ISSN: 0976-3104
SPECIAL ISSUE: Emerging Technologies in Networking and Security (ETNS)
Gowthami et al.

THE IIOAB JOURNAL

ARTICLE OPEN ACCESS

ANALYZING AND REMOVING UNUSED ANDROID INTER-APP PERMISSIONS

M. Gowthami*, S. Sriraj, G. Jitesh Kumar, G. Vishal

Department of Computer Science and Engineering, Vel Tech High Tech Dr. Rangarajan Dr.Sakunthala Engineering College, Avadi, Chennai, INDIA

ABSTRACT

Aims: *The aircrafts are playing Vital role in Military and Commercial purposes. It is very difficult to communicate with the aircrafts when it is flying. The aircrafts requires more secure communication because it is used in defense. Radar communication has so many drawbacks if signal loss then we cannot track the aircrafts and also it does not provides secure communication. If any Eavesdropper wants to hacks our confidential aircrafts communication easily they can hack.* ***Materials and methods:*** *Android's enforcement of the permissions is at the level of individual apps, allowing multiple malicious apps to collude and combine their permissions or to trick vulnerable apps to perform actions on their behalf that are beyond their individual privileges. In this paper, we present COVERT, a tool for compositional analysis of Android inter-app vulnerabilities. COVERT's analysis is modular to enable incremental analysis of applications as they are installed, updated, and removed. It statically analyzes the reverse engineered source code of each individual app, and extracts relevant security specifications in a format suitable for formal verification.* ***Results:*** *Given a collection of specifications extracted in this way, a formal analysis engine (e.g., model checker) is then used to verify whether it is safe for a combination of applications—holding certain permissions and potentially interacting with each other—to be installed together.* ***Conclusion:*** *Our experience with using COVERT to examine over 200 real-world apps corroborates its ability to find inter-app vulnerabilities in bundles of some of the most popular apps on the market.*

Published on: 18th– August-2016

KEY WORDS

Formal verification, static analysi, Android, Inter-App vulnerabilities

***Corresponding author: Email:** gowthamimailme@gmail.com **Tel.:** +91-90474-91577; **Fax:** +91-44-26840 249

INTRODUCTION

Mobile app markets are creating a fundamental model shift in the way software is delivered to the end users. The benefits of this software supply model are plenty, including the ability to rapidly and effectively acquire, introduce, maintain, and enhance software used by the consumers. By providing a medium for reaching a large consumer market at a nominal cost, app markets have leveled the software development industry, allow-ing small entrepreneurs to compete with prominent soft-ware development companies.

Application frameworks are the key enablers of these markets. An application framework, such as the one provided by Android, ensures apps developed by a wide variety of suppliers can interoperate and coexist together in a single system (e.g., a phone) as long as they conform to the rules and constraints imposed by the framework.

This paradigm shift, however, has given rise to a new set of security challenges. In parallel with the emergence of app markets, we are witnessing an increase in the security threats targeted at mobile platforms. This is nowhere more evident than in the Android market (i.e., Google Play), where many cases of apps infected with malwares and spywares have been reported [1]. Numerous culprits are at play here, and some are not even technical, such as the general lack of an overseeing authority in the case of open markets and inconsequential implication for those caught provisioning applications with vulnerabilities or malicious capabilities.

In this context, Android's security has been a thriving subject of research in the past few years. Leveraging program analysis techniques, these research efforts have investigated weaknesses from various perspectives, including detection of information leaks [2–4], analysis of the least-privilege principle [5,6], and enhancements to Android protection mechanisms [7–9]. The majority of these approaches, however, are subject to a common limitation: they are intended to detect and mitigate vulnerabilities in a single app, but fail to identify vulnerabilities that arise due to the interaction of multiple apps. Vulnerabilities due to the interaction of multiple

www.iioab.org
THE IIOAB JOURNAL
www.iioab.webs.com

THE IIOAB JOURNAL
ISSN: 0976-3104

www.iioab.org

THE IIOAB JOURNAL

www.iioab.webs.com

apps, such as collusion attacks and privilege escalation chaining [5], cannot be detected by techniques that analyze a single app in isolation. Thus, security analysis techniques in such domains need to become compositional in nature.

