Supplementary Event-Listener Injection Attack in Smart Phones

S. Fouzul Hidhaya[1], Angelina Geetha[2]
B. Nandha Kumar[1], Loganathan Venkat Sravanth[1] and A. Habeeb[1]
[1] Department of Computer Technology, Anna University
Chennai, Tamil Nadu – 600 044 – India
[e-mail: fouzul_hameed@yahoo.com]
[2] Department of Computer science, B.S. Abdur Rahman University,
Chennai, Tamil Nadu – 600 048 - India
[e-mail: anggeetha@yahoo.com]
*Corresponding author: S. Fouzul Hidhaya

Received April 7, 2015; revised July 6, 2015; accepted August 5, 2015; published October 31, 2015

Abstract

WebView is a vital component in smartphone platforms like Android, Windows and iOS that enables smartphone applications (apps) to embed a simple yet powerful web browser inside them. WebView not only provides the same functionalities as web browser, it, more importantly, enables a rich interaction between apps and webpages loaded inside the WebView. However, the design and the features of WebView lays path to tamper the sandbox protection mechanism implemented by browsers. As a consequence, malicious attacks can be launched either against the apps or by the apps through the exploitation of WebView APIs. This paper presents a critical attack called Supplementary Event-Listener Injection (SEI) attack which adds auxiliary event listeners, for executing malicious activities, on the HTML elements in the webpage loaded by the WebView via JavaScript Injection. This paper also proposes an automated static analysis system for analyzing WebView embedded apps to classify the kind of vulnerability possessed by them and a solution for the mitigation of the attack.

Keywords: Android Security, WebView, Embedded browser, Smart Phone Security, Malicious Attacks, Java Script Injection

1. Introduction

Smartphones are one of the greatest inventions of the 21st century. As the popularity grows, so does the threats on the mobile platforms grows[1]. A major factor that has contributed to the wide spread popularity of smartphones are their applications. Most of these applications are web-based, owing to the societal requirements and demanding nature of updated information in people's day-to-day life. These applications get information from the web servers, display their contents in attractive forms and allow the users to interact with them. Though their functionality seems to be similar to the desktop web browsers there exists a significant difference between them. Desktop web browsers are generic and their features are independent from web applications whereas web based applications on smart phones are customized for specific web applications. This is achieved with the help of WebView component in smartphones. This component is known as WebView on Android OS [2], Windows Browser on Windows phone OS [3], UIWebView on iOS [4], Cascades WebView on BlackBerry OS [5] , Mojo WebView on Palm [6]. In general the term WebView is commonly used to refer all of the above. It includes the basic functionalities of desktop browsers such as page rendering, navigation and JavaScript execution. It also enables a two-way communication between the webpage and applications (apps). During apps to webpage interaction, apps can invoke JavaScript code within webpages, insert their own scripts into the webpages, monitor-intercept-respond to the events occurred within the webpages. During webpage to app interaction, JavaScript code in the embedded webpages accesses the native Java code using the interface registered to the WebView[7]. Bowsers can also be used to view these web applications in smartphones, but the extra step of opening a browser and typing in the address to open the application makes the use of WebView desirable in smartphones.

2. WebView- An Overview

WebView was designed to ease the use of web applications in smartphones. Apart from the 'ease of use' factor, Webviews are also designed , to provide the best 'look and feel' effect for the application. This can be materialised by using user- interface based Application Programming Interfaces (APIs) and Web based APIs. WebView as such can be benign, but may contain a vulnerable code. Certain methods of the APIs are defined to provide programming flexibility, so that the hosted application can be clear and concise. The use of these APIs can serve as a loop hole to launch an attack on system. The WebView class allows developers to display data from webpages. WebViews can load data using HTTP/HTTPS protocol and they can also load HTML files that are stored locally. Developers can also specify the layout and behavior of WebView according to their requirements thereby creating a custom embedded web browser. This customization feature available in the WebView component can be manipulated by the probers to inject their attack on the underlying operating system. To understand how malicious attacks are launched on WebView, it is essential to have a brief overview on the basic functionalities provided by them.

