

Guess Where I am:

Android模拟器躲避的检测与应对

胡文君(MindMac) 肖梓航(Claud Xiao)

# 模拟器检测技术

# 模拟器技术应用广泛

- Google、安全企业使用模拟器检测恶意代码
- 为什么用模拟器而不是真机？
  - 经济成本低
  - 高度可定制
  - 易于开发
  - 容易部署

# 模拟器检测技术现状

- Timothy Vidas and Nicolas Christin, Evading Android Runtime Analysis via Sandbox Detection, ASIACCS'14
- 赵闽 and 倪超, 逃离安卓动态检测&订票助手一日谈, HitCon 2013
- Tim Strazzere, Dex Education 201:Anti-Emulators, HitCon 2013
- Patrick Schulz, Android Emulator Detection by Observing Low-level Caching Behavior
- Felix Matenaar and Patrick Schulz, Detecting Android SandBoxes
- Jon Oberheide and Charlie Miller, Dissecting the Android Bouncer, SummerCon 2012
- Nicholas J. Percoco and Sean Schuler, Adventures in BouncerLand, Black Hat USA 2012
- Vaibhav Rastogi, Yan Chen and William Enck, AppsPlayGround: Automatic Security Analysis of Smartphone Applications, CODASPY' 13

# 模拟器检测技术的分类

- 模拟检测技术从上至下可分为4类



# 基于用户层行为和数据检测模拟器

- 存在API Demos、Dev Tools等一般模拟器上的应用程序
- 联系人、短信、电话记录、相册是否为空？
- 应用程序安装数量很少 or 只有模拟器上默认的应用程序
- logcat一直处于运行状态？Log中记录敏感数据信息，如短信发送的目的地址和内容？

# 基于Android系统层特征检测模拟器

- 电话号码 == 15555215554-5584, etc
- 电池状态与电量
- wifi、GPS等硬件特征
- Build.Device == generic, etc
- 反射调用SystemProperties.get获取属性值
- 读取/system/build.prop文件
- Monkey行为模拟事件

# 基于Linux系统层特征检测模拟器

- 通过驱动信息特征检测模拟器
- 通过设备文件特征检测模拟器
- 通过执行shell命令检测模拟器
- 通过Native Code检测模拟器



# 基于模拟器体系结构特征检测模拟器

- 模拟器CPU信息异于真实机器
- DexLabs-Qemu二进制翻译技术
- BlueBox-模拟器底层缓存行为

# 基于模拟器体系结构特征检测模拟器

- 模拟器CPU信息异于真实机器
  - adb shell
  - cat /proc/cpuinfo

```
CPU part      : 0xc08
CPU revision  : 0
Hardware      : Goldfish
Revision      : 0000
Serial        : 0000000000000000
```

```
CPU part      : 0x06f
CPU revision  : 0
Hardware      : Qualcomm MSM 8974 HAMMERHEAD (Flattened Device Tree)
Revision      : 000b
Serial        : 0000000000000000
```

# 反模拟器对抗的应用不理想

- 使用模拟器的分析系统考虑到了模拟器检测行为的存在
- 各分析系统针对性地使用了反模拟器检测技术
- 最新研究结果表明实际效果并不理想
  - Timothy Vidas and Nicolas Christin, Evading Android Runtime Analysis via Sandbox Detection, ASIACCS'14

	Andrubis	CopperDroid	ForeSafe
<b>Detection method</b>			
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

图片来源: Evading Android Runtime Analysis via Sandbox Detection, ASIACCS'14

# 我们的问题

- 如何判断Android应用程序是否存在反模拟器行为？
- 真实世界中，有多大比例的应用程序会进行模拟器检测？采用何种手段进行检测？检测模拟器的目的？
- 如何构造更真实的模拟器，欺骗应用程序其运行在真机环境？

如何判断Android应用程序是否存在反模拟器行为？

# 反模拟器行为真实案例

- 通过Build.MODEL获取设备型号
- 与特定字符串进行比较

```
.method private static a()Z
.locals 2

.prologue
.line 117
sget-object v0, Landroid/os/Build;->MODEL:Ljava/lang/String;
const-string v1, "google_sdk"
invoke-virtual {v0, v1}, Ljava/lang/String;->compareToIgnoreCase(Ljava/lang/String;)I
move-result v0

if-eqz v0, :cond_0

sget-object v0, Landroid/os/Build;->MODEL:Ljava/lang/String;
const-string v1, "sdk"
invoke-virtual {v0, v1}, Ljava/lang/String;->compareToIgnoreCase(Ljava/lang/String;)I
move-result v0

if-eqz v0, :cond_0

const/4 v0, 0x0

:goto_0
return v0

:cond_0
const/4 v0, 0x1

goto :goto_0
.end method
```

模拟器标识

获取设备型号

与特定字符串比较

模拟器检测

# 反模拟器行为检测思路

- 反编译APK
- 搜索特定API以及字符串
- 若存在获取系统信息，并与特定字符串比较，则认为存在模拟器检测行为

# 反模拟器行为特征(40条特征)

- TelephonyManager类的API
  - (getDeviceId, 0000000000000000)
  - (getDeviceId, 012345678912345)
  - (getSubscriberId,310260000000000)
  - (getVoiceMailNumber, 15552175049)
  - ...
- Build类字段
  - (BRAND, generic)
  - (DEVICE, generic)
  - (HARDWARE, goldfish)
  - (HOST, android-test)
  - ...



