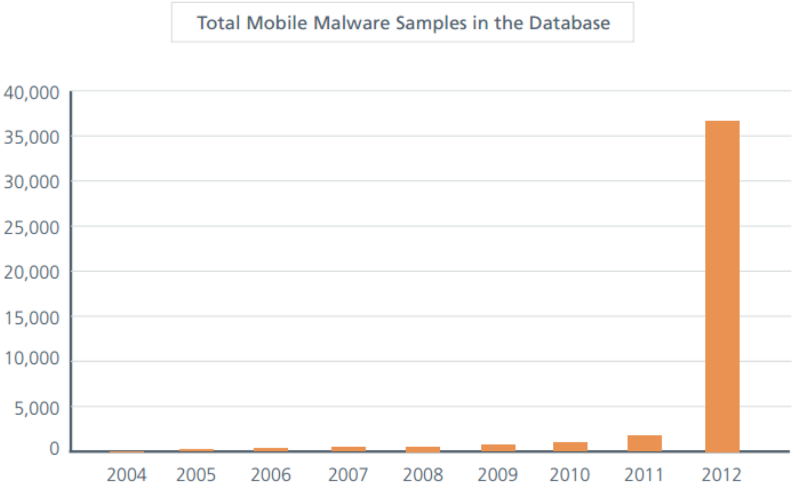
A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors

Alessandro Reina[†], **Aristide Fattori**[†], Lorenzo Cavallaro[‡]

† Università degli Studi di Milano ‡ Royal Holloway, University of London

6th European Workshop on System Security
Prague, Czech Republic, April 14, 2013

Android Malware: the Rise



Source: McAfee Threat Reports 2012

Android Malware: the Rise

Why?

Android is rapidly becoming the Windows of Mobile OSes

- **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
  (BYOD: worst nightmare ever for security guys)

Android Malware: the Rise

Why?

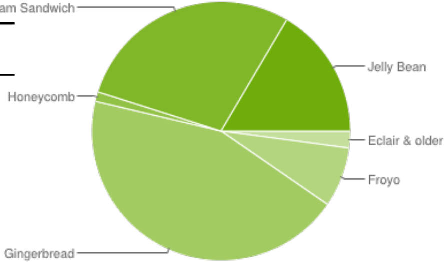
Android is rapidly becoming the Windows of Mobile OSes

- 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
  (BYOD: worst nightmare ever for security guys)

Android Malware: the Rise

Why?

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

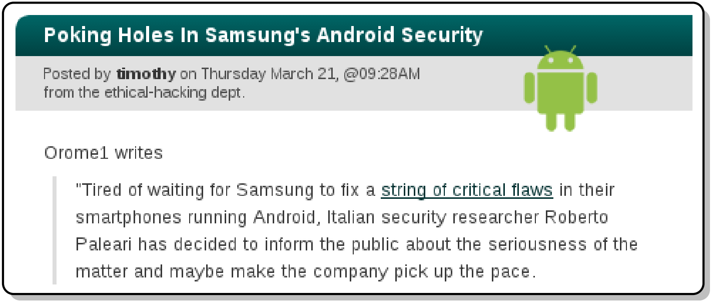
Why?

Android is rapidly becoming the Windows of Mobile OSes

- Widely adopted on heterogeneous devices
- Producers push patches/updates slowly
- **Operators' and Producers' customizations**
  Often Closed-Source
- **Rooted Devices, Jailbreaks**
- **Custom ROMS: CyanogenMod, MIUI**
- **Custom kernels, modems**
- A number of interesting information on a phone
  (BYOD: worst nightmare ever for security guys)

Android Malware: the Rise

Why?



Poking Holes In Samsung's Android Security

Posted by **timothy** on Thursday March 21, @09:28AM
from the ethical-hacking dept.

Orome1 writes

> "Tired of waiting for Samsung to fix a string of critical flaws in their smartphones running Android, Italian security researcher Roberto Paleari has decided to inform the public about the seriousness of the matter and maybe make the company pick up the pace.

