你不知道的Android WebView漏洞

笔记本:	程序员		
创建时间:	2020/8/13 星期四 15:28	**更新时间:**	2020/8/13 星期四 15:28
标签:	Android, WebView, 编程		
URL:	https://juejin.im/post/6844903482437156877		

你不知道的Android WebView漏洞

前言

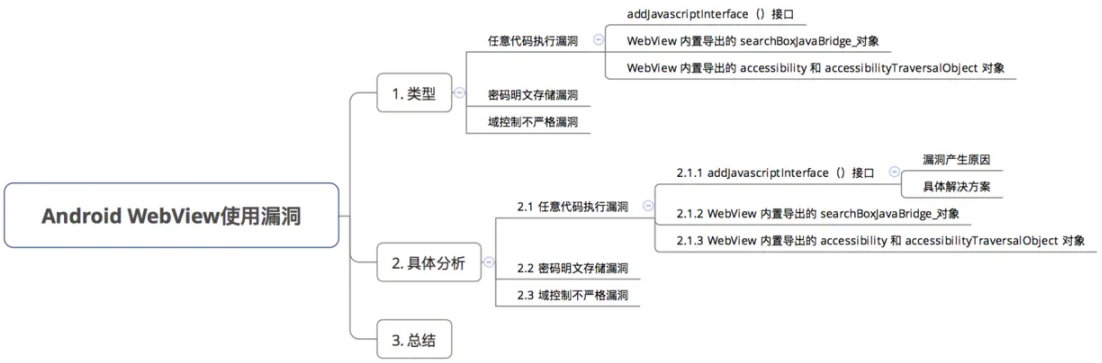
- 现在很多App里都内置了Web网页（Hyprid App），比如说很多电商平台，淘宝、京东、聚划算等等，如下图

京东首页

- 上述功能是由 **Android的WebView** 实现的，但是 WebView 使用过程中存在许多漏洞，容易造成用户数据泄露等等危险，而很多人往往会忽视这个问题
- 今天我将全面介绍 **Android WebView的使用漏洞** 及其修复方式

> 阅读本文前请先阅读：
> Android开发：最全面、最易懂的Webview详解
> 最全面总结 Android WebView与 JS 的交互方式
> 手把手教你构建 Android WebView 的缓存机制 & 资源预加载方案

目录



目录

1. 类型

WebView中，主要漏洞有三类：

- 任意代码执行漏洞
- 密码明文存储漏洞
- 域控制不严格漏洞

2. 具体分析

2.1 WebView 任意代码执行漏洞

出现该漏洞的原因有三个：

- WebView 中 `addJavascriptInterface（）` 接口
- WebView 内置导出的 `searchBoxJavaBridge_` 对象
- WebView 内置导出的 `accessibility` 和 `accessibilityTraversal` Object 对象

2.1.1 addJavascriptInterface 接口引起远程代码执行漏洞

A. 漏洞产生原因

JS调用Android的其中一个方式是通过 `addJavascriptInterface` 接口进行对象映射：

复制代码

```
 webView.addJavascriptInterface(new JSObject(), "myObj");
// 参数1：Android的本地对象
// 参数2：JS的对象
// 通过对象映射将Android中的本地对象和JS中的对象进行关联，从而实现JS调用Android的
```

所以，漏洞产生原因是：当JS拿到Android这个对象后，就可以调用这个Android对象中所有的方法，包括系统类（java.lang.Runtime 类），从而进行任意代码执行。

> 如可以执行命令获取本地设备的SD卡中的文件等信息从而造成信息泄露

具体获取系统类的描述：（结合 Java 反射机制）

- Android中的对象有一公共的方法：getClass()；
- 该方法可以获取到当前类 类型Class
- 该类有一关键的方法： Class.forName；
- 该方法可以加载一个类（可加载 java.lang.Runtime 类）
- 而该类是可以执行本地命令的

以下是攻击的Js核心代码：

复制代码

```
function execute(cmdArgs)
{
    // 步骤1：遍历 window 对象
    // 目的是为了找到包含 getClass （）的对象
    // 因为Android映射的JS对象也在window中，所以肯定会遍历到
    for (var obj in window) {
        if ("getClass" in window[obj]) {

      // 步骤2：利用反射调用forName（）得到Runtime类对象
            alert(obj);
            return  window[obj].getClass().forName("java.lang.Runtime")

      // 步骤3：以后，就可以调用静态方法来执行一些命令，比如访问文件的命令
getMethod("getRuntime",null).invoke(null,null).exec(cmdArgs);
```

```
// 从执行命令后返回的输入流中得到字符串，有很严重暴露隐私的危险。
// 如执行完访问文件的命令之后，就可以得到文件名的信息了。
        }
    }
}
```