2.1 WebView APIs

WebView APIs can be broadly classified into two categories namely web-based and user-interface based APIs. The web-based APIs are used to load, display and navigate webpages while the UI based APIs are used for interacting with the webpages, altering the

webpage characteristics (properties) and handling the events that occur on the webpage loaded by the WebViews. The categories of the WebView APIs are given in **Table 1**.

Table 1. WebView API Classification

Web-based APIs	User-Interface based APIs
Communication – loadURL, addJavaScriptInterface.	Event Processing – dispatchKeyEvent, onTouchEvent.
Navigation – WebViewClient, ShouldOverrideUrlLoading.	Layout Management – setLayoutParams, scrollBy, scrollTo.
Display – pageDown, pageUp.	Properties – setAplha, setBackgroundColor.

2.2 WebView Event Handling

WebView enables Android applications to monitor, intercept and respond to the events that occur within the webpages. This is accomplished with the help of the functions provided by the WebViewClient class. WebViewClient provides a list of functions that will be triggered when their intended events take place inside the webpages. Once triggered these functions (hooks) can access event information and might even change the outcome of those events. Hooks that are commonly used and its associated event (purpose) is shown in **Table 2**.

Table 2. WebViewClient Hooks and its associated Events

HOOK	PURPOSE
onLoadResource	Notify the host application that the WebView will load the resource specified by the given url.
onPageStarted, onPageFinished	Notify the host application that a page has started/finished loading.
shouldOverrideUrlLoading	Notify to take over the control when a new url is about to be loaded in the current WebView.
shouldOverrideKeyEvent	Give the host application a chance to handle the key event synchronously.

2.3 Loading Webpages in WebView

Once a WebView control has been created, webpages can be loaded using the loadURL () API which takes the requested URL as its parameter. WebView supports loading resources from Web or from the assets or resource (res) folder of device. The syntax for each type is shown below along with an example in **Table 3**.

Table 3. Various modes for loading webpages inside WebView

- Loading mywebpage.html from the assets folder :
  myWebView.loadUrl ("file:///android_asset/mywebpage.html");
- Load an image logo.png from the res folder :
  myWebView.loadUrl("file:///android_res/drawable/logo.png");
- Loading a web based URL, Facebook in this case :
  myWebView.loadUrl("http://www.facebook.com");

3. Prerequisites for the Attack

The attacks that can be launched through WebView can be classified based on its APIs classification as Web-based attacks and User-Interface (UI) based attacks. JavaScript Injection [8], Excess Authorization [9], Cross-site scripting [10], Event sniffing and hijacking [8] belong to Web-based attack while Event simulation, KeyStroke Hijacking and Touch Jacking belong to UI based attack[11]. The attack being presented in this paper is based on the exploitation of JavaScript injection facility possessed by the WebView. Like the other attacks, this attack also makes an assumption just like other attacks that the application has been granted with android.permission.INTERNET permission which is essential for interacting with web servers and it is granted in most of the web-based applications.

3.1 WEBPAGE TO DEVICE BINDING

Android Java objects can be binded to the WebView by creating an interface using addJavaScriptInterface method provided by WebView. This interface can then be utilized for communication between the webpage and the device (specifically to call Java methods from JavaScript inside the webpages). The syntax for creating the interface is given in **Table 4**.

Table 4. Syntax for creating Interface

```
JavaScriptInterfaceClass JSInterface_obj = newJavaScriptInterfaceClass(this);
myWebView.addJavaScriptInterface(JSInterface_Obj, "Interface_Name");
```

- Where '*JavaScriptInterfaceClass*' is the Java class created by the developer which has the data (methods) to be accessed during the interaction between the webpage and device.

Supplementary Event-Listener Injection attack creates such an interface between the webpage and device to send the stolen information from the webpage to the device (Java code) for uploading it to the attacker's remote server.

