Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (200-204), Month: January-April 2016, Available at: www.noveltyjournals.com

Permission Analysis System for Android Applications

K.KAVIYA[1], S.V.HEMALATHA[2], P.MEENAKSHI [3], DR.K.VALARMATHI [4]

[1,2,3] Student, [4]Professor, Department of CSE, Panimalar Engineering College, Chennai, India

Abstract: **Android applications are open source and can be developed by anybody, testing is not done. The objective of this paper is to remove the unused/redundant permissions in the android applications by breaking it and extracting the permissions to prevent the permission gap. High level Permission Checking Framework on Android Applications that were previously uploaded by breaking the .apk files to analyze in code level by decompiling it in a efficient way. This is a compositional analysis for Android inter app vulnerabilities. Both developer and user can test and modify manifest file of an application.**

Keywords: **Android SDK, JDK, Apache ANT, APK, QR codes, dalvik byte code, DEX.**

Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (200-204), Month: January-April 2016, Available at: www.noveltyjournals.com

1. INTRODUCTION

Android applications are usually developed in the Java language using the Android Software Development Kit. Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play**, **SlideME**, **Opera Mobile Store**,**Mobango**,**F-droid**.

Applications for Android are written in Java and compiled into Dalvik byte code. Dalvik byte code is optimized to run on devices where memory and processing are scarce. An Android application is packaged into an Android package file which contains the Dalvik byte code, data (pictures, sounds. . .) and a metadata file called the "manifest". The application's developer has declared permissions in the application manifest. For installing an application, the user has to approve all the permissions. If all permissions are approved then the application is installed and receives group memberships. The group memberships are used to check the permissions at runtime. Missing permission causes the application to crash. Adding too many of them is not secure. In the latter case, injected malware can use those unused permissions to achieve malicious goals. The unused permissions are called as "permission gap". Any permission gap results in insecure, suspicious or unreliable applications

2. RELATED WORKS

Android security has received a lot of attention in recently published literature, due mainly to the popularity of Android as a platform of choice for mobile devices, as well as increasing reports of its vulnerabilities. Here, we provide a discussion of the related efforts in light of our research.

A large body of work [7], [10], [13], [19] focuses on performing program analysis over Android applications for security, which can be categorized based on their underlying static or dynamic analysis technique. FlowDroid [20] introduces a precise approach for static taint flow analysis in the context of each application component. CHEX [16] also takes a static method to detect component hijacking vulnerabilities within an app.Apart from techniques based on static analysis, several tools use dynamic analysis to detect vulnerabilities in smart phone applications. TaintDroid [3] detects information leak vulnerabilities using dynamic taint flow analysis at the system level.

3. PROBLEM DEFINITION

As android applications are open source and can be developed by anybody, testing is not mandatory and hence it is more vulnerable. Android application developed by users are directly uploaded to Google play store and no code level testing's are done. Since the developers upload only compiled, packed (.apk) files no further investigation is done on the application.

A basic call graph can only give the number of permission checks but not the actual names of the checked permissions because of the lack of string analysis to extract permission names from the byte code CHA-Android which leverages the service redirection, service identity inversion and entry point construction components.

Spark specific issues such as entry point initialization or Android specific issues such as service initialization. Spark to get a first understanding of the main problems that occur when analyzing the Android API. This gives us a key insight, Spark discards 96 percent of the API methods to be analyzed. The reason is that Spark does not work on receiver objects whose value is null.

4. PROPOSED WORK

We propose a High level Permission Checking Framework on Android Applications that were previously uploaded by breaking the .apk files to analyze in code level by decompiling it in a efficient way. We also innovate to recompile the vulnerable free code for secure use with the end users. We further make a proposal to Google Play Services to implement this kind of Frameworks so as to avoid Fake Applications that steals user's Private data and make some vulnerability.

Android 2.2 defines 134 permissions in the android. Manifest permission system class, whereas Android 4.0.1 defines 166 permissions. This gives us an upper-bound on the number of permissions which can be checked in the Android

Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (200-204), Month: January-April 2016, Available at: www.noveltyjournals.com

framework. Android has two kinds of permissions: "high-level" and "low-level" permissions. High-level permissions are only checked at the framework level (that is, in the Java code of the Android SDK). We focus on the high-level permissions that are only checked in the Android Java framework Compositional analyses for extracting permission checks. In essence, each analysis constructs a call graph from the byte code, finds permission check methods and extracts permission names.

We have presented a generic approach to reduce the attack surface of permission-based software in order to automatically add or remove permission enforcement points at the level of application or the framework.