# 反模拟器行为特征(40条特征)

- 特征文件
  - /dev/socket/qemud
  - /dev/qemu\_pipe
  - /dev/qemu\_trace
  - .....
- 系统属性
  - (ro.hardware, goldfish)
  - (ro.product.device, generic)
  - (ro.product.model, sdk)
  - (ro.product.name, sdk)
  - .....

无法检测基于用户层行为和数据、模拟器体系结构特征的反模拟器行为！

真实世界中，有多大比例的应用程序会进行模拟器检测？采用何种手段进行检测？检测模拟器的目的？

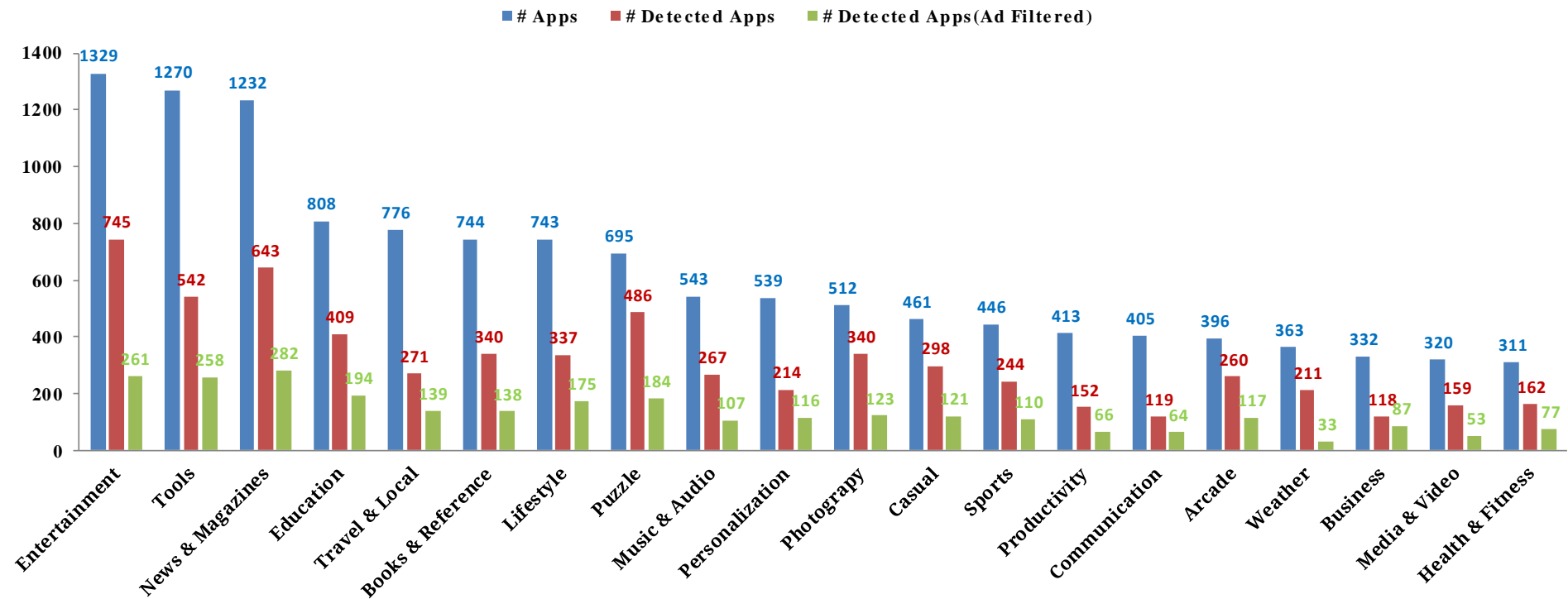
# 实验样本空间

- 正常应用程序
  - 来源: Google Play 2013
  - 样本规模: 14,195
- 恶意代码样本
  - 来源: [AndroMalShare](#)
  - 样本规模: 8,939

