

ADVANCED ANDROID APPLICATION SECURITY CASE STUDIES

VULNERABILITIES HIDING IN MILLIONS OF APPS

Flanker

KEEN TEAM

GeekPwn Shanghai, June 2015

KEEN

TABLE OF CONTENTS

1 INTRODUCTION

- About

2 ANDROID SECURITY BACKGROUND

- The Sandbox
- Component IPC

3 CONTEXT AND GOAL

4 CASE STUDIES

- API Misuse
- Capability Leak
- Dataflow vulnerability

5 SDKs SECURE?

- Umeng SDK
- JPush

6 CONCLUSION

KEEN

ABOUT ME

Security researcher at KEEN, pwner, coder. I'm currently focusing on mobile security, including:

- Application Security
- Android Framework and System Security
- Vulnerability Exploitation (Fun with buffer overruns!)
- Program Analysis

KEEN

ABOUT KEEN

- As audience of GeekPwn, I assume you should already know us. Shouldn't you? :)
- If not, Mr. Lu would like to talk a bit with you.

**KEEN**

OBJECTIVE OF THIS TALK

- Give a basic description of Android Security Mechanism
- Vulnerability Case Studies
- Another Case Study: 0day vulnerabilities in millions of apps

KEEN

ANDROID SECURITY BACKGROUND

APPLICATION SANDBOX

Coarse access control implemented in Linux Kernel

- File access control based on UID
 - Each app gets its own UID on installation (In general, I know you want to say sharedUID and system UID)
 - Access private files from one app to another is forbidden (If developers create their files correctly)
- Resource access control based on GID
 - Applications access network with **inet** gid
 - Applications access camera with **camera** gid
 - See more mappings at `/data/etc/platform.xml`

KEEN

ANDROID SECURITY BACKGROUND

APPLICATION SANDBOX

Fine-grained access control using permission, supported by Binder

- Application ask for permission upon installation
 - Some key permissions are signatureOrSystem, e.g. **INSTALL_PACKAGES**
 - Changed in M Preview with runtime enforcement
- Custom control using **enforceCallingPermission** and **getCallingUid**
 - Frequently seen in system_server
 - Kernel guarantees results from getCallingUid cannot be forged

KEEN

ANDROID SECURITY BACKGROUND

COMPONENT SECURITY

Inter-component communication is a key functionality in Android

- Components declared in AndroidManifest
 - Activity, Broadcast Receiver, Content Provider, Service
 - Can be exported or internal-only
 - Can be protected by permission
- Dynamic registered BroadcastReceiver
 - Implicitly exported
 - Can be protected by permission
- Access another application's un-exported component is considered sandbox escape
 - Un-exported components usually contains sensitive actions and do not sanitize input
 - Lead to serious security impact, will see it later

ANDROID SECURITY BACKGROUND

COMPONENT SECURITY

Local vs Remote attacks

- Service, Broadcast Receivers, Providers cannot be accessed remotely (in theory).
 - Of course practice go beyond theory sometimes
 - Some custom code by someone: in JavascriptInterface, use `parseUri`, etc
- Certain Activity can be invoked through URL
 - Use SEL to bypass restrictions on old browsers
 - Up-to-date only allows BROWSABLE category

KEEN

CONTEXT AND GOAL

GOAL

Attack another application from local or remote, to

- Denial of service
- Read/write private files/resources
- Abuse victim's permissions
- Affect victim's internal logic
- Steal sensitive information
- Code execution
- etc

CONTEXT

High-value applications are juicy targets, including

EN

API MISUSE - INSECURE RSA

BACKGROUND

RSA asymmetric algorithm, For encryption we have

$$c \equiv m^e \bmod n$$

For decryption we have

$$m \equiv c^d \bmod n$$

Where (e, n) is the public key, (d, n) is private one. c is encrypted text, m is cleartext.

MULTIPLE VULNERABILITIES EXIST IN TAOBAO LOGIN SDK

Taobao Login SDK use http channel to transport user's password when login. RSA encryption is adopted to defeat MITM sniffing. However multiple issues exist

- Affect all mobile clients of Alibaba
- Reported in 2014.5, fixed in late 2014
- Typical example of API misuse

MULTIPLE VULNERABILITIES EXIST IN TAOBAO LOGIN SDK

USE RSA THEN YOU'RE REALLY SECURE?

Taobao Login SDK use http channel to transport user's password when login, and use RSA to encrypt the traffic. However multiple issues exist

- The cipher suite is chosen without padding
 - Cipher.getInstance("RSA")
- e is chose as 3
 - Too small for a large n
- So we have exactly $c = m^3$, i.e. $m = c^{\frac{1}{3}}$
 - The password is cleartext for attacker to sniff even it's encrypted by RSA!

