13th International ISC Conference on Information Security and Cryptology (ISCISC2016)
September 7-8, 2016; Shahid Beheshti University – Tehran, Iran

Fine-Grained Access Control for Hybrid Mobile Applications in Android Using Restricted Paths

Shahrooz Pooryousef
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
pooryousef@ce.sharif.edu

Morteza Amini
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
amini@sharif.edu

***Abstract*—Hybrid Mobile Applications are a new generation of mobile applications that have recently introduced new security challenges. In these applications, untrusted web content, such as an advertisement inside an embedded browser, has the same privileges as the entire application and can directly access the device resources. Unfortunately, existing access control mechanisms are very coarse-grained and do not provide adequate facilities for fine-grained access rule definition and enforcement in hybrid mobile applications. In this paper, we propose a fine-grained access control mechanism for privilege separation in hybrid mobile applications. Our proposed access control mechanism, called RestrictedPath, enables developers to define separate paths inside the application in which each path has restricted permissions. For preparing a fine-grained access control at the Android framework layer, RestrictedPath enforces access control at two different levels; browser level and Android access control system level. We have developed a proof-of-concept prototype of RestrictedPath for the Android open source project version 4.4.3 to illustrate its feasibility and to evaluate its overhead on the system. Our experiments show that RestrictedPath is practical, easy to use for developers, and has low performance overhead (in average 10 percent) on the device.**

I. Introduction

Hybrid Mobile Applications are a new generation of mobile applications that integrate features of web and native mobile applications [1]. In these applications, like web based applications, most of the content is loaded from web servers at run time. An embedded browser, for example WebView in Android, loads and executes web content like HTML, CSS and JavaScript libraries [2]. With exposing interfaces on embedded browser, web content can invoke native codes (for example Java in Android), and reach device resources outside the browser [3]. As the main part of these applications is developed with web technologies, they can run on multiple platforms such as Android, iOS, and Windows Phone [1]. Because of Android huge market share [4], in this paper, we target Android as a reference platform and the WebView as the hybrid mobile application's embedded browser. As Gartner report in 2016 [5], half of the mobile applications will be hybrid mobile applications in near future.

Hybrid mobile applications have recently introduced new security challenges. Exposing interface on embedded browser, breaks browser sand box property [3]. Browser sand box property contains the behaviors of web content and does not allow JavaScript to access the system resources outside the browser [3]. Hybrid mobile applications can load untrusted web content such as advertisements and social web content inside themselves. This untrusted web content has the same privileges as the entire application and can invoke application native APIs. However, when developers include some untrusted web content in the application embedded browser, it is treated as part of the host application web content. Same Origin Policy [6] in an embedded browser, only protects web content from different origins inside the browser, and does not apply to resources outside the browser.

Access control in hybrid mobile applications has been considered in several studies [2], [1], [7]. Some studies such as [3], [8] and [2] have focused on breaking browser sandbox property and proposed different mechanisms for privilege separation in these applications. Unfortunately, proposed privilege separation solutions are not so useful in restricting untrusted web content behaviors. Specifically, these approaches suffer from two problems, which are elaborated in the following.

a) Coarse-Grained Access Control: Some of the proposed access control systems such as NoFRAK [1], are very coarse-grained and do not prepare any mechanism for developers to define separate permissions for web content of different origins. In other words, if web content from an origin inside an embedded browser be allowed to invoke native APIs, it would be able to access all of the application native APIs. While in hybrid mobile applications, web content from different origins may need different permissions and invoke different APIs at the native side.

b) Unprotected Native APIs: Some other access control models which propose a fine-grained access control, cannot restrict undesirable behaviors from untrusted web content completely. For example, mechanisms such as definition of Android permissions for web content origins [2], cannot protect all native APIs from untrusted web content. Because invoking application native APIs does not invoke Android access control mechanism necessarily. For example, untrusted web content can use native APIs for sending messages to other applications, while we do not have any permission in Android for restricting this action. Having access to device public resources and APIs can be dangerous in some situations [9], [10], and these APIs must be protected from untrusted web

978-1-5090-3949-4/16/$31.00 ©2016 IEEE

content in hybrid mobile applications.