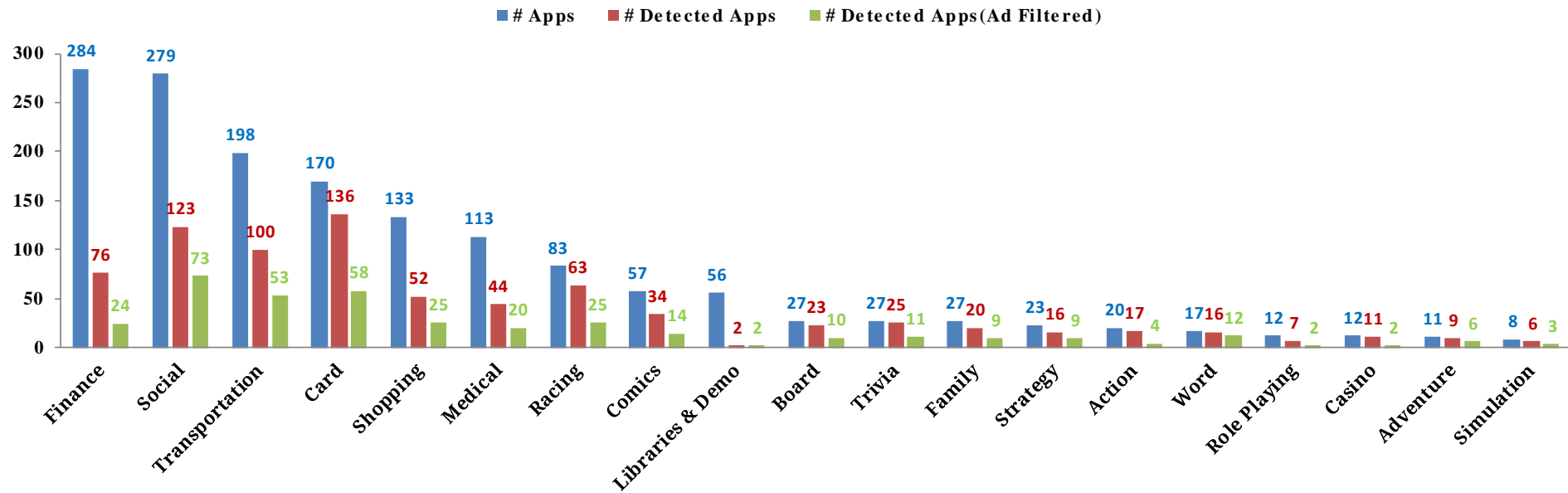
# 正常应用有近50%命中特征

- 49.996%的样本命中特征
- 分析发现大部分命中特征来源于广告模块
- 过滤广告模块，仍然有21.606%样本命中特征

# 基于正常应用的统计数据



# 基于正常应用的统计数据



# 反模拟器行为多来自于第三方库

- 大部分应用程序自身并没有反模拟器行为，其模拟器检测部分代码来自于以下几类
  - 广告模块: Google Ad, Millennial Media, etc
  - 社交类库: Facebook, Twitter, etc
  - 支付类库: PayPal, Amazaon, etc
  - 视频类库: Youtube, etc
  - 游戏引擎: LGame, etc
  - 其他第三方库: SamSung S-Pen, Mozilla JavaScript, etc

# 模拟器检测方法-Google Ad

- 反射调用SystemProperties.get方法获取系统属性
- 比较模拟器对应的特征值检测模拟器

```
try
{
    Method localMethod = Class.forName("android.os.SystemProperties").getDeclaredMethod("get",
        new Class[] { String.class });
    localMethod.setAccessible(true);
    boolean bool = "1".equals(localMethod.invoke(null, new Object[] { "ro.kernel.qemu" }));
    return bool;
}
catch (Exception localException) {}
return false;
}
```

```
if ((str2 != null) && (!f()))
{
    Uri localUri = d(paramUri, Base64.encodeToString(c(arrayOfByte, str2.getBytes()), 11),
        "ms");
    return localUri;
}
```



# 模拟器检测方法-PayPal

- 调用TelephonyManager.getDeviceId获取设备Id
- 与字符串000000000000000000比较判断是否为模拟器

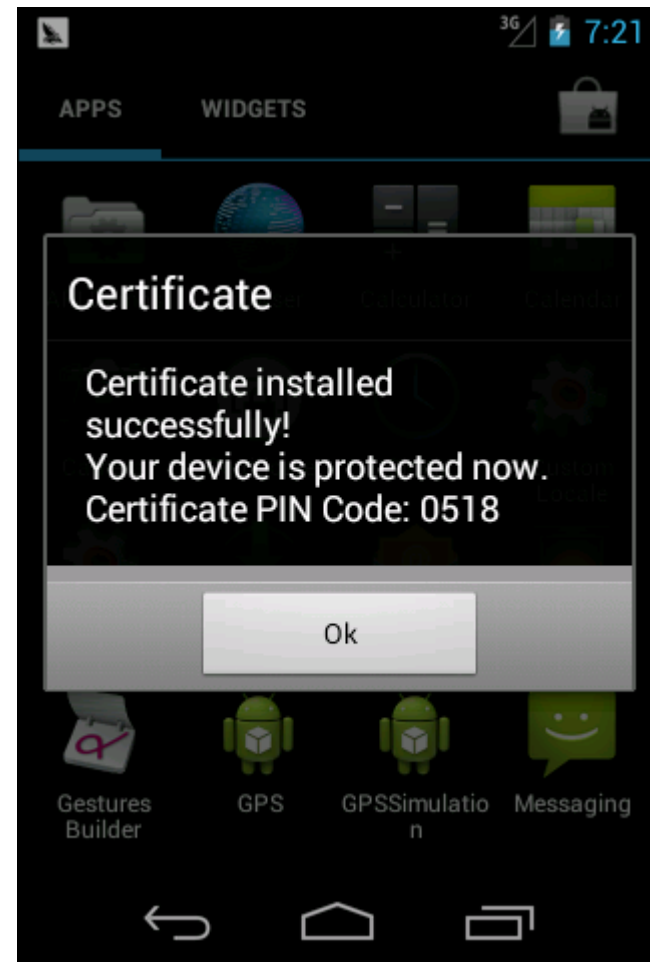
```
private static boolean x()
{
    String str = ((TelephonyManager)PayPal.getInstance().getParentContext().getSystemService("phone")).getDeviceId();
    if (str == null) {
        str = ((WifiManager)PayPal.getInstance().getParentContext().getSystemService("wifi")).
            getConnectionInfo().getMacAddress();
    }
    if ((PayPal.getInstance().getServer() == 1) && (str.equals("0000000000000000")))
    {
        Intent localIntent = new Intent(PayPalActivity.FATAL_ERROR).putExtra("FATAL_ERROR_ID", "-1").
            putExtra("FATAL_ERROR_MESSAGE", h.a("ANDROID_simulator_payment_block"));
        PayPalActivity.getInstance().paymentFailed((String)PayPalActivity._networkHandler.c("CorrelationId"),
            (String)PayPalActivity._networkHandler.c("PayKey"), "-1", h.a("ANDROID_simulator_payment_block"), false, true);
        PayPalActivity.getInstance().sendBroadcast(localIntent);
        return false;
    }
    return true;
}
```

