COPPERDROID
Automatic Reconstruction of Android Malware Behaviors

Network and Distributed System Security Symposium (NDSS)
San Diego, USA, February 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
Università 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.
Over 1.75 billion Mobile users worldwide 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.
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 → User)
EXTRACTING BEHAVIORS

- 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)

```
open(file, ..., ...) = 0x13
socket(..., ..., ...) = 0x26
write(0x13, ..., "IMEI:" ) = ...
connect(0x26, IP, ...) = ...
ioctl(..., BC_TR, ...) = ...
write("1037", 0x13)
ioctl(..., BC_RE, ...) = ...
close(0x13) = ...
sendto(0x26, ..., ..., ..., ...) = ...
ioctl(..., BC_TR, ...) = ...
```

- Shortened, simplified system call trace:
- Omitted parameters
- Omitted process names, PID/UIDs

- File creation
- Network Connection
- Inter Process Communications

Reconstructed behaviors and actual files
EXTRACTING BEHAVIORS

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

```c
open( file, ..., ... ) = 0x13
socket( ..., ..., ... ) = 0x26
write( 0x13, ..., "IMEI:" ) = ...
connect( 0x26, IP, ... ) = ...
ioctl( ..., BC_TR, ... ) = ...
write("1037", 0x13)
ioctl( ..., BC_RE, ... ) = ...
close( 0x13 ) = ...
sendto( 0x26, ..., ..., ..., ... ) = ...
ioctl( ..., BC_TR, ... ) = ...
```

Shortened, simplified system call trace:
- Omitted parameters
- Omitted process names, PID/UIDs

Reconstructed behaviors and actual files
THE BINDER PROTOCOL

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
Application

PendingIntent sentIntent = PendingIntent.getBroadcast(SMS.this, 0, new Intent("SENT"), 0);
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage("7855551234", null, "Hi There", sentIntent, null);
public void sendTextMessage(...) {
    ...
    ISms iccISms = ISms.Stub.asInterface(ServiceManager.getService("isms"));
    if (iccISms != null)
        iccISms.sendText(destinationAddress, scAddress, text, sentIntent, deliveryIntent);
    ...
}
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

Application

android.telephony.SmsManager

com.android.internal.telephony.ISms

ioctl

Kernel (drivers/staging/android/binder.c)
IPC BINDER CREATION: TOP TO BOTTOM

Application

android.telephony.SmsManager

com.android.internal.telephony.ISms

CopperDroid

Kernel (drivers/staging/android/binder.c)
IPC BINDER CREATION: TOP TO BOTTOM

Application

android.telephony.SmsManager

ioctl(4, 0xc0186201, ...
\x4b\x00\x00\x00\x49\x00\x20\x00\x74\x00\x61\x00
\x6b\x00\x65\x00\x70\x00\x6c\x00\x61\x00\x75\x00
\x72\x00\x65\x00\x74\x00\x73\x00 ...)

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

android.telephony.SmsManager

ioctl(/dev/binder, BINDER_WRITE_READ, ...
   InterfaceToken = com.android.internal.telephony.ISms,
   method: sendText,
   destAddr = 785551234,
   scAddr = ,
   text = Hi There ...)

Kernel (drivers/staging/android/binder.c)
CopperDroid inspects the Binder protocol in detail by intercepting a subset of the ioctl's issued by userspace Apps.

```c
ioctl(binder_fd, BINDER_WRITE_READ, &binder_write_read);
```
CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs.

CopperDroid uses a modified AIDL parser to automatically generate signatures of each method (use codes) for each interface (uses InterfaceToken).
**CopperDroid** analyzes BC.TRANSACTIONs and BC.REPLYs.
CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs

```java
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_TRANSACTIONS and BC_REPLYs.

```c
ISms.sendText("7855551234", ...)
```

```
struct binder_transaction_data
    target
    code
    uid
    ...
    data_size
    buffer

InterfaceToken | Param 1 | Param 2 | Param 3 | ...
```

```
ISms.sendText("7855551234", ...)
```
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
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\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... ]
ORACLE ACTION:
Type[0] = Primitive "string"
at offset 0: ReadString()
increment offset by len(string)

ORACLE OUTPUT:
com.android.internal.telephony.ISms.sendText(
    destAddr = 785551234
)
ORACLE ACTION:
Type[2] = Primitive "string"
at offset 18: ReadString()
increment offset by len(string)

ORACLE OUTPUT:
com.android.internal.telephony.ISms.sendText(
  destAddr = 7855551234
  srcAddr = null
  text = "Hi there"
)
ORACLE ACTION:
Type[3] = IBinder "PendingIntent"
at offset 18: Parse IBinder for handle increment offset by sizeof(IBinder)

ORACLE OUTPUT:
com.android.internal.telephony.ISms.sendText(
  destAddr = 7855551234
  srcAddr = null
  text = "Hi there"
  sentIntent {
    type = BINDER_TYPE_HANDLE
    flags = 0x7F | FLAT_BINDER_FAULT
    handle = 0xa
    cookie = 0x0
  }
)
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\x00\x48\x00
\x69\x00\x20\x00\x74\x00\x68\x00\x65
\x00\x72\x00 \x65\x00\x85*hs\7f\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 ... ]
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](http://copperdroid.isg.rhul.ac.uk)
Open source soon: [http://s2lab.isg.rhul.ac.uk/projects/mobsec/](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

<table>
<thead>
<tr>
<th>#</th>
<th>Malware Family</th>
<th>Stim.</th>
<th>Samples w/ Add. Behav.</th>
<th>Behavior w/o Stim.</th>
<th>Incr. Behavior w/ Stimuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADRD</td>
<td>3.9</td>
<td>17/21</td>
<td>7.24</td>
<td>4.5 (63%)</td>
</tr>
<tr>
<td>2</td>
<td>AnserverBot</td>
<td>3.9</td>
<td>186/187</td>
<td>31.52</td>
<td>8.2 (27%)</td>
</tr>
<tr>
<td>3</td>
<td>BaseBridge</td>
<td>2.9</td>
<td>70/122</td>
<td>16.44</td>
<td>5.2 (32%)</td>
</tr>
<tr>
<td>4</td>
<td>BeanBot</td>
<td>3.1</td>
<td>4/8</td>
<td>0.12</td>
<td>3.8 (3000%)</td>
</tr>
<tr>
<td>5</td>
<td>CruseWin</td>
<td>4.0</td>
<td>2/2</td>
<td>1.00</td>
<td>2.0 (200%)</td>
</tr>
<tr>
<td>6</td>
<td>GamblerSMS</td>
<td>4.0</td>
<td>1/1</td>
<td>1.00</td>
<td>3.0 (300%)</td>
</tr>
<tr>
<td>7</td>
<td>SMSReplicator</td>
<td>4.0</td>
<td>1/1</td>
<td>0.00</td>
<td>6.0 (⊥)</td>
</tr>
<tr>
<td>8</td>
<td>Zsone</td>
<td>5.0</td>
<td>12/12</td>
<td>16.67</td>
<td>3.8 (23%)</td>
</tr>
</tbody>
</table>
**IBINDER HANDLE/INTENT SYSTEM CALLS**

1. Register SMS Service

2. Send Intent

3. Send Intent’s handle

4. Please send this SMS