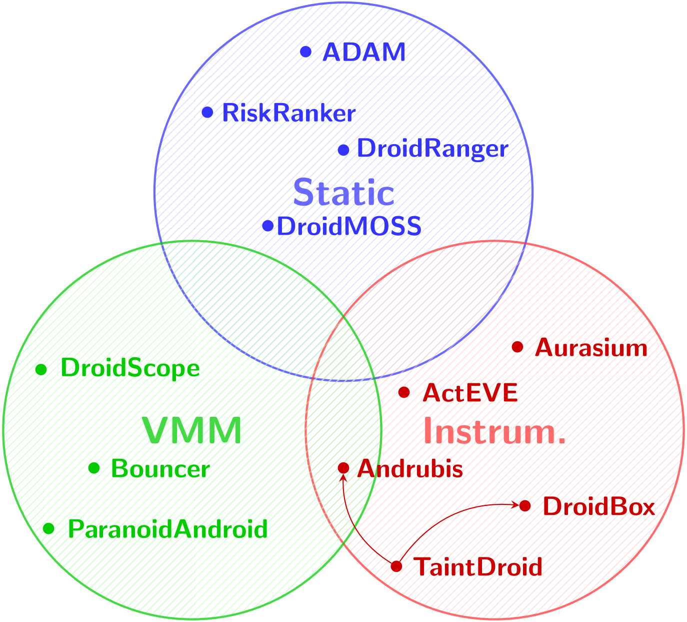
Android Malware: the Rise

Why?

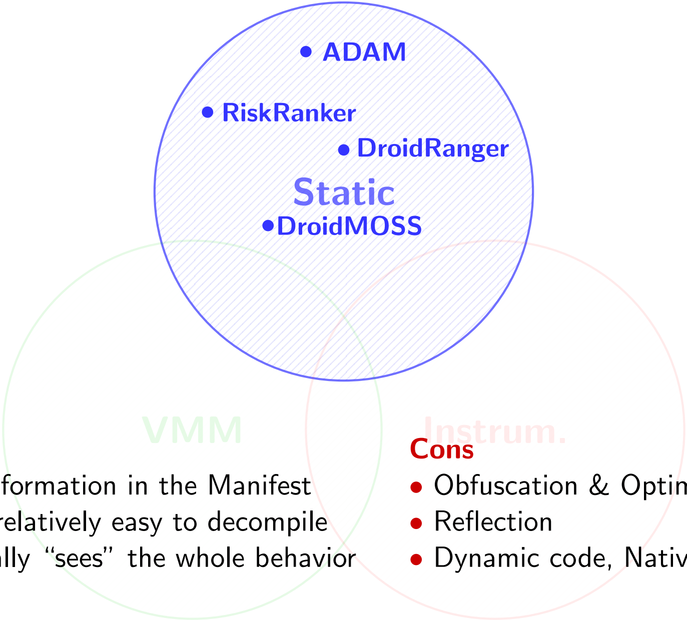
Android is rapidly becoming the Windows of Mobile OSes

- 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**
  (BYOD: worst nightmare ever for security guys?)

Malware Analysis



Malware Analysis: Static



- ADAM
- RiskRanker
- DroidRanger

Static

- DroidMOSS

VMM

Instrum.