# 恶意样本反模拟器行为低于正常样本

- 19.029% 的恶意样本命中特征
- 部分命中特征仍来源于广告模块，但远低于正常应用中所占比例
- 过滤广告模块，仍然有15.360% 恶意样本命中特征

# 模拟器检测方法-Pincer

- MD5
  - 2D66D7942148DE2D9F08EAB403921C89
- 收集设备信息并上传
  - 设备型号
  - 运营商信息
  - 电话号码
- 通过短信接受远程命令控制
  - start\_sms\_forwarding
  - start\_call\_blocking
  - send\_sms
- 检测运行环境
  - getDeviceId
  - getLine1Number



# 模拟器检测方法-Pincer

- com.security.cert.a.a.c

```
str1 = com.security.cert.b.b.b(paramContext);
str2 = com.security.cert.b.b.c(paramContext);
str3 = com.security.cert.b.b.d(paramContext);
if (str2 == null) {
    str2 = "";
}
if (str1 == null) {
    str1 = "";
}
if (str3 == null) {
    str3 = "";
}
if ((str3.toLowerCase().equals("android")) || (str1.equals("0000000000000000")) ||
    (str1.equals("012345678912345")) || (str2.equals("15555215554")) ||
    (Build.MODEL.toLowerCase().equals("sdk")) || (Build.MODEL.toLowerCase().equals("generic"))) {
    com.security.cert.b.a.a.a(paramContext, true);
}
```

通过比较DeviceId、Phone Number检测模拟器

# 模拟器检测方法-Pincer

- com.security.cert.b.b

```
public static String b(Context paramContext)
{
    return ((TelephonyManager)paramContext.getSystemService("phone")).getDeviceId();
}

public static String c(Context paramContext)
{
    return ((TelephonyManager)paramContext.getSystemService("phone")).getLine1Number();
}

public static String d(Context paramContext)
{
    return ((TelephonyManager)paramContext.getSystemService("phone")).getNetworkOperatorName();
}
```

获取DeviceId、Phone Number以及Network Operator

# 模拟器检测方法-Pincer

- com.security.cert.b.a.a

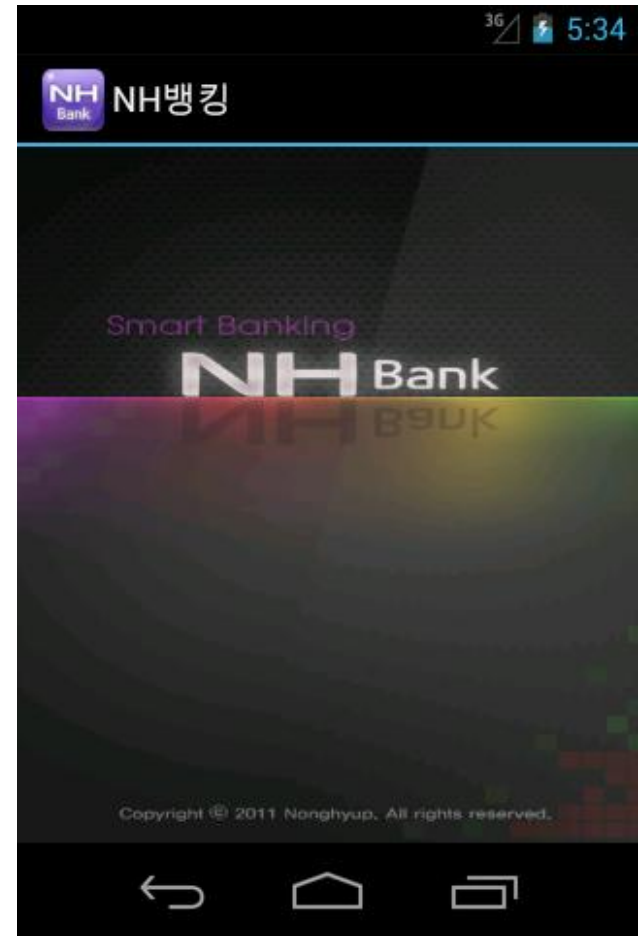
```
public static void a(Context paramContext, Class paramClass, boolean paramBoolean)
{
    if (paramBoolean) {}
    for (int i = 1;; i = 2)
    {
        ComponentName localComponentName = new ComponentName(paramContext, paramClass);
        paramContext.getPackageManager().setComponentEnabledSetting(localComponentName, i, 1);
        return;
    }
}

public static void a(Context paramContext, boolean paramBoolean)
{
    a(paramContext, OnBootReceiver.class, paramBoolean);
    a(paramContext, SmsReceiver.class, paramBoolean);
    a(paramContext, PhoneCallReceiver.class, paramBoolean);
    a(paramContext, SmsSentReceiver.class, paramBoolean);
}
```

