Advanced Android Application Security Case Studies

Vulnerabilities hiding in millons of apps

Flanker

KEEN TEAM

GeekPwn Shanghai, June 2015

KEEN

Table of Contents

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? :)

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)

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

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

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

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

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).

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

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

KEEN

Context and Goal

Context

High-value applications are juicy targets, including

- System Application with critical permissions
  - *From zero permission to system-level backdoor in Samsung phones* via *QuarksLab*
- Financial/Input/Widely-used/Sensitive Applications
- Widely used SDKs
  - We'll see later

Context and Goal

Attack surfaces

Rank by Access Vector and Exploitability Metrics

- Remote attack
- Local App
- MITM
- adb or physical access

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?

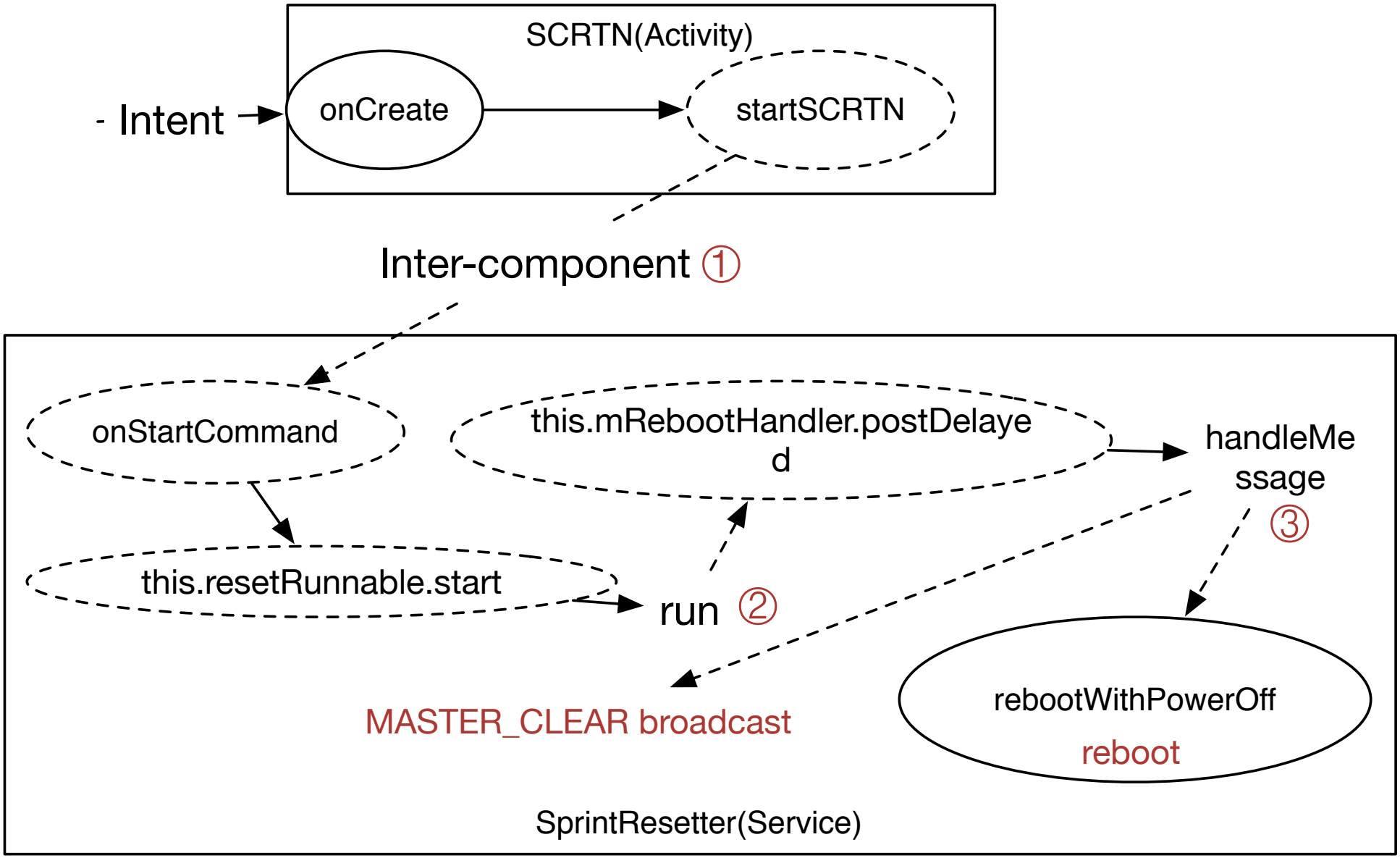
And the recent *unzip* directory traversal from NowSecure

