ROYAL HOLLOWAY UNIVERSITY OF LONDON

COPPERDROID

Automatic Reconstruction of Android Malware Behaviors

Network and Distributed System Security Symposium (NDSS)
San Diego, USA, Februrary 9, 2015

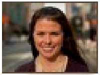
Kimberly Tam*, Salahuddin J. Khan*
Aristide Fattori∓, Lorenzo Cavallaro*

*Systems Security Research Lab and Information Security Group
Royal Holloway University of London

∓Dipartimento di Informatica
Universita` degli Studi di Milano

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.*

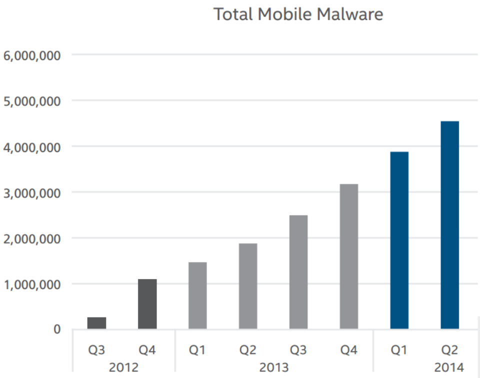


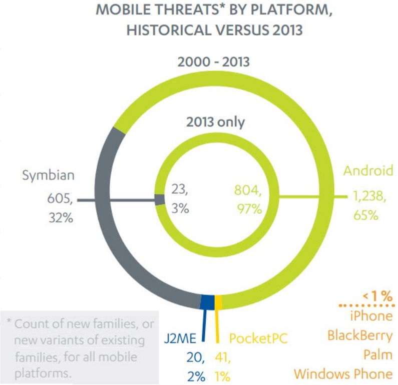
By Rachel King for Between the Lines | September 3, 2013 -- 17:36 GMT (18:36 BST)

Follow @rachelking

THE RISE IN ANDROID MALWARE

Over 1.75 billion Mobile users world wide in 2014 [eMarketer]





Source: McAfee Labs.

Problem: Analyses dependent on Android version

One way to analyze high-level behaviors is to modify runtime

- Unstable and prone to error
- Runtime internals may change
- Runtime itself may change (e.g., Dalvik VM, ART)

Can we do better?

- No modification to Android internals
- Can still analyze high-level behaviors

Key Insight

All interesting behaviors achieved through system calls

- Low-level, OS semantics (e.g., network access)
- High-level, Android semantics (e.g., phone call)

Goal

- Automatically reconstruct behaviors from system calls
- With no changes to the Android OS image