Disable相关组件

# 模拟器检测方法-Wroba

- MD5
  - 0BDD5C05FE8B2C5D235CF54CAD21DC48
- 伪装成韩国NH银行应用
- 收集用户短信
- 发送短信
- 查询已安装应用程序
- 与远程服务器通信
- 判断是否为模拟器，核心后台服务在模拟器环境下不运行



# 模拟器检测方法-Wroba

- nh.four.MainService

```
public void onCreate()
{
    super.onCreate();
    BaseMessage localBaseMessage = new BaseMessage();
    if ((!localBaseMessage.isEmulator()) && (!localBaseMessage.isContant(this).booleanValue()))
    {
        SQLiteHelper.CreateSQLiteHelper(this);
        this.isrun = true;
        new Thread(this).start();
    }
}
```

- nh.four.BaseMessage
  - 通过android.os.Build获取系统属性

```
public boolean isEmulator()
{
    return (Build.MODEL.equals("sdk")) || (Build.MODEL.equals("google_sdk"));
}
```



# 模拟器检测的目的

- 兼容性检查
- 数据收集
- 根据模拟器/真机推送不同数据内容
- 软件崩溃时的日志记录
- 防止自动化行为，如发送垃圾信息
- 隐藏恶意行为
- ...

 <p><b>自动打招呼加好友</b> 全自动多账号循环打招呼推送信息，对附近的人推送新的产品及活动信息。软件同时支持多个微信账号添加固定号为好友。</p>	 <p><b>全自动群发信息</b> 支持单账号与客户进行聊天和交流，也支持多账号全自动群发助手群发信息，取代传统手机短信群发，节省费用，效率极高。</p>	 <p><b>批量关注公众号</b> 狂人微信营销软件支持全自动批量关注企业微信公众号，快速认证。</p>
 <p><b>建立营销数据库</b> 本微信营销系统支持全自动验证加微信或打招呼好友，建立属于自己的营销数据库，以便后期营销跟进和客户关系管理。</p>	 <p><b>漂流瓶群发</b> 支持多个账号批量扔漂流瓶和检漂流瓶，方便用户更多手法推广自己的产品或活动。</p>	 <p><b>批量分享</b> 狂人微信营销软件新增全自动批量分享公众号的广告给自己小号的好友功能，让营销效果更加明显。全网独创！</p>
 <p><b>自动识别验证码</b> 软件内置打码平台接口，自动识别验证码，适合软件24小时全自动挂机使用。</p>	 <p><b>后台静默操作</b> 狂人微信营销软件可完全后台执行任务，不控制鼠标，不锁定屏幕，不占用键盘，完全不影响电脑系统的其他工作。</p>	 <p><b>Bluestacks模拟器</b> 狂人微信营销软件是基于Bluestacks模拟器基础上自主研发的独立操作系统，软件运行顺畅、稳定。</p>

# 正常样本与恶意样本检测结果对比

- 正常应用中近50%的样本有反模拟器行为，远高于恶意样本
- 大量第三方库使用了模拟器检测
- 去除广告库的干扰，正常应用仍然比恶意应用的比例高
- 通过判断是否有模拟器检测行为，不能作为判定样本恶意性的主要依据之一
- 反模拟器技术应用普遍，直接影响应用程序在模拟器上运行时的行为，同时恶意样本会隐藏恶意行为

**研究反模拟器对抗技术意义重大！**

如何构造更真实的模拟器，欺骗  
应用程序其运行在真机环境？

# 反模拟器对抗的两种基本方法

- 源码修改
  - 更改字段值、API行为
  - 编译源码生成system.img
  - 加载system.img运行模拟器
- Runtime Hook
  - 运行时动态修改API调用行为
  - Java层Hook、Linux层Hook

# 源码修改缺点明显

- 优点

- 可直接修改硬编码字段、文件内容等
- 修改后的内容在Android系统最初的启动阶段就可以生效
- 不需要Root权限

- 缺点

- 下载、编译源码的软硬件需求高
- Android碎片化严重，不同的版本都需要进行源码修改
- 调试不便，编译时间较长
- 后期修改、维护麻烦
- 无法动态更改API行为

- A Linux or Mac system. It is also possible to build Android in a virtual machine on unsupported systems such as Windows. If you are running Linux in a virtual machine, you need at least 16GB of RAM/swap and 30GB or more of disk space in order to build the Android tree.
- 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.