ALSO SOME GOOD LONG-LIVING EXAMPLES..

- Javascript addJsInterface code execution
- SharedPreferences and openFileOutput modes
- HTTPS setHostnameVerifier

SOMETIMES APIs HURT, WHOSE FAULT?

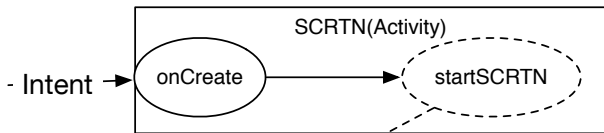
AND THE RECENT *unzip* DIRECTORY TRAVERSAL FROM NOWSECURE

- Samsung code execution via MITM
- Directory traversal in zip entry: ../../../../pwned.dex
 - MITM Swift keyboard update zip via HTTP link, insecure as we know :(
 - The app blindly unzip the file using ZipInputStream, extracting all files
 - Overwrite odex file, inject code, trigger execution
 - Get system shell
- Who's to blame?

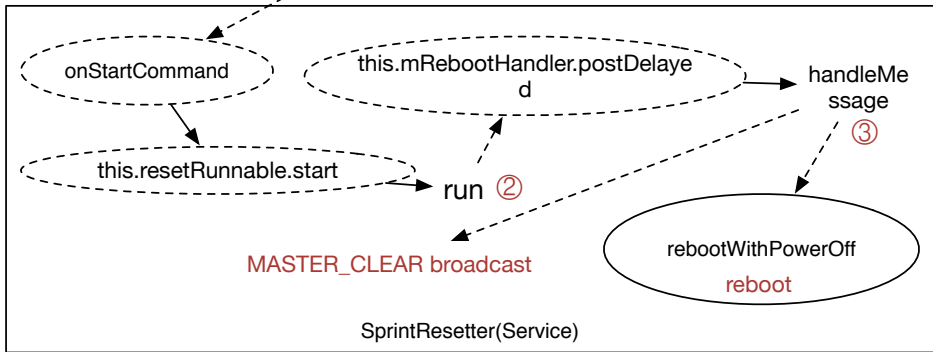
ADVERSARY FORCES VICTIM APP TO PERFORM PRIVILEGED ACTION VIA IPC

- Perform action on behave of victim app
- Bypass permission sandbox
- Especially useful when bug exists in system app

■ Nexus 5 local DOS



Inter-component ①



TAINTED DATA FLOWIn/SENSITIVE DATA FLOWOut

TAINTED DATA FLOWIn FROM INTENT

- Dataflow from incoming attacker controlled Intent
- To sensitive API call

TAINTED DATA FLOWIn/SENSITIVE DATA FLOWOut

SOGOU INPUT METHOD RCE

- Triggered via Intent scheme
- Code execution in Input Method can lead to password leak -
Your keyboard is mine

dummyMain (for exported components)

...(omit)

```
virtualinvoke $r149.<sogou.mobile.explorer.hotwords.minibrowser.MiniWebViewActivity:  
void onCreate(android.os.Bundle)>($r150);  
...(omit)
```

MiniWebActivity.onCreate



```
$r3 = virtualinvoke $r0.<sogou.mobile.explorer.hotwords.minibrowser.MiniWebViewActivity:  
android.content.Intent getIntent()>()
```

```
$r0.<sogou.mobile.explorer.hotwords.minibrowser.MiniWebViewActivity: void  
processExtraData()>()
```

←← attacker controlled data

arg1: intent(\$r3)

MiniWebActivity.processExtraData

```
$r3 = virtualinvoke $r2.<java.lang.String: android.content.Intent getDataString()>()  
<com.tencent.smtt.sdk.WebView: void loadUrl($r3)>
```

arg1: String(\$r3)

```
$r2.<com.tencent.smtt.sdk.WebView$SystemWebView: void loadUrl(java.lang.String)>($r1)
```

(CHA needed)



```
import java.lang.reflect.Method;  
  
class WebView$SystemWebView extends WebView {  
    public WebView$SystemWebView(com.tencent.smtt.s  
        com.tencent.smtt.sdk.WebView.this = arg4;  
        super(arg5);  
    }  
}
```

TAINTED DATA FLOWIn/SENSITIVE DATA FLOWOut

SAMSUNG KNOX RCE

- Triggered via URL scheme
- Flow through **URLRequest**
- Finally reaches **PackageManager.installPackage**