In this paper, we propose a fine-grained access control mechanism for hybrid mobile applications in Android. Using our proposed access control, developers can define different permissions for application's different parts. For resolving the above mentioned problems, our proposed access control, called RestrictedPath, enforces access control at two different points. For access control at first point, we have modified WebView component so that this component controls all method invocations between the application's web and native side. This can help us to control all native APIs invoking, even those that do not invoke Android privileged APIs. Since we cannot identify the Android APIs which are invoked on application's native APIs at this point, we need to control Android APIs invoking at another point. We have added a new module, called Path Manager, to Android framework layer to control Android privilege services namely SMS and contact list. Path Manager in framework layer, controls Android sensitive security APIs and makes decisions based on the permissions that developers have defined for the web and native side.

In this paper, we make the following contributions:

- We introduce a fine-grained access control mechanism at Android framework level for hybrid mobile applications. Our access control could restrict untrusted web content behaviors in applications and can be used easily by developers.
- To demonstrate the applicability of our approach, we designed and implemented a prototype of RestrictedPath in the Android open source project version 4.4.3.

In the rest of this paper, we first overview the structure of hybrid mobile applications and Android access control model in Section II. Our proposed access control mechanism is discussed in Section III. In Section IV the overhead of RestrictedPath is analyzed. Related work is surveyed in Section V, and the paper is concluded in Section VI.

II. Background

A. Hybrid Mobile Applications Structure

The structure of hybrid mobile application is depicted in Fig. 1, which consists two parts. In one side we have application's native APIs which is developed with native languages such as Java in Android and Objective-C in iOS. In the other side there is an embedded browser that loads and executes web content. Usually developers use middleware frameworks such as PhoneGap [11] for developing hybrid mobile applications. These frameworks prepare diverse native plugins for mobile device resources such as SMS, telephony, and contact list. Hybrid mobile applications in Android, can register Java objects to WebView through AddJavaScriptInterface method [3]. Then, application web content can invoke all the public methods of these Java objects. An example of using this method is shown in Fig. 2. Web content such as JavaScript functions within the WebView can use FUtil or CM to invoke the methods in FileUtils and ContactManager Java classes.

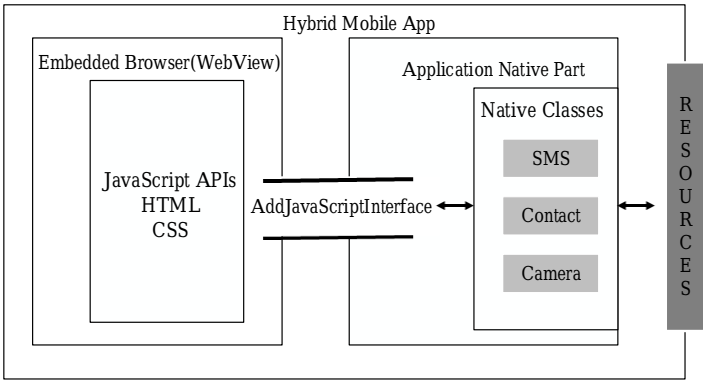

Fig. 1. Hybrid mobile application architecture [12].

```
WebView WV = new WebView();
WV.addjavascriptinterface(new FileUtils(),"Futil");
WV.addjavascriptinterface(new ContactManager(),"CM");
```

Fig. 2. Registering Java objects into WebView [3].

B. Android Access Control Model

Android is the most popular operation system in mobile platforms market share [4]. Android applications are usually developed in Java language. Android applications are distributed as APK (Android Package) files, which consist of the application's Manifest, resources, and application bytecode. Hybrid mobile applications are installed in Android such as Android native applications and a unique user ID (UID) is assigned to each of them at the installation time [13]. This allows the OS to differentiate between installed applications.

In order to protect privileged resources, Android uses the permission concept. For some resources such as SMS, telephony, and contacts list, Android provides some services which have been implemented at framework layer of Android architecture [13]. Applications can interact with these services using Binder [14] mechanism. Binder is Android's lightweight inter-process communication mechanism. Android access control module at framework level (Package Manager) checks whether applications have sufficiently permissions for calling these services or not. For some other permissions, such as SD Card, Bluetooth, and Internet, access control is applied at kernel level, which leverages application's UID and resources group ID (GID) for this purpose [15].