# Runtime Hook轻量灵活

- 优点

- 开发成本低，软硬件需求不高
- 可针对不同的Android版
- 调试方便，类似于普通应用程序开发
- 轻量级，可以APK、动态链接库形式存在
- 部署方便
- 高度可定制
- 运行时可动态切换具体行为
- 后期修改维护方便

- 缺点

- Hook生效时间较晚
- 硬编码字段无法修改
- 需要Root权限

# Android Runtime Hook 框架

- 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

# Xposed基本原理

- 替换app\_process
- 将需要Hook的Java函数替换成JNI函数
- 所有需要Hook的函数首先由xposedCallHandler处理
- XposedCallHandler负责调用注册的beforeHookedMethod和afterHookedMethod

参考: MindMac-Xposed框架Java部分  
<http://bbs.pediy.com/showthread.php?t=181561>



# 基于Hook的模拟器隐藏

- 主要针对Android系统层、Linux系统层
  - 用户层数据和行为：可以通过数据构造，如增加联系人信息、短信、通话记录解决
  - 模拟器体系结构特征：模拟器的本质问题，同时实际应用中较少

# 针对Android系统层的Hook

# 基于TelephonyManager API的模拟器检测

- 对抗方法
  - Hook对应的API, 在afterHookedMethod函数中设置区别于模拟器值的返回结果
  - 可进一步返回随机且合理值(更不易被探测)

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



设置返回值

# 基于电池电量和状态特征的模拟器检测

- 获取当前电池电量和状态的方法
  - BatteryManager将电池信息通过Sticky Intent广播
  - 调用registerReceiver并传入值为null的广播接收器
  - registerReceiver返回的Intent实例中包含电池电量信息

```
IntentFilter intentFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);  
Intent batteryStatus = context.registerReceiver(null, intentFilter);  
int batteryLevel = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
```

# 基于电池电量和状态特征的模拟器检测

- 对抗方法
  - Hook android.content.Intent的getIntExtra等函数
  - 在afterHookedMethod函数中判断Intent的Action是否为ACTION\_BATTERY\_CHANGED,若是, 则根据参数修改返回值

```
protected void afterHookedMethod(MethodHookParam param) throws Throwable {  
    Intent intent = (Intent) param.thisObject;  
    if(intent != null && intent.getAction().  
        equals(Intent.ACTION_BATTERY_CHANGED)){  
        String key = (String) param.args[0];  
        if(key.equals(BatteryManager.EXTRA_LEVEL))  
            param.setResult(78);  
    }  
}
```

获得this对象

获得函数参数

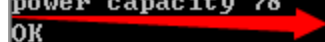
# 基于电池电量和状态特征的模拟器检测

- 对抗方法(另一种方法)
  - telnet localhost <emu-port>
  - power capacity 78

```
Android Console: type 'help' for a list of commands
OK
power help
allows to change battery and AC power status

available sub-commands:
  display      display battery and charger state
  ac           set AC charging state
  status       set battery status
  present      set battery present state
  health       set battery health state
  capacity     set battery capacity state

KO: bad sub-command
power capacity 78
OK
```



# 基于wifi、GPS等硬件特征的模拟器检测

- Wifi

- 模拟器上不存在wifi硬件
- 获取到的MAC地址为null
- 真机上即使在wifi未打开情况下仍然可以正确获取MAC地址

```
WifiManager wifiMgr = (WifiManager) getSystemService(Context.WIFI_SERVICE);  
WifiInfo wifiInfo = wifiMgr.getConnectionInfo();  
String macAddress = wifiInfo.getMacAddress();
```


- GPS

- 模拟器上不存在GPS设备
- LocationManager.getLastKnownLocation返回值为null

# 基于wifi、GPS等硬件特征的模拟器检测

- 对抗方法(wifi)
  - Hook android.net.wifi.WifiInfo的getMacAddress函数
  - 返回“真实”的MAC地址(可使用随机方法每次返回不同值)
- 对抗方法(GPS)
  - Hook LocationManager的getLastKnownLocation函数
  - 在afterHookedMethod中实例化Location对象，并设置坐标值
  - 返回实例化的Location对象

```
protected void afterHookedMethod(MethodHookParam param) throws Throwable {  
    Location location = new Location(LocationManager.GPS_PROVIDER);  
    location.setLatitude(10.03);  
    location.setLongitude(25.05);  
    param.setResult(location);  
}
```

 设置坐标值



# 基于Build字段的模拟器检测

- Build.Device等属于静态字段，在android.os.  
Build类加载时完成赋值
- Xposed等只提供对函数的Hook操作，无法对字段值进行动态修改
- Xposed等Hook生效时间要晚于Build类的加载

# 基于Build字段的模拟器检测

- 对抗方法
  - 修改Android源码，更改BRAND字段值
  - 解压system.img文件；修改build.prop;重新生成system.img
  - andwise, [Unpack/repack ext4 Android system images](#), xda developers

如何不修改源码达到Hook效果？

# 基于Build字段的模拟器检测

- 对抗方法(Smali Hook)
  - 反编译生成smali code
  - 将对Landroid/os/Build的引用修改为自定义的类
  - 重新编译、签名生成APK

```
.line 26
:cond_0
sget-object v0, Landroid/os/Build; -> BRAND:Ljava/lang/String;
.line 27
.local v0, brand:Ljava/lang/String;
const-string v1, "generic"
invoke-virtual {v0, v1}, Ljava/lang/String; -> equals(Ljava/lang/Object;)Z
move-result v1
if-eqz v1, :cond_1
```

