2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering

Let me join two worlds! Analyzing the Integration of Web and Native Technologies in Hybrid Mobile Apps

Shahrooz Pouryousef
College of Computer Science
University of Massachusetts Amherst
MA,USA
shahrooz@cs.umass.edu

Mariam Rezaiee
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
mrezaiee@ce.sharif.edu

Ata Chizari
Faculty of Science and Technology
University of Twente
Netherlands
a.chizari@utwente.nl

***Abstract*—We can notice that security problems of inappropriate integration of native and web technologies in hybrid mobile applications (apps) have been covered in the related state-of-the-art research. However, analyzing hybrid mobile apps' unique behaviors has been seldom addressed. In this paper, we explore the influence of native and web technologies integration in hybrid mobile apps on the generated profile of mobile applications. Specifically, we analyze type of Security Sensitive APIs (SS-APIs) exposed to web content and identify the corresponding usage patterns by systematically tracking function-call-graphs of a large number of hybrid and native mobile apps. Our investigations indicate that the generated profiles for hybrid and native mobile apps are considerably different. Using our proposed tool, called Hybrid-scanner, for tracking and analyzing internal behaviors of hybrid mobile apps, we show that there are more trace of API calling for triggering an specific SS-API in a hybrid mobile app in comparison with Android native mobile apps. In addition, we have found that almost 40% of SS-APIs in hybrid mobile apps are invoked by third party libraries, e.g. advertisement libraries. This knowledge, however, is crucial for designing appropriate malware detection or vulnerability mitigation strategies. Based on our results, we discuss two main approaches in Android malware analysis field and enumerate some suggestions which should be considered in order to successfully detect malicious behaviors in such new type of apps.**

I. Introduction

Third-party mobile applications (apps) are usually written in native languages such as Java and C [1]. In this paper, native language in native mobile apps is referred as native technology. It can interact with system native application programming interfaces (APIs) directly and provide the users with a pleasant experience. However, developers need to extend their native apps using different languages in order to support multiple platforms. In contrast to native apps, Web-based mobile apps are developed in web technologies which are platform-independent. Such apps can be maintained and managed easily by changing the web content on the server. Hybrid mobile apps are a new generation of mobile apps which integrate features of the web and native mobile apps [2]. By integrating native and web technologies, hybrid mobile apps realize satisfactory user experience and also can run across different platforms. As a result, more and more apps are developed in the hybrid form [3].

Enhancing web content functionality and integrating native and web technologies in hybrid mobile apps can be a double-edged sword. Although app's functionality acquires good performance by exposing system native APIs to web content, inappropriate combination of such technologies potentially create new security and privacy concerns [2], [4]. There is a broad body of work on analyzing inappropriate access control models of mobile platforms for these type of apps [2], [5]–[10]. What such studies have in common is that they mostly investigate fracking attacks on mobile apps which allow untrusted web content being included into a hybrid app, e.g. advertisements (ads), in order to directly access device resources [2], [4], [11]. However, there have been no rigorous studies to quantify hybrid mobile apps' unique behaviors and the way the integrating native and web technologies can influence application generated profile. It has been also noticed that the proposed approaches for auditing and filtering out malicious apps for third-party mobile apps have been designed based on software stack of native mobile apps. In hybrid mobile apps, in contrast, there exists a combination of native and web technologies and consequently a different software stack.

The widespread use of hybrid mobile apps necessitates proposing new approaches to detect, understand, and measure their unique behaviors. The objective of this paper is to analyze unique features of hybrid mobile apps in addition to exploring their influence on the state-of-the-art schemes proposed for mobile malware analysis. Specifically, we made an attempt to address the following questions:

1. What is the effect of combining native and web technologies in hybrid mobile apps on the generated profile of apps? Dynamic nature of hybrid mobile apps could cause different patterns of **S**ecurity **S**ensitive **API**s (**SS-API**s) invoking at runtime. Dynamicity in native app's source code can be detected by searching keywords such as DexClassLoader [12] or JavaReflection [13]. Nevertheless, in hybrid mobile apps independent of using such APIs, dynamicity in application

2324-9013/18/31.00 ©2018 IEEE
DOI 10.1109/TrustCom/BigDataSE.2018.00274

behavior can be achieved.

