

## 移动应用三方SDK安全质量保障实践

OPPO安珀实验室

🕒 2021-02-20 14:59:49

🔥 146182

本文整理自OPPO技术开放日第六期——《应用与数据安全防护》活动，分享嘉宾为向波。

主要内容如下:

## 三方SDK安全现状

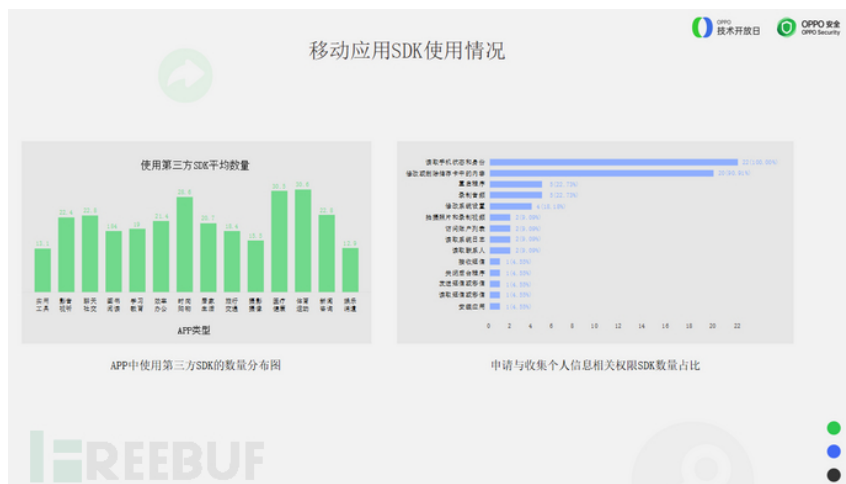
## 三方SDK安全案例

## 三方SDK安全检测方法

## 三方SDK安全保障在SDL中的实践

## 一、三方SDK安全现状

众所周知，应用与数据安全防护的一个重点关注方向，就是三方SDK的安全问题。作为一个基础性的安全问题，SDK广泛应用于APP当中，这就导致三方SDK的安全问题可能影响更为深远。



三方SDK在现在的日常生活当中使用非常广泛，封装了很丰富的功能，对于开发者来说可以直接拿来使用十分便利。同时它包含的类型、采集的信息也是各种各样，对于安全分析也带来了很多的挑战。2020年的315大会上就曾曝光了某SDK检测收集个人隐私信息的问题。



三方SDK的特征可以简单归结为四点:

**共享APP权限。** SDK存在形式包括jar、aar和so等方式，封装有丰富的功能，同时还有一个特点，就是可以共享APP权限。

**黑盒无法检测。**开发者可能更关注SDK是否使用便捷，但是SDK是否搜集用户的隐私信息、是否存在安全漏洞、有没有恶意行为，开发者是不知道的。

**攻击面多。**引入了SDK也就等于引入了更多的攻击面，而且在我们实际测试当中发现三方SDK的安全问题往往多于APP自身或自研的SDK。

**影响广泛。**常用的SDK装机量可能是以亿计或者十亿计，如果出现一个安全问题，可能影响的就是上亿或者上十亿的用户或者终端。

请 [登录](#) / [注册](#) 后在FreeBuf发布内容哦

19

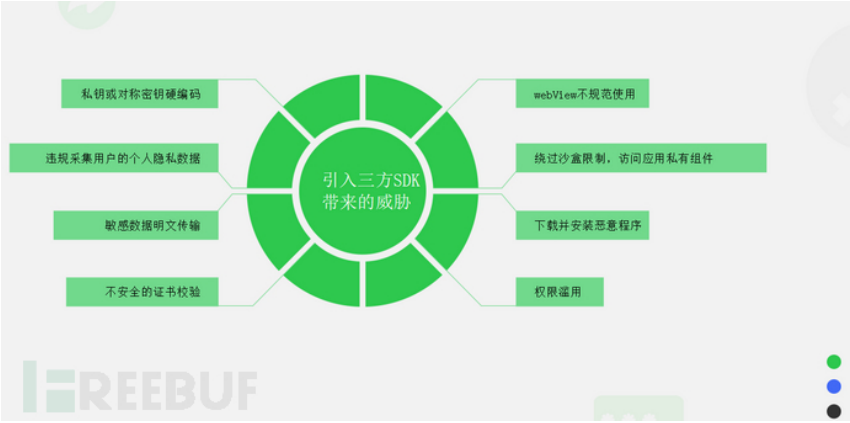
0

+ 收入专辑

...