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

```
.line 26
:cond_0
sget-object v0, Ldndroid/os/Build; -> BRAND:Ljava/lang/String;
.line 27
.local v0, brand:Ljava/lang/String;
const-string v1, "generic"
invoke-virtual {v0, v1}, Ljava/lang/String; -> equals(Ljava/lang/Object;)Z
move-result v1
if-eqz v1, :cond_1
```

部分应用会进行自校验，此方法会导致APK无法运行！

# 基于反射调用获取系统属性的模拟器检测

- `SystemProperties`提供对系统属性的访问和设置
- `SystemProperties`类默认没有导出
- 需要通过Java反射机制调用

# 基于反射调用获取系统属性的模拟器检测

- 对抗方法1

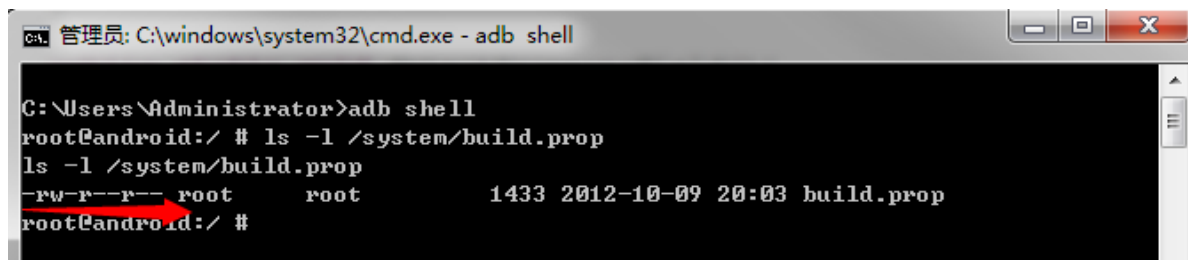
- Hook反射调用的目标函数，如SystemProperties.get函数
- 根据参数情况，修改返回值，注意返回值的合理性(系统在启动阶段同样会调用该函数，不合理的值会导致系统无法启动)

- 对抗方法2

- 直接Hook java.lang.reflect.Method类的invoke函数
- 根据反射调用的类名和方法名判断目标函数
- 进一步解析参数值，根据参数修改返回值

# 基于/system/build.prop检测模拟器

- 读取/system/build.prop文件检测模拟器
  - /system/build.prop文件记录了系统属性值
  - Android属性系统服务启动时会读取该文件内容并将属性值存入共享内存
  - 普通应用程序具有可读权限
  - 可读取该文件内容并判断是否包含特定字符串



```
C:\Users\Administrator>adb shell
root@android:/ # ls -l /system/build.prop
ls -l /system/build.prop
-rw-r--r-- root root 1433 2012-10-09 20:03 build.prop
root@android:/ #
```

```
ro.build.tags=test-keys
ro.product.model=sdk
ro.product.brand=generic
ro.product.name=sdk
ro.product.device=generic
```

# 基于/system/build.prop检测模拟器

- 对抗方法
  - Hook IO相关API, 在读取文件时, 篡改文件路径
  - 需要Hook所有IO类?
  - 所有IO操作最终调用libcore.io.IoBridge类中的API
  - Hook open函数, 在beforeHookedMethod中修改path参数

```
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";  
    }  
}
```

仅针对普通应用程序进行篡改

修改path参数值

# 基于Monkey事件模拟检测模拟器

- 大部分自动分析系统会采用Monkey生成随机事件，用于模拟人机交互，触发应用程序更多行为
- `ActivityManager.isUserAMonkey`函数可检测当前用户是否为Monkey，返回值为true表示存在Monkey事件模拟



# 基于Monkey事件模拟检测模拟器

- 对抗方法
  - Hook ActivityManager.isUserAMonkey函数
  - 在afterHookedMethod中将返回值设置为false

# 针对Linux系统层的Hook

# 通过驱动信息特征检测模拟器

- /proc/tty/drivers 驱动信息文件包含特征字符串 goldfish
- 普通应用程序具有可读权限
- 读取该驱动信息文件检测是否包含 goldfish

```
2!root@android:/ # cat /proc/tty/drivers
cat /proc/tty/drivers
/dev/tty          /dev/tty          5          0 system:/dev/tty
/dev/console     /dev/console     5          1 system:console
/dev/ptmx        /dev/ptmx        5          2 system
/dev/vc/0        /dev/vc/0        4          0 system:vtmaster
goldfish         /dev/ttyS        253 0-7 serial
pty_slave       /dev/pts         136 0-1048575 pty:slave
pty_master      /dev/ptm         128 0-1048575 pty:master
unknown         /dev/tty         4 1-63 console
root@android:/ # ls -l /proc/tty/drivers
ls -l /proc/tty/drivers
-r--r--r-- root  root          0 2014-06-02 04:05 drivers
```