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

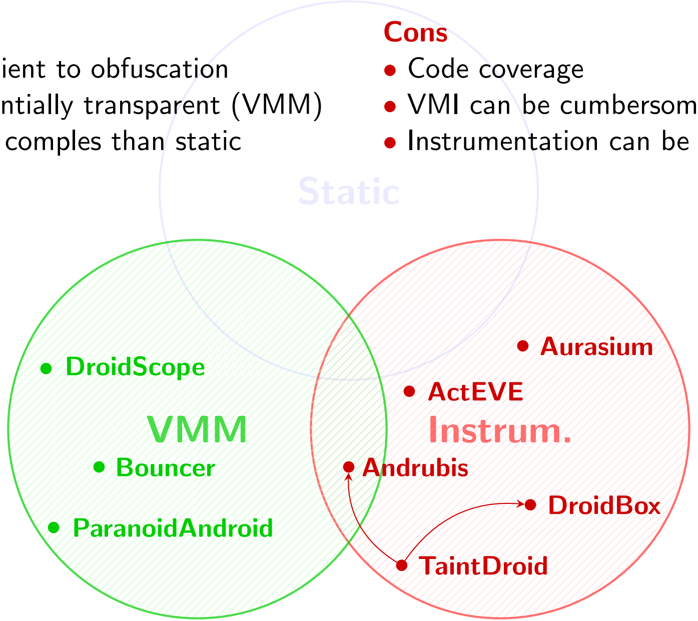
Malware Analysis: Dynamic

Pros

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

Cons

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



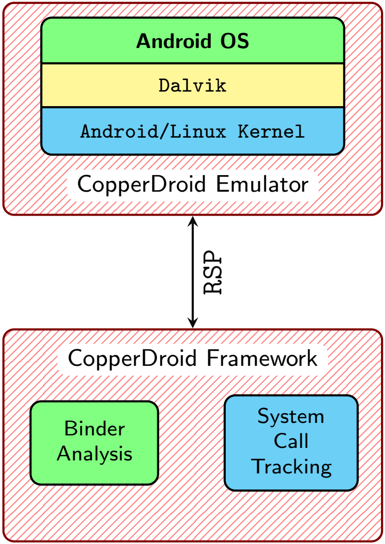
CopperDroid

An *unified* dynamic analysis technique to characterize the behavior of android malware.

Features

1. Automatically reconstructs the behaviors of Android malware
2. System-call centric analysis
   (everything is based on system interaction, i.e., syscalls)
3. Android version independent
4. Dynamically stimulates Apps to disclose additional behaviors

Architecture



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
  (**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`)

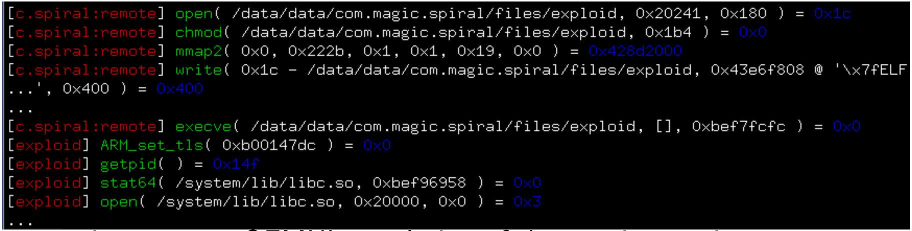
CopperDroid's Approach

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

Tracking System calls

System call Analysis

- Intercept when a syscall is invoked
- We need to intercept return to user-space too!



```
[c.spiral:remote] open( /data/data/com.magic.spiral/files/exploid, 0x20241, 0x180 ) = 0x1c
[c.spiral:remote] chmod( /data/data/com.magic.spiral/files/exploid, 0x1b4 ) = 0x0
[c.spiral:remote] mmap2( 0x0, 0x222b, 0x1, 0x1, 0x19, 0x0 ) = 0x428d2000
[c.spiral:remote] write( 0x1c - /data/data/com.magic.spiral/files/exploid, 0x43e6f808 @ '\x7fELF
...', 0x400 ) = 0x400
...
[c.spiral:remote] execve( /data/data/com.magic.spiral/files/exploid, [], 0xbef7fcfc ) = 0x0
[exploid] ARM_set_tls( 0xb00147dc ) = 0x0
[exploid] getpid( ) = 0x14f
[exploid] stat64( /system/lib/libc.so, 0xbef96958 ) = 0x0
[exploid] open( /system/lib/libc.so, 0x20000, 0x0 ) = 0x3
...
```

