 - zygote] mprotect( 0x40008000, 0x1000, 0x1 )
= 0x0
[c1c18000 - 293 - 293 - zygote] write ( 0x13 , /acct/uid/0/tasks,
0xa24c0 "'0'", 0x1 ) = 0x1
[c1c18000 - 293 - 293 - zygote]  close ( 0x13 ) = 0x0
[c1c18000 - 293 - 293 - zygote] prctl( 0x8, 0x1, 0x0, 0x0, 0x0 ) = 0x0
[c1c18000 - 293 - 293 - zygote] setgroups32( 0x2, 0xbef7fa20 ) = 0x0
[c1c18000 - 293 - 293 - zygote] setgid32( 0x2722 ) = 0x0
[c1c18000 - 293 - 293 - zygote] open( /acct/uid/10018/tasks, 0x20242,
0x1b6 ) = 0xfffffffe
```

Recreates file "tasks"
with path /acct/uid/0/tasks and "0" written to it

APPS STIMULATION

(Android) malware needs to be properly stimulated to trigger additional 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)

EVALUATION

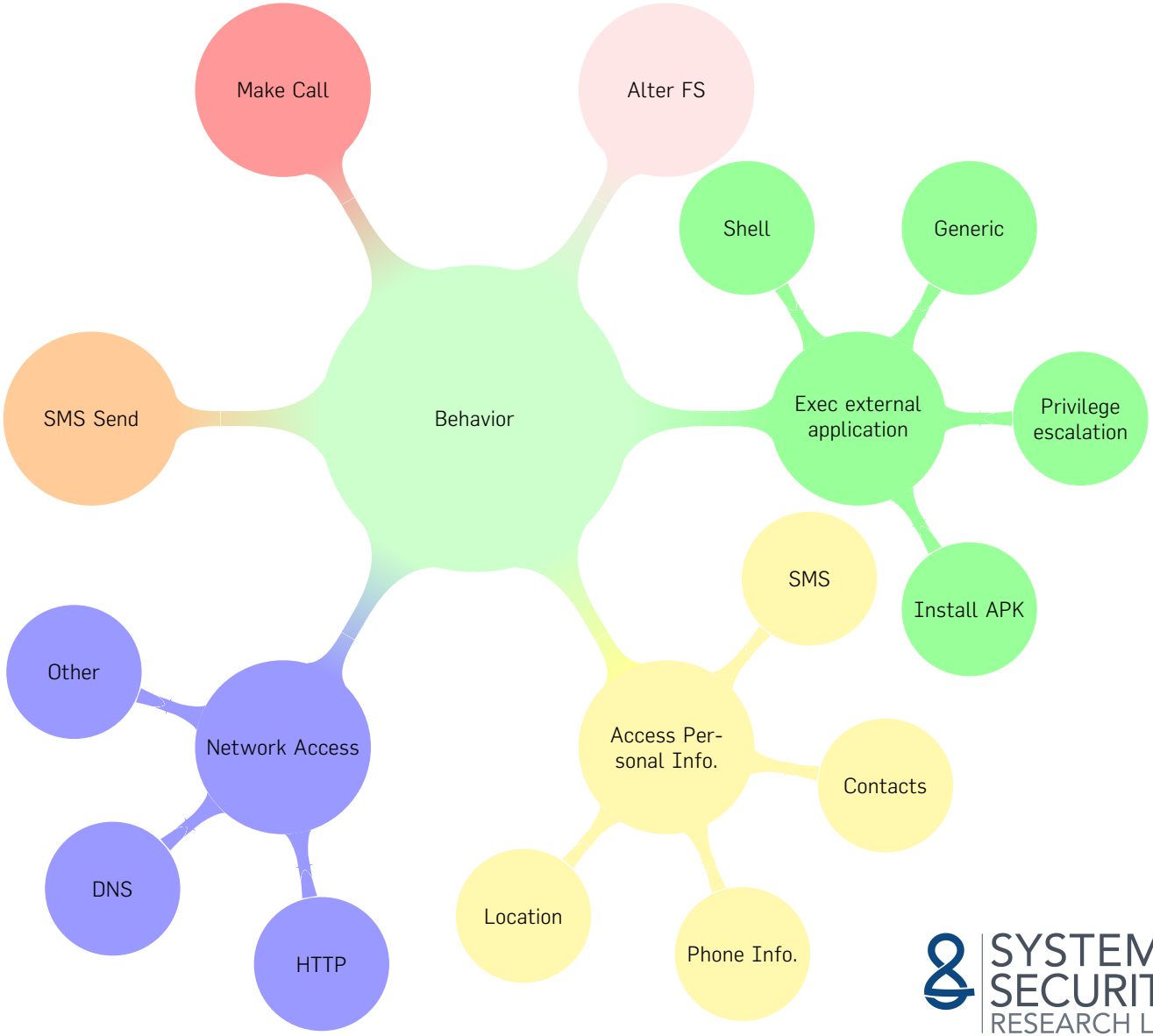
1,200 malware from the Android Malware Genome Project, 395 from the Contagio repository, and 1,300+ from McAfee

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	(∞)
8	Zsone	5.0	12/12	16.67	3.8	(23%)

OBSERVED BEHAVIORS

BEHAVIORAL MINDMAP



Behavior Class	No Stimulation	Stimulation
FS Access	889/1365 (65.13%)	912/1365 (66.81%)
Access Personal Info.	558/1365 (40.88%)	903/1365 (66.15%)
Network Access	457/1365 (33.48%)	461/1365 (33.77%)
Exec. External Appf.	171/1365 (12.52%)	171/1365 (12.52%)
Send SMS	38/1365 (2.78%)	42/1365 (3.08%)
Make/Alter Call	1/1365 (0.07%)	55/1365 (4.03%)

Table: Overall behavior breakdown of McAfee samples.