3.2 JAVASCRIPT INJECTION

JavaScript, a dynamic programming language, used in the webpages allows client side scripts to interact with users, communicate asynchronously with servers and provide functionalities to modify the contents (DOM - Document Object Model) of the webpage. WebView paves a way to solicit JavaScript from Java. A WebView makes it possible to inject JavaScripts into the webpages loaded on them, thereby making the manipulation of the page contents possible. Though JavaScript execution is disabled by default it can be enabled by calling setJavaScriptEnabled method. JavaScript Injection is accomplished with the help of WebView's loadUrl API. If the URL string (loadUrl's parameter) begins with "javascript:" WebView treats it to be JavaScript content and executes the code followed by it on behalf of the web page (imposes the injected script to be of the webpage's origin) that is loaded by the WebView. A sample script that is injected to display an alert message in the loaded webpage is given in **Table 5**.

Table 5. Sample script

```
String alert_message=”THIS MESSAGE IS INJECTED FROM WEBVIEW” ;
myWebView.getSettings().setJavaScriptEnabled(true);
myWebView.loadUrl(“javascript:alert(alert_message);”);
```

Supplementary Event-Listener Injection attack uses this mechanism to inject the malicious script into the webpage for launching the attack.

4. Supplementary Event- Listener Injection Attack

Supplementary Event-Listener Injection attack registers subsidiary event listeners to HTML of the web pages, that are loaded by the WebView. This is in addition to their already existing primary event listeners (also known as default event listeners). This registration is through the JavaScript Injection mechanism discussed previously. Upon the occurrence of those events both the default and auxiliary event listeners will be triggered. While the default event listener proceeds with its intended task, the supplementary event listeners are used to manipulate the page contents (DOM) thereby enabling access to sensitive information present in the web pages through them.

To demonstrate this attack we use Gmail and Yahoo web pages as victims. The Login credentials of the users are stolen from those webpages and transmitted to the Java code using the webpage-to-device interface registered with the WebView. Prior to attack launch, the identifier (ID) of the necessary HTML elements are pre-determined. They will act as hooks for stealing the information associated with those elements. As most of the web-based applications are intended for loading specific webpages, it’s not a challenging task to get the ID of the required HTML elements from the web pages, by viewing the source-code of the webpage.In our attack demonstration, after the page loading event is complete, we associate a blur event to the input textboxes (email and password field) as the secondary event listener.

Blur event takes place when the HTML elements associated with it lose their focus. So when the user completes typing-in his/her Email id and then switches to password input field, blur event affiliated with the Email input field will be triggered. Then the data entered into the Email input field is retrieved with the help of its identifier (hook). Similarly when password field loses its focus it triggers the blur event linked with it which collects the data entered into the password field in the same fashion.

Now the gathered login credentials are sent to the Server of the hacker through the HTTP protocol. For demonstration purpose, the details are displayed using Android’s toast message to affirm the arrival of stolen information at the server end. The code snippet that performs the Supplementary Event Injection attack is given in **Table 6**.

Table 6. Snippets of SEI Attack.

```
String EMAIL_ID="Email", PASS_ID="Passwd";
 // GMAIL page input elements ID.
// YAHOO page input elements ID: EMAIL_ID="login-username", PASS_ID="login-passwd"

WebView WV= (WebView) findViewById(R.id.webview);
WV.getSettings().setJavaScriptEnabled(true);
WV.getSettings().setDomStorageEnabled(true);
WV.addJavascriptInterface(new MyJavaScriptInterface(),"webdroid");
```

