**Q/NBY**

**努比亚技术有限公司企业标准**

**（软件研发模板）**

**Q/NBY XX XX.XXX-X.X**

**Android安全编码规范**

**2017-03-15发布 2017-03-25实施**

**努 比 亚 技 术 有 限 公 司**

目 录

[**1** **目的** 4](#_Toc478229875)

[**2** **范围** 4](#_Toc478229876)

[**3** **术语、定义** 4](#_Toc478229877)

[**4** **组件安全** 4](#_Toc478229878)

[4.1 Activity安全 4](#_Toc478229879)

[4.2 Service安全 5](#_Toc478229880)

[4.3 ContentProvider安全 6](#_Toc478229881)

[4.4 Broadcast Receiver安全 8](#_Toc478229882)

[**5** **存储安全** 9](#_Toc478229883)

[5.1 禁止将应用的敏感数据保存在外置SD上 9](#_Toc478229884)

[5.2 应用创建文件时，禁止使用MODE\_WORLD\_READABLE、MODE\_WORLD\_WRITABLE模式 10](#_Toc478229885)

[5.3 通过NDK创建的文件时，必须指定文件权限 10](#_Toc478229886)

[5.4 Log中禁止携带敏感数据 10](#_Toc478229887)

[5.5 用户隐私数据必须加密保存或传输 11](#_Toc478229888)

[**6** **WebView安全** 11](#_Toc478229889)

[6.1 如果应用程序不直接使用 WebView 中的 JavaScript，请禁止调用 setJavaScriptEnable(). 11](#_Toc478229890)

[6.2 不允许WebView通过file://访问敏感的本地资源 11](#_Toc478229891)

[**7** **通信安全** 12](#_Toc478229892)

[7.1 敏感数据必须使用HTTPS传输 12](#_Toc478229893)

[7.2 HTTPS通信时，必须对SSL证书进行强校验 12](#_Toc478229894)

[7.3 禁止使用 localhost 网络端口处理敏感数据。 12](#_Toc478229895)

[7.4 禁止Socket绑定INADDR\_ANY。 12](#_Toc478229896)

**修改记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本号 | 修改章节 | 修改内容简述 | 修改人 | 修订日期 | 审批人 |
|  |  | 无 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 注1：每次更改归档文件时，需填写此表。  注2：文件第一次归档时，“修改内容简述”栏写“无”，其它栏空。 | | | | | |

# **目的**

本文档旨在规范软件开发人员的安全编码。

# **范围**

本规范适用于Android平台安全编码。

# **术语、定义**

反例：不合规的样例

正例：合规解决方案

注：对样例代码的解读

# **组件安全**

## Activity安全

### 必须声明android:exported的值，不允许使用默认值

默认值：如果包含有intent-filter 默认值为true; 没有intent-filter默认值为false

反例：

<activity

android:name=".MainActivity"

android:label="@string/app\_name"

android:uiOptions="splitActionBarWhenNarrow">

</activity>

正例：

<activity

android:name=".MainActivity"

android:label="@string/app\_name"

android:uiOptions="splitActionBarWhenNarrow"

android:exported="false">

</activity>

### 声明为公开可访问的Activiy，必须设置访问权限，权限级别必须是signature或signatureOrSystem

反例：

<activity

android:name=".MainActivity"

android:label="@string/app\_name"

android:uiOptions="splitActionBarWhenNarrow"

android:exported="true">

<intent-filter>

<action android:name="cn.nubia.intent.action.ENTRY" />

</intent-filter>

</activity>

正例：

<permission

android:name="cn.nubia.permission.START\_MY\_ACTIVITY"

android:protectionLevel="signatureOrSystem" >

</permission>

<activity

android:name=".MainActivity"

android:label="@string/app\_name"

android:uiOptions="splitActionBarWhenNarrow"

android:exported="true"

android:permission="cn.nubia.permission.START\_MY\_ACTIVITY">

<intent-filter>

<action android:name="cn.nubia.intent.action.ENTRY" />

</intent-filter>

</activity>

**signature：**：只有当申请权限的应用程序的数字签名与声明此权限的应用程序的数字签名相同时（如果是申请系统权限，则需要与系统签名相同），才能将权限授给它；

**signatureOrSystem**：签名相同，或者申请权限的应用为系统应用（在system image中）。

## Service安全

### 必须声明android:exported的值，不允许使用默认值

反例：

<service

android:name=".MyService"">

</service>

正例：

<service

android:name=".MyService"

android:exported="false">

</service>

### 声明为公开可访问的Service，必须设置访问权限，权限级别必须是signature或signatureOrSystem

注：如果一个服务被导出并且没有受到强权限的保护，那么任何应用程序都可以启动并绑定到服务。根据特定服务的职责，它可能泄漏信息或执行未授权的任务，为了防止此类可能性，导出的服务应始终以强权限保护。