III. Proposed Access Control Mechanism

Fig. 3 provides an overview of our proposed access control mechanism called RestrictedPath. The highlighted modules are new modules in Android structure which is required for our proposed access control mechanism. In the following we describe these modules and the proposed access control procedure.

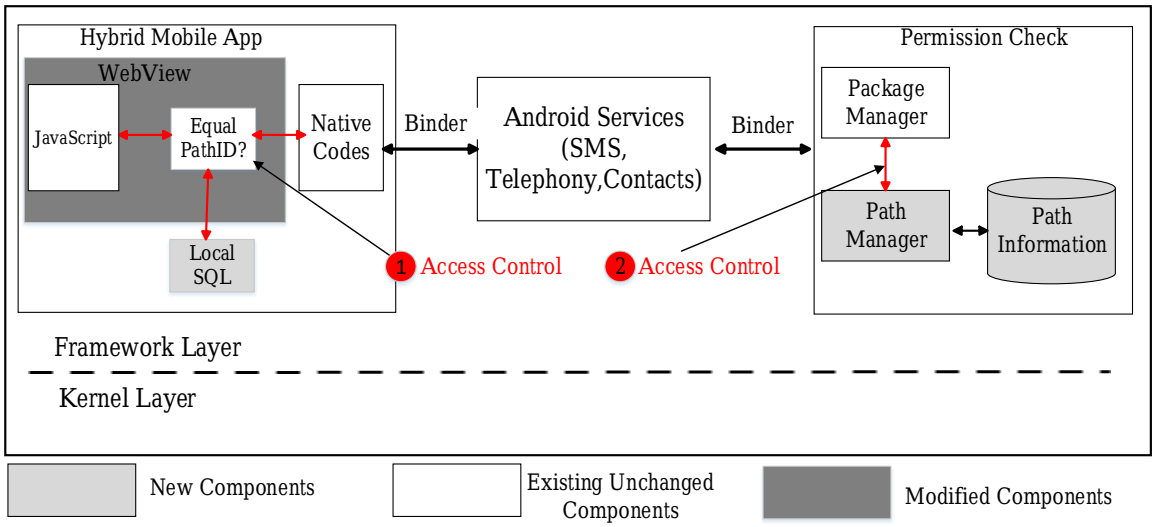

Fig. 3. The Architecture of proposed access control mechanism.

A. Path Definition for Privilege Separation

In order to provide a fine-grained access control in hybrid mobile applications, we need to define an entity or concept for privilege separation inside the application. Using only web content origin is not enough for privilege separation in hybrid mobile applications; because this yields a coarse-grained access control. In hybrid mobile applications, to allow a JavaScript to access Android APIs, first a JavaScript from a specific origin, invokes application's native APIs, and then these APIs interact with Android APIs. Hence we have a specific path from the JavaScript to Android APIs for invoking Android APIs. Our proposed access control helps developers to define different paths in applications and access control goes on the permissions of these paths. In other words, using this approach, we will have separate paths which behaviors of each path are restricted.

For path definition in applications, we have defined new tags in Android system. These include <PathID>, <origin>, and <PathPermission> tags. Developers can define their path information such as origin of web content and path permissions at development time. An example of path definition has been shown in Fig. 4. These tags are parsed by Package Manager when an application is installed. In our design, we extended Package Manager module to retrieve paths information by parsing <PathID>, <origin>, and <PathPermission> tags in the Manifest file. In attaching native objects to WebView component with AddjavascriptInterface method, we have added a new argument to AddJavaScriptInterface method. Thus, developers can determine the path ID in interface definition as depicted in Fig. 5. The path ID, native method name, and the origin name of related path are saved in a local database in the application. Our modified WebView uses these information for checking the web and native interaction at run time.

```
<Path android:name = "facebookPath"
Android:id = "1000"
 origin:value = "facebook.com"
PathPermission:name = android:permission:INTERNET  />

<Path android:name = "MyAppPath"
Android:id = "2000"
 origin:value = "myapp.com"
PathPermission:name = android:permission:INTERNET
PathPermission:name = android:permission:CONTACTS />
```

Fig. 4. Path definition in application Manifest file.

```
WebView WV = new WebView();
WV.addjavascriptinterface(new FileUtils(),"Futil",1000);
WV.addjavascriptinterface(new ContactManager(),"CM",2000);
```

Fig. 5. Interface definition in modified WebView.