- 当一些 APP 通过扫描二维码打开一个外部网页时，攻击者就可以执行这段 js 代码进行漏洞攻击。
- 在微信盛行、扫一扫行为普及的情况下，该漏洞的危险性非常大

B. 解决方案

B1. Android 4.2版本之后

Google 在Android 4.2 版本中规定对被调用的函数以 `@JavascriptInterface` 进行注解从而避免漏洞攻击

B2. Android 4.2版本之前

在Android 4.2版本之前采用**拦截prompt（）**进行漏洞修复。
具体步骤如下：

- 继承 WebView ，重写 `addJavascriptInterface` 方法，然后在内部自己维护一个对象映射关系的 Map；

  > 将需要添加的 JS 接口放入该Map中

- 每次当 WebView 加载页面前加载一段本地的 JS 代码，原理是：

  - 让JS调用一Javascript方法：该方法是通过调用prompt（）把JS中的信息（含特定标识，方法名称等）传递到Android端；
  - 在Android的onJsPrompt（）中，解析传递过来的信息，再通过反射机制调用Java对象的方法，这样实现安全的JS调用Android代码。

> 关于Android返回给JS的值：可通过prompt（）把Java中方法的处理结果返回到Js中

具体需要加载的JS代码如下：

复制代码

```
javascript:(function JsAddJavascriptInterface_(){
// window.jsInterface 表示在window上声明了一个Js对象

//   jsInterface = 注册的对象名
// 它注册了两个方法，onButtonClick(arg0)和onImageClick(arg0, arg1, arg2)
// 如果有返回值，就添加上return
    if (typeof(window.jsInterface)!='undefined') {
        console.log('window.jsInterface_js_interface_name is exist!!');}
    else {
        window.jsInterface = {

     // 声明方法形式：方法名: function(参数)
            onButtonClick:function(arg0) {
// prompt（）返回约定的字符串
// 该字符串可自己定义
// 包含特定的标识符MyApp和 JSON 字符串（方法名，参数，对象名等）
                return prompt('MyApp:'+JSON.stringify({obj:'jsInterface',fun
            },


            onImageClick:function(arg0,arg1,arg2) {
         return
prompt('MyApp:'+JSON.stringify({obj:'jsInterface',func:'onImageClick',args:[
            },
        };
    }
}
)()

// 当JS调用 onButtonClick（） 或 onImageClick（） 时，就会回调到Android中的 onJ
// 我们解析出方法名，参数，对象名
// 再通过反射机制调用Java对象的方法
```

关于该方法的其他细节

细节1：加载上述JS代码的时机

- 由于当 WebView 跳转到下一个页面时，之前加载的 JS 可能已经失效
- 所以，通常需要在以下方法中加载 JS：

复制代码

```
onLoadResource（）；
doUpdateVisitedHistory（）；
onPageStarted（）；
onPageFinished（）；
onReceivedTitle（）；
onProgressChanged（）；
```

细节2：需要过滤掉 Object 类的方法

- 由于最终是通过反射得到Android指定对象的方法，所以同时也会得到基类的其他方法（最顶层的基类是 Object类）
- 为了不把 getClass（）等方法注入到 JS 中，我们需要把 Object 的共有方法过滤掉，需要过滤的方法列表如下：

复制代码

```
getClass()
hashCode()
notify()
notifyAl()
equals()
toString()
wait()
```

总结

- 对于Android 4.2以前，需要采用**拦截prompt（）**的方式进行漏洞修复
- 对于Android 4.2以后，则只需要对被调用的函数以 @JavascriptInterface进行注解
- 关于 Android 系统占比，Google公布的数据：截止 2017 .1 .8 ， Android4.4之下占有约15%，所以需要重视。

> 具体数据如下：

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.1%
4.1.x	Jelly Bean	16	4.0%
4.2.x		17	5.9%
4.3		18	1.7%
4.4	KitKat	19	22.6%
5.0	Lollipop	21	10.1%
5.1		22	23.3%
6.0	Marshmallow	23	29.6%
7.0	Nougat	24	0.5%