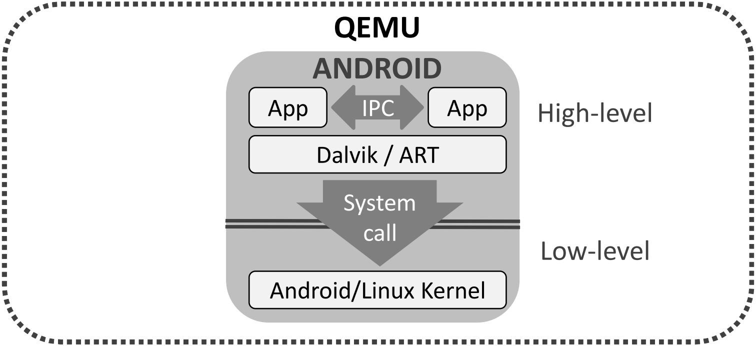
COPPERDROID

Traditional Roots

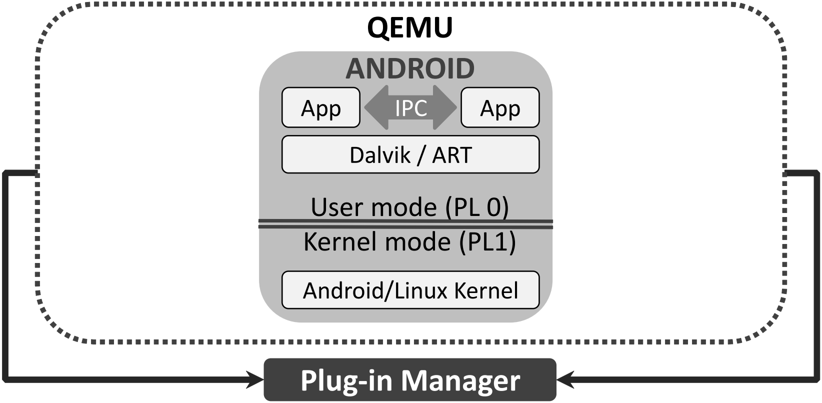
A well-established technique to characterize process behaviors

Can it be applied to Android?

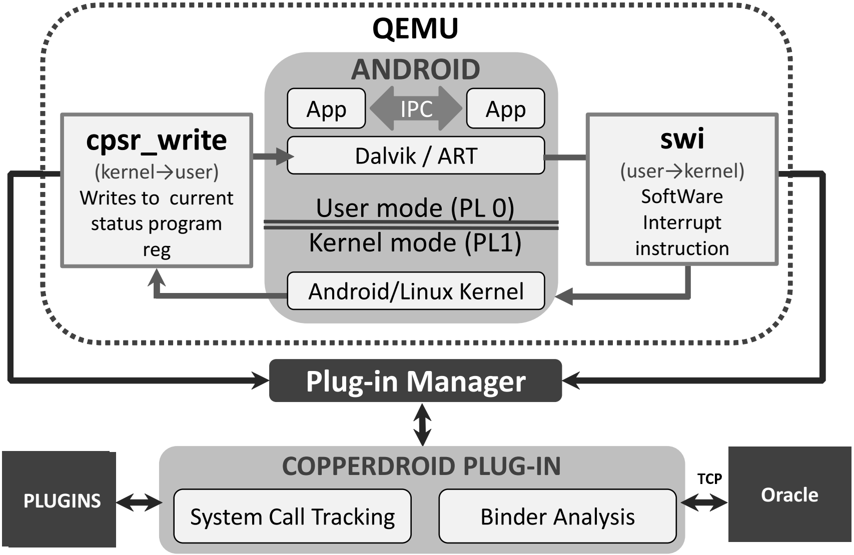
- Android architecture is different to traditional devices
- Are all behaviors achieved through system calls?
  - → Android-specific behaviors (e.g., Dalvik) (e.g., SMS, phone calls)
  - → OS interactions (e.g., creating a file, network communication)



- Android emulator built on QEMU
- Android applications are isolated
- Apps communicate via IPC or system calls



- Small modification to QEMU to allow CopperDroid plugin
- No modification to Android image
- Increases portability and reduces runtime overhead.


QEMU
ANDROID
App
IPC
App
Dalvik / ART
User mode (PL 0)
Kernel mode (PL1)
Android/Linux Kernel
cpsr_write
(kernel→user)
Writes to current status program reg
swi
(user→kernel)
SoftWare Interrupt instruction
Plug-in Manager
PLUGINS
COPPERDROID PLUG-IN
System Call Tracking
Binder Analysis
TCP
Oracle

SYSTEM CALLS ON LINUX ARM

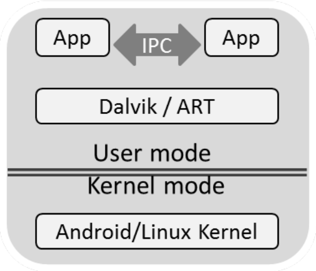
A system call induces a User -> Kernel transition

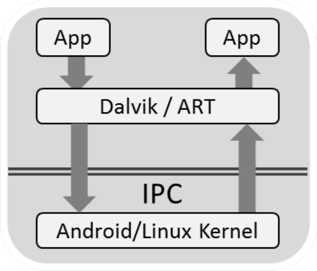
- On ARM invoked through the `swi` instruction (SoftWare Interrupt)
- `r7`: invoked system call number
- `r0-r5`: parameters
- `lr`: return address