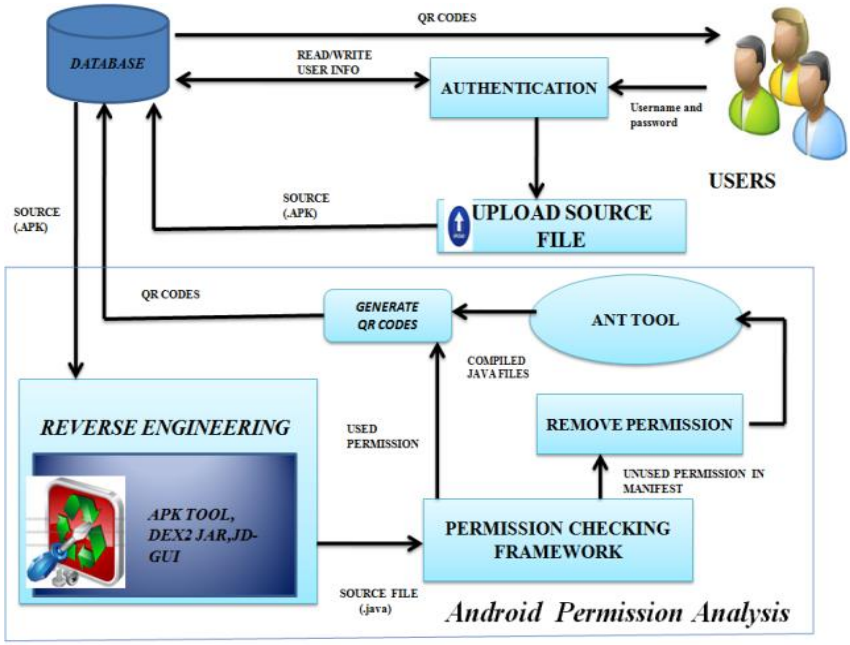

Fig.1 Overview of permission analysis system

5. WORKING MODEL

5.1 Login / Registration And Upload:

User enters the personal information for registration and the user input fields are validated and records are stored in Database. After registration the User can Login with his credentials and can upload source code. The uploaded source is securely stored in server side. If you are uploading a source code it should in a zip format which can be done by any zip until tools. The uploaded zip contents are automatically unzipped in code level in server side.

5.2 Reverse Engineering The Apk File:

In this Module, user can upload both source and apk files. The apk file is breaked by using APK Tool and the generated (.dex) files are converted to (.jar) files by de2jar.The layout and resource files are retained. The jar files are extracted to get the .class files. Now we use the jad API to convert the .class files to .java files. Then these files are written to the src folder of android code base retaining the package name. Thus the Server automatically Decompile the .apk file by reverse engineering.

5.3 Permission Check's In Source Code:

Android applications contain much permission to use the services. Developer must declare the permission in manifest to use that service. Once the permission is declared, the android application packager in the mobile phone will ask the users for accepting the permission usage while application installation. For installing an application, the user has to approve all the permissions that application's developer has declared in the application manifest.

If all permissions are approved, the application is installed and receives group memberships. The group memberships are used to check the permissions at runtime. Now the decompiled apk files are validated for permissions in the manifest.xml file. Now our high level permission checking framework examines the code written for each permission in java files and validates it. If the any of the permissions fails the validation process, it is tagged as Unused/Redundant permissions.

5.4 Removing Unused Permissions:

In this module, if unused permissions are declared, their respective service is also running in mobile. Missing permission causes the application to crash. Adding too many of them is not secure. Injected malware can use those declared, yet unused permissions, to achieve malicious goals. So the unused permissions found by our framework are removed in the Manifest.xml file.

The modified/Permission checked source code is recompiled and harmless apk's are generated which can be downloaded using Qrcode. Only the uploaded source codes are recompiled