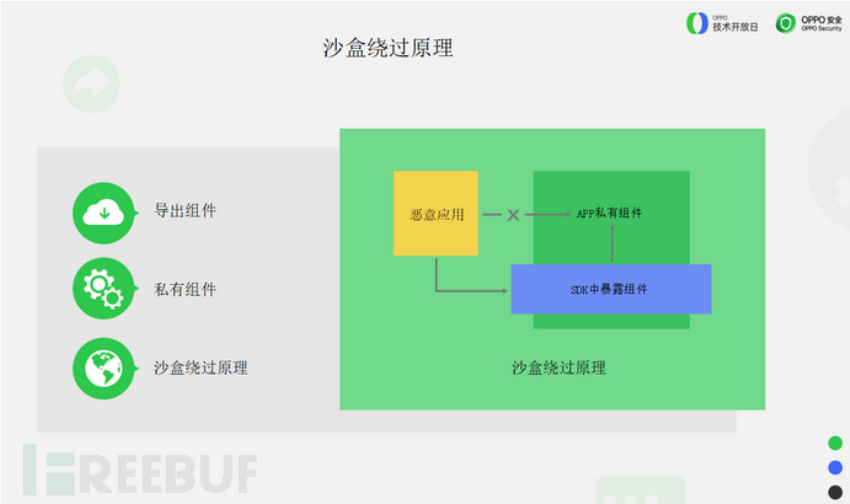
文章目录

- 二、三方SDK安全
- 三、三方SDK安全
- 四、三方SDK安全



在实际测试当中可以发现，三方SDK带来的常见威胁大概分为八个方面：

- 第一是私钥或对称密钥硬编码；
- 第二违规采集用户的个人隐私信息；
- 第三敏感数据明文传输；
- 第四不安全的证书校验；
- 第五WebView不规范使用；
- 第六绕过沙盒限制，访问应用私有组件；
- 第七下载并安装恶意使用；
- 第八权限滥用。



组件一般可以分为**导出组件**和**私有组件**。  
**导出组件**可以被任意应用访问，**私有组件**只能被自己的应用访问，而私有组件中往往会存在一些敏感操作，比如修改密码、修改帐号、设置密码，访问敏感数据等操作。  
同时开发者一般会觉得私有组件没有对外暴露所以是足够安全，所以对用户外面的一些输入没有做强校验。  
如果是存在漏洞的话，那就有可能带来严重的威胁。  
恶意应用本身是无法直接访问APP当中的一些私有组件，但是它通过一个桥，通过SDK当中的一些暴露的组件，这个暴露的组件里面又可以直接去访问私有组件，这样就达到了绕过的原理。

终端安全

19

FreeBuf

主站 招聘站 公开课 用户服务 企业服务 行业专区

云沙箱 搜索 登

1

<activity android:name="com.examples.xxxActivity">  
<intent-filter>  
<action android:name="android.intent.action.MAIN" />  
<category android:name="android.intent.category.LAUNCHER" />  
</intent-filter>  
</activity>

2

xxxActivity中接收外部Intent传入字符串，未经校验情况下，将传入字符串作为ActivityName进行保存  
this.ActivityName = getIntent().getStringExtra(Constants.SHARE\_START\_ACTIVITY);

3

在当前应用Context中调用startActivity启动ActivityName指定Activity  
Intent intent = new Intent();  
intent.setClassName(this, this.ActivityName);  
intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);  
startActivity(intent);

文章目录

二、三方SDK安全类

三、三方SDK安全类

四、三方SDK安全类

二、三方SDK安全案例

接下来看一个实例，这里有三部分内容，第一是导出组件；  
第二是接收外部传入的数据，然后不做任何校验保存为ActivityName；  
第三在当前应用当中启动ActivityName，就达到了绕过沙盒的结果。  
解决方法就是不需要导出的组件一定要关闭，同时对外部输入进行严格校验。