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

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

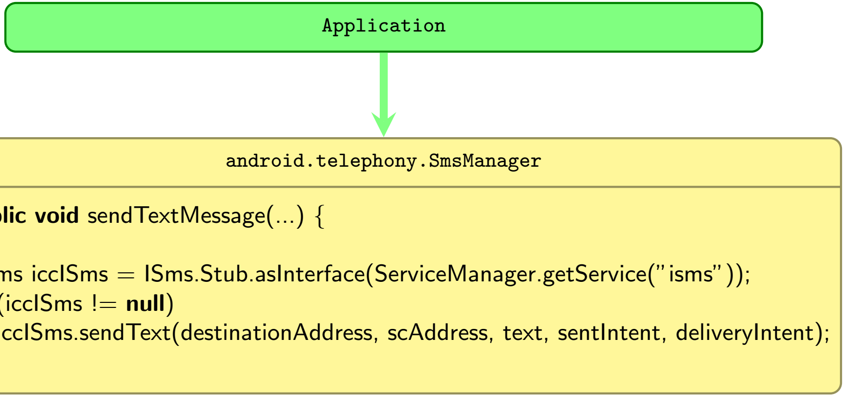
For example, applications cannot send SMSs on their own, but must invoke (RPC) the proper system service to do that.

Binder

Application

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

Binder

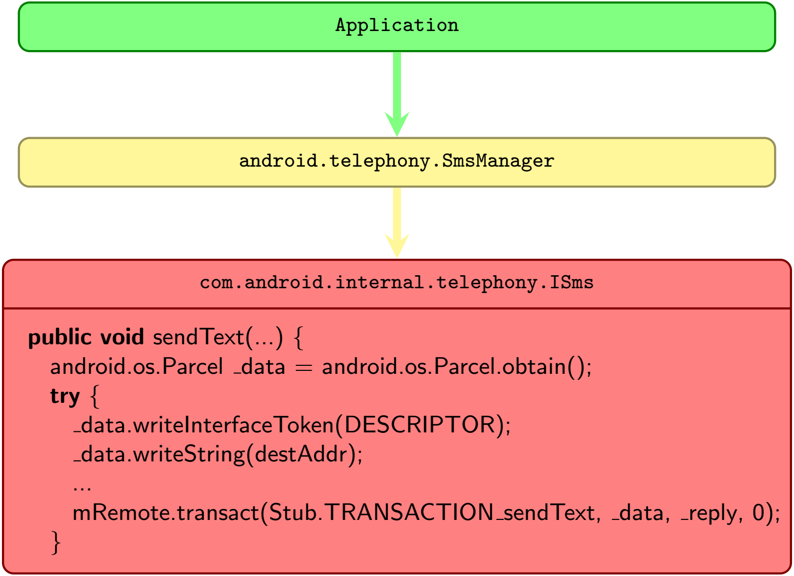


Application