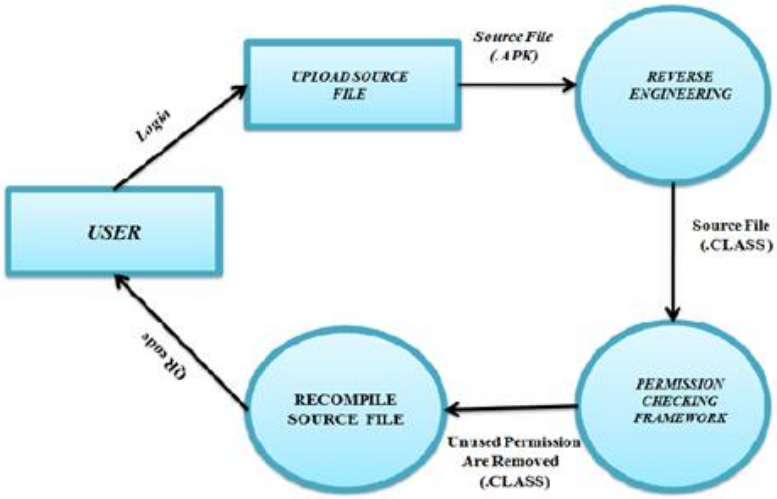

Fig.2 Data Flow For Permission Analysis System

6. BENEFITS OF PROPOSED SYSTEM

The proposed system will remove all unused permissions in the android applications. A static code level check is done on the code. Missing permission causes the application to crash. Adding too many of them is not secure. In the latter case, injected malware can use those unused permissions to achieve malicious goals. The unused permissions are called as "permission gap". Permission gap results in insecure, suspicious or unreliable applications. QR codes are generated for each recompiled applications

7. CONCLUSION

This paper concludes that extracted the permission checks and to removed the unused permissions to prevent the permission gap and Applications were builded using Apache Ant tool and Qrcode were generated.Application are downloaded using qr code

REFERENCES

[1] R. Valle _e-Rai, P. Co, E. Gagnon, L. Hendren, and V. Lam, and P.Sundaresan, "Soot—a Java bytecode optimization framework," inProc. Conf. Centre Adv. Stud.Collaborative Res., 1999, p. 13.

[2] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proc. ACM Conf. Comput. Commun.Security, 2009, pp. 235–245.

[3] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scandroid: Automated security certification of android applications," Dept. Comput.Sci., Univ. Maryland, College Park, MD, USA, Tech. Rep. CSTR-4991, 2009.

[4] D. Barrera, H. Kayacik, P. Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in Proc. 17th ACM Conf. Comput.Commun. Security, 2010, pp. 73–84.

[5] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application Communication in android," in Proc. 9th Int.Conf. Mobile Syst., Appl. Services, 2011, pp.239–252.

[6] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in Proc. 20th USENIX Conf. Security,2011, p. 21.

[7] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel,and A. N. Sheth,:"Taintdroid An information-flow tracking system for realtime privacy monitoring on smartphones" [7], in Proc.9th USENIX Conf. Operating Syst. Des. Implementation, 2011.

[8] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proc. ACM Conf. Comput. Commun. Security, 2011, pp. 627–638

[9] A. Bartel, J. Klein, Y. LeTraon, and M. Monperrus, "Dexpler: Converting android dalvik bytecode to jimple for static analysis with soot," in Proc. ACM SIGPLAN Int. Workshop State of the Art Java Program Anal., 2012, pp. 27–38.

[10] E. Fragkaki, L. Bauer, L. Jia, and D. Swasey. (2012). " Modeling and enhancing android's permission system", Proc. ESORICS [Online]. Available:http://link.springer.com/chapter/10.1007/978-3-642-33167-1_1

[11] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker:Scalable and accurate zero-day android malware detection," in Proc. Int. Conf. Mobile Syst., Appl. Services, 2012, pp. 281–294.

[12] Y. Zhou, Z. Y. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in Proc. 19th Netw. Distrib. Syst. SecuritySymp., 2012.

[13] C. Mann and A. Starostin, "A framework for static detection of privacy leaks in android applications," in Proc. 27th Annu. ACM Symp. Appl. Comput., 2012, pp. 1457–1462.

[14] M. Grace,Y. Zhou, Z. Wang, and X. Jiang, "Systematic detection of capability leaks in stock android smartphones," in Proc. 19thAnnu. Symp. Netw. Distrib. Syst. Security, 2012.

[15] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the android permission specification," in Proc. ACM Conf. Comput.Commun. Security, 2012, pp. 217–228.

[16] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: Statically vetting android apps for component hijacking vulnerabilities,"in Proc. ACM Conf. Comput. Commun. Security, 2012, pp. 229–240.

[17] S. Bugiel, L. David, Dmitrienko, T. A. Fischer, A. Sadeghi, and B.Shastry, "Towards taming privilege-escalation attacks on android," presented at the NDSS Symp., San Diego, CA, USA,2012.

[18] S. Holavanalli, D. Manuel, V. Nanjundaswamy, B. Rosenberg, F.Shen, S. Y. Ko, and L. Ziarek, "Flow permissions for android," in Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng., 2013,pp. 652–657.

[19] Y. Zhou and X. Jiang, "Detecting passive content leaks and pollution in android applications," in Proc. 20th Netw. Distrib. Syst.Security Symp., 2013.

[20] S. Arzt, S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D.Octeau, and P.McDaniel, "FlowDroid: Precise context, flow, field,object-sensitive and lifecycle-aware taint analysis for android apps," in Proc. 35th Annu. ACM SIGPLAN Conf.