2. How third-party JavaScript and advertisement libraries can influence the generated profile of APIs invoking in hybrid apps? Considering a hybrid mobile app that integrates untrusted third-party libraries such as advertisement libraries in its content, it is demanding to provide knowledge about the number of application's **SS-API**s being triggered by such libraries.

To answer those questions, we experimentally track byte-codes traces extracted from a large corpus of real-world hybrid mobile apps and their web codes using a data set including $5,534$ hybrid and $2,534$ native mobile apps. Specifically, we analyze the type of **SS-API**s exposed to web content and investigate their usage patterns by extracting application function call graph using our proposed framework, called Hybrid-Scanner. Among other results, we find that the generated profiles of function-call-graph for hybrid and native mobile apps are considerably different. As a measurement of application behavior dynamicity, we show that there is more trace of API calling for triggering a specific **SS-API** in hybrid mobile apps in comparison with Android native mobile apps. In addition, we show that almost 40% of **SS-API**s invoking in hybrid mobile apps are being triggered by third-party libraries such as advertisement libraries.

We believe that this knowledge must be considered in those mobile app analysis approaches which focus on application high-level behavior analysis or those which have been designed based on native mobile apps' software stack, for instance, [14] and [15]. In Section V, two main approaches proposed for Android malware analysis will be discussed followed by limitations regarding hybrid mobile apps analysis. Then, we enumerate some suggestions which should be considered in a malware analysis or a mobile fuzzer tool to successfully detect malicious behavior in hybrid mobile apps.

The rest of this paper is organized as follow. Section II presents the Hybrid-Scanner tool as a static analysis framework proposed for hybrid mobile apps analysis. Then the Hybrid-Scanner component's functionality will be described describe. After that, we present the results of applying Hybrid-Scanner on a large number of hybrid and native mobile apps in Section III. Finally, Section IV presents related work and in Section V we will conclude the paper and discuss our research, its limitations, our suggestions, and future works.

II. Proposed Static Analysis Tool

In this section, we explain our proposed tool, entitled Hybrid-Scanner, for tracking and analyzing internal behaviors of hybrid mobile apps. Fig. 1 shows main modules of our proposed framework. Hybrid-Scanner tracks app's web and native sides interactions, SS-APIs, and correlates native and web codes.

At first, app's APK file is disassembled to Smali files using android-Apktool [16]. Smali codes [17] are a representation of the app's code using Android Dalvik opcodes, essentially an intermediate representation of the code between the original Java source and the processor-specific assembly instructions. For application's native side analysis, **Native Tracker** extracts all **SS-API**s, call graph of these **SS-API**s, and all native exposed APIs to the web side. Our definition of **SS-API**s are Android privileged APIs but not so accurate to exhibit application behaviors at low-level aka application's system calls. Since we directly work on the application Smali codes and neither need to get the source codes of the application, nor decompile the Dalvik byte-codes to Java source code, obfuscation methods are not taken into account in our analysis process.

A. Definition of Path Concepts

In hybrid mobile apps, each **SS-API** from application web or native side can trigger or be triggered by another **SS-API** from another side. We define a path as a trace of function calls which its entry point is an API at native or web side, and a series of API traces at either web or native side, respectively.

Web-to-native-side-path definition covers a flow of function calls from JavaScript to Android privileged APIs at application native side. In web side, the whole interactions between JavaScript functions, which at the end will invoke one of the application interfaces, are detected and assigned to separate path-IDs. On the other hand, native-to-web-side-path concept includes all events during which the triggering points are on the application native side. These events could be one of the events that have been mentioned in [15].