反例：

<service

android:name=".MyService"

android:exported="true">

<intent-filter>

<action android:name="cn.nubia.intent.action.ENTRYMYSERVICE" />

</intent-filter>

</service>

正例：

<permission

android:name="cn.nubia.permission.START\_MY\_SERVICE"

android:protectionLevel="signatureOrSystem" >

</permission>

<service

android:name=".MyService"

android:exported="true"

android:permission="cn.nubia.permission.START\_MY\_SERVICE">

<intent-filter>

<action android:name="cn.nubia.intent.action.ENTRYMYSERVICE" />

</intent-filter>

</service>

### 如果Service支持绑定并且允许应用程序调用所提供的方法，必须为这些方法增加权限校验，控制对方法的访问

反例：

public void writeData(String content) {

…

}

正例：

public void writeData(String content) {

boolean hasPermission = checkCallingPermission();

if (hasPermission) {

…

}

}

## ContentProvider安全

### 必须声明android:exported的值，不允许使用默认值

注：ContentProvider类提供了一种用于管理和与其他应用程序共享数据的机制。 当与其他应用程序共享提供程序的数据时，应仔细实施访问控制，以禁止未经授权的访问敏感数据。

反例：

<provider

android:name=".MyProvider"

android:authorities="cn.nubia.provider.myprovider">

</provider>

正例：

<provider

android:name=".MyProvider"

android:authorities="cn.nubia.provider.myprovider"

android:exported="false">

</provider>

### 声明为公开可访问的Provider，必须设置读写权限，权限级别必须是signature或signatureOrSystem

反例：

<provider

android:name=".MyProvider"

android:authorities="cn.nubia.provider.myprovider"

android:exported="true">

</provider>

正例：

<permission

android:name="cn.nubia.permission.READ\_MY\_PROVIDER"

android:protectionLevel="signatureOrSystem" >

</permission>

<permission

android:name="cn.nubia.permission.WRITE\_MY\_PROVIDER"

android:protectionLevel="signature" >

</permission>

<provider

android:name=".MyProvider"

android:authorities="cn.nubia.provider.myprovider"

android:exported="true"

android:readPermission="cn.nubia.permission.READ\_MY\_PROVIDER"

android:writePermission="cn.nubia.permission.WRITE\_MY\_PROVIDER">

</provider>

### openFile()接口中必须对访问文件路径进行有效判断

注：通过使用ContentProvider.openFile（）方法，您可以为另一个应用程序提供访问应用程序数据（文件）的功能。 根据ContentProvider的实现，使用该方法可能导致目录遍历漏洞。 因此，在通过内容提供者交换文件时，路径应在使用前进行规范化。

反例：

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);

}

正例：

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);

}

## Broadcast Receiver安全

### 如果receiver 注册的 intent 仅来自某个特定的应用，必须设置权限，权限级别必须是signature或signatureOrSystem，只接收具有该权限的广播

反例：

<receiver

android:name=".MyReceiver"

android:exported="true">

<intent-filter>

<action android:name="cn.nubia.intent.action.MYRECEIVER" />

</intent-filter>

</receiver>

正例：

<permission

android:name="cn.nubia.permission.START\_MY\_RECEIVER"

android:protectionLevel="signatureOrSystem" >

</permission>

<receiver

android:name=".MyReceiver"

android:exported="true"

android:permission="cn.nubia.permission.START\_MY\_RECEIVER">

<intent-filter>

<action android:name="cn.nubia.intent.action.MYRECEIVER" />

</intent-filter>

</receiver>

### 如果广播内容包含敏感数据，则必须设置权限，具有该权限才能接收该广播

正例：

Intent intent = new Intent(“cn.nubia.intent.action.MYRECEIVER”);

context.sendBroadcast(intent, “cn.nubia.permission.START\_MY\_RECEIVER”);

### 同一个应用内broadcast/receiver Intent，必须使用LocalBroadcastManager注册receiver和broadcast Intent

正例：

IntentFilter filter = new IntentFilter();

filter.addAction("my-sensitive-event");

mReceiver = new BroadcastReceiver();

mLocalBroadcastManager.registerReceiver(mReceiver, filter);