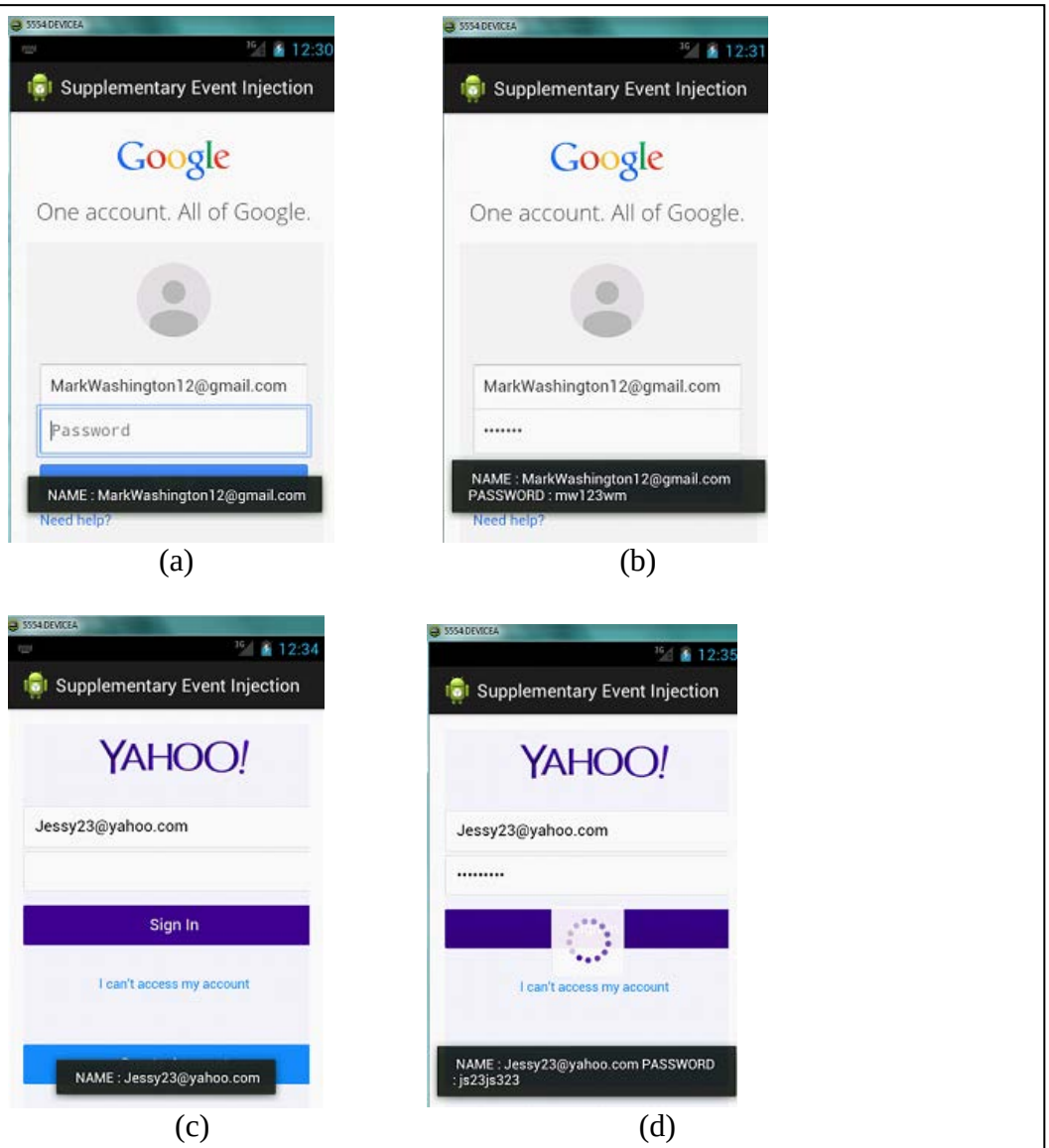
```
WV.setWebViewClient(new WebViewClient()
{
public void onPageFinished(WebView view, String URL)
{
super.onPageFinished(view, URL);
WV.loadUrl("javascript:document.getElementById(EMAIL_ID).addEventListener
  ('blur', function() { window.webdroid.send(' NAME : '+ document.getElementById
  (EMAIL_ID).value);}, true);");
WV.loadUrl("javascript:document.getElementById(PASS_ID).addEventListener
  ('blur', function() { window.webdroid.send(' PASSWORD : '+ document.getElementById
  (PASS_ID).value);}, true);");
 }
});

class MyJavaScriptInterface
{
public void send(String data)
  {
          Toast.makeText(MainActivity.this, data, Toast.LENGTH_LONG).show();
         //data can be uploaded to attacker's server from here.
   }
 }
```