CopperDroid's Approach

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

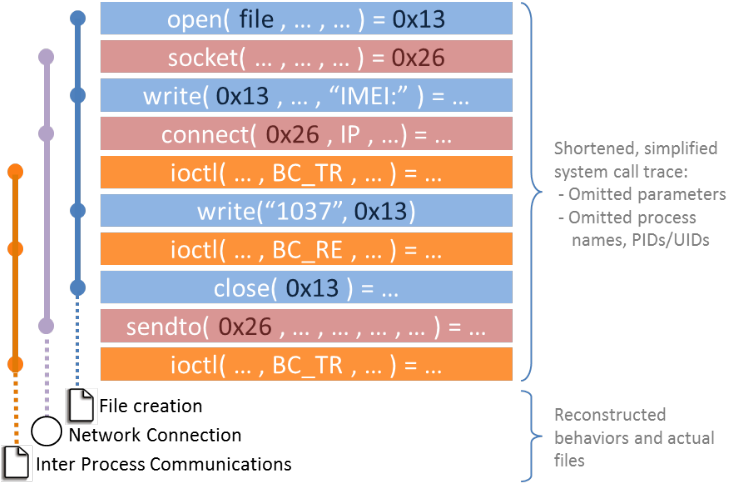
- OS functionality (e.g., open, read, write)
- Android functionality (Send SMS, Phone Call etc.)
  - → Inspect the Binder (IPC) protocol via I/O control system calls to the binder kernel driver





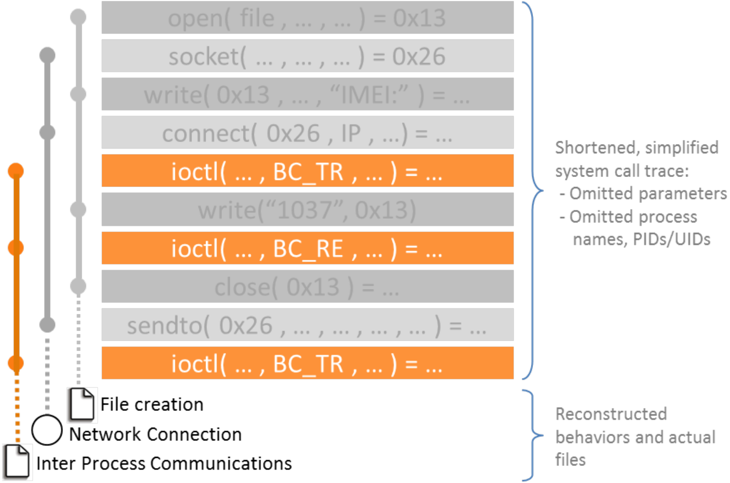
EXTRACTING BEHAVIORS

- Android functionality (Send SMS, Phone Call etc.)
- OS functionality (e.g., open, read, write)



EXTRACTING BEHAVIORS

- Android functionality (Send SMS, Phone Call etc.)
- OS functionality (e.g., open, read, write)



IPC/RPC

- Binder protocols enable fast inter-process communication
- Allows apps to invoke other app component functions
- Binder objects handled by Binder Driver in kernel
  - → Serialized/marshalled passing through kernel
  - → Results in input output control (ioctl) system calls

Android Interface Definition Language (AIDL)

- AIDL defines which/how services can be invoked remotely
- Describes how to marshal method parameters
- We modified AIDL parser to understand marshalled Binders

IPC BINDER CREATION: TOP TO BOTTOM

Application

```
PendingIntent sentIntent = PendingIntent.getBroadcast(SMS.this,
            0, new Intent("SENT"), 0);
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage("7855551234", null, "Hi There", sentIntent, null);
```