SDK中ZIP目录穿越漏洞

1

while ((zipEntry = zipInputStream.getNextEntry()) != null) {  
Log.i(TAG, zipEntry.getName());  
File f = new File(destDir + zipEntry.getName());  
if (zipEntry.isDirectory()) {  
f.mkdirs();  
} else {  
FileOutputStream fileOutputStream = new FileOutputStream(f);  
byte[] buf = new byte[1024];  
int len = 1;  
while ((len = zipInputStream.read(buf)) != -1) {  
fileOutputStream.write(buf, 0, len);  
}  
fileOutputStream.flush();  
fileOutputStream.close();  
}  
}

2

ZipInputStream zis = new ZipInputStream(new  
BufferedInputStream(Go));  
while ((zipEntry = zis.getNextEntry()) != null) {  
File f = new File(DIR, zipEntry.getName());  
String canonicalPath = f.getCanonicalPath();  
if (!canonicalPath.startsWith(DIR)) {  
// handle exception  
}  
// Finish unzipping  
}

SDK当中还存在常见的ZIP目录穿越漏洞。  
这种ZIP压缩包里面的路径存在../这样的字符串，攻击者可以利用多个../就可以穿越到其他地方。  
想象一下，如果跳跃到其他地方，这个文件和用户要加载的文件名字相同，就可以直接覆盖掉后者。  
如果覆盖的是一个可执行程序，这个替换后的可执行程序在被加载时，轻则可以造成拒绝服务，重则可以造成代码执行漏洞，会严重危害用户设备安全。  
在考虑解决方法的时候，最需要关注的就是在ZipEntry取得绝对路径时，需要确保里面没有../，同时校验是否为正常访问的目录，是的话可以执行，如果不是就需要抛出异常。

终端安全

19

①

```
<service android:name="com.example.messageService">
<intent-filter>
<action
android:name="android.intent.action.MAIN" />
<category
android:name="android.intent.category.DEFAULT" />
</intent-filter>
</service>
```

②

```
protected void onMessage(Context context) throws JSONException
{
String v0_1 = intent.getStringExtra("body");
String v1 = intent.getStringExtra("id");
String v2 = intent.getStringExtra("task_id");
aMessage message = new aMessage(new JSONObject(v0_1));
if("pullapp".equals(message.display_type)) {
Intent intent = new Intent();
intent.setClassName(message.piled_package, message.piled_service);
startService(intent);
}
}
```

文章目录

二、三方SDK安全第

三、三方SDK安全格

四、三方SDK安全保

对这两个漏洞做一个总结。

如果有访问私有路径，再加上路径穿越的话，那对于攻击者来说，可以执行的操作就在于其想象的空间。

相对于第一个漏洞来说，这个漏洞的构造方式可能稍微复杂一点，首先这是一个导出组件，将外部传入的数据设置成v0\_1，通过JsonObject创建一个Message，这个Message也是没有做校验的。

攻击者可以在插件加载过程当中覆盖掉一些可执行文件，达到恶意代码注入和代码执行的结果。

某SDK开放网络端口漏洞案例

①

```
this.c = new ServerSocket();
this.c.bind(this.a != null ? new InetSocketAddress(this.a, this.b) : new InetSocketAddress(this.a));
this.e = new Thread(new v(this));
this.a.setDaemon(true);
this.a.setName("HandlerThread Main Listener");
this.a.start();
```

②

```
a.put("geoLocation", b + "GetLocLiteString");
a.put("getApp", b + "GetApp");
a.put("getServiceInfo", b + "GetServiceInfo");
a.put("getPackageInfo", b + "GetPackageInfo");
a.put("sendIntent", b + "SendIntent");
a.put("getAppId", b + "GetAppId");
a.put("getLocString", b + "GetLocString");
a.put("scanDownloadFile", b + "ScanDownloadFile");
a.put("addContactInfo", b + "AddContactInfo");
a.put("getAppList", b + "GetAppList");
a.put("downloadFile", b + "DownloadFile");
```