android.telephony.SmsManager

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

Binder



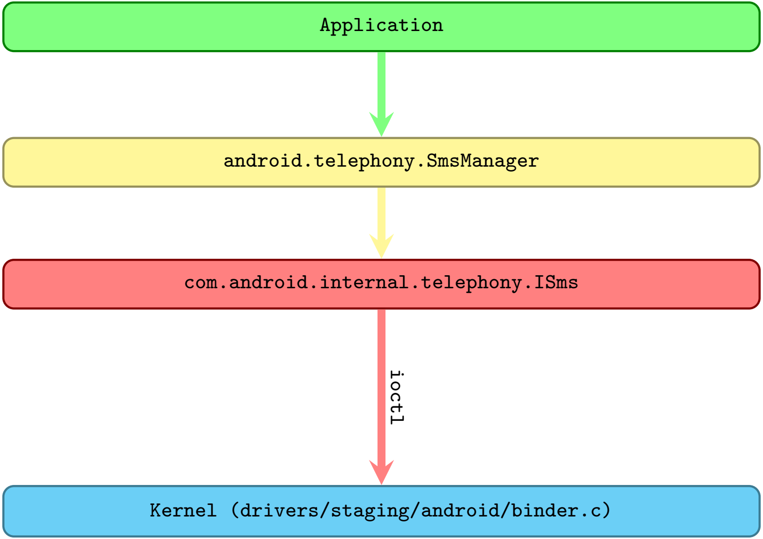
Application

android.telephony.SmsManager

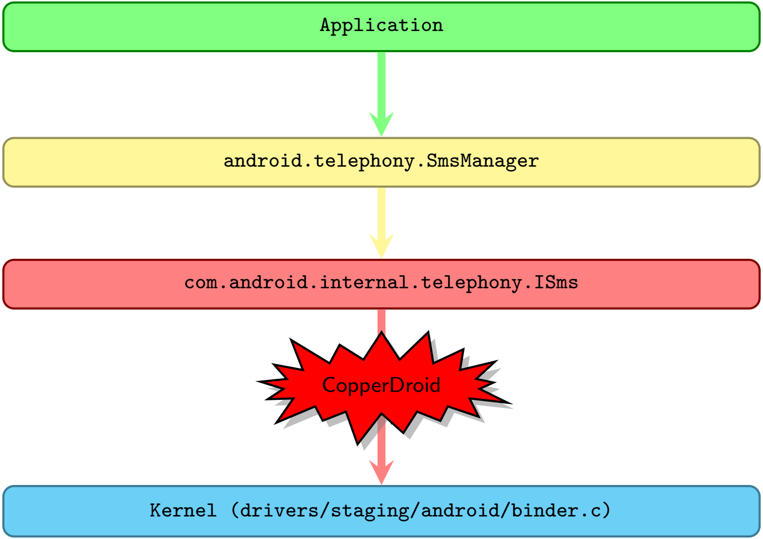
com.android.internal.telephony.ISms

```
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);
  }
```

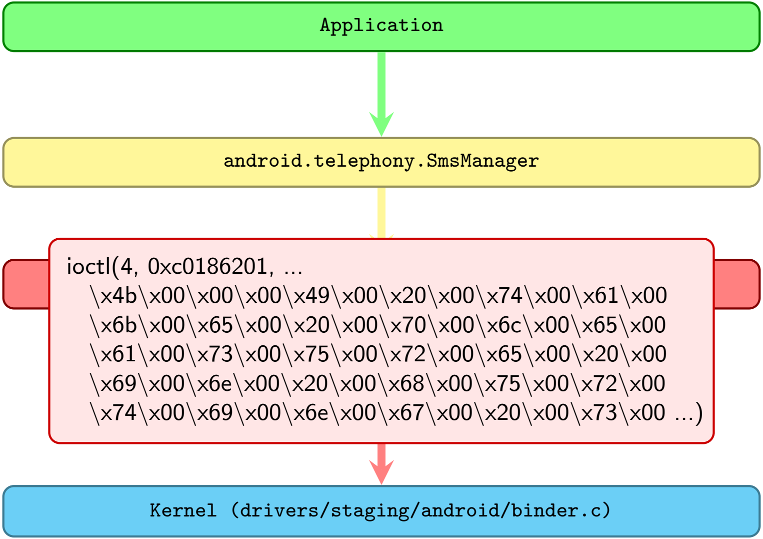
Binder


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)