IPC BINDER CREATION: TOP TO BOTTOM

Application

android.telephony.SmsManager

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

IPC BINDER CREATION: TOP TO BOTTOM

Application

↓

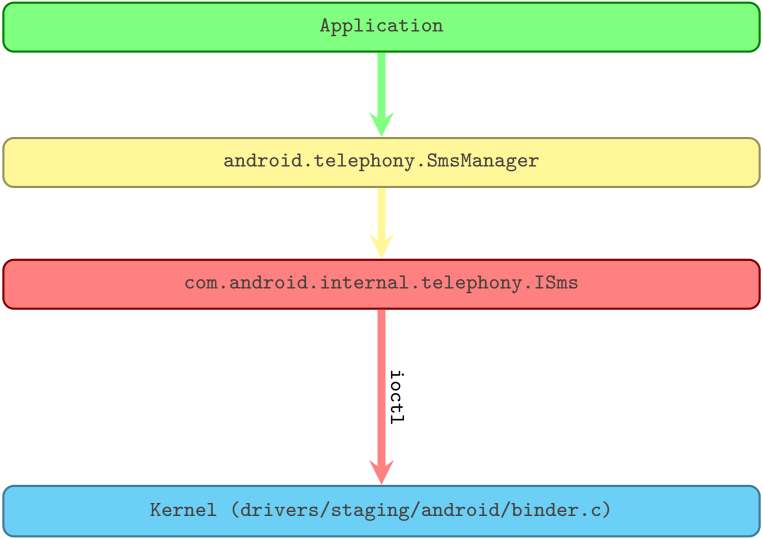
android.telephony.SmsManager

↓

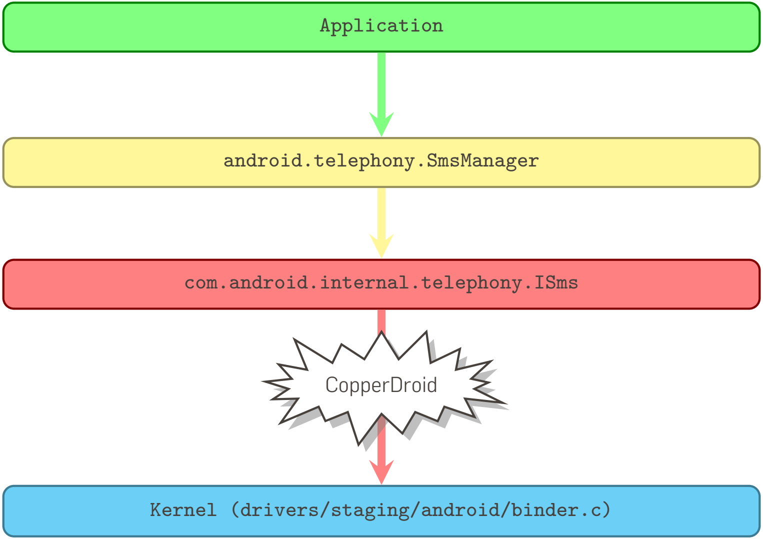
com.android.internal.telephony.ISms

```
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);
  }
```