B. Native and Web Tracker

Native and **Web Tracker** modules construct application interaction paths. After disassembling APK files in Smali format, **Native Tracker** module extracts all native APIs exposed to application web side. Each application in Android must have an activity which is application's main entry point. In order to find this activity inside the application's Smali files, we use "LUANCHER" tag in activity's definition codes. We use this activity as a first entry point of our **SS-API** extraction process. For identifying application **SS-API**s, we use Felt. et al. [18] research results. For extracting Android privileged APIs from our apps byte-codes, we use Androguard set of tools [19].

In **Web Tracker** module, we scrap apps JavaScript files. For extracting API and other information from existing JavaScript files after application APK file disassembling, we use Python scripts. For each JavaScript function at web side which invokes native APIs, we have first extracted all possible traces of API invoking inside the application which will reach an Android privileged API at native side. In **Internal Interaction Tracker** component, all application interactions, their constructed paths, and relationships are extracted and reported as application profile.

III. Analysis Results

In this section, we will apply our proposed framework to a large number of hybrid and native mobile apps. We will use three different metrics regarding application behavior for analyzing our data set. These metrics are dynamicity of API invoking inside the application, passing data to or receiving

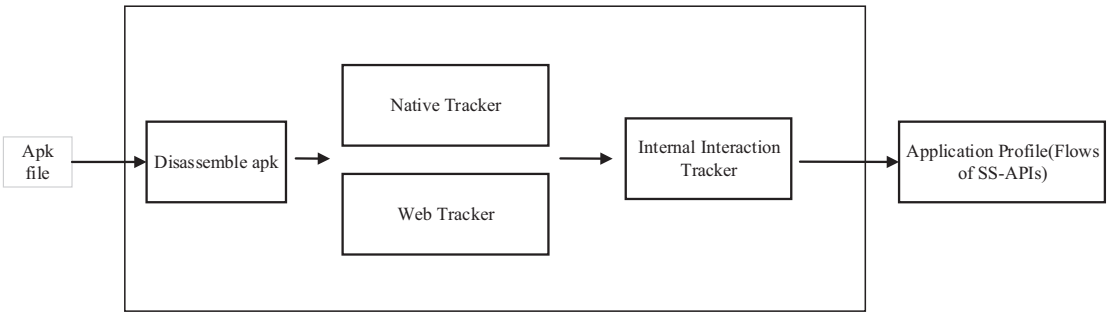

Fig. 1. Hybrid-Scanner architecture

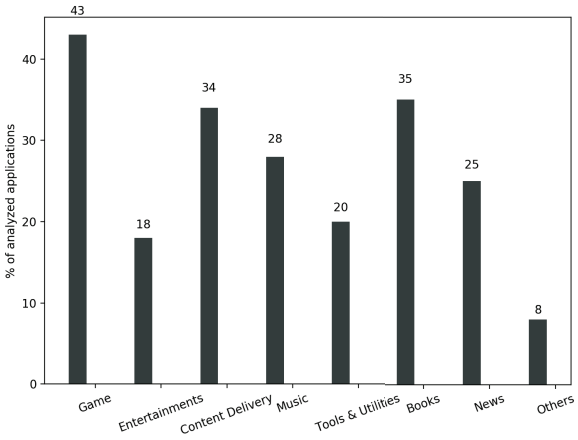

Fig. 2. The genre of analyzed hybrid mobile apps

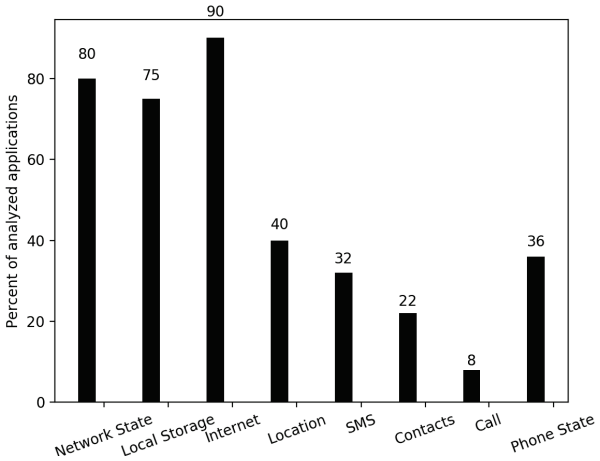

Fig. 3. Most sensitive used APIs in the analyzed data set of hybrid apps

data from application's other side and usage of ad libraries in the application and the influence of them on the generated profile of API invoking.