③

```
if (b2 != null) {
if ((b2.applicationInfo.flags & 1) == 1) {
if (b.a(this.b, "android.permission.INSTALL_PACKAGES") != 0) {
File file = new File(str);
if (a0) {
new c(this, "SystemMonitor_InstallAPKByPackageInstaller", file, context, str).start();
return;
}
a(Uri.fromFile(file), new SilentPackageInstaller(context, str), 0, context.getPackageName());
}
} else if (com.module.a.h.a(context).a0) {
com.module.a.h.a(context).a("pm install -r " + str + ".apk");
}
}
```

再看另外一类。

这一类也是影响比较大的一个漏洞；这种方式就是在本地APP当中创建http server Socket，该server在APP启动的时候就运行，同时绑定到任意地址，然后监听网络端口。

此外可以执行一些强大的命令，比如说上传、下载文件，添加联系人，甚至可以判断手机是不是Root，如果是Root可以静默安装更多的程序。

使用Socket存在的一个风险点就是弱校验。我们的修复建议就是**禁止Socket绑定0.0.0.0这样的地址**，改为绑定到127.0.0.1，并且对接入数据进行过滤、验证。

三方SDK任意代码执行案例

①

```
public void searchClass(Context context) {
for (PackageInfo
info:context.getPackageManager().getInstalledPackages(0));
String packageName = info.packageName;
if(packageName.startsWith("com.example.module.")){
try {
invokeModuleMethod(context, packageName);
} catch (PackageManager.NameNotFoundException e) {
// handle exception exception
}
}
}
```

②

```
protected void invokeModuleMethod(Context context, String packageName)
throws PackageManager.NameNotFoundException {
Context myContext = context.createPackageContext(packageName,
Context.INCLUDE_CODE | Context.IGNORE_SECURITY);
ClassLoader classLoader = myContext.getClassLoader();
try {
Object object =
classLoader.loadClass("com.example.module.testClass").getMethod("myFunction");
invoke(object);
} catch (IllegalAccessException e) {
// handle exception
}
```

③

绕过方法

```
package com.example.module;

public class testClass {
protected void myFunction() {
try {
Runtime.getRuntime().exec("command");
} catch (IOException e) {
// handle exception
}
}
}
```

终端安全

信息,但是没有判断可加载的可执行程序具体从哪儿来,攻击者可以伪造同类名和方法名的方式来构造可执行程序,导致被classloader执行。

建议大家对APP的签名信息做校验。

文章目录

19

SDK绕过安全检测违规获取用户隐私信息案例

①

```
Method method = getMethod(G.getClass(), "ZZVUT0$JlXph0$=",
    new Class[0]);
String str = null;
try {
    str = (String) method.invoke(G, new Object[0]);
} catch (IllegalAccessException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
}
return str;
}
```

②

```
public Method getMethod(Class cls, String str, Class[]...clsArr) {
    Method declaredMethod = null;
    try {
        declaredMethod = cls.getDeclaredMethod(base64decode(str),
            clsArr);
    } catch (NoSuchMethodException e) {
        e.printStackTrace();
    }
    return declaredMethod;
}
```

③

ZZVUT0\$JlXph0\$=
getLocation

某些三方SDK开发商还会采用一些方式绕过常规安全工具的检测来获取用户隐私。

上图左边的硬编码字符串是一个敏感操作方法getLocation用于获取用户的地址信息并做了Base64编码,到右边中使用Base64去解码,还原成getLocation方法,再使用反射方式去调用,从而达到绕过常规安全检测工具的检测的目的。



三、三方SDK安全检测方法

那么,三方SDK应该如何保障安全呢?