- 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


SCRTN(Activity)
- Intent
onCreate
startSCRTN
Inter-component ①
onStartCommand
this.mRebootHandler.postDelaye
d
handleMe
ssage
③
this.resetRunnable.start
run ②
MASTER_CLEAR broadcast
rebootWithPowerOff
reboot
SprintResetter(Service)

Tainted data flowIn/Sensitive data flowOut

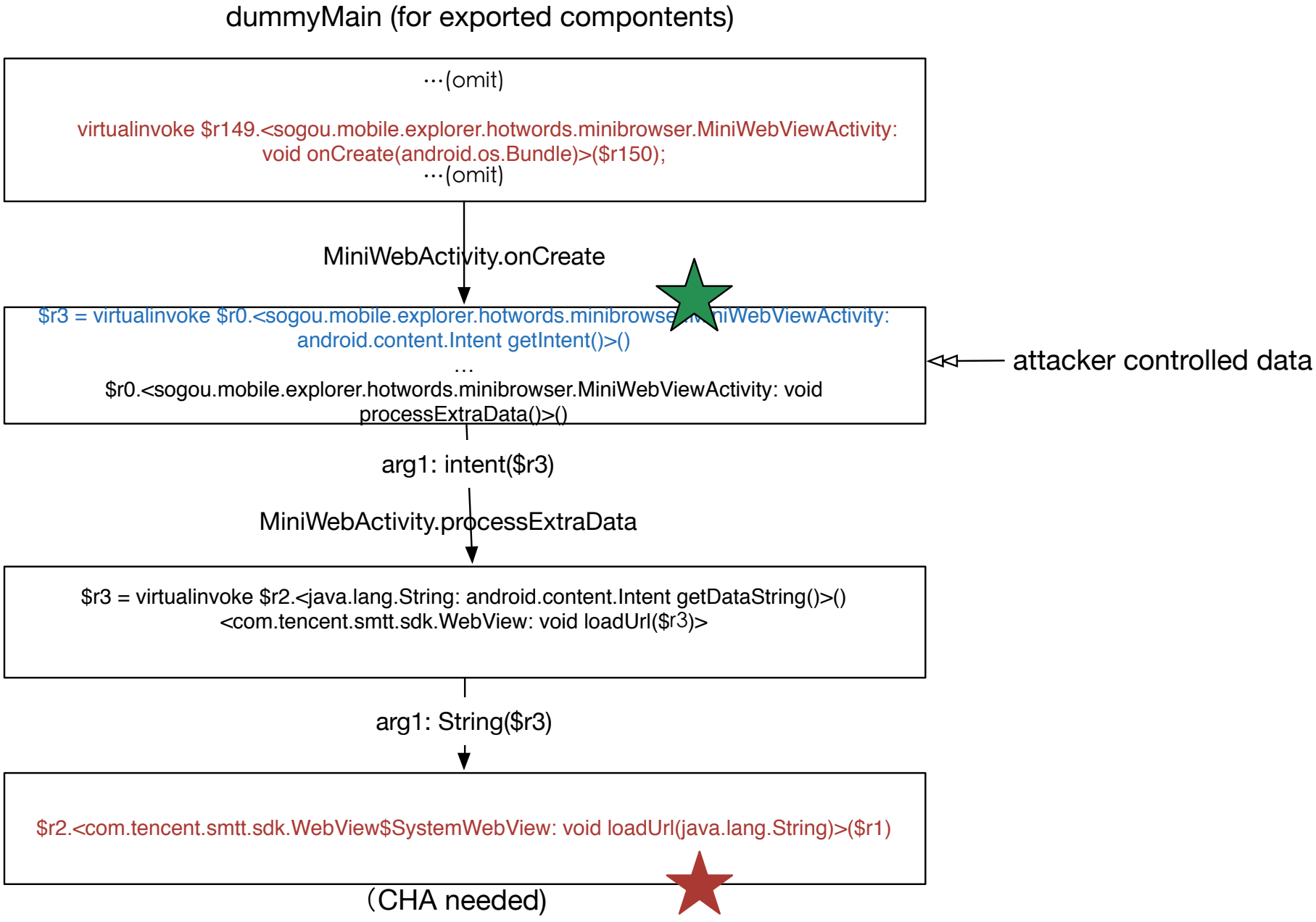
Tainted data flowin from Intent

- 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



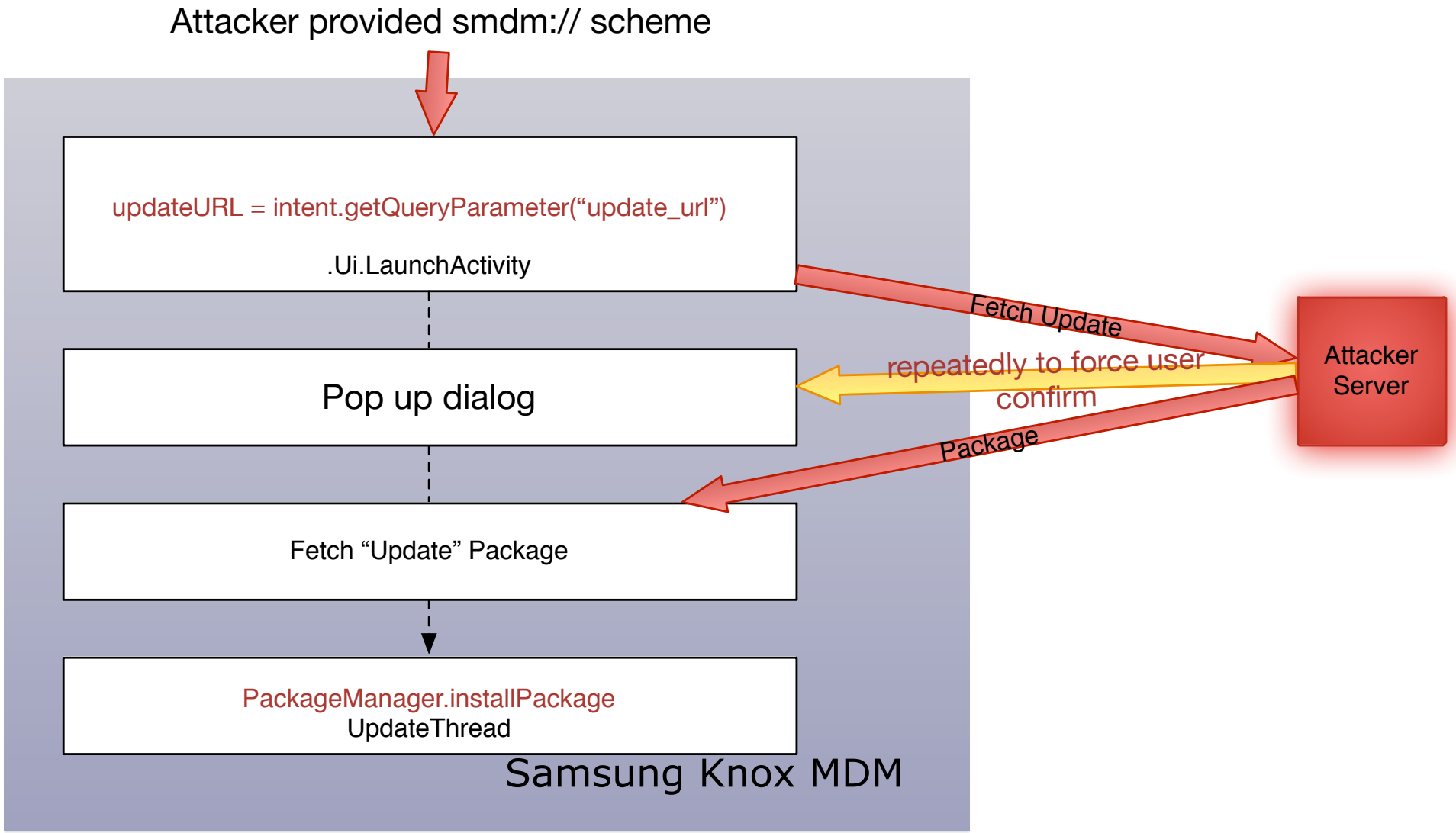

```
import java.lang.reflect.Method;

class WebView$SystemWebView extends WebView {
    public WebView$SystemWebView(com.tencent.smtt.s
        com.tencent.smtt.sdk.WebView.this = arg4;
```