Attacker provided smdm:// scheme



```
updateURL = intent.getQueryParameter("update_url")
```

```
.Ui.LaunchActivity
```

Pop up dialog

Fetch "Update" Package

```
PackageManager.installPackage  
UpdateThread
```

Samsung Knox MDM

Fetch Update

repeatedly to force user
confirm

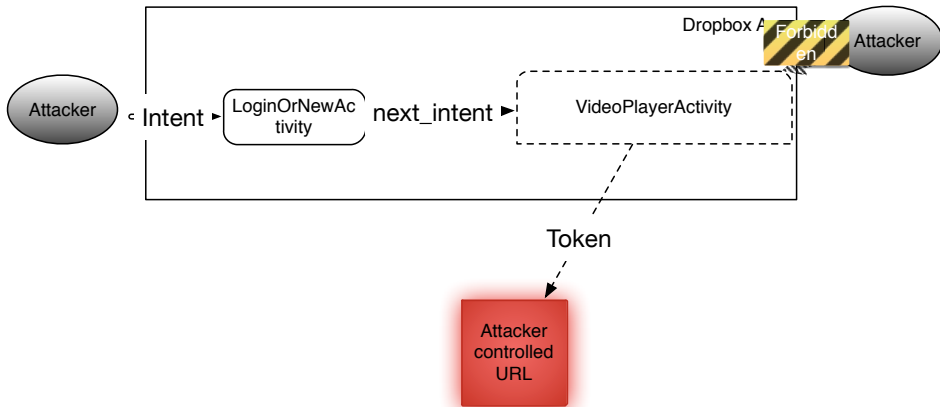
Package

Attacker
Server

TAINTED DATA FLOWIn/SENSITIVE DATA FLOWOut

Dropbox Next-Intent Attack

KEEN



VULNERABILITY HIDDEN IN MILLIONS OF APPS

DEVELOPERS LOVES SDKs

- Include them as blackbox JAR/SO



- Rich functionalities!
- Message pushing, activating app, URL pushing
- Millions of apps use them

KEEN

ATTACK SCENARIO

However, design flaws in those SDK allows an zero-permission attacking app can

- Fake notification message
- Start arbitrary activity - **bypassing sandbox**
- Private file stolen
- Code execution!

in arbitrary target app bundled with vulnerable SDK via IPC.

KEEN

CASE STUDY: VULNERABILITIES IN PUSH SDKs

Umeng SDK: one of the most famous push SDK in China



友盟消息推送，帮助开发者建立于用户直接沟通的渠道。将APP的内容推送给用户，让用户第一时间获取到相关信息，有效提升用户活跃度和忠诚度。



do you know embedding it will break your app's sandbox?

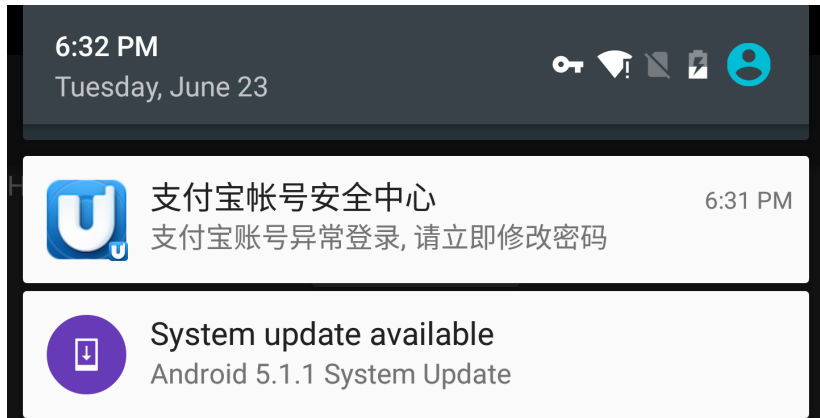
KEEN

OTHER SDKs ARE ALSO VULNERABLE

- Xg-Push SDK
- JPush
- Even earlier versions of GCM

FORGE NOTIFICATION

- A zero permission attacking app can forge victim's notification



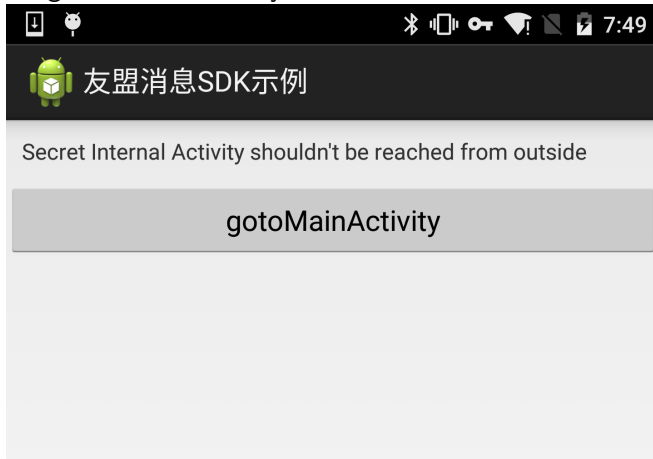
- A zero permission attacking app can start arbitrary activity of victim, including **unexported** ones.
- Use official SDK-sample as example

```
<activity
    android:name="com.umeng.message.example.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name="com.umeng.message.example.TestActivity" />
```