Intent intent = new Intent("my-sensitive-event");

intent.putExtra("event", "this is a test event");

LocalBroadcastManager.getInstance(this).sendBroadcast(intent);

# **存储安全**

## 禁止将应用的敏感数据保存在外置SD上

注：在Android 4.1之前，保存到外部存储的文件是世界上可写的。从Android 4.1到Android 4.3，只有WRITE\_EXTERNAL\_STORAGE权限才能让应用程式写入任何应用程式储存的任何外部储存空间档案。从Android 4.4开始，基于目录结构创建文件的组和模式，允许应用程序根据其软件包名称来管理/读取/写入目录结构中的文件。从Android 4.4开始，用户（包括作为用户的应用）与Android设备控制的其他应用的主要外部存储空间隔离。由于缺少上述限制，写入外部存储器的文件可以通过安装在设备上的其他应用程序（对于允许读/写的Android版本）以及任何有权访问文件的任何人进行修改或读取。

反例：

private String filename = "myfile"

private String string = "sensitive data such as credit card number"

FileOutputStream fos = null;

try {

File file = new File(getExternalFilesDir(TARGET\_TYPE), filename);

fos = new FileOutputStream(file, false);

fos.write(string.getBytes());

} catch (FileNotFoundException e) {

} catch (IOException e) {

} finally {

if (fos != null) {

try {

fos.close();

} catch (IOException e) {

}

}

}

正例：

private String filename = "myfile"

private String string = "sensitive data such as credit card number"

FileOutputStream fos = null;

try {

fos = openFileOutput(filename, Context.MODE\_PRIVATE);

fos.write(string.getBytes());

fos.close();

} catch (FileNotFoundException e) {

} catch (IOException e) {

} finally {

if (fos != null) {

try {

fos.close();

} catch (IOException e) {

}

}

}

## 应用创建文件时，禁止使用MODE\_WORLD\_READABLE、MODE\_WORLD\_WRITABLE模式(android7.1会直接crash)

反例：

openFileOutput("someFile", Context.MODE\_WORLD\_READABLE);

openFileOutput("someFile", Context.MODE\_WORLD\_WRITEABLE);

正例：

openFileOutput("someFile", MODE\_PRIVATE);

如果一定要供其他应用可读写，请使用openFile()方法进行实现：

public ParcelFileDescriptor openFile(Uri paramUri, String paramString) throws FileNotFoundException {

}

## 通过NDK创建的文件时，必须指定文件权限

反例：

FILE \* fp = fopen("/data/data/com.mine.work/file.txt", "a");

fprintf(fp, "Don't alter this content.\n");

fclose(fp);

正例：

const char \* fn = "/data/data/com.mine.work/file.txt";

const char \* content = "Don't corrupt this content.\n";

fd = open(fn, O\_CREAT|O\_RDWR, S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IWGRP|S\_IROTH);

err = write(fd, content, strlen(content));

close(fd);

O\_RDONLY 只读打开

O\_WRONLY 只写打开

O\_RDWR 可读可写打开

允许文件的所有者阅读它；允许文件所有者写它；允许文件组读取它；允许文件组编写它。其他读

## Log中禁止携带敏感数据

反例：

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

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

## 用户隐私数据必须加密保存或传输

加密算法：需要采用对称式加密，建议使用AES/CBC/PKCS5Padding(128)，不要对AES使用Android加密默认值，AES加密方式使用默认的ECB快加密模式，不要使用非随机IV进行CBC加密。

秘钥：加/解密使用的秘钥不允许写在代码中，应该由程序产生随机秘钥，不要使用静态种子来种植securerandom()。

秘钥存储：秘钥保存在自己应用的/data/data目录下，建议可以将秘钥分成几段乱序存储，并且将秘钥经过一些计算后作为加/解密的秘钥，如SHA-256。

# **WebView安全**

## 不要在包含不受信任内容的WebView中提供addJavascriptInterface方法访问（API级别JELLY或更低）

注：对于API级别JELLY\_BEAN或更低版本，允许应用程序在WebView中使用addJavascriptInterface方法与不受信任的内容，使应用程序容易受到使用反射从JavaScript访问公共方法的脚本攻击。 不受信任的内容示例包括来自任何HTTP URL（而不是HTTPS）和用户提供的内容的内容。

反例：

WebView webView = new WebView(this);

setContentView(webView);

...

class JsObject {

     private String sensitiveInformation;

     ...

     public String toString() { return sensitiveInformation; }

}

 webView.addJavascriptInterface(new JsObject(), "injectedObject");

 webView.loadData("", "text/html", null);

 webView.loadUrl("[http://www.example.com"](http://www.example.com/));

正例1：

合规代码可以避免调用addJavascriptInterface（）方法。

WebView webView = new WebView(this);

setContentView(webView);

...