Our applications dataset contains $5,534$ hybrid mobile apps and $2,534$ Android native apps downloaded from Google Play in June 2016. Google Play has appropriate APIs for searching and downloading different apps able to be programmed on its market. For downloading applications automatically, we developed a Python script which automatically searches an application package name on the Google Play. If it exists, the script downloads application's APK file. For identifying hybrid mobile apps on Google Play, we have used the applications' list of [2] and [20]. Both have used hybrid mobile apps (specifically developed by PhoneGap hybrid mobile framework) in their evaluation section. In addition, we have used $2,532$ Android native apps in our dataset, downloaded from Google Play and local market shares. In Fig. 2, the genre of our downloaded hybrid mobile apps has been depicted.

A. Measuring Program Dynamism

An experiment has been carried out on both most frequently used APIs in the downloaded hybrid mobile apps and the degree of dynamicity of API invoking. We define dynamicity property of an API as a concept which illustrates the number of possible paths of API invoking which can trigger the target of this API. For instance, if an **SS-API** is invoked from two different traces of API flows, we define the degree of dynamicity for this **SS-API** as 2.

In Fig. 3, the most frequently used APIs in apps' web and native side in our data-set of hybrid mobile apps have been shown. Among them is the connection to the Internet and access to local storage APIs. As in hybrid mobile app most application's content is downloaded from remote servers and loaded in application embedded browser at runtime, using the Internet-related APIs is very usual in these type apps. Using access to local storage and file APIs in these type of application is for enhancing the user experience by caching content downloaded from the server.

Fig. 4 demonstrates the degree of dynamicity in API invoking for hybrid and native mobile apps. For finding the dynamicity of different APIs, we selected a subset of our hybrid and native apps in which they share the same permission set in their manifest file. As it has been shown in Fig. 4, we have a different pattern in the degree of dynamicity in API

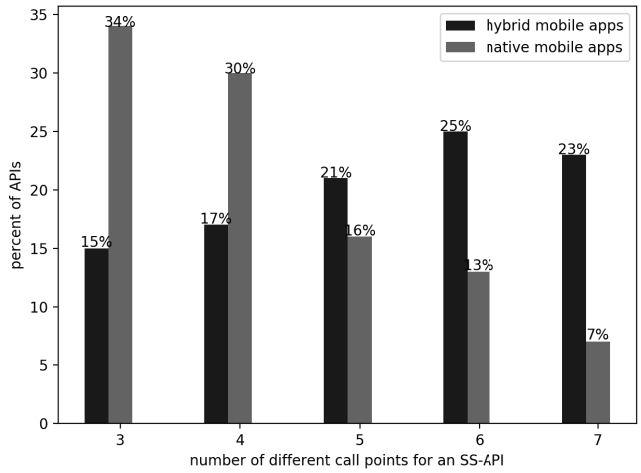

Fig. 4. degree of dynamicity in API invoking for hybrid and native mobile apps

TABLE I
PASSING DATA TO OR FROM APPLICATION WEB AND NATIVE SIDE

Passing data to web side	Passing data to native side
49%	52%

invoking for hybrid and native mobile apps. For instance, the number of different call points (possible trace calls of APIs) in hybrid mobile apps for almost 25 percent of application security sensitive APIs is more than 6. That means, 25 percent of application **SS-API**s can be triggered from more than 6 different points inside the application source code. However, only 13 percent of security sensitive APIs in native apps have 6 different call points and most of security sensitive APIs have solely 3 different call points.