Behavior Class	Subclass	No Stim	Stim
Network Access	Generic	483	489
	HTTP	309	318
	DNS	416	416
FS Access	Write	889	912
Access Personal Info.	SMS	32	266
	Phone	510	559
	Accounts	51	672
	Location	143	147
Exec. External App.	Generic	132	132
	Priv. Esc.	103	103
	Shell	73	73
	Inst. APK	8	8
Send SMS	---	38	42
Make/Alter Call	---	1	55

Table: Detailed behavior breakdown of McAfee samples.

CLUSTHEDROID

WHAT IS CLUSTERING?

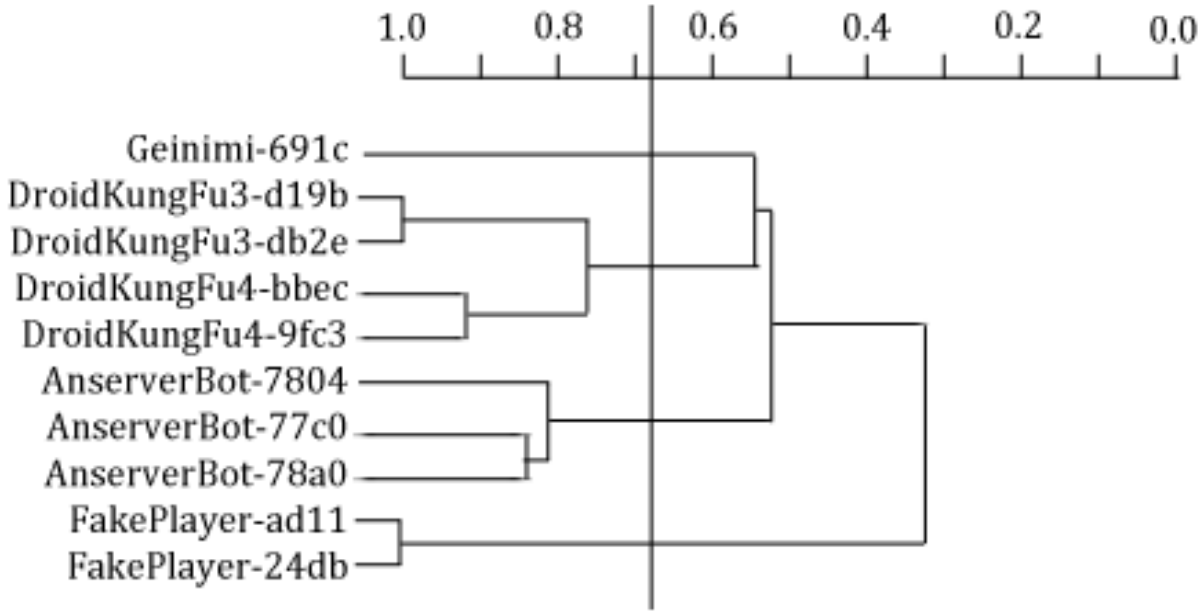
1. Finding natural groupings among objects.
2. Organize data into clusters such that:
   - → high intra-cluster similarity
   - → low inter-cluster similarity

SIMILARITY

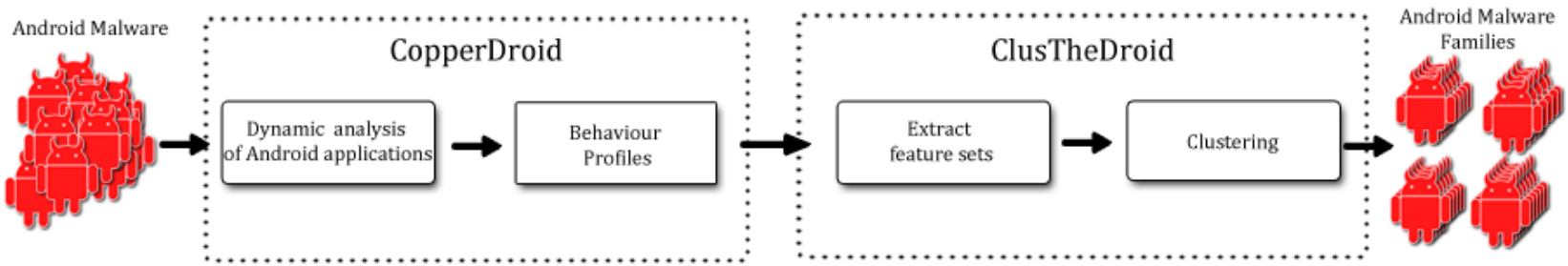
- Similarity is subjective!
- We represent elements through features and similarity between elements are thus a notion of how similar their feature sets are.
- Defined by distance (or similarity) function $d(x, y)$ which must satisfy:
  1. $d(x, y) \geq 0$
  2. $d(x, y) = 0 \iff x = y$
  3. $d(x, y) = d(y, x)$
  4. $d(x, z) \leq d(x, y) + d(y, z)$

HIERARCHICAL CLUSTERING

Hierarchical clustering algorithms are either divisive (top-down) or agglomerative (bottom-up). Divisive algorithms initially have all objects in the same cluster and proceed by splitting clusters, where agglomerative algorithms start with all objects in singleton clusters and proceed by merging clusters.



CLUSTERING ANDROID MALWARE



FEATURES

Disclaimer

Extremely naïve approach

FEATURES

Disclaimer

Extremely naïve approach

For each behavior, we compute:

- the frequency of a specific behavioral class