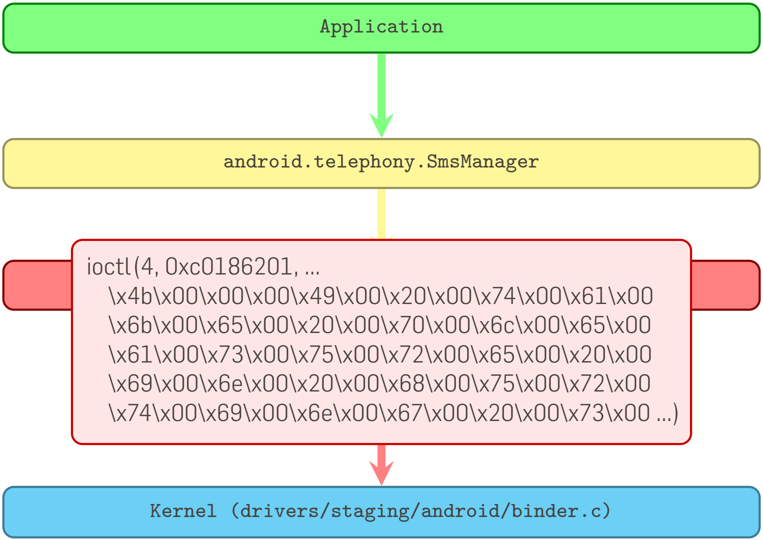
IPC BINDER CREATION: TOP TO BOTTOM



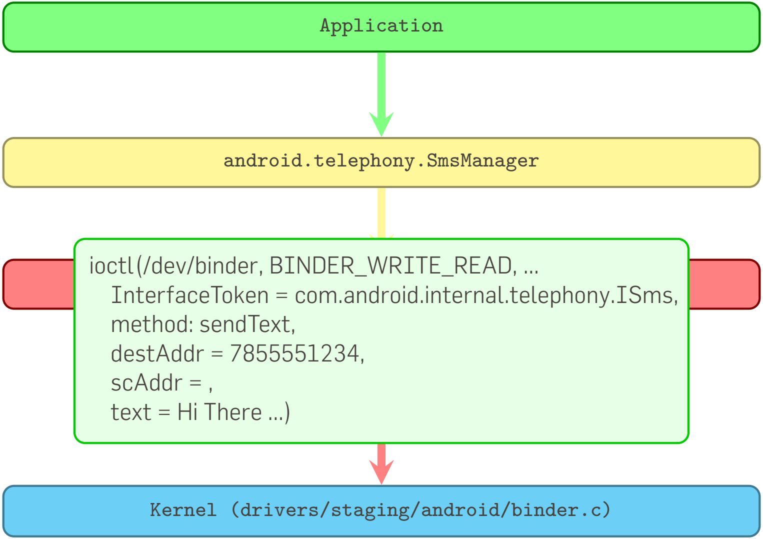
IPC BINDER CREATION: TOP TO BOTTOM



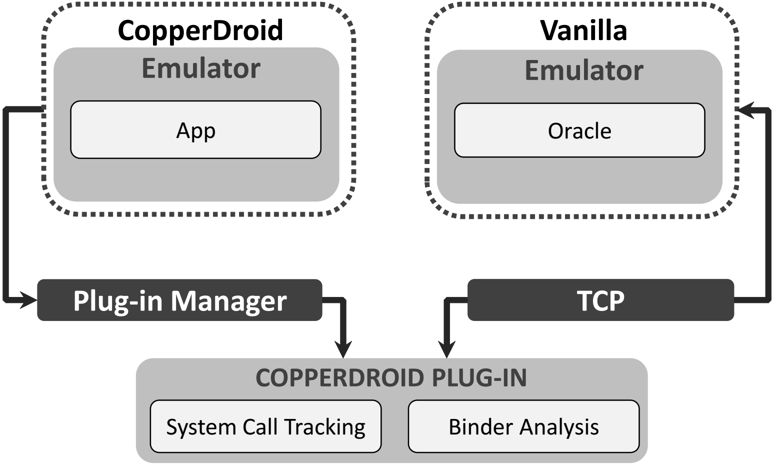
IPC BINDER CREATION: TOP TO BOTTOM



IPC BINDER CREATION: TOP TO BOTTOM



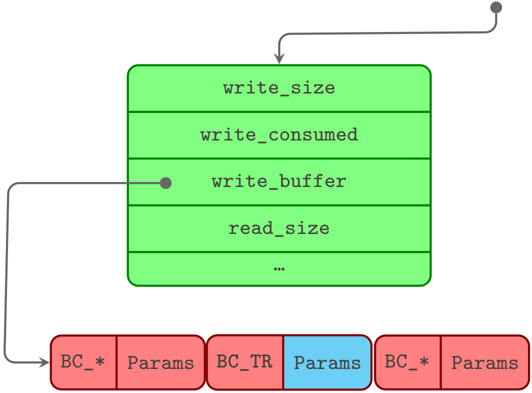
AUTOMATIC ANDROID OBJECTS UNMARSHALLING


CopperDroid
Emulator
App
Vanilla
Emulator
Oracle
Plug-in Manager
TCP
COPPERDROID PLUG-IN
System Call Tracking
Binder Analysis