B. Transferring data between native and web codes

Hybrid mobile apps can use different mechanisms for invoking events and transferring data between two technologies, native and web. [2], [21]. Some of these event triggering mechanisms are handled by hybrid framework's libraries such as PhoneGap [22]. In this section, we have analyzed percent of application APIs in native or web side which pass data to or receive a message as an argument from the other side of the application. This concept is important for developing fuzzers for testing applications functionality and finding vulnerabilities by triggering application different parts [23]. Table I indicates that in hybrid mobile apps, more data has been passed to application's APIs in the native side from web side. That means we have a big flow of data from web side of the application to native side. This is important as these arguments can be changed in different situations as application web side has a dynamic nature. Having a good picture of application messages and different function arguments, we can generate specific inputs in fuzzing tools.

C. Advertisements libraries

In third-party mobile apps, developers use advertising services inside application's source code as a source of revenue. Unfortunately, ad plug-ins inside hybrid mobile apps inherit all application's permissions and can exploit this capability to steal users private information [7], [2]. While this concern has been covered in many articles such as [2], [24] and [7], we will analyze the influence of using ad plug-ins on the generated profile of API invoking.

We found that at least 20% of applications in our dataset have more than one ad plug-in from an ad network and almost 40% of applications **SS-API**s at the native side are being triggered from advertisement web content.

IV. RELATED WORK

Security concerns of hybrid mobile apps and in common, exposing native APIs to web content have been addressed in recent articles actively [2], [4], [5], [7], [21], [25]–[32]. The studies in this area mainly fall in analyzing access control limitations of different mobile platforms for hybrid mobile apps including third-party libraries and web content inside application's source code. These works highlight unsuitability of existing access controls in web or native side of the application and try to propose a new access control model which is specific for hybrid mobile apps' structure. Based on our knowledge, this paper is the first study that conducts an analysis of these types of applications' unique features and their impacts on application generated profile.

Luo *et al.* [33], Chin and Wagner [31] and Tongbo *et al.* [32] analyze security concerns of breaking browser sandbox in hybrid mobile apps in a large data set of apps. NoFRAK [2] conducts an analysis of hybrid mobile frameworks access control vulnerabilities. Hybrid-Guard [9] is a novel policy enforcement framework proposed for preparing capability for developers to define their fine-grained policies for web content from different domains. At runtime, Hybrid-Guard enforces access control on multiple origins of web content loaded from untrusted domains inside the applications without requiring any modification in the source code of hybrid mobile frameworks or platforms. Draco [27] is a uniform and fine-grained access control that works for all applications that use embedded browsers and restricts all channels that prepare a way for web content to access device resources. Brucker *et al.* [34] propose a static analysis approach for analyzing foreign language calls inside hybrid mobile apps using Cordova framework. Mohamed *et al.* [21] propose a new policy based access control for the hybrid mobile app by re-designing Cordova structure to provide limited access to different pages of applications. Restricted-Path [35] is a new access control mechanism for hybrid mobile app implemented at browser and OS level in Android. EOEDroid [36] is a new approach which can help researchers to study web events, native event handlers, and their triggering points automatically in hybrid mobile apps. Based on the evaluation of almost 3,652 most popular apps, the authors have found 97 total vulnerabilities in 58 hybrid mobile apps.

Chen *et al.* [4] and Jin *et al.* [37] present new types of XSS attacks through new channels on the HTML5-based mobile apps. In these types of attacks, the attacker could inject her malicious JavaScript code in the HTML5-based mobile app through channels like SMS and QR. Hassanshahi *et al.* [20] present an automated framework for finding and confirming these attacks. Other studies [25], [30], [37], [38] propose different approaches for finding new attacks in hybrid mobile apps. In [25], an approach using the state machine of application's behaviors for detecting injected behaviors in the HTML5-based mobile app has been proposed. Using state machine, they can capture application profile and the contexts where actions take place and whenever malicious code injection happens, the injected behaviors will be detected. Using application high-level analysis, a new feature based on inter-app communication analysis has been proposed in [39].