  → e.g., Access Personal Info, Network Access
- the frequency of a specific behavioral sub-class
  → e.g., Account, Contacts, Call
- the frequency of a specific OS- and Android-specific observed behavior

SIMILARITY AND CLUSTER-TO-VECTOR COMPARISON

Disclaimer

Somewhat simplified!

SIMILARITY AND CLUSTER-TO-VECTOR COMPARISON

Disclaimer

Somewhat simplified!

- Jaccard Similarity: $J(A, B) = \frac{A \cap B}{A \cup B}$
- Single-linkage (minimal distance between elements)

SIMILARITY AND CLUSTER-TO-VECTOR COMPARISON

Disclaimer

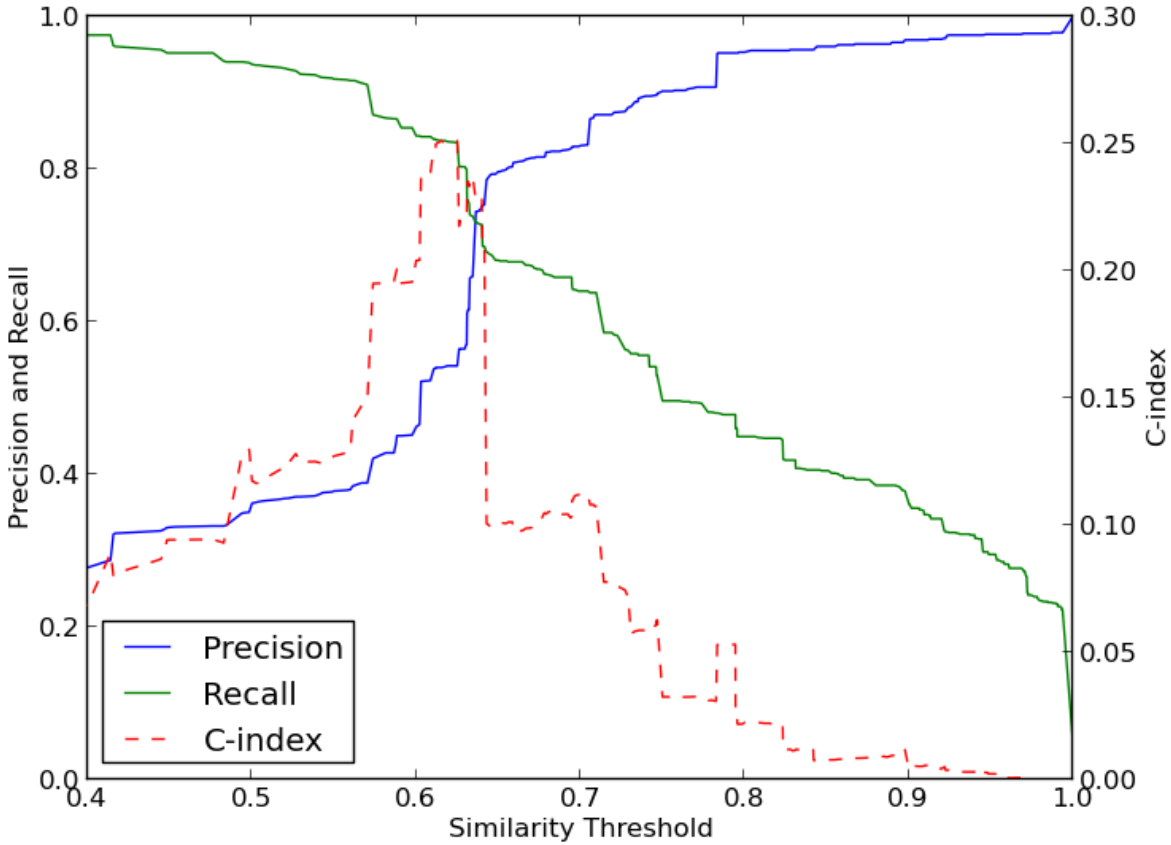
Somewhat simplified!

- Jaccard Similarity: $J(A, B) = \frac{A \cap B}{A \cup B}$
- Single-linkage (minimal distance between elements)

Dataset

Android Malware Genome Project (1260 malware in 49 families)

EXPERIMENT


Precision and Recall
C-index
Similarity Threshold
Precision
Recall
C-index
1.0
0.8
0.6
0.4
0.2
0.0
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.4
0.5
0.6
0.7
0.8
0.9
1.0

COMPARATIVE EVALUATION

We evaluate our results to those of Bayer et al.

Approach	Malware	Sim. Threshold	Precision	Recall
Bayer	Windows PE	0.7	0.98	0.93
Our 1	.apk	0.63	0.74	0.73
Our 2a	.apk	0.64	0.84	0.73
Our 2b	.apk	0.58	0.77	0.91

SO WHAT DID WE GET?

- A rigorous attempt to cluster Android malware families.
- A solution that performance-wise satisfies the requirements of today's demand and has room for future growth of Android malware.
- A successfull first attempt to use the C-index as a means of determining near-optimal clusters regarding malware families.

CONCLUSIONS

CONCLUSIONS