BINDER STRUCTURE WITHIN IOCTL

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

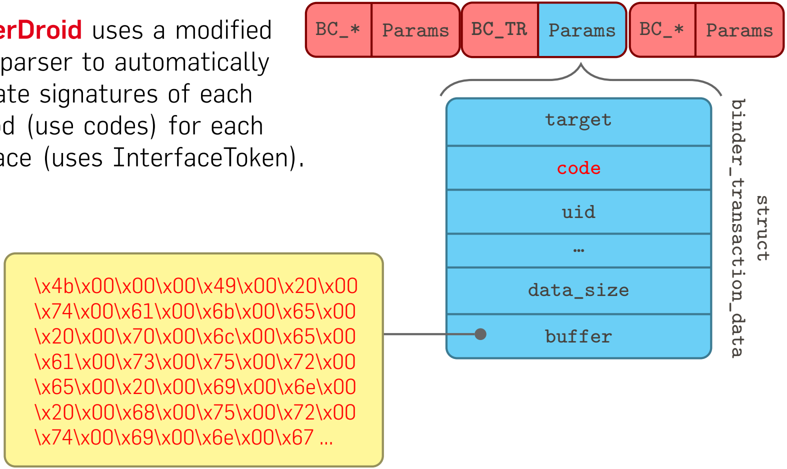
```
ioctl(binder_fd, BINDER_WRITE_READ, &binder_write_read);
```



AUTOMATIC BINDER UNMARSHALLING

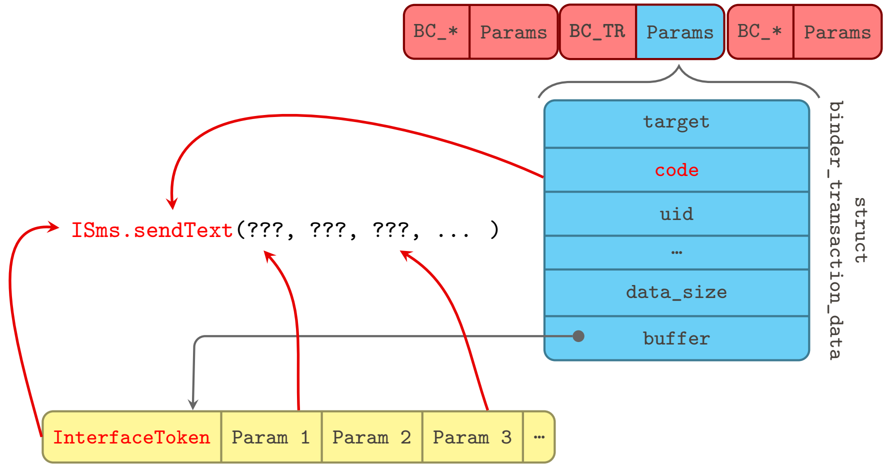
CopperDroid analyzes `BC_TRANSACTION`s and `BC_REPLY`s

CopperDroid uses a modified AIDL parser to automatically generate signatures of each method (use codes) for each interface (uses InterfaceToken).