首先, **组件尽量不要导出**,如果必须导出需要设置好保护级别,且需要对外部传入的数据并且校验并确认调用者的身份。 **通信方面,需要防止中间人攻击,传输内容的做加密。**

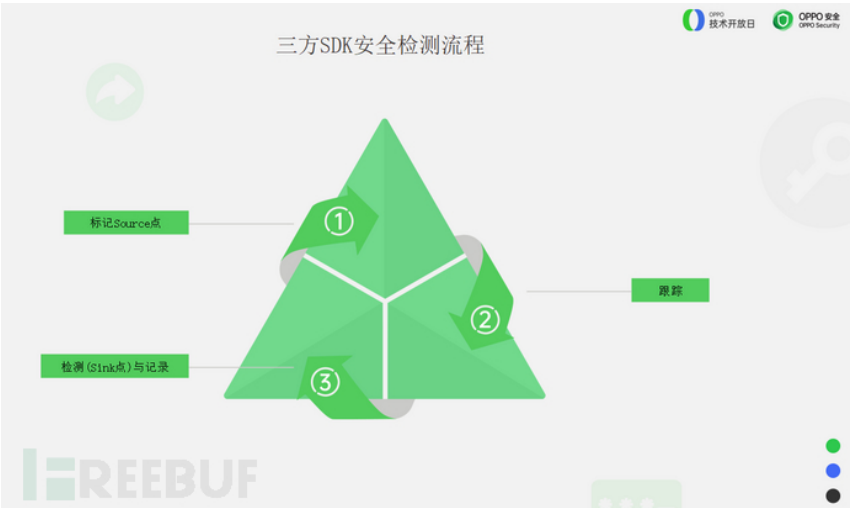
此外, **数据方面确保权限最小**,防范路径穿越。加载的内容要可信,可以使用**加密和校验的方式**。



三方SDK的安全检查是非常重要的。

我们的检测内容包含三个方面：**第一隐私合规检测，第二漏洞检测，第三恶意行为检测。**

结合工作实践，从这三个维度出发，可以达到尽可能保障用户端应用安全的目的。



检测方面，可以用静态的污点分析方法。

污点分析分三个阶段。

**首先是标记source点（即敏感信息），然后进行跟踪，采用特定的规则跟踪分析污点信息，最后检测与记录访问路径。**

举个例子。

针对沙盒绕过，首先需要判断是否传入了intent。如果没有做校验就打开另一个组件，则这种是有安全风险的。



终端安全

四、三方SDK安全保障在SDL中的实践

那么，三方SDK的安全检测在SDL流程中是如何融合的呢？

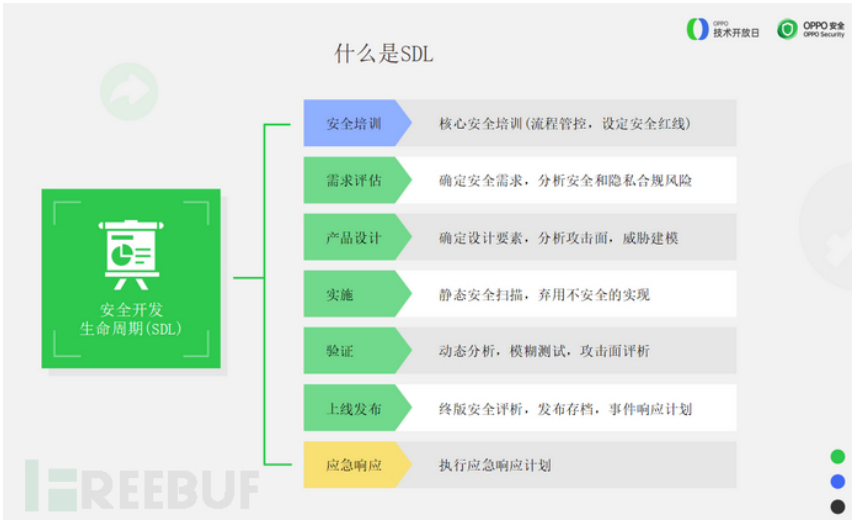
首先简单介绍一下SDL。

SDL(Security Development Lifecycle)即“安全开发生命周期”，主要指在帮助开发人员构建更安全的软件 and 解决安全合规需求的同时，降低开发成本的一个开发过程。