SAMPLE TARGET

Target internal activity

**KEEN**

DEMO VIDEO

- Forge notification of App containing XgPush and UmengPush SDK
- Start private activity of App containing XgPush and UmengPush SDK

KEEN

STATUS

- Reported and under fix procedure
- Will publish detail after fix

KEEN

PRIVATE FILE THEFT AND CODE EXECUTION

- Vulnerability in JPush earlier than 1.7.2, fixed in 1.7.3
- Affect 100k apps? (estimated)
- Developers are recommended to upgrade immediately

VULNERABILITY DEMONSTRATION

```

..de/bul... [java (java)] /tmp (zsh) ...Mac... sh (sh) ...roid_3... ..Fuck2... ..(zsh) python... ..(zsh) ...1.0.1... 3D
c nc
system 301 178 592532 52568 ffffffff 00000000 S system_server
system 310 1 59372 6860 ffffffff 00000000 S /system/bin/surfaceflinger
root 323 2 0 0 ffffffff 00000000 S flush-8:16
wifi 386 1 9364 2248 ffffffff 00000000 S /system/bin/wpa_supplicant
u0_a38 392 178 521040 46524 ffffffff 00000000 S com.android.systemui Initial-scaled-0. max
u0_a52 415 178 515800 32776 ffffffff 00000000 S com.alipay.pushservice
dhcp 450 1 6396 1220 ffffffff 00000000 S /system/bin/dhpcd
u0_a18 480 178 492016 30584 ffffffff 00000000 S com.android.inputmethod.latin
radio 585 178 505816 29380 ffffffff 00000000 S com.android.phone
u0_a19 535 178 516496 45932 ffffffff 00000000 S com.android.launcher
u0_a43 603 178 488884 23340 ffffffff 00000000 S com.android.smspush
u0_a0 621 178 519308 36736 ffffffff 00000000 S android.process.acore
u0_a11 668 178 491364 26872 ffffffff 00000000 S com.android.deskclock
u0_a52 682 178 547824 49496 ffffffff 00000000 S com.eg.android.AlipayGphone .getMethod("get
u0_a7 764 178 489628 26648 ffffffff 00000000 S com.android.providers.calendar
u0_a12 779 178 492452 28746 ffffffff 00000000 S android.process.media
u0_a13 804 178 499700 23524 ffffffff 00000000 S com.android.email
u0_a41 878 178 489020 23584 ffffffff 00000000 S com.android.voicerecorder .get
u0_a6 907 178 497008 25836 ffffffff 00000000 S com.android.calendar
u0_a54 931 178 998976 47168 ffffffff 00000000 S com.icsclub.android
u0_a10 1788 178 490028 24296 ffffffff 00000000 S com.android.defcontainer .getActivityClassname
u0_a26 1806 178 489000 23584 ffffffff 00000000 S com.android.musicfx
u0_a31 1823 178 488900 23292 ffffffff 00000000 S com.svox.pico
u0_a37 1838 178 490368 24056 ffffffff 00000000 S com.noshufou.android.su
u0_a34 1854 178 490264 24392 ffffffff 00000000 S com.android.autotextsearchbox
root 2432 56 6384 1196 ffffffff 00000000 S /system/bin/sh
root 2434 2432 7400 2232 ffffffff 00000000 S logcat
system 2494 178 488980 25472 ffffffff 00000000 S com.android.keychain
root 4587 56 6396 1420 ffffffff 00000000 S /system/bin/sh
root 6423 56 6396 1388 ffffffff 00000000 S /system/bin/sh
u0_a53 7838 178 1000076 54948 ffffffff b75bd587 S com.oayou.android
u0_a55 7978 178 495276 29588 ffffffff 00000000 S com.example.misp
u0_a53 7998 7838 6388 1348 c0101ff8 b7df50e S /system/bin/sh
root 8007 56 6624 1636 ffffffff 00000000 S logcat
u0_a53 8213 7998 6444 1216 00000000 b7df50e R ps
id
gid=10053(u0_a53) gid=10053(u0_a53) groups=1006(camera),1015(sdcard_rw),1028(sdcard_r),3003(inet)
ps | grep u0_a53
u0_a53 7838 178 1000076 54948 ffffffff b75bd587 S com android
u0_a53 7998 7838 6388 1348 c0101ff8 b7df50e S /system/bin/sh
u0_a53 8256 7998 6444 1216 00000000 b7df50e R ps
u0_a53 8257 7998 7160 1200 c010afc56 b7df50e S grep