AUTOMATIC BINDER UNMARSHALLING

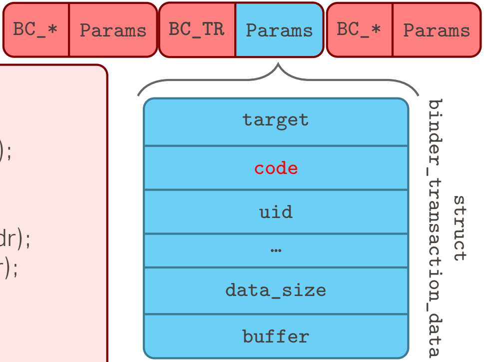
CopperDroid analyzes `BC_TRANSACTION`s and `BC_REPLY`s



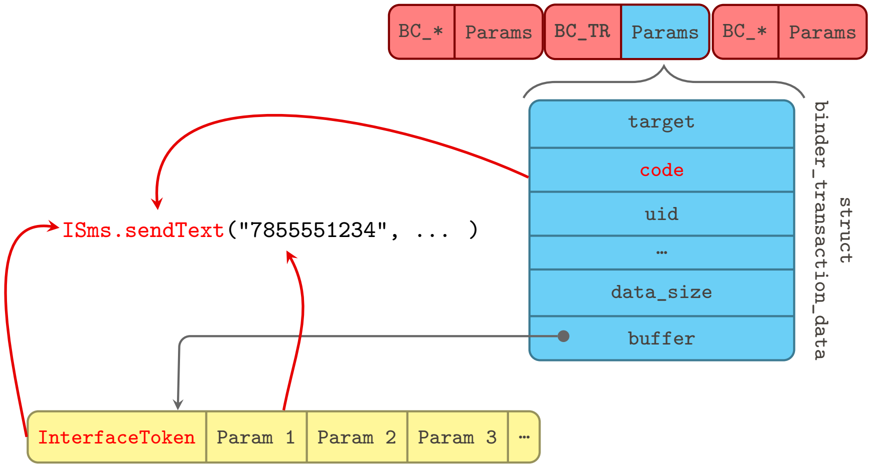
AUTOMATIC BINDER UNMARSHALLING

CopperDroid analyzes `BC_TRANSACTION`s and `BC_REPLY`s

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



CopperDroid analyzes `BC_TRANSACTION`s and `BC_REPLY`s



AUTOMATIC ANDROID OBJECTS UNMARSHALLING

- Primitive types (e.g., String text)
  - → A few manually-written procedures
- Complex Android objects
  - → 300+ Android objects (can't unmarshal manually)
  - → Finds object ``creator field"
  - → Use reflection (type introspection, then intercession)
- IBinder object reference
  - → A handle (pointer) sent instead of marshalled object
  - → Look earlier in trace to map each handle to an object

CopperDroid's Oracle unmarshalls all three automatically

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

INPUT: Types ["string", "string", "string", "PendingIntent", "PendingIntent"]

INPUT: Data [\x0A\x00\x00\x00\x34\x00 \x38\x00\x35\x00\x35\x00\x35\x00\x35 \x00\x31\x00\x32\x00\x33 \x00\x34\x00 \x00\x00\x00\x08\x00\x00\x00\x48\x00 \x69\x00\x20\x00\x74\x00\x68\x00\x65 \x00\x72\x00 \x65\x00\x85*hs\x7f\x00 \x00\x00\xa0\x00\x00\x00\x00\x00\x00 \x00 ...]

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

ORACLE ACTION:
Type[0] = Primitive "**string**"
at offset 0: ReadString()
increment offset by len(string)

ORACLE OUTPUT:
com.android.internal.tele-
phony.ISms.sendText(
destAddr = 7855551234
)

INPUT: Types ["**string**", "string", "string", "PendingIntent", "PendingIntent"]

INPUT: Data [\x0A\x00\x00\x00\x34\x00\x38\x00\x35\x00\x35\x00\x35\x00\x35\x00\x31\x00\x32\x00\x33 \x00\x34\x00\x00\x00\x00\x08\x00\x00\x00\x48\x00\x69\x00\x20\x00\x74\x00\x68\x00\x65\x00\x72\x00 \x65\x00\x85*hs\x7f\x00\x00\x00\xa0\x00\x00\x00\x00\x00\x00\x00 ...]