Similar to blur event it is also possible to add a click event as well as focus event as secondary event listeners to the HTML elements. To demonstrate the usage of these events, we will explain how to make the user auto-login into some malicious Email account (created by attacker to spread spam mails) without user interference.

This is accomplished by adding a focus event listener to the Email input text field so that when the user interacts with Email input field for typing the name it will trigger the focus event which can be utilized to set the malicious Email id and its associated password in their respective input fields. The submit button's click event can be triggered using elementid.click (). So, the user is forced to login into the attacker's malicious Email account without interruption. **Table 8** shows the JavaScript utilized for launching the specified attack. **Fig. 1** shows the screenshot of the attacks.

Table 8. Snippet of the attack script

```
wv.loadUrl("javascript:document.getElementById('Email').addEventListener('focus', function()
 {
 document.getElementById('Email').value='malicious_mail_id@gmail.com';
document.getElementById('Passwd').value='malicious_mail_password';
document.getElementById('signIn').click();
}, true);");
```



(a, c) – Shows the Name (Email id) in Toast message once Email TextBox loses its Focus.
(b, d) – Shows the Password appended with Email id in Toast message once Password TextBox loses its Focus.

Fig. 1. Screenshot of the attack

5. Automated Static Analysis System

An automated static analysis system was developed to analyze the WebView embedded applications to predict the rate of vulnerability possessed by them, taking into account all the malicious attacks launchable through the WebViews.

The analysis system utilizes decision tables for generating inference rules, which will then be used for predicting the kind of vulnerability possessed by various WebView embedded apps. The system uses unique decision table for every malicious attack. The Analysis results are

presented in a comprehensive format to the users. The system is helpful to developers and distributors for analyzing the WebView embedded applications.
The decision table is to be related as follows:

- Each column of the decision table corresponds to an inference rule
- Each row of the table corresponds to a condition.
- Each entry of the decision table corresponds to any of the three values, namely YES (Y), NO (N) and not_signifiant (empty cells).
- Upon the execution and verification of all the inference rules the input APK is then classified in to one of the following categories:
    1. Harmless WebView
    2. Oblivious WebView
    3. Malicious WebView
- Harmless WebView: The input APK containing a WebView component is named as a Harmless WebView if and only if any of the inference rules produces negative results. That is, this kind of WebView does not have a vulnerability hole in its code. It doesn't own any code segment that opens portal for launching malicious attacks.
- Oblivious WebView: The input APK containing a WebView component is named as an Oblivious WebView if and only if the WebView contains code segments which can become potentially harmful upon manipulation. Though these WebViews don't pose any threat to the users they are on the verge of causing trouble.
- Malicious WebView: The input APK containing a WebView component is named as a Malicious WebView if and only if any of the inference rules produces positive results. This kind of WebView may cause an attack on the user's smartphone.

Some Inference rules classify the APK under more than one action namely a combination of Harmless and Oblivious WebView or Oblivious and Malicious WebView. In such cases the one with greater impact is given more importance namely Oblivious and Malicious for the above two cases respectively.

The decision table used for checking the existence of Supplementary Event Injection attack in WebView embedded apps is given in **Table 9**. The table is to be read as follows:
Rule 1:

a) If the APK has methods 'addJavaScriptInterface', 'setJavaScriptEnabled', 'setWebViewClient' , 'onPageFinished' , 'JavaScriptInterface', 'loadUrl', 'addEventListener', 'Focus Event added' available in the java code
b) And 'DOM element attributes', 'Click Event added', 'Blur Event added', 'setWebViewClient' method not included in the java code

Then the application is vulnerable to the supplementary event-listener Injection attack and so it is considered malicious.