Paste_Image.png

2.1.2 searchBoxJavaBridge_接口引起远程代码执行漏洞

A. 漏洞产生原因

- 在Android 3.0以下，Android系统会默认通过 `searchBoxJavaBridge_` 的Js接口给 WebView 添加一个JS映射对象：`searchBoxJavaBridge_` 对象
- 该接口可能被利用，实现远程任意代码。

B. 解决方案

删除 `searchBoxJavaBridge_` 接口

```
// 通过调用该方法删除接口
removeJavascriptInterface（）；
```

2.1.3 `accessibility` 和 `accessibilityTraversal` 接口引起远程代码执行漏洞

问题分析与解决方案同上，这里不作过多阐述。

2.2 密码明文存储漏洞

2.2.1 问题分析

WebView默认开启密码保存功能：

```
mWebView.setSavePassword(true)`
```

- 开启后，在用户输入密码时，会弹出提示框：询问用户是否保存密码；
- 如果选择"是"，密码会被明文保到 /data/data/com.package.name/databases/webview.db 中，这样就有被盗取密码的危险

2.2.2 解决方案

关闭密码保存提醒

```
WebSettings.setSavePassword(false)
```

2.3 域控制不严格漏洞

2.3.1 问题分析

先看Android里的 *WebViewActivity.java*:

```
复制代码
public class WebViewActivity extends Activity {
    private WebView webView;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_webview);
        webView = (WebView) findViewById(R.id.webView);

        //webView.getSettings().setAllowFileAccess(false);
        //webView.getSettings().setAllowFileAccessFromFileURLs(true);
        //webView.getSettings().setAllowUniversalAccessFromFileURLs(true);
        Intent i = getIntent();
        String url = i.getData().toString(); //url = file:///data/local/tmp/
        webView.loadUrl(url);
    }
 }

/**Mainifest.xml**/
// 将该 WebViewActivity 在Mainifest.xml设置exported属性
// 表示：当前Activity是否可以被另一个Application的组件启动
android:exported="true"
```

即 A 应用可以通过 B 应用导出的 Activity 让 B 应用加载一个恶意的 file 协议的 url，从而可以获取 B 应用的内部私有文件，从而带来数据泄露威胁

> 具体：当其他应用启动此 Activity 时， intent 中的 data 直接被当作 url 来加载（假定传进来的 url 为 file:///data/local/tmp/attack.html ），其他 APP 通过使用显式 ComponentName 或者其他类似方式就可以很轻松的启动该 WebViewActivity 并加载恶意url。

下面我们着重分析WebView中getSettings类的方法对 WebView 安全性的影响：

- setAllowFileAccess（）
- setAllowFileAccessFromFileURLs（）
- setAllowUniversalAccessFromFileURLs（）

1. setAllowFileAccess（）

复制代码

```
// 设置是否允许 WebView 使用 File 协议
webView.getSettings().setAllowFileAccess(true);
// 默认设置为true，即允许在 File 域下执行任意 JavaScript 代码
```

使用 file 域加载的 js代码能够使用进行**同源策略跨域访问**，从而导致隐私信息泄露

> 1. 同源策略跨域访问：对私有目录文件进行访问
> 2. 针对 IM 类产品，泄露的是聊天信息、联系人等等
> 3. 针对浏览器类软件，泄露的是cookie 信息泄露。

如果不允许使用 file 协议，则不会存在上述的威胁；

复制代码

```
webView.getSettings().setAllowFileAccess(true);
```

但同时也限制了 WebView 的功能，使其不能加载本地的 html 文件，如下图：

> 移动版的 Chrome 默认禁止加载 file 协议的文件

Paste_Image.png

解决方案：

- 对于不需要使用 file 协议的应用，禁用 file 协议；

```
setAllowFileAccess(false);
```

- 对于需要使用 file 协议的应用，禁止 file 协议加载 JavaScript。

```
setAllowFileAccess(true);

// 禁止 file 协议加载 JavaScript
if (url.startsWith("file://") {
    setJavaScriptEnabled(false);
} else {
    setJavaScriptEnabled(true);
}
```

2. setAllowFileAccessFromFileURLs（）

```
// 设置是否允许通过 file url 加载的 Js代码读取其他的本地文件
webView.getSettings().setAllowFileAccessFromFileURLs(true);
// 在Android 4.1前默认允许
// 在Android 4.1后默认禁止
```