V. Conclusion and Discussion

In this paper, we analyzed the generated profile of function call graph for hybrid and native mobile apps. We found that the generated profiles for hybrid and native mobile apps are considerably different from different point of views. Our study uncovered a number of aspects of hybrid mobile apps' behaviors which comes from their unique structure. First, we showed that there is more trace of API calling for triggering a specific **SS-API** in hybrid mobile apps in comparison with Android native mobile apps using Hybrid-Scanner framework as a first specific static analysis tool developed for hybrid mobile app behavior analyzing. In addition, we found that almost $\%40$ of **SS-API** invoking is done by third-party libraries, such as Ad libraries, in hybrid mobile apps. This knowledge, however, is crucial for designing appropriate malware detection or vulnerability mitigation strategies in hybrid mobile apps. As such, it is important to discuss some recent body of work on Android native mobile apps analysis and examine possible consideration to explore future approaches in hybrid mobile application security. We have listed some of the key points here which should be considered regarding hybrid mobile apps analysis:

1. Watch out your decision: With respect to hybrid mobile apps, making a decision regarding benign, malicious, or abnormal actions by analyzing application behavior will be a challenging task. This is mostly due to dynamic nature of hybrid mobile apps. Even though malicious actions at the level of system calls have unique patterns, due to the unique structure of smart-phone platforms; this has been shown that analyzing application behaviors at high-level is more useful than that of application generated system calls at low-level [14], [15], [40]. We believe that our extracting features and analysis must be considered in those mobile app analysis approaches which use application high-level behavior analysis and have been designed based on mobile native app software stack. Using internal analysis methods such as [14] for hybrid mobile applications, we need to identify all native and web side events which could impact on application's another side. For instance, resource delivery points in hybrid mobile apps in contrast-to-native-apps can be at two points, namely native and web side.

2. Generate more customized inputs: We showed that many of **SS-API**s in hybrid mobile apps are being triggered from application web content. The main functionality of hybrid mobile application can be triggered by receiving an event from web content. This feature will be important in developing fuzzer tools for hybrid mobile apps. For example, Intellidroid [23] is a fuzzer for dynamic analysis of Android malwares which can trigger applications malicious functionality via specific inputs. As hybrid mobile applications, in case of huge inputs come from web side we need a customized version of IntelliDroid to generate specific inputs for such applications.

Acknowledgment

The authors would like to thank Suman Jana, Xing Joe, Farahnaz Hasanshahi and specially Martin Goergive for their help in gathering data set used in the experiments.

References

[1] M. Sun and G. Tan, "Nativeguard: Protecting android applications from third-party native libraries," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 2014, pp. 165–176.

[2] 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)*, vol. 2014. NIH Public Access, 2014, p. 1.

[3] 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

[4] 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 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 66–77.

[5] X. Jin, L. Wang, T. Luo, and W. Du, "Fine-grained access control for html5-based mobile applications in android," in *Information Security*. Springer, 2015, pp. 309–318.

[6] M. Georgiev, S. Jana, and V. Shmatikov, "Rethinking security of web-based system applications," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 366–376.

[7] S. Son, D. Kim, and V. Shmatikov, "What mobile ads know about mobile users," in *23nd Annual Network and Distributed System Security Symposium, NDSS*, 2016, pp. 21–24.

[8] R. McPherson, S. Jana, and V. Shmatikov, "No escape from reality: Security and privacy of augmented reality browsers," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 743–753.

[9] P. H. Phung, A. Mohanty, R. Rachapalli, and M. Sridhar, "Hybrid-guard: A principal-based permission and fine-grained policy enforcement framework for web-based mobile applications."

[10] J. Mao, H. Ma, Y. Chen, Y. Jia, and Z. Liang, "Automatic permission inference for hybrid mobile apps," *Journal of High Speed Networks*, vol. 22, no. 1, pp. 55–64, 2016.

[11] B. Hassanshahi, Y. Jia, R. H. Yap, P. Saxena, and Z. Liang, "Web-to-application injection attacks on android: Characterization and detection," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 577–598.