This paper contributes a novel approach, called COVERT, for compositional analysis of Android inter-app vulnerabilities. Unlike all prior techniques that focus on assessing the security of an individual app in isolation, our approach has the potential to greatly in-crease the scope of application analysis by inferring the security properties from individual apps and checking them as a whole by means of formal analysis. This, in turn, enables reasoning about the overall security pos-ture of a system (e.g., a phone device) in terms of the security properties inferred from the individual apps.

COVERT combines static analysis with formal meth-ods. At the heart of our approach is a modular static analysis technique for Android apps, designed to enable incremental and automated checking of apps as they are installed, removed, or updated on an Android de-vice.

Through static analysis of each app, our approach extracts essential information and captures them in an analyzable formal specification language. These formal specifications are intentionally at the architectural level to ensure the technique remains scalable, yet represent the true behavior of the implemented software, as they are automatically extracted from the installation artifacts.

The set of models extracted in this way are then checked as a whole for vulnerabilities that occur due to the inter-action of apps comprising a system. COVERT uses Alloy as a specification language [10], and the Alloy Analyzer as the analysis engine. Alloy is a formal specification language based on first order logic, optimized for auto-mated analysis.

Since COVERT's analysis is compositional, it provides the analysts with information that is significantly more useful than what is provided by prior techniques. Our experiences with a prototype implementation of the approach and its evaluation against one of the most prominent inter-app vulnerabilities, i.e. privilege escalation, in the context of hundreds of real-world Android apps collected from variety of repositories have been very positive. The results, among other things, corroborate its ability to find vulnerabilities in bundles of some of the most popular apps on the market.

Contributions: This paper makes the following contributions:

Formal model of Android framework: We develop a formal specification representing the behavior of Android apps that is relevant for the detection of inter-app vulnerabilities. We construct this formal specification as a reusable Alloy module to which all extracted app models conform.

Modular analysis: We show how to exploit the power of our formal abstractions by building a modular model extractor that uses static analysis techniques to automatically extract formal specifications (models) of apps form their installation artifacts.

Implementation: We develop a prototype implementation on top of our formal framework for compositional security analysis of Android apps.

Experiments: We present results from experiments run on over 200 real-world apps, corroborating COVERT's ability in effective compositional analysis of Android inter-app vulnerabilities in the order of minutes.

The study was conducted in the Department of Conservative Dentistry and Endodontics, Sinhgad Dental College and Hospital, Pune with technical aid from the Department of Microbiology, SKN Medical College and Hospital, Pune. All the participants were informed about the study and necessary informed consent was taken. Ethical clearance was obtained from the ethical committee of college. Total 20 aprons of dental healthcare professionals (interns, PG students, faculty members) were included in the study.

THE IIOAB JOURNAL
ISSN: 0976-3104

www.iioab.org

THE IIOAB JOURNAL

www.iioab.webs.com

PROPOSED SYSTEM

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.

Android application, (ex: appwrong), which is able to communicate with external servers since it is granted the INTERNET permission. Moreover, appwrong has declared permission CAMERA while it does not use any code related to the camera. The CAMERA permission allows the application to take pictures without user intervention, i.e., the permission gap consists of a single permission: CAMERA. In this particular example the attacker would be able to write code to use the camera, take a picture and send the picture to a remote host on the Internet.

The problematic consequences of having more permissions than necessary and showed that the problem can be mitigated using compositional analysis. The approach has been fully implemented for Android, a permission-based platform for mobile devices. Android application stores indeed suffer from permission gaps.

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 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.

IMPLEMENTATION

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 power 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".

For installing an application, the user has to approve all the permissions the 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.

Missing permission causes the application to crash. Adding too many of them is not secure. In the latter case, injected malware can use those declared, yet unused permissions, to achieve malicious goals. We call those unused permissions, "permission gap". Any permission gap results in insecure, suspicious or unreliable applications.

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.

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.