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

ORACLE ACTION:
Type[2] = Primitive "**string**"
at offset 18: ReadString()
increment offset by len(string)

ORACLE OUTPUT:
com.android.internal.tele-
phony.ISms.sendText(
destAddr = 7855551234
srcAddr = null
text = "Hi there"
)

INPUT: Types ["string", "string", "**string**",
"PendingIntent", ...]

INPUT: Data [\x0A\x00\x00\x00\x34\x00
\x38\x00\x35\x00\x35\x00\x35\x00\x35
\x00\x31\x00\x32\x00\x33 \x00\x34\x00
\x00\x00\x00\x08\x00\x00\x00\x48\x00
\x69\x00\x20\x00\x74\x00\x68\x00\x65
\x00\x72\x00 \x65\x00\x85*hs\x7f\x00
\x00\x00\xa0\x00\x00\x00\x00\x00\x00
\x00 ...]

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

ORACLE ACTION:
Type[3] = IBinder "**PendingIntent**"
at offset 18: Parse IBinder for handle
increment offset by sizeof(IBinder)

ORACLE OUTPUT:

```
com.android.internal.tele-
   phony.ISms.sendText(
      destAddr = 7855551234
      srcAddr = null
      text = "Hi there"
      sentIntent {
        type = BINDER_TYPE_HANDLE
        flags = 0x7F | FLAT_BINDER_
                  FLAG_ACCEPT_FDS
        handle = 0xa
        cookie = 0x0
}
)
```

INPUT: Types ["string", "string", "string",
"**PendingIntent**", ...]

INPUT: Data [\x0A\x00\x00\x00\x34\x00
\x38\x00\x35\x00\x35\x00\x35\x00\x35
\x00\x31\x00\x32\x00\x33 \x00\x34\x00
\x00\x00\x00\x08\x00\x00\x00\x48\x00
\x69\x00\x20\x00\x74\x00\x68\x00\x65
\x00\x72\x00 \x65\x00\x85*hs\x7f\x00
\x00\x00\xa0\x00\x00\x00\x00\x00\x00
\x00 ...]

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

ORACLE ACTION:
Type[3] = IBinder "**PendingIntent**"
at offset 18: Unmarshal com.Android.Intent (AIDL)
increment offset by sizeof(IBinder)

ORACLE OUTPUT:
com.android.internal.telephony.ISms.sendText(
destAddr = 7855551234
srcAddr = null
text = "Hi there"
sentIntent {
Intent("SENT") }
)

INPUT: Types ["string", "string", "string", "**PendingIntent**", ...]

INPUT: Data [\x0A\x00\x00\x00\x34\x00 \x38\x00\x35\x00\x35\x00\x35\x00\x35 \x00\x31\x00\x32\x00\x33 \x00\x34\x00 \x00\x00\x00\x08\x00\x00\x00\x48\x00 \x69\x00\x20\x00\x74\x00\x68\x00\x65 \x00\x72\x00 \x65\x00\x85*hs\x7f\x00 \x00\x00\xa0\x00\x00\x00\x00\x00\x00 \x00 ...]

INPUT: Found with reference 0xa
Data [... \x01\x00\x00\x00 \x04\x00 \x00\x00S\x00E\x00N\x00T ...]

CONCLUSIONS

CopperDroid: automatic reconstruction of Android apps behaviors

Key Insight

All Android behaviors eventually manifest as system calls

- Challenge: reconstruction of Android semantics from low-level events

System call-centric analysis on unmodified Android

- Unmarshalling oracle to reconstruct Android semantics
- Agnostic to underlying runtime (Dalvik vs. ART)
- Opens possibility of a realistic in-device monitoring

Available at: http://copperdroid.isg.rhul.ac.uk
Open source soon: http://s2lab.isg.rhul.ac.uk/projects/mobsec/

BACKUP SLIDES

STIMULATION 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%)

IBINDER HANDLE/INTENT SYSTEM CALLS