Table 9. Decision Table for SEI Attack.

	RULES									
CONDITIONS	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
C1 : Check for addJavaScriptInterface	Y	Y	Y	Y	Y	Y	Y	Y	N	N
C2 : Check for setJavaScriptEnabled	Y	Y	Y	Y	Y	Y	Y	Y	N	Y
C3 : Check for setWebViewClient	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
C4 : Check for onPageFinished	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
C5 : Check for setDomStorageEnabled				Y	Y		Y			

C6 : Check for loadUrl	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
C7 : Check for JavaScript injection	Y	Y	Y	Y	Y	Y	Y	N	N	Y
C8 : Check for DOM element attributes	N	N	N	Y	Y	N	Y			N
C8.1 : Value attribute accessed				Y	N		Y			
C8.2 : Click attribute accessed				N	Y		Y			
C9 : Check for addEventListener	Y	Y	Y	N	N	Y	N			N
C9.1 : Focus Event added	Y	N	N			Y				
C9.2 : Click Event added	N	N	Y			Y				
C9.3 : Blur Event added	N	Y	N			N				
ACTIONS	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A1 : Harmless WebView								X	X	X
A2 : Oblivious WebView				X	X		X	X		X
A3 : Malicious WebView	X	X	X			X	X			

6. Solution for Attack Mitigation

Supplementary event-listener injection attack can be prevented by counter-acting mechanism. Adding auxiliary event listeners to existing ones contributes to the major part in this attack, so if we can prevent them from adding secondary event listeners this attack can be avoided.

This can be accomplished with the help of **event.stopImmediatePropagation()** property provided by the HTML which prevents subsidiary event propagation and activates only the default event listener associated with an HTML element.

Therefore, it is the responsibility of the webpage developer to set this property for all the elements capable of having an event listener, so that even if hackers try to inject supplementary event listeners through malicious script injection mechanism, those event listeners will not be triggered and only the default listener will be active all the time. Only then sensitive information held by the webpages can be safeguarded.

We implemented this solution on the WebViews which tried to inject supplementary event listeners. The event listeners injected by the malicious WebViews were not triggered at runtime and only the default event listeners were triggered.

7. Related Work

There had been considerable research carried out to address the security issues of WebView. Many researchers have pointed out the different attacks that are possible using the WebView feature of the smartphone and they have provided mitigating techniques for these attacks.
Luo and Hao [8] in their work, classifies the attacks as web-based and user-interface based attacks based on the exploitation of WebView APIs. They further classified web-based attacks into two categories namely attacks from webpages and attacks from applications. The attacks such as Excess-authorization attack, JavaScript Injection, Event sniffing and Hijacking, Frame confusion attacks and cross-site scripting attacks have been dealt.

Chin and Wagner [9] have developed a tool, Bifocals, for detecting the vulnerabilities inflicted to the smartphones. It characterizes the prevalence of vulnerable code in smartphone applications. Cross site scripting attacks done through the usage of WebView and its

associated APIs is dealt by Bhavani [10]. A mechanism for launching cross-site scripting attacks without using core WebView APIs is proposed. HTTP Client APIs are used to load malicious scripts dynamically from remote servers for launching cross-site scripting attacks. This paper presents session hijacking attack by stealing the cookies information. Accessing and extracting sensitive information like contacts stored on device using HTTP GET and HTTP POST APIs is also addressed in this work.

Xing Jin and Tongbo Luo [11] in their work introduced different types of touchjacking attacks such as WebView redressing attack, Invisible WebView/Transparent WebView hijacking attack, Key stroke hijacking attack and Event simulation attack.