Tainted data flowIn/Sensitive data flowOut

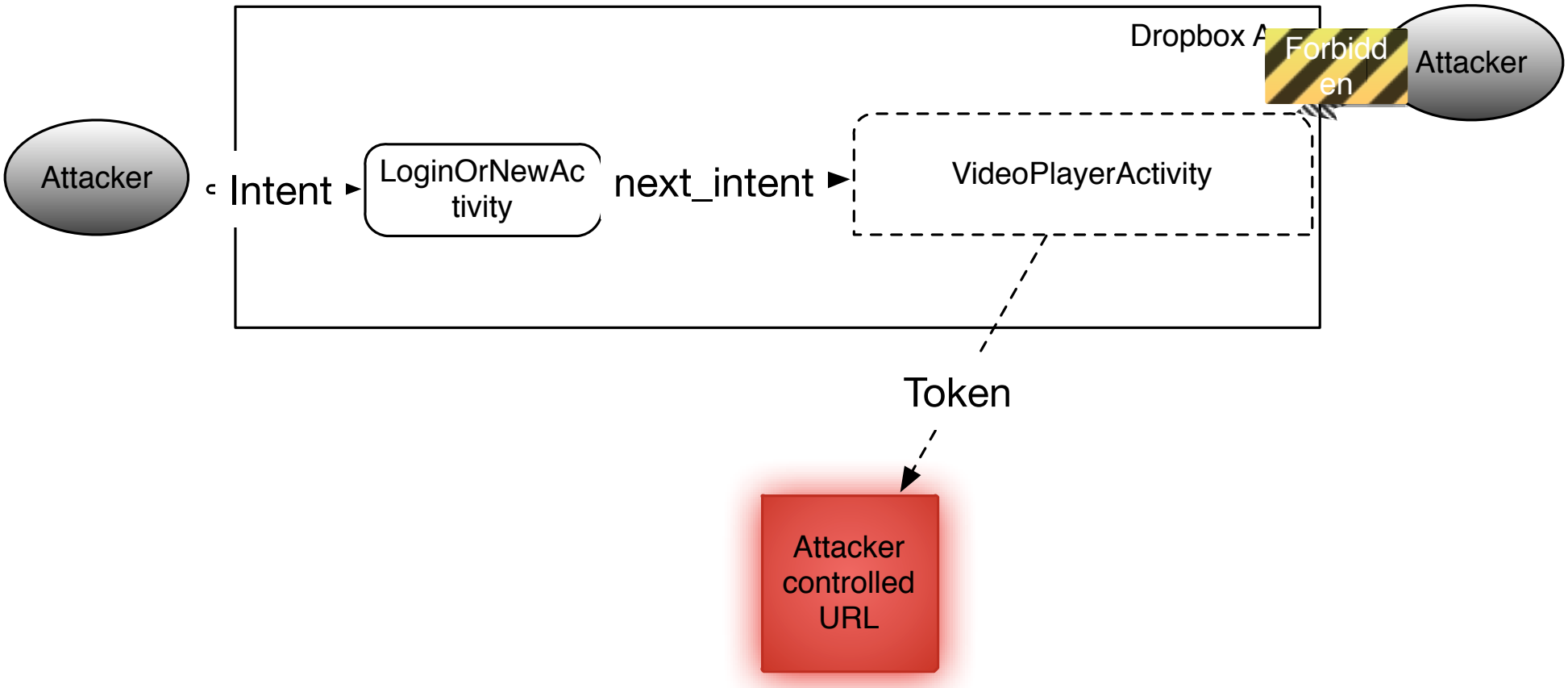
Samsung KNOX RCE

- 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

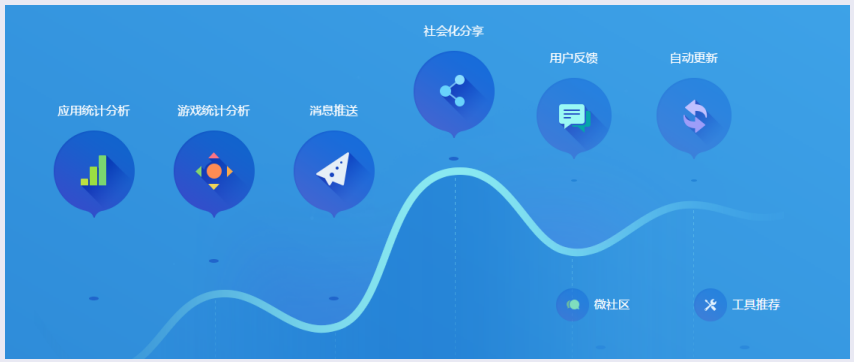
Dropbox Next-Intent Attack


Dropbox A
Forbidd
en
Attacker
Attacker
Intent
LoginOrNewAc
tivity
next_intent
VideoPlayerActivity
Token
Attacker
controlled
URL

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

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
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