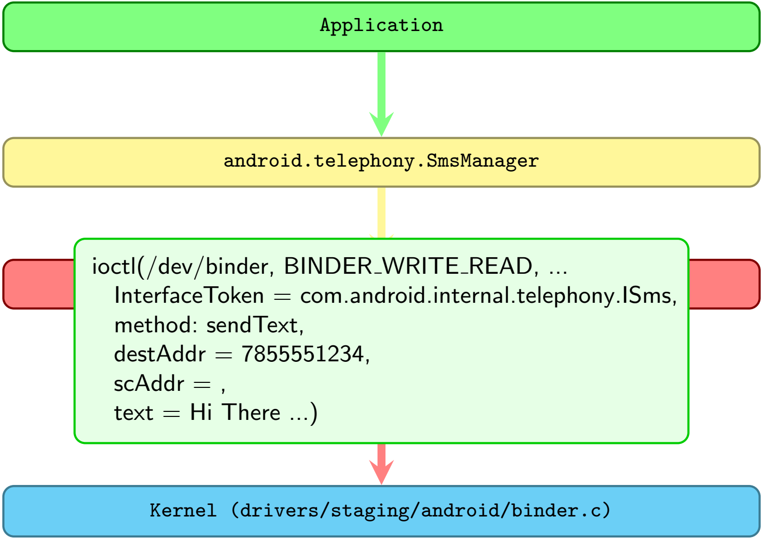
Binder


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 ...)
```

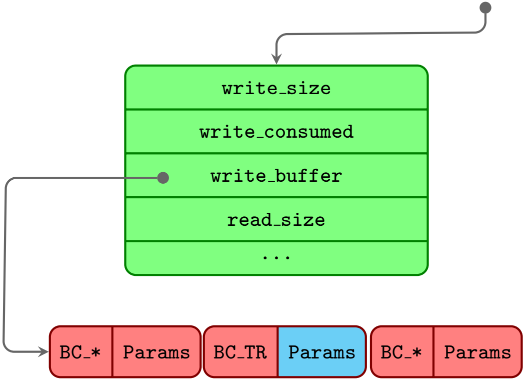
`Kernel (drivers/staging/android/binder.c)`

Binder

CopperDroid Analysis

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

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



write_size

write_consumed

write_buffer

read_size

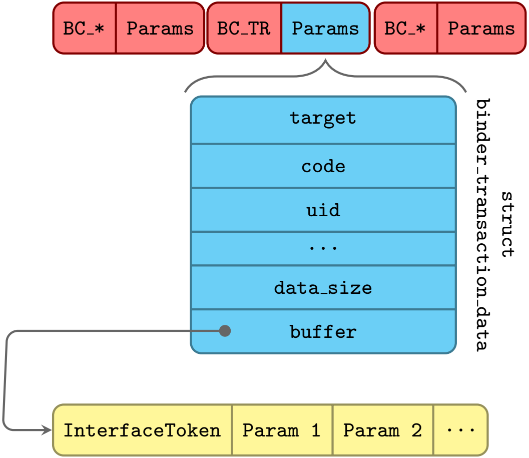
...

BC_*	Params	BC_TR	Params	BC_*	Params

Binder

write_buffer operations

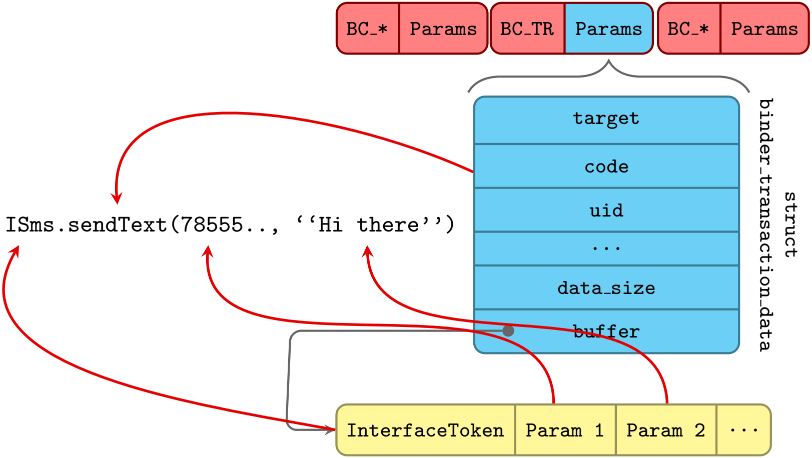
CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs



Binder

write_buffer operations

CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs

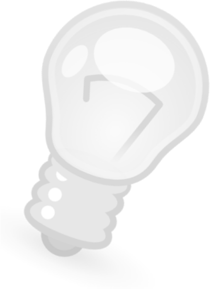


Stimulation

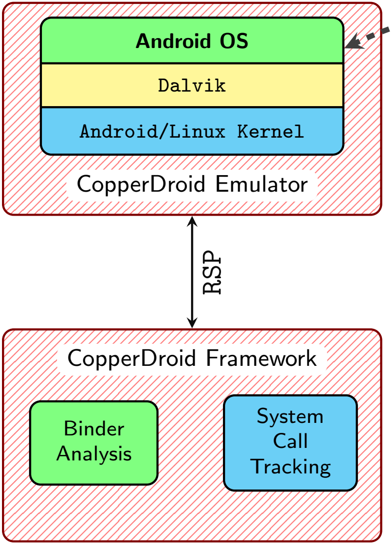
Android malware needs to be properly stimulated to trigger more malicious behaviors and increase coverage of dynamic analysis.

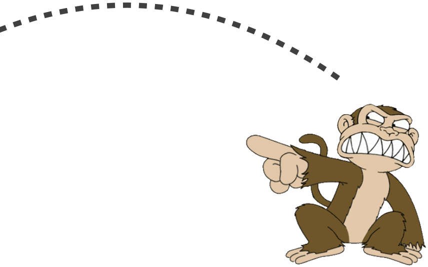
CopperDroid Ad-Hoc Stimuli

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)



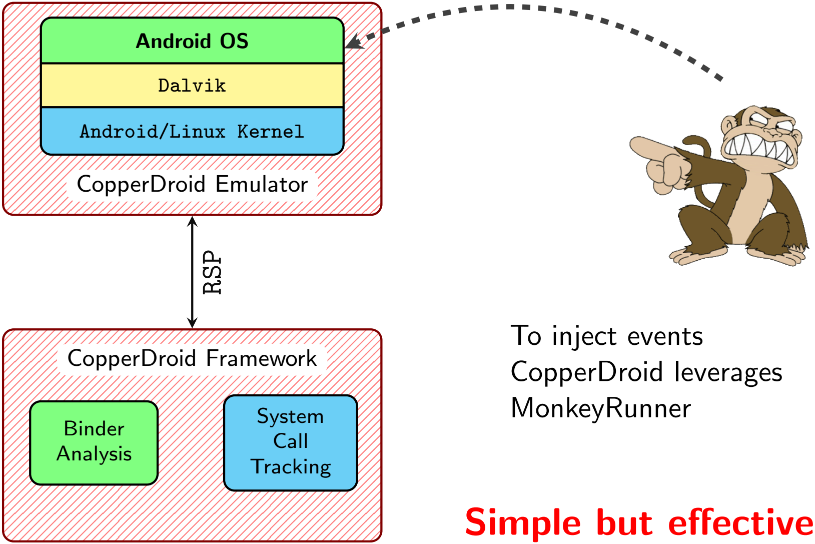
Stimulation





To inject events CopperDroid leverages MonkeyRunner

Stimulation



To inject events CopperDroid leverages MonkeyRunner

Simple but effective!

Evaluation

CopperDroid analyzed 1,200 malware from the Android Malware Genome Project and 395 from the Contagio repository.

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

Conclusions

CopperDroid Analysis Framework

Automatically reconstructs the behaviors of Android malware

- Unified analysis that avoid multi-layered VMI

  All the behaviors are eventually achieved via system interactions
- Dynamically stimulates Apps to disclose additional behaviors
- Extensive evaluation on ∼1,600 Android malware

Conclusions

CopperDroid Analysis Framework

1. Available at `http://copperdroid.isg.rhul.ac.uk`
2. Ongoing project
   2.1 Automatic AIDL Unmarshalling ✓
   2.2 Detailed stimulation ✓
   2.3 Extensive evaluation (McAfee support) ✓
   2.4 Behavioral attribution
   2.5 Detection
   2.6 . . .

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

http://copperdroid.isg.rhul.ac.uk/

Thank you!
Any questions?

Aristide Fattori
joystick@security.di.unimi.it

Backup Slides

Binder

Some examples of interesting binder transactions

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