当 `AllowFileAccessFromFileURLs（）` 设置为 true 时，攻击者的JS代码为：

```


// 通过该代码可成功读取 /etc/hosts 的内容数据
```

解决方案：设置 `setAllowFileAccessFromFileURLs(false);`

> 当设置成为 false 时，上述JS的攻击代码执行会导致错误，表示浏览器禁止从 file url 中的 javascript 读取其它本地文件。

3. setAllowUniversalAccessFromFileURLs（）

```
// 设置是否允许通过 file url 加载的 Javascript 可以访问其他的源(包括http、https等
webView.getSettings().setAllowUniversalAccessFromFileURLs(true);

// 在Android 4.1前默认允许（setAllowFileAccessFromFileURLs（）不起作用）
// 在Android 4.1后默认禁止
```

当 `AllowFileAccessFromFileURLs（）` 被设置成true时，攻击者的JS代码是：

```
// 通过该代码可成功读取 http://www.so.com 的内容
```

解决方案：设置 `setAllowUniversalAccessFromFileURLs(false);`

4. setJavaScriptEnabled（）

```
// 设置是否允许 WebView 使用 JavaScript（默认是不允许）
webView.getSettings().setJavaScriptEnabled(true);

// 但很多应用（包括移动浏览器）为了让 WebView 执行 http 协议中的 JavaScript，都会
```

即使把 `setAllowFileAccessFromFileURLs（）` 和 `setAllowUniversalAccessFromFileURLs（）` 都设置为 false，通过 file URL 加载的 javascript 仍然有方法访问其他的本地文件：**符号链接跨源攻击**

> 前提是允许 file URL 执行 javascript，即 `webView.getSettings().setJavaScriptEnabled(true);`

这一攻击能奏效的原因是：**通过 javascript 的延时执行和将当前文件替换成指向其它文件的软链接就可以读取到被符号链接所指的文件。**具体攻击步骤：

1. 把恶意的 js 代码输出到攻击应用的目录下，随机命名为 xx.html，修改该目录的权限；
2. 修改后休眠 1s，让文件操作完成；
3. 完成后通过系统的 Chrome 应用去打开该 xx.html 文件

4. 等待 4s 让 Chrome 加载完成该 html，最后将该 html 删除，并且使用 ln -s 命令为 Chrome 的 Cookie 文件创建软连接

> 注：在该命令执行前 xx.html 是不存在的；执行完这条命令之后，就生成了这个文件，并且将 Cookie 文件链接到了 xx.html 上。

于是就可通过链接来访问 Chrome 的 Cookie

> 1. Google 没有进行修复，只是让Chrome 最新版本默认禁用 file 协议，所以这一漏洞在最新版的 Chrome 中并不存在
> 2. 但是，在日常大量使用 WebView 的App和浏览器，都有可能受到此漏洞的影响。通过利用此漏洞，容易出现数据泄露的危险

如果是 file 协议，禁用 javascript 可以很大程度上减小跨源漏洞对 WebView 的威胁。

> 1. 但并不能完全杜绝跨源文件泄露。
> 2. 例：应用实现了下载功能，对于无法加载的页面，会自动下载到 sd 卡中；由于 sd 卡中的文件所有应用都可以访问，于是可以通过构造一个 file URL 指向被攻击应用的私有文件，然后用此 URL 启动被攻击应用的 WebActivity，这样由于该 WebActivity 无法加载该文件，就会将该文件下载到 sd 卡下面，然后就可以从 sd 卡上读取这个文件了

最终解决方案

- 对于不需要使用 file 协议的应用，禁用 file 协议；

```
复制代码
// 禁用 file 协议；
setAllowFileAccess(false);
setAllowFileAccessFromFileURLs(false);
setAllowUniversalAccessFromFileURLs(false);
```

- 对于需要使用 file 协议的应用，禁止 file 协议加载 JavaScript。

```
// 需要使用 file 协议
setAllowFileAccess(true);
setAllowFileAccessFromFileURLs(false);
setAllowUniversalAccessFromFileURLs(false);

// 禁止 file 协议加载 JavaScript
if (url.startsWith("file://") {
    setJavaScriptEnabled(false);
} else {
    setJavaScriptEnabled(true);
}
```