CopperDroid Goal

Automatically reconstructs the behaviors of Android malware

- Unified system call-centric analysis that avoid 2-level VMIs
  - → All the behaviors are eventually achieved via system interactions
  - → Suitable for in-device (data) collection!
- Automatic unmarshalling of Android objects
  - → Online/offline Oracle analysis
- Dynamically stimulates Apps to disclose additional behaviors
- Evaluation on 2,900+ Android malware
  (28% additional behaviors on 60% of Genome samples)
  (22% additional behaviors on 73% of Contagio samples)
  (28% additional behaviors on 61% of McAfee samples)

CONCLUSIONS

CopperDroid Goal

1. Team work! Thanks to Kim, Salah, Aristide, and Alessandro
2. Available at http://copperdroid.isg.rhul.ac.uk
3. 4-year EPSRC-funded project within the Systems Security Research Lab (S^2Lab)[a] at Royal Holloway University of London
4. Ongoing research directions:
   - 4.1 Automatic policy enforcement
   - 4.2 Information leakage detection (no taint-tracking!)
   - 4.3 Benign / Malicious Android malware detection
   - 4.4 Automatic clustering and classification
   - 4.5 UI-driven/aided symbolic execution
   - 4.6 Hardware-supported virtualization for in-device analyses

[a] http://s2lab.isg.rhul.ac.uk


Visit S²Lab at ~~Hogwarts~~ Royal Holloway University of London

THANK YOU

lorenzo.cavallaro@rhul.ac.uk
@lcavallaro

BACKUP

ORACLE > UNMARSHALLING PRIMITIVES

Type[0] = ``int"
at offset 0: ReadInt()
increment offset 4 bytes

OUTPUT[0] =[``int{12345}"]

INPUT: Types ["int", "string",
"FloatArray", "SparseBooleanArray", ...]