[12] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute this! analyzing unsafe and malicious dynamic code loading in android applications." in *In Proceeding of the Network and Distributed System Security Symposium (NDSS 2014)*, vol. 14, 2014, pp. 23–26.

[13] I. R. Forman, N. Forman, and J. V. Ibm, "Java reflection in action," 2004.

[14] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 611–622.

[15] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 303–313.

[16] Google. (2014) Apktool. [Online]. Available: https:code.google.compandroid-apktool Access time: 2016, May

[17] J. Hoffmann, M. Ussath, T. Holz, and M. Spreitzenbarth, "Slicing droids: program slicing for smali code," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1844–1851.

[18] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 627–638.

[19] A. Desnos. (2014) Androguard-reverse engineering, malware and goodware analysis of android applications. [Online]. Available: https: code.google.com/p/androguard Access time: 2013, May

[20] B. Hassanshahi, Y. Jia, R. H. Yap, P. Saxena, and Z. Liang, "Web-to-application injection attacks on android: Characterization and detection," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 577–598.

[21] 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.

[22] J. M. Wargo, *PhoneGap essentials: Building cross-platform mobile apps*. Addison-Wesley, 2012.

[23] M. Y. Wong and D. Lie, "Intellidroid: A targeted input generator for the dynamic analysis of android malware," in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2016.

[24] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, "A large-scale study of mobile web app security," in *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.

[25] J. Mao, R. Wang, Y. Chen, and Y. Jia, "Detecting injected behaviors in html5-based android applications," *Journal of High Speed Networks*, vol. 22, no. 1, pp. 15–34, 2016.

[26] M. L. Hale and S. Hanson, "A testbed and process for analyzing attack vectors and vulnerabilities in hybrid mobile apps connected to restful web services," in *Services (SERVICES), 2015 IEEE World Congress on*. IEEE, 2015, pp. 181–188.

[27] G. S. Tuncay, S. Demetriou, and C. A. Gunter, "Draco: A system for uniform and fine-grained access control for web code on android," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 104–115.

[28] J. Knockel, A. Senft, and R. Deibert, "Privacy and security issues in bat web browsers," in *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*. USENIX Association.

[29] S. Lee, J. Dolby, and S. Ryu, "Hybridroid: Static analysis framework for android hybrid applications," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 2016, pp. 250–261.

[30] Y. C. X. P. S. Z. Vaibhav Rastogi, Rui Shao 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, February 2016.

[31] E. Chin and D. Wagner, "Bifocals: Analyzing webview vulnerabilities in android applications," in *International Workshop on Information Security Applications*. Springer, 2013, pp. 138–159.

[32] T. Sutcliffe and A. Taylor, "The lifetime of android api vulnerabilities: Case study on the javascript-to-java interface," in *Security Protocols XXIII: 23rd International Workshop, Cambridge, UK, March 31-April 2, 2015, Revised Selected Papers*, vol. 9379. Springer, 2015, p. 126.

[33] 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.

[34] A. D. Brucker and M. Herzberg, "On the static analysis of hybrid mobile apps," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2016, pp. 72–88.

[35] S. Pooryousef and S. Amini, "Fine-grained access control for hybrid mobile applications in android using restricted paths," in *13th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), 2016*. IEEE, 2016, pp. 85–90.

[36] G. Yang, J. Huang, and G. Gu, "Automated generation of event-oriented exploits in android hybrid apps." NDSS, 2018.

[37] Y.-L. Chen, H.-M. Lee, A. B. Jeng, and T.-E. Wei, "Droidcia: A novel detection method of code injection attacks on html5-based mobile apps," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 1014–1021.

[38] S. Pooryousef and M. Amini, "Enhancing accuracy of android malware detection using intent instrumentation." in *ICISSP*, 2017, pp. 380–388.

[39] S. Pooryousef and K. Fouladi, "Proposing a new feature for structure-aware analysis of android malwares," in *14th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), 2016*. IEEE, 207.

[40] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors," in *NDSS*, 2015.