原则上要求尽量从源头上解决安全问题，避免开发当中的常见安全错误不再犯。

文章目录

- 二、三方SDK安全基
- 三、三方SDK安全检
- 四、三方SDK安全保



SDL可以简单的划分为三大块，第一块是安全培训,第二块是SDK上线之前打点的流程（上图绿色部分），最后是应急响应。

主要来看看第二部分的打点流程。

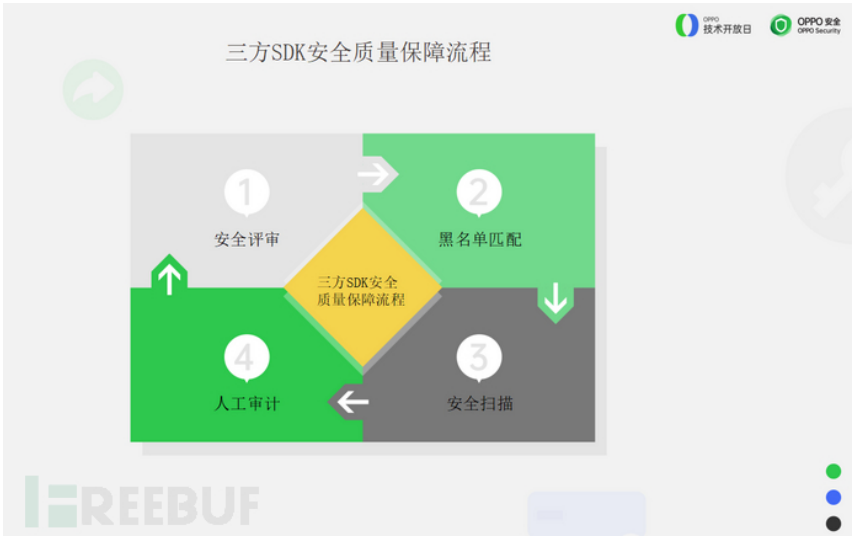
需求评估方面，在设计阶段不能仅仅考虑功能的实现，还要重视安全是否合规，是否违反了安全红线。

而产品设计阶段要分析攻击面，进行威胁建模。

到实施阶段可以执行静态安全扫描，摒弃不安全的实现。并在验证当中对集成SDK后的APP做一个动态扫描和模糊测试。

这样在最终版安全上线的时候，可以看到APP在安全维度的分数，如果低于要求则不能上线，同时还要处理好事件的响应计划。

对于手机终端安全人员来说，SDK检测的难点，主要集中两个方面，一个是SDK量大且迭代频繁，增加检测工作的难度。另外就是怎么做好投入产出比的计算。



三方SDK安全保障流程可以从四个方面做到闭环。

首先从安全评审，直接在设计或者SDK引入阶段，对引入的SDK安全能力做一个评估，执行威胁建模



终端安全

19

三方SDK安全质量保障流程

OPPO 技术开放日

OPPO 安全

1

安全评审覆盖威胁建模，减少安全和隐私风险

2

黑名单检测，保证APP接入的SDK是最新版本

3

安全检测平台化和自动化，扫描结果覆盖全面

4

重点项目专家深入分析，开发和复用安全编码库

优点

REEBUF

文章目录

二、三方SDK安全

三、三方SDK安全

四、三方SDK安全

三方SDK安全质量保障流程有几个方面需要注意。

首先是安全评审方面，**在最初阶段就要进行威胁建模，减少SDK的安全和隐私风险。**

其次是黑名单检测，**同时确保APP接入的SDK始终是最新的版本，如果有存在漏洞需要保证漏洞补丁及时打上。**

第三是安全检测平台化和自动化，**扫描结果覆盖全面，提前发现并帮助业务方自行解决典型的安全漏洞**，避免在上线/终版审核之前业务方大量的修复和升级的工作量。

同时扫描结果尽量详细，把安全团队的人员从冗余工作当中解放出来，研究更好的技术，给用户带来更多的安全体验。

第四对于重点项目建议组织专家深入分析，**同时开发和复用安全编码库，采用复用安全编码库提高一致性，从而提高工作效率。**

通过以上三方SDK安全质量保障流程的实践，在投入少的情况下使得SDK的安全性得到了更好的保证，保护用户终端的安全性。

本文作者：OPPO安珀实验室， 转载请注明来自FreeBuf.COM

# 应用安全

# 终端安全

被以下专辑收录，发现更多精彩内容

+ 收入我的专辑

App安全合规

Android自动化检测

评论

按热度排序

请 [登录](#) / [注册](#) 后在FreeBuf发布内容哦

相关推荐