INPUT: Data [\x39\x30\x00\x00\x0C
\x00\x00\x00\x48\x00\x65\x00\x6C
\x00\x6C\x00\x6F\x00\x20\x00\x57
\x00\x6F\x00\x72\x00\x6C\x00\x64
\x00\x21\x00\x00\x00\x00\x00\x04
\x00\x00\x00\x00\x00\x00\x42\x00
\x00\x00\x42\x00\x00\x00\x42\x00
\x00\x00\x42\x02\x00\x00\x00\x01
\x00\x00\x00\x01\x00\x00\x00\x02
\x00\x00\x00\x00\x00\x00\x00 ...]

ORACLE > UNMARSHALLING PRIMITIVES

Type[1] = ``string"
at offset 4: ReadString()
increment offset by 32

OUTPUT[1] = ``string{Hello World!}"

NOTE: As the first four bytes of strings, arrays, and lists show the number of items written, we can correctly increment the buffer offset

INPUT: Types ["int", "string",
"FloatArray", "SparseBooleanArray", ...]

INPUT: Data [\x39\x30\x00\x00\x0C
\x00\x00\x00\x48\x00\x65\x00\x6C
\x00\x6C\x00\x6F\x00\x20\x00\x57
\x00\x6F\x00\x72\x00\x6C\x00\x64
\x00\x21\x00\x00\x00\x00\x00\x04
\x00\x00\x00\x00\x00\x00\x42\x00
\x00\x00\x42\x00\x00\x00\x42\x00
\x00\x00\x42\x02\x00\x00\x00\x01
\x00\x00\x00\x01\x00\x00\x00\x02
\x00\x00\x00\x00\x00\x00\x00 ...]

ORACLE > UNMARSHALLING PRIMITIVES

Type[2] = ``FloatArray"
at offset 36: ReadFloatArray()
increment offset by 20

OUTPUT[2] =
``FloatArray{[32.0, 32.0, 32.0, 32.0]}"

INPUT: Types ["int", "string",
"FloatArray", "SparseBooleanArray", ...]

INPUT: Data [\x39\x30\x00\x00\x0C
\x00\x00\x00\x48\x00\x65\x00\x6C
\x00\x6C\x00\x6F\x00\x20\x00\x57
\x00\x6F\x00\x72\x00\x6C\x00\x64
\x00\x21\x00\x00\x00\x00\x00\x04
\x00\x00\x00\x00\x00\x00\x42\x00
\x00\x00\x42\x00\x00\x00\x42\x00
\x00\x00\x42\x02\x00\x00\x00\x01
\x00\x00\x00\x01\x00\x00\x00\x02
\x00\x00\x00\x00\x00\x00\x00 ...]

ORACLE > UNMARSHALLING PRIMITIVES

Type[3] = ``SparseBooleanArray"
at offset 56: SparseBooleanArray()
increment offset by 20

OUTPUT[3] =
``SparseBooleanArray{[(1, true),
(2, true)]]"

INPUT: Types ["int", "string",
"FloatArray", "SparseBooleanArray",...]

INPUT: Data [\x39\x30\x00\x00\x0C
\x00\x00\x00\x48\x00\x65\x00\x6C
\x00\x6C\x00\x6F\x00\x20\x00\x57
\x00\x6F\x00\x72\x00\x6C\x00\x64
\x00\x21\x00\x00\x00\x00\x00\x04
\x00\x00\x00\x00\x00\x00\x42\x00
\x00\x00\x42\x00\x00\x00\x42\x00
\x00\x00\x42\x02\x00\x00\x00\x01
\x00\x00\x00\x01\x00\x00\x00\x02
\x00\x00\x00\x00\x00\x00\x00 ...]

ORACLE > UNMARSHALLING PRIMITIVES

- Unmarshalling Android (complex) objects is more challenging
  1. Create class from type
  2. From the class, generate parcelable instance
  3. Locate the (static) CREATOR field
  4. Use the CREATOR to read from parcelable
- Extra challenge for others (e.g., PendingIntent)
  - → Objects are shared and references passed along (via ashmem)
  - → Collect objects from caller's address space (need value-based data flow analysis; see next)