B. Access Control Process in RestrictedPath

For each path, we must apply access control at two points. In RestrictedPath, when web content invokes application native APIs, our modified WebView checks the origin of web content and identify its PathID. If web content's PathID and native method's PathID be same, invocation is accepted. Note that, we do not check the permission of paths at this point. Invoked native APIs may not invoke Android privileged APIs necessarily, but must be protected from untrusted web content. As we mentioned in Section I-0b, some of interactions between application's two sides do not invoke Android privileged APIs necessarily at the end but they could be dangerous [9], [10].

If we only enforce access control at this point, our access

control would be coarse-grained. Because at this point, we cannot identify which Android privileged APIs the native API is going to invoke. Thus, we need to control Android APIs which are invoked from the application's native part too. We modified the Package Manager module, at the core of Android access control, to redirect the permission checking control flow to our Path Manager module whenever a permission check event occurs. In particular, Path Manager identifies the PathID from the native object name and then checks the path permissions. If it has the requested permission for invoking that API, Path Manager returns a true to Package Manager.

IV. Evaluation

In order to evaluate the run time overhead and efficiency of RestrictedPath approach, we implemented our access control mechanism in Android open source project [16], version 4.4.3 (Jelly-Bean). For overhead evaluation, compiled source code has been run in compiled source code default emulator in Ubuntu 14.04 (64bit) with 8G RAM and dual core CPU. Also we developed a sample hybrid application with compiled source code. With this sample application we tested our access control mechanism for preparing fine-grained access control at path level.

A. RestrictedPath Overhead

For performance evaluation, we compared consumed time for invoking Android privileged API in original and our modified version of Android operating system. For evaluation of consumed time, we employed the following pseudo-code in our experiments:

- time_start := System.current Time
- for i:=1 to 8 do
  - invoke read SMS API 8*i times
- time_end := System.current Time
- execute_time := time_end time_start

Permission checking overhead is shown in Fig. 6. The maximum difference between the original and the modified version using our proposed access control mechanism is 70 ms. The experimental results show that the proposed access control mechanism (RestrictedPath) imposes 8% runtime overhead. This is due to the fact that permissions are checked at two points in RestrictedPath access control process; first in Package Manager and then in Path Manager.

We also used Antutu benchmark to check the RestrictedPath overhead on system performance. This benchmark produces an overall score for performance of a device using different measures such as memory and CPU performance. A higher number for overall performance indicates the better performance. By using RestrictedPath in different loads (Low, Medium, and High), we have maximum 50 score difference in comparison to the case of using the original access control mechanism in the system (see Fig. 7). Low, Medium, and High load means we ran 4, 8, and 12 hybrid mobile applications on the device respectively.

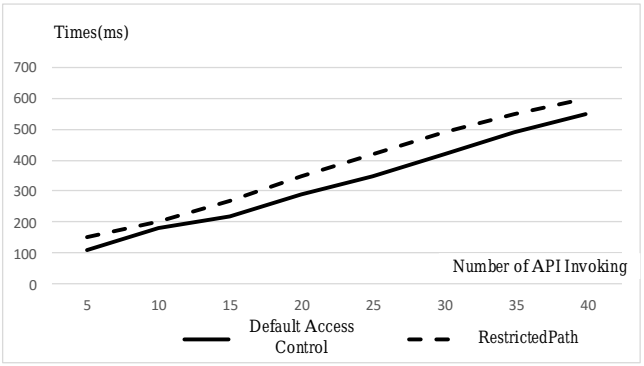

Fig. 6. Path permission checking overhead in RestrictedPath.