Li and Greg Clark [12] investigated the potential threats that affect the mobile phone's security. They mentioned the risk factors involved in smartphone platforms, where the iOS and Android mobile OS implement some form of application privilege separation model. That is, the underlying kernel isolates each application into an execution sandbox that shields the application from unauthorized access to its data. An application makes access requests to gain the necessary permissions to protected resources—these permissions are granted either by the kernel or explicitly by the user. They also argue that the web-based applications on both iOS and Android have all the vulnerabilities found in their desktop counterparts. They criticize that the WebView technology adopted by both the iOS and Android systems to embed browser functionalities inside the applications can be exploited to tamper the protection mechanisms in the underlying mobile Operating system.

Yu and Yamauchi [13] in their work discussed the Javascript attacks carried out through webview. They have provided a solution for mitgating the attack dynamically by performing an access control on the certain APIs that are responsible for the security of the contents of the device.

Matthias Neugschwandtner et.al [14] gives an overview of the webview targeted threats. These threats has been distinguised as the server compromising attacks, like the SQL injection attacks, Cross-Site scripting attacks and the traffic compromising attacks, like the man-in-the-middle attack , HTML attacks, Javascript attacks.

Veelasha Moonsamy and Lynn Batten [15] in their work claim that 10% of the application from their dataset extract information using third party advertising libraries. A leaky application is defined to be an vulnerable application that is used by an third party to extract information. Webview was found to be one of the vulnerable application. This paper discusses how the leak occurs and the destination of the leaked data.

Batyuk and Leonid [16] in their work, targeted the problem of detection and mitigation of unwanted activities within Android applications. Their contribution is two fold. First, a static analysis service was described. This gives a detailed study of the application's internals and if it is comprehensible for an end-user. Second, a strategy for monitoring unwanted activities was presented.

A.D. Schmidt et.al [17] in their paper present an analysis of security mechanism in Android Smartphones with a focus on Linux. These results can be applicable to Android as well as Linux-based smartphones. Android framework and the Linux kernel have been analysed and the security functionalities have been compared. Well accepted security mechanisms and tools which could increase device security were surveyed. Details on how to adopt these security tools on Android platform and overhead analysis of techniques in terms of resource usage have been illustrated. Their second contribution focuses on malware detection techniques at the kernel level. The applicability of existing signature and intrusion detection methods in android platform was tested. Observation on the kernel, like identifying critical kernel event, log file,

file system and network activity events, and making efficient mechanisms to monitor them in a resource restricted setting was made. Using these observations a simple decision tree for deciding the suspiciousness of the application was constructed.

8. Conclusion and Future Work

Though WebView provides great features for the development of hybrid web-based applications they also create the portal for launching malicious attacks on smartphones and webpages. In this paper we have introduced the Supplementary Event-Listener Injection attack, which can be launched through JavaScript Injection mechanism. An automated static analysis system for scanning vulnerable WebViews has been developed. This is a critical attack as it results in the loss of sensitive private information from webpages and hence making the users insecure. In the future work we will enhance the automation, to perform dynamic analysis on the WebViews at run time and reduce the impact caused by the malicious attacks.