```

VULNERABILITY ANALYSIS

The SDK adds exported `cn.jpish.android.ui.PushActivity` in `AndroidManifest`

```
protected void onCreate(Bundle arg5) {
    int i = 0x400;
    x.c();
    super.onCreate(arg5);
    if(this.getIntent() != null) {
        Intent intent = this.getIntent();
        this.jpishdata = intent.getSerializableExtra(PushActivity.z[1]);
        if(this.jpishdata != null && this.jpishdata.z == 2) {
            this.jpishdata.z = 1;
            this.jpishdata.p = 3;
            m.a(((Context)this), this.jpishdata);
        }

        this.requestWindowFeature(1);
        if(this.jpishdata.q) {
            Window window = this.getWindow();
            window.setFlags(i, i);
        }

        intent = this.getIntent();
        this.processData(intent);
    }
    else {
```

VULNERABILITY ANALYSIS

```
private void processData(Intent arg9) {  
    int i = 4;  
    int i1 = 2;  
    this.jpushdata = arg9.getSerializableExtra(PushActivity.z[1]);  
    switch(this.jpushdata.o) {  
        case 0: {  
            goto label_44;  
        }  
        case 1: {  
            goto label_36;  
        }  
        case 2: {  
            return;  
        }  
    }  
    //omit  
    goto label_34;  
label_36:  
    this.msghandler.removeMessages(i1);  
    this.msghandler.removeMessages(3);  
    this.msghandler.sendMessageDelayed(i1, 1);  
    return;  
label_44:  
    this.msghandler.removeMessages(i);  
    this.msghandler.removeMessages(5);  
    this.msghandler.sendMessageDelayed(i, 1); //target path
```

VULNERABILITY ANALYSIS

```
public final void handleMessage(Message arg8) {
    Handler handler;
    int i = 5;
    int i1 = 3;
    long l = 0x3E8;
    switch(arg8.what) {
        //case 0,2,6 omit
        case 4: {
            this.a.setRequestedOrientation(1);
            handler = PushActivity.getHandler(this.a);
            handler.removeMessages(4);
            handler = PushActivity.getHandler(this.a);
            handler.removeMessages(i);
            this.sendEmptyMessageDelayed(i, l); //notice this line send out msg of 5
            break;
        }
        case 5: {
            PushActivity.processJpushData(this.a); //key path
            break;
        }
        //omit
    }
}
```

VULNERABILITY ANALYSIS

```

static void processJpushData(PushActivity arg8) {
    //omit
    JPushData1 pushdata = arg8.jpushdata;
    String string = ((s)pushdata).a;
    if(((s)pushdata).W == 0) {
        if(p.a(string)) {
            String string1 = ((s)pushdata).ab;
            if(((s)pushdata).q) {
                arg8.d = new JsInterfaceWebView1(((Context)arg8), pushdata);
                JsInterfaceWebView1 a = arg8.d;
                if(!TextUtils.isEmpty(((CharSequence)string1))) {
                    string2 = string1.replace(PushActivity.z[i], "");
                    file = new File(string2);
                    if(file.exists()) {
                        arg8.d.loadURL(string1); //arbitrary load from file
                        goto label_37;
                    }
                }

                arg8.d.loadURL(string); //arbitrary URL load with addJsInterface enabled, game
                over
            }
        }
    }
}

```

- The issue is fixed at 2015.2
- However apps distributed at 2015.5 still contain the old vulnerable SDK

TO SDK DEVELOPERS

- Be responsible
- Offer SDK upgrade channel for App developers and publish security advisories in time

KEEN

TO APP DEVELOPERS

- Perform assessment first when using blackbox SDK
- Upgrade your app more often

STATUS

- Umeng SDK: reported
- XgPush SDK: reported
- JPush SDK: fixed

KEEN

CREDITS

- Jashui Wang (@moonflow)
- Shi Wu (@rock509)
- Some referenced disclosed vulnerabilities belong to their respective owners

KEEN

THANKS!

- Any questions?

KEEN