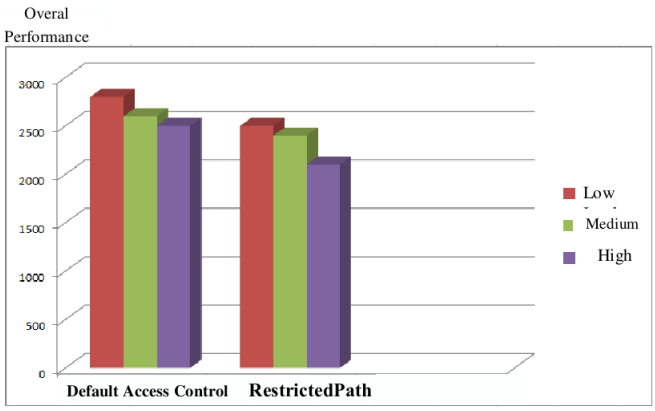

Fig. 7. System performance in case of using RestrictedPath.

B. Attack Scenarios and Defense in RestrictedPath

Here we use a sample application- which has been developed by our compiled source code- in order to show how RestrictedPath can achieve privilege separation for different paths. In our sample application, web content is loaded from two different origins and there are two paths. In the first attack scenario, the web content from one path invokes a native method from another path. As shown in Fig. 8, this interaction has been denied with modified WebView component. In the second attack scenario, a method of a native object from one path which does not have SEND_SMS permission, tries to read and send the user's SMSs. Fig. 9 illustrates that this action is denied by our Path Manager component.

V. Related Work

In [17], [8], [18] and [19] security concerns of WebView component in Android applications are analyzed. For first time Lou et al. [3] described the problems of using addJavascriptInterface method in WebView. After that, access control in hybrid mobile applications was considered in several studies. MobileIFC [20] controls information flows inside hybrid mobile applications and prepares a mechanism for developers to define new chunks to control information flows. By modifying hybrid mobile framework, developers can define policy for these chunks using XACML language

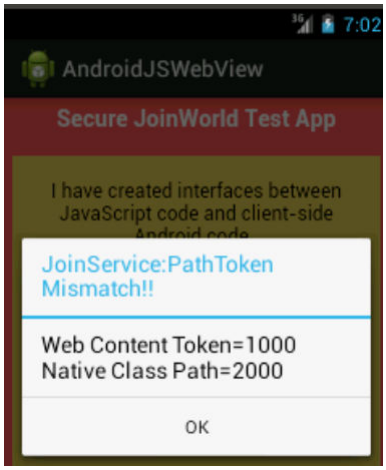

Fig. 8. Native method invocation control in RestrictedPath.

```
Caused by: java.lang.SecurityException: Permission Denial: opening provider com.android.providers.telephony.SmsProvider from ProcessRecord{410373e8 916:com.example.androguard/u0a10065} (pid=916, uid=10065) requires android.permission.READ_SMS or android.permission.WRITE_SMS
```

Fig. 9. The error message shown when the access to the Android SendSMS API is denied.

[21]. Rastogi et al. [22] provide a general understanding of attacks in mobile applications through the app-web interface in which a user may go to a malicious Web destination via advertisements inside the application. Georgiev et al. [1] have analyzed hybrid mobile frameworks white list mechanism and found that hybrid frameworks do not properly compose access control policies governing web code. NoFRAK [1] is a coarse-grained access control mechanism which enforces same origin policy using a local DB (SQLite) to store whitelisted domains with unique tokens. For enforcing access control in NoFRAK, some JavaScript libraries have been added to hybrid mobile frameworks. Jin et al. [2] changed Android access control model and defined Android permissions for frames in the web side.

In Table I, we briefly compare existing access control mechanisms with RestrictedPath. As Table I summarizes, none of the surveyed mechanisms can restrict all undesirable behaviors from untrusted web content. In comparison to the existing approaches, RestrictedPath, by applying access control at two different points, can prepare a fine-grained access control for Android privileged services at framework level. The comparison has been categorized based on the four features that have been introduced for this purpose in the literature.

Advertisements in native mobile applications have the same privileges as their host applications. Several works have provided mechanisms for separating advertisements' privileges from the host applications' privileges. AdSplit [23] runs advertisements' components in a separate process and sets limited permissions for advertisements hosting process. In AdDroid [24] a customized SDK (Software Development Kit) has been proposed which supports specific permissions for advertisements. Some studies such as [25], [26] and [27] conduct an study on the hidden costs of ad rendering, including network usage, and battery power. [28] suggests to track the flow of information between users and ad networks to optimize energy consumption in mobile advertisements. In practice, employing privilege separation in native mobile applications for access control in hybrid mobile applications is not a proper solution; due to the fact that the structures of the hybrid mobile applications is different from the native ones. In native mobile applications, advertisements have separate components and developers load these components in their applications. In contrast, in hybrid mobile applications, advertisements and untrusted web content are loaded in the embedded browser and can invoke applications' native APIs.