References

[1] "Threats increase on mobile platforms – especially Android – as popularity grows," *Network Security*, vol. 2014, Issue 3, March 2014, pp. 1-2, ISSN 1353-4858. Article (CrossRef Link).
[2] http://developer.android.com/reference/android/webkit/WebView.html (Accessed in June 2014)
[3] https://msdn.microsoft.com/en-us/library/windows/apps/ff431797(v=vs.105).aspx (accessed on June 2014)
[4] https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/ (accessed on June 2014)
[5] https://developer.blackberry.com/native/documentation/ui/webview/loadingwebcontent.html (accessed on June 2014)
[6] Allen, Mitch, "Palm webOS," *O'Reilly Media, Inc.*, 2009.
[7] Luo, Tongbo., "Attacks and countermeasures for WebView on mobile systems," 2014.
[8] T. Luo, H. Hao, W. Du, Y. Wang and H. Yin, "Attacks on WebView in the Android System," in *Proc. of 27th Annual Computer Security Applications Conference (ACSAC)*, pp. 343-352, 2011. Article (CrossRef Link)
[9] Erika Chin, and David Wagner, "Bifocals: Analyzing webview vulnerabilities in android applications," *Information Security Applications*, pp. 138-159, Springer International Publishing, 2014. Article (CrossRef Link).
[10] Bhavani, A. B., "Cross-site Scripting Attacks on Android WebView," *International Journal of Computer Science and Network (IJCSN)*, vol. 2, Issue2, April 2013.
[11] Tongbo Luo, Xing Jin, Ajai Ananthanarayanan, and Wenliang Du, "Touchjacking attacks on web in android, iOS, and windows phone," *Foundations and Practice of Security*, pp. 227-243. Springer Berlin Heidelberg, 2013. Article (CrossRef Link).
[12] Li, Qing, and Greg Clark, "Mobile security: A look ahead," *Security & Privacy, IEEE* 11, no. 1 (2013): 78-81. Article (CrossRef Link)
[13] Jing Yu and Toshihiro Yamauchi, "Access Control to Prevent Attacks Exploiting Vulnerabilities of WebView in Android OS," *High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC)*, pp. 1628-1633, IEEE, 2013. Article (CrossRef Link).
[14] Matthias Neugschwandtner, Martina Lindorfer, and Christian Platzer, "A View to a Kill: WebView Exploitation," *LEET*, 2013.
[15] Veelasha Moonsamy, and Lynn Batten, "Android applications: Data leaks via advertising libraries," *Information Theory and its Applications (ISITA), 2014 International Symposium on*, pp. 314-317, IEEE, 2014.

[16] Batyuk, Leonid, Markus Herpich, Seyit Ahmet Camtepe, Karsten Raddatz, A-D. Schmidt, and Sahin Albayrak, "Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications," *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*, pp. 66-72, IEEE, 2011.
Article (CrossRef Link).
[17] Schmidt, Aubrey-Derrick, Hans-Gunther Schmidt, Jan Clausen, Kamer A. Yuksel, Osman Kiraz, Ahmet Camtepe, and Sahin Albayrak, "Enhancing security of linux-based android devices," in *Proc. of 15th International Linux Kongress. Lehmann*, 2008.

S. Fouzul Hidhaya received Master in computer Science and Engineering from Anna University, Chennai, India and Bachelors in Electrical and Electronics Engineering from Madras University, Chennai, India. She is a Ph. D candidate in the Department of Computer Science and Engineering, B.S. Abdur Rahman University, Chennai. She is now working as a Teaching Fellow in Anna University, Department of Computer Technology. Her research interest includes Web Application security, Smartphone Security, Android applications and its vulnerabilities, etc..

Dr. Angelina Geetha received her Ph. D degree and Masters in engineering in computer Science and Engineering from Anna university, Chennai, in 2008 and 2001 respectively. She is now working as professor at the Department of Computer Science and Engineering, B.S. Abdur Rahman University, Chennai , India. Her area of expertise is Web Mining and information retrival. She has published numerous papers in International journals and conferences. She has been serving as chairs, program committee, or organizing committee chair for many international conferences and workshops.

B. Nandha Kumar received his Bachelors in Engineering in Computer science and Engineering from MIT Campus, Anna univerity, Chennai in 2015. He is now working as a programmer at Intellect Design Arena Ltd- A polaris group of company, Chennai , India. His area of interests are software development and testing.

Loganathan Venkat Sravanth received his Bachelors in Engineering in Computer science and Engineering from MIT Campus, Anna univerity, Chennai in 2015. He has a commendable score in GRE and TOFEL and is yet to get admission to a masters program. His area of interests are Software Engineering and Object Oriented analysis and Design.

A. Habeeb received his Bachelors in Engineering in Computer science and Engineering from MIT Campus, Anna univerity, Chennai in 2015. He is now a programmer at NMS labs Inc. Chennai, India. His area of interests are software development and testing.