正例2：

在应用程序的清单中指定应用程序仅适用于API级别JELLY\_BEAN\_MR1及更高版本。 对于这些API级别，只有使用JavascriptInterface注释的公共方法可以从JavaScript访问。 API级别17是JELLY\_BEAN\_MR1。

<manifest>

<uses-sdk android:minSdkVersion="17" />

...

</manifest>

## 如果应用程序不直接使用 WebView 中的 JavaScript，请禁止调用 setJavaScriptEnable().

## 不允许WebView通过file://访问敏感的本地资源

反例：

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);

}

}

正例：

String intentUrl = getIntent().getStringExtra("url");

String localUrl = "about:blank";

if (!intentUrl.startsWith("file:")) {

loadUrl = intentUrl;

}

# **通信安全**

## 敏感数据必须使用HTTPS传输

## HTTPS通信时，必须对SSL证书进行强校验 （正例）

包括：签名CA是否合法、证书是否是自签名、主机域名是否匹配、证书是否过期。

反例：

public class extends SSLSocketFactory {

    SSLContext sslContext;

public MySSLSocketFactory (KeyStore truststore) throws NoSuchAlgorithmException, KeyManagementException,KeyStoreException, UnrecoverableKeyException {

    super(truststore);

    this.sslContext = SSLContext.getInstance("TLS");

    this.sslContext.init (null, new TrustManager[] {new X509TrustManager() {

            public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException

            {

            }

            public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException

            {

            }

            public X509Certificate[] getAcceptedIssuers() {

                return null;

            }

        }}, null);

}

    public Socket createSocket() throws IOException {

        return this.sslContext.getSocketFactory().createSocket();

    }

    public Socket createSocket(Socket socket, String host, int port, boolean autoClose) throws IOException

                                                                                               , UnknownHostException {

        return this.sslContext.getSocketFactory().createSocket(socket, host, port, autoClose);

    }

}

 public static HttpClient getNewHttpClient() {

            DefaultHttpClient v6;

            try {

                KeyStore v5 = KeyStore.getInstance(KeyStore.getDefaultType());

                v5.load(null, null);

                MySSLSocketFactory mySSLScoket = new MySSLSocketFactory(v5);

                if(DefineRelease.sAllowAllSSL) {

                    ((SSLSocketFactory)mySSLScoket).setHostnameVerifier(SSLSocketFactory.ALLOW\_ALL\_HOSTNAME\_VERIFIER);

                }

                BasicHttpParams v2 = new BasicHttpParams();

                HttpConnectionParams.setConnectionTimeout(((HttpParams)v2), 30000);

                HttpConnectionParams.setSoTimeout(((HttpParams)v2), 30000);

                HttpProtocolParams.setVersion(((HttpParams)v2), HttpVersion.HTTP\_1\_1);

                HttpProtocolParams.setContentCharset(((HttpParams)v2), "UTF-8");

                SchemeRegistry v3 = new SchemeRegistry();

                v3.register(new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));

                v3.register(new Scheme("https", ((SocketFactory)mySSLScoket), 443));

                v6 = new DefaultHttpClient(new ThreadSafeClientConnManager(((HttpParams)v2), v3), ((HttpParams)v2));

            }

            catch(Exception v1) {

                v6 = new DefaultHttpClient();

            }

            return ((HttpClient)v6);

}

注：在上面的示例中，checkClientTrusted（）和checkServerTrusted（）被覆盖以使空白实现，以便SSLSocketFactory不验证SSL证书。 MySSLSocketFactory类用于在应用程序的另一部分中创建HttpClient的实例。

sAllowAllSSL是DefineRelease类的静态成员，在其静态构造函数中初始化为true。 这将启用SSLSocketFactory.ALLOW\_ALL\_HOSTNAME\_VERIFIER的使用。 因此，在建立SSL连接时应进行的主机名验证被禁用，并且将导致与所有证书都受信任的情况相同的情况。

## 禁止使用 localhost 网络端口处理敏感数据。（后面这两个应该是服务端的问题）

## 禁止Socket绑定INADDR\_ANY。（INADDR\_ANY就是指定地址为0.0.0.0的地址，这个地址事实上表示不确定地址，或“所有地址”、“任意地址”。 一般来说，在各个系统中均定义成为0值）