Using advertisements in pure web applications can endanger the security of users and host applications. Many works have been proposed to limit privileges of untrusted JavaScript inside web applications. Adjail [29] loads advertisement content in a separate sandbox with least privileges. ConScript [30] defines policy in client side for untrusted web content. Some works namely Adsafe [31] and Caja [32] use safe JavaScript subset for developing web applications in which dangerous APIs have been eliminated. Akhawe et al. [33] propose a mechanism for separation of HTML5 based application parts using an approach based on temporary origin.

VI. Discussion and Conclusion

Preparing the security of hybrid mobile applications is essential for success in their vast popularity. In this paper, we proposed a new access control mechanism, called RestrictedPath, which enforces access control on applications in different points. In our proposed mechanism, developers can define separate paths with restricted permissions and behaviors in their applications. We have implemented RestrictedPath on Android source code version 4.4.3 and evaluated its overhead on system performance and Android APIs invoking. In practice, our proposed access control has some limitations. RestrictedPath can control access to privileged resources at framework layer only. In other words, the proposed solution can not prepare access control for resources that are controlled by Android kernel. In future work, we will modify Android kernel access control for restricting resources such as SD Card and Bluetooth.

Acknowledgment

The authors would like to thank Yifei Wang and Xing Jin for providing the source code of their implementation.

References

[1] M. Georgiev, S. Jana, and V. Shmatikov, "Breaking and Fixing Origin-based Access Control in Hybrid Web/Mobile Application Frameworks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014, pp. 1–15.

[2] X. Jin, L. Wang, T. Luo, and W. Du, "Fine-Grained Access Control for HTML5-based Mobile Applications in Android," in *Information Security*, ser. Lecture Notes in Computer Science. Springer, 2015, vol. 7807, pp. 309–318.

TABLE I
COMPARE RESTRICTEDPATH WITH PREVIOUS PROPOSED ACCESS CONTROL MECHANISMS.

Proposed Access Control	Fine-grained	Restricting Undesirable Behaviors	Need to Change Application Source Code	Need to Modify OS Access Control
MobileIFC [20]	✓	✓	×	✓
NoFRAK [1]	×	×	✓	✓
Jin et al. [2]	✓	×	✓	×
RestrictedPath	✓	✓	✓	×

[3] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, "Attacks on WebView in the Android System," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 343–352.

[4] I. D. Corporation. (2014) International Data Corporation Says Android Dominated the Smartphone Market with a Share of 82.8% in 2015 Q2. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp Access time:1 April. 2016

[5] Gartner. (2007) Gartner Recommends a Hybrid Approach for Business-to-Employee Mobile Apps. [Online]. Available: http://gartner.com/newsroom/id/2429815 Access time:13 April. 2015

[6] J. Ruderman. (2001) The Same Origin Policy. [Online]. Available: http://www.mozilla.org/projects/security/components/same-origin.html5 Access time:1 July. 2015

[7] M. Shehab and A. AlJarrah, "Reducing Attack Surface on Cordova-based Hybrid Mobile Apps," in *Proceedings of the 2nd International Workshop on Mobile Development Lifecycle*. ACM, 2014, pp. 1–8.

[8] E. Chin and D. Wagner, "Bifocals: Analyzing Webview Vulnerabilities in Android Applications," in *Proceedings of the International Workshop on Information Security Applications*. Springer, 2013, pp. 138–159.

[9] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, Location, Disease and More: Inferring Your Secrets from Android Public Resources," in *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*. ACM, 2013, pp. 1017–1028.

[10] R. Wang, L. Xing, X. Wang, and S. Chen, "Unauthorized Origin Crossing on Mobile Platforms: Threats and Mitigation," in *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*. ACM, 2013, pp. 635–646.