# 通过驱动信息特征检测模拟器

- 对抗方法
  - 与基于/system/build.prop的模拟器检测对抗方法类似
  - Hook IoBridge的open函数后，篡改文件路径，重定向至伪造的驱动信息文件

# 通过设备文件特征检测模拟器

- 模拟器上存在/dev/socket/qemud、/dev/qemu\_pipe等表征模拟器的设备文件
- 通过判断这些文件的存在性检测模拟器

```
public static boolean isEmulator(){  
    File qemu_file = new File("/dev/qemu_pipe");  
    if(qemu_file.exists())  
        return true;  
    else  
        return false;  
}
```

# 通过设备文件特征检测模拟器

- 对抗方法
  - Hook java.io.File类的exists函数
  - 获取当前文件路径
  - 若当前文件路径为/dev/qemu\_pipe, 设置返回值为false

```
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);  
}
```

获取文件路径

检查文件路径, 设置返回值

# 通过执行shell命令检测模拟器

- `getprop`命令可以获取当前系统的属性信息
- 通过检查某些属性信息判断是否为模拟器，如 `ro.product.name`
- 调用 `Runtime.exec` 函数可以执行shell命令
- 读取shell执行结果判断是否存在特征字符

# 通过执行shell命令检测模拟器

- 对抗方法1
  - Runtime.exec函数调用后返回Process实例
  - 执行的shell结果通过Process.getInputStream获取
  - Runtime.exec最终会调用ProcessManager.exec，返回ProcessImpl实例
  - Hook ProcessImpl类的getInputStream函数，返回篡改后文件的inputStream对象

```
findAndHookMethod("java.lang.ProcessManager$ProcessImpl", lpparam.classLoader, "getInputStream",  
new XC_MethodHook() {  
    @Override  
    protected void beforeHookedMethod(MethodHookParam param)  
        throws Throwable {  
        InputStream inputStream = new FileInputStream("/data/local/tmp/fake-build.prop");  
        param.setResult(inputStream);  
    }  
});
```

内部类

before函数中进行参数修改，防止内存泄露

重定向InputStream



# 通过执行shell命令检测模拟器

- 对抗方法2
  - 替换Linux系统函数，如替换getprop命令
  - /system/core/toolbox/getprop.c
  - make toolbox
  - /out/target/product/generic/system/bin/toolbox
  - 使用重新编译生成的toolbox替换系统toolbox

```
-rwxr-xr-x root    shell      5580 2012-10-09 20:16 gdbjithelper
-rwxr-xr-x root    shell     186112 2012-10-09 20:08 gdbserver
lrwxr-xr-x root    shell           2012-10-09 20:16 getevent -> toolbox
lrwxr-xr-x root    shell           2012-10-09 20:16 getprop -> toolbox
-rwxr-xr-x root    shell      9512 2012-10-09 20:16 gzip
```

```
static void record_prop(const char* key, const char* name, void* opaque)
{
    strlist_t* list = opaque;
    char temp[PROP_VALUE_MAX + PROP_NAME_MAX + 16];
    if(strcmp(key, "ro.product.name") == 0){
        name = "google";
    }
    snprintf(temp, sizeof temp, "[%s]: [%s]", key, name);
    strlist_append_dup(list, temp);
}
```

# 通过Native Code检测模拟器

- NDK提供了\_\_system\_property\_get函数获取系统属性
- 需要包含sys/system\_properties.h头文件

```
JNIEXPORT jboolean JNICALL Java_com_emulator_detect_DetectUtils_detectGetpropDirectly
( JNIEnv* env, jobject this )
{
    int len;
    char buf[1024];
    len = __system_property_get("ro.product.name", buf);
    return (strcmp(buf, "sdk") == 0);
}
```

# 通过Native Code检测模拟器

- Xposed仅支持对Java层函数进行Hook，无法对Linux系统层函数的Hook操作
- adbi支持Native Code的Hook

# 通过Native Code检测模拟器

- 使用adbi Hook Linux系统函数
  - 指定入口点

```
void __attribute__((constructor)) my_init(void);
```

- 设置hook函数

```
void my_init(void)
{
    hook(&eph, getpid(), "libc.", "__system_property_get",
        my_system_property_get_arm, my_system_property_get);
}
```

目标函数所在的库

目标函数名称

ARM指令集下的hook函数

Thumb指令集下的hook函数

# 通过Native Code检测模拟器

- 使用adbi Hook Linux系统函数
  - Hook函数实现

参数与目标函数一致

```
int my_system_property_get(const char *name, char *value)
{
    if(strcmp(name, "ro.product.name") == 0){
        value = "google";
    }
    return 6;
}
```

Demo

# 总结

- 设计实现了反模拟器行为的检测
- 通过对真实样本的测试，我们发现
  - 反模拟器行为在真实世界中应用十分普遍
  - 大部分第三方库进行了模拟器环境检测
  - 正常样本中模拟器检测行为比例高于恶意样本
- 通过Hook解决针对Android、Linux系统层的模拟器检测
  - 开发容易
  - 部署方便
  - 定制灵活

# 谢谢!

- MindMac <mindmac.hu@gmail.com>
- Claud Xiao <secmobi@gmail.com>

HideAndroidEmulator:

<https://github.com/MindMac/HideAndroidEmulator>