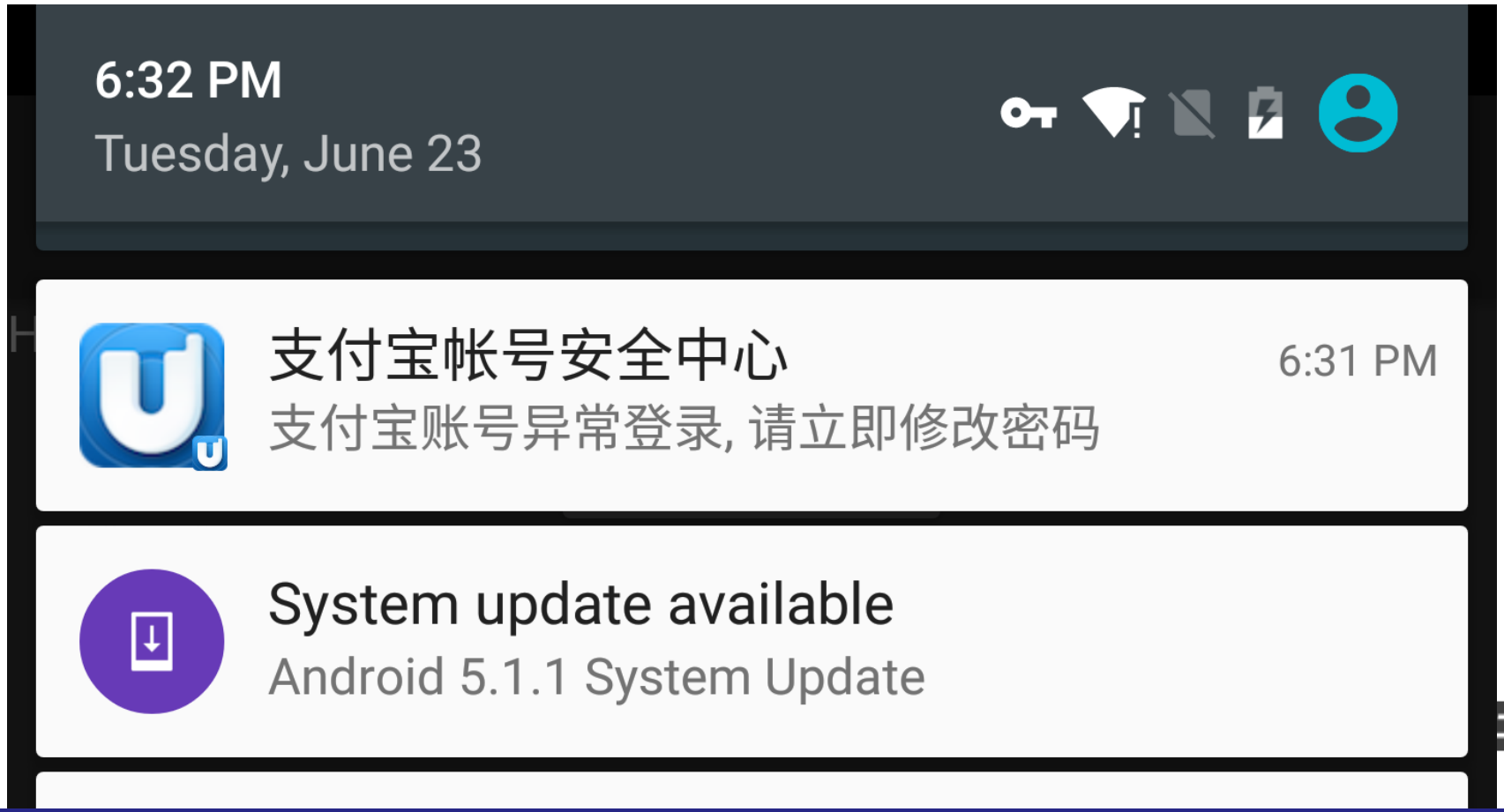
- Xg-Push SDK
- JPush
- Even earlier versions of GCM