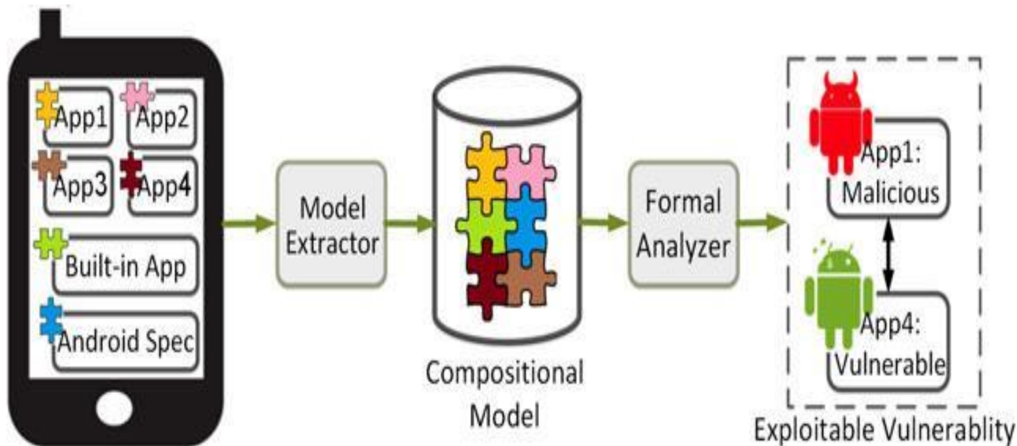

Fig. 1: Architecture

- **Login / Registration and Upload.**
- **Reverse Engineering the apk file.**
- **Permission Check's in Source Code.**
- **Remove Unused Permissions.**

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.

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.

CONCLUSION

This paper presents a new approach for compositional learning of Android inter-app vulnerabilities. Our move toward employs static analysis to automatically recover models that reflect Android apps and interactions among them. It is able to leverage these models to identify vulnerabilities due to interaction of multiple apps that cannot

www.iioab.org THE IIOAB JOURNAL www.iioab.webs.com

THE IIOAB JOURNAL
ISSN: 0976-3104

www.iioab.org

be detected with prior techniques relying on a single app analysis. We formalized the basic elements of our analysis in an analyzable specification language based on relational logic, and developed a prototype implementation, COVERT, on top of our formal analysis framework. The experimental results of evaluating COVERT against privilege escalation—one of the most prominent inter-app vulnerabilities—in the context of hundreds of real-world Android apps corroborates its ability to find vulnerabilities in bundles of some of the most popular apps on the market.

CONFLICT OF INTEREST

The authors declare no conflict of interests.

ACKNOWLEDGEMENT

None

FINANCIAL DISCLOSURE

The authors report no financial interests or potential conflicts of interest.

REFERENCES

[1] *World Health Population*11(3): 44–54. A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," IEEE Security Privacy, vol. 8, no. 2, pp. 35–44, Mar./Apr. 2010.

[2] 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.

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

[4] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," in Proc. ACM Conf. Comput. Commun. Security, 2011, pp. 639–652.

[5] 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.

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

[7] 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

[8] 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.

[9] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach, "Quire: Lightweight provenance for smart phone operating sys-tems," in Proc. 20th USENIX Seurity Symp., 2011.

[10] D. Jackson. (2002). Alloy: A lightweight object modelling notation. TOSEM [Online]. 11(2), pp. 256–290. Available: http://portal. acm.org/citation.cfm?doid=505145.505149

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

[12] A. Bartel, J. Klein, Y. LeTraon, and M. Monperrus, "Dexpler: Con-verting 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.

[13] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," ACM Comput. Surv., vol. 41, no. 4, pp. 19:1–19:36, Oct. 2009.

[14] P. Zave, "A practical comparison of alloy and spin," Formal Asp. Comput. vol. 27, no. 2, pp. 239–253, 2015.

[15] Android API reference document [Online]. Available: http:// developer.android.com/reference, Oct. 2014.

[16] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Princi-ples, Techniques, and Tools, 2nd ed. Boston, MA, USA: Addison-Wesley, 2006.

[17] Android developers guide [Online]. Available: http://developer. android.com/guide/topics/fundamentals.html, Oct. 2014.

[18] 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.

**DISCLAIMER: *This published version is uncorrected proof; plagiarisms and references are not checked by IIOABJ; the article is published as provided by author and checked/reviewed by guest editor.*

THE IIOAB JOURNAL
www.iioab.webs.com