[11] J. M. Wargo, *PhoneGap Essentials: Building Cross-Platform Mobile Apps*. Addison-Wesley, 2012.

[12] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri, "Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 66–77.

[13] M. Backes, S. Bugiel, S. Gerling, and P. von Styp-Rekowsky, "Android Security Framework: Extensible Multi-layered Access Control on Android," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 46–55.

[14] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Towards Taming Privilege-Escalation Attacks on Android," in *Proceedings of the Network & Distributed System Secuity Symposium (NDSS)*, vol. 17, 2012, p. 19.

[15] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege Escalation Attacks on Android," in *Proceedings of the International Conference on Information Security*. Springer, 2010, pp. 346–360.

[16] Google. (2007) Android Open Source Project. [Online]. Available: https://source.android.com// Access time:1 April. 2016

[17] T. Luo, X. Jin, A. Ananthanarayanan, and W. Du, "Touchjacking Attacks on Web in Android, iOS, and Windows Phone," in *Proceedings of the International Symposium on Foundations and Practice of Security*. Springer, 2012, pp. 227–243.

[18] T. Sutcliffe and A. Taylor, "The Lifetime of Android API Vulnerabilities: Case Study on the JavaScript-to-Java Interface," in *Proceedings of the 23rd International Workshop on Security Protocols*, vol. 9379. Springer, 2015, pp. 126–138.

[19] B. Hassanshahi, Y. Jia, R. H. Yap, P. Saxena, and Z. Liang, "Web-to-Application Injection Attacks on Android: Characterization and Detection," in *Proceedings of the European Symposium on Research in Computer Security*. Springer, 2015, pp. 577–598.

[20] K. Singh, "Practical Context-aware Permission Control for Hybrid Mobile Applications," in *Proceedings of the International Workshop on Recent Advances in Intrusion Detection (RAID)*. Springer, 2013, pp. 307–327.

[21] M. Verma. (2004) XML Security: Control Information Access with XACML. [Online]. Available: http://www.ibm.com/developerworks/xml/library/x-xacml/ Access time:12 June. 2016

[22] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. Riley, "Are These Ads Safe: Detecting Hidden Attacks through the Mobile App-Web Interfaces," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2016.

[23] S. Shekhar, M. Dietz, and D. S. Wallach, "Adsplit: Separating Smartphone Advertising from Applications," in *Proceedings of the 21st USENIX Security Symposium*, 2012, pp. 553–567.

[24] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege Separation for Applications and Advertisers in Android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. ACM, 2012, pp. 71–72.

[25] J. Gui, S. Mcilroy, M. Nagappan, and W. G. Halfond, "Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers," in *Proceedings of the 37th International Conference on Software Engineering*, vol. 1. IEEE Press, 2015, pp. 100–110.

[26] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating Ad Fraud in Android Applications," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2014, pp. 123–134.

[27] A. Albasir, K. Naik, B. Plourde, and N. Goel, "Experimental Study of Energy and Bandwidth Costs of Web Advertisements on Smartphones," in *Proceedings of the 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*. IEEE, 2014, pp. 90–97.

[28] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo, "Don't Kill My Ads! Balancing Privacy in an Ad-Supported Mobile Application Market," in *Proceedings of the 12th Workshop on Mobile Computing Systems & Applications*. ACM, 2012.

[29] M. Ter Louw, K. T. Ganesh, and V. Venkatakrishnan, "AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements," in *Proceedings of the 19th USENIX Security Symposium*, 2010, pp. 371–388.

[30] L. A. Meyerovich and B. Livshits, "ConScript: Specifying and Enforcing Fine-Grained Security Policies for Javascript in the Browser," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 481–496.

[31] D. Crockford. (2008) Adsafe. [Online]. Available: http://www.adsafe.org// Access time:1 April. 2015

[32] Google. (2008) A Source-to-Source Translator for Securing JavaScript-based Web Content. [Online]. Available: http://code.google.com/p/google-caja/ Access time:1 April. 2016

[33] D. Akhawe, P. Saxena, and D. Song, "Privilege Separation in HTML5 Applications," in *Proceedings of the 21st USENIX Security Symposium*, 2012, pp. 429–444.