Forge notification

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

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

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

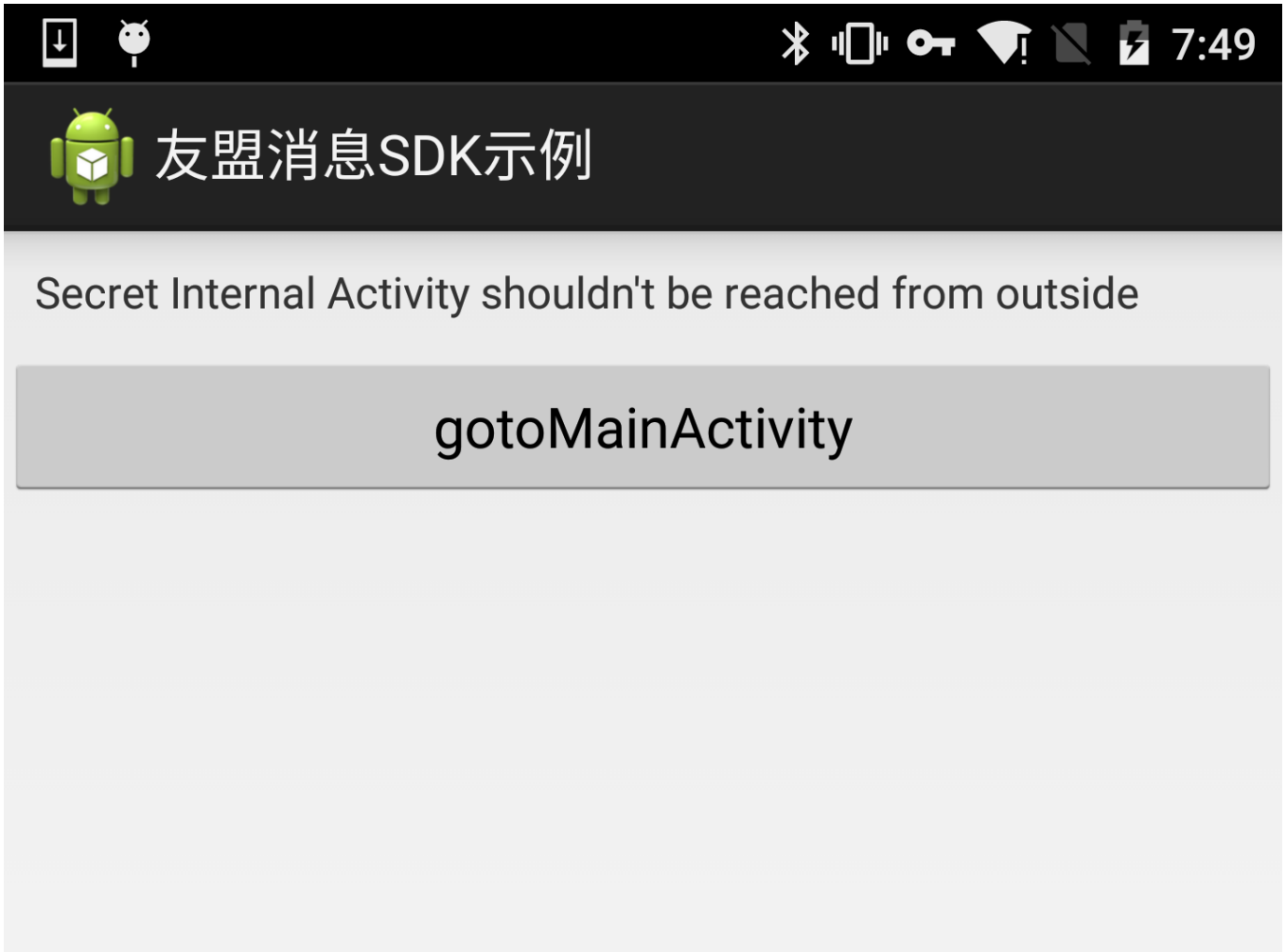
```
<activity
    android:name="com.umeng.message.example.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

KEEN

Sample target

Target internal activity



Demo video

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

Status

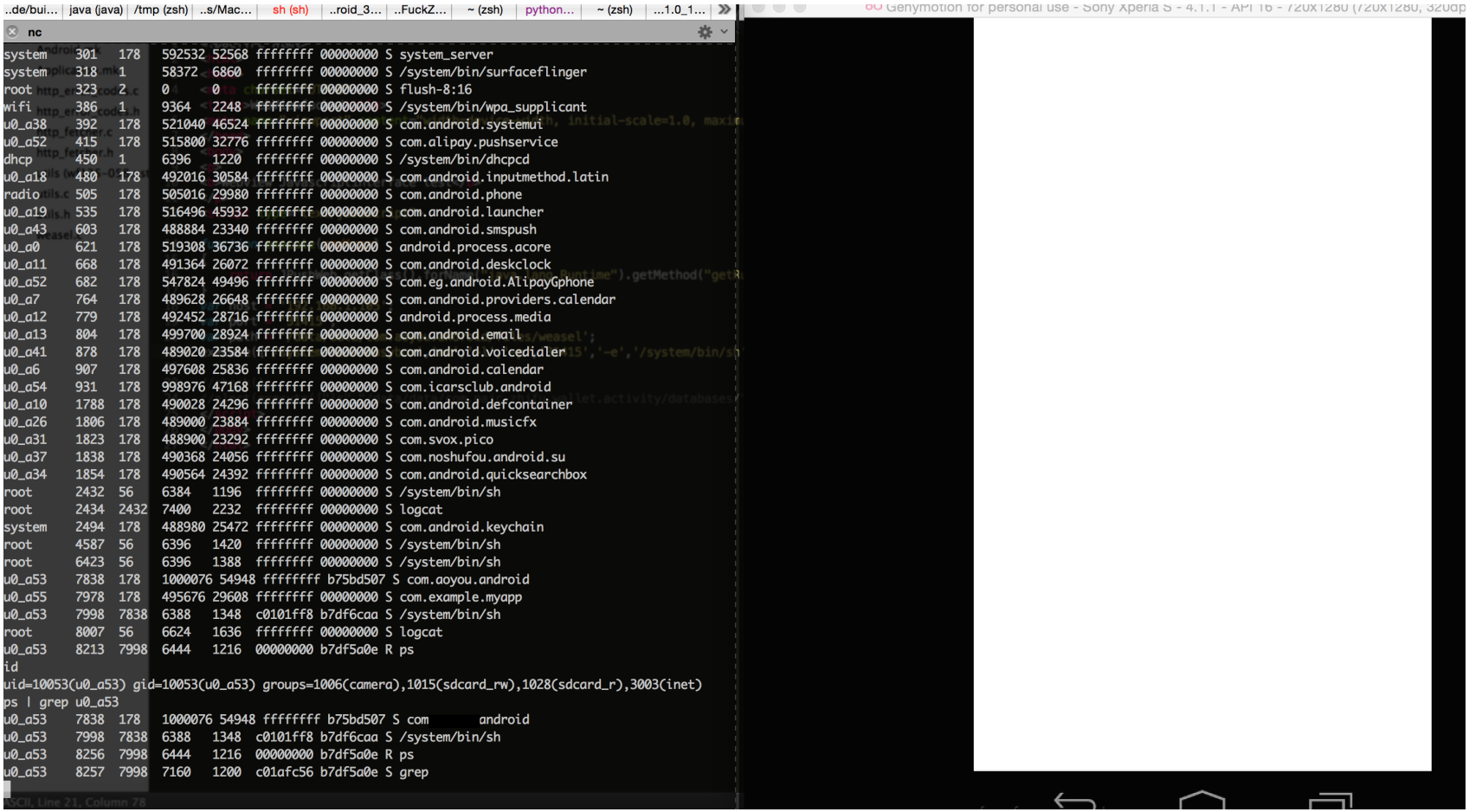
- Reported and under fix procedure
- Will publish detail after fix

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

KEEN

Vulnerability Demonstration



KEEN

Figure : Shell from app bundled with vulnerable SDK

Vulnerability Analysis

The SDK adds exported cn.jpush.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.jpushdata = intent.getSerializableExtra(PushActivity.z[1]);
        if(this.jpushdata != null && this.jpushdata.z == 2) {
            this.jpushdata.z = 1;
            this.jpushdata.p = 3;
            m.a(((Context)this), this.jpushdata);
        }

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

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

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.sendEmptyMessageDelayed(i1, l);
    return;
label_44:
    this.msghandler.removeMessages(i);
    this.msghandler.removeMessages(5);
    this.msghandler.sendEmptyMessageDelayed(i, l);//target path
    return;
```

KEEN

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
    }
}
```

KEEN

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

Credits

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

KEEN

Thanks!

- Any questions?