Your phone is leaking data! Evaluating Android content provider permissions

Prajit Kumar Das
University of Maryland, Baltimore County
prajit1@umbc.edu

Sandeep Narayanan
University of Maryland, Baltimore County
sand7@umbc.edu

Stanislav Bobovych
University of Maryland, Baltimore County
stan@umbc.edu

Anupam Joshi
University of Maryland, Baltimore County
joshi@umbc.edu

Nilanjan Banerjee
University of Maryland, Baltimore County
nilanb@umbc.edu

Ryan Robucci
University of Maryland, Baltimore County
robucci@umbc.edu

ABSTRACT

In 2010, the number of mobile devices in the world surpassed the number of personal computers. Mobile devices now carry sensitive personal data, captured through sensors on the phone, as well as confidential corporate data through work emails and apps. As a result, mobile devices have become a lucrative target for attackers, and privacy and security of these devices have become a vital issue. The existing access control mechanisms on these devices, which mostly relies on a one-time permission grant, is too restrictive and inadequate. Such a mechanism is incapable of controlling contextual or custom app-data flows. In this paper we focus on this scenario and show how data leakages may occur due to developer inadequacy and not having proper checks for such leakages. We describe a design flaw in the Android permission verification mechanism and a way to capture such a vulnerability on a user's mobile device. We show a mechanism of injecting such a vulnerability into any app.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*

Keywords

Access Control, Android Content Providers, Permission Control

1. INTRODUCTION

Mobile devices have become ubiquitous due to its low cost and Android is the biggest player in the market. Latest reports from Google boasts of more than a billion active 30 day user [12]. According to the International Data Corporation's Worldwide Quarterly Mobile Phone Tracker report Android has a 85% market share in the smartphone category. Apps from the Google Play Store and a variety of other app-stores like Amazon App Store and Samsung Galaxy Apps provide a plethora of ways through which Android users can get their apps [2]. According to Statista [11], as of July 2015, there are more than 1.6 million android apps in the Google Play Store.

The proliferation of smartphones has led to the popularity of the BYOD(Bring-Your-Own-Device) paradigm, whereby users' use their personal devices for corporate purposes. Naturally, this creates a greater need to ensure strong access control mechanisms for the data on such devices. In certain domains the access control needs are of a critical nature. For example in the category of Medical and Health & fitness apps it is essential that user data and if being used in hospitals, corporate data security be maintained to the utmost level. Hospitals today use various hardware devices that are smart enough to communicate with smartphones and may even contain sensitive medical data. In addition to that android apps are capable of collecting a huge amounts of data about the smartphone user, often without the knowledge of the user.

In this paper, we introduce Heimdall [1], a heuristics based system that is currently in-the-works in our group. Heimdall is capable of detecting common vulnerabilities on an android device that can cause leakage of app data. Heimdall has been created with a BYOD scenario in mind where a part of the system resides on the mobile device and part of it resides on the server side. The server side includes a dashboard that gets notifications of apps being installed on the mobile devices used by the employees of the organization. The system is then able to analyze and detect if the app is vulnerable with respect to a list of previously known heuristics. We are adding new heuristics as we discover and study them. We are also including mechanisms to prevent data leakages by injecting code into the android framework which would allow us to intervene in the permission check process and thereby control the actual data flow on the phone.

For the current paper we have focused on vulnerability in custom permissions created by app developers. These permissions are there to protect the app developers' data avail-

HotMobile'16 February 23-24, 2016, St. Augustine, Florida, USA

[1]Heimdall is the all-seeing and all-hearing guardian sentry of Asgard who stands on the rainbow bridge Bifröst to watch for any attacks on Asgard

able through their own content providers. It is advised by Google that, if an app developer creates a content provider for allowing access to there own data, they should also create a permission to control access to it. However, this requirement is not a stringent one and one might simply ignore creating such a permission. We show in this paper how such a vulnerability might lead to leakage of app data. We use two different mechanisms to demonstrate the issue. We show that it is possible to exploit this vulnerability using our own data access app and content provider app pair. We also show that it's possible to reverse engineer and repackage any standard app to create this vulnerability. We did observe that it is possible to check for this issue in your code instead of delegating this issue to android but through our evaluation we show that such a check might be beyond standard practices.

Previous work found in the literature points out the extensive research that has gone into various mechanisms to study vulnerabilities in android apps. The mechanisms have ranged from app metadata analysis by Pandita et. al. [9], to detecting malwares by studying their characteristics like installation methods, activation mechanisms and malicious payload nature by Zhou et.al. [13]. Such studies indicate a need for better mobile anti-malware solutions and access control mechanisms. There have been multiple attempts at achieving the goal of properly managing access control on mobile (Android) devices. Efforts have been made by the open source community through the XPrivacy project (needs a rooted phone), the Privacy Guard project (available on Cyanogenmod, a custom Android ROM), the PDroid application (needs a rooted device). Research project by Conti et. al. [3](CRePe), Enck et al. [4](TaintDroid) and Jagtap et al. [6](Preserving Privacy in Context-Aware Systems) have made similar efforts. CRePE described a system where security policy enforcement was carried out based on context of the smart phone. TaintDroid was a research effort where the data flow on an Android device was studied to figure out when sensitive data left the system via an untrusted application. The work of Jagtap et al. [6] focused on constraining data flow in a context-aware system using a policy-based framework. A related work by Ghosh et al. [5] used a similar policy driven approach to constrain application permissions based on context.

We can understand from the extensive work done that there is significant knowledge about vulnerabilities on Android and ways to detect them. In this paper we present Heimdall, a system which can detect such vulnerabilities and we show an example of how one of these vulnerabilities can be detected using our system. The rest of the paper is organized as follows. We describe our system overview in the section 2. That is followed by a description of the problem at hand in section 3. We also present a way such a loophole can be introduced in any android app in this section. We present a working prototype that is capable of detecting such a vulnerability in section 4. We conclude the paper with a discussion of related work in section 5 and future research directions that can lead to more vulnerability discoveries in section 6.

2. SYSTEM OVERVIEW

Heimdall has two components. The first component is an app that is installed on a user's mobile device. The second component is a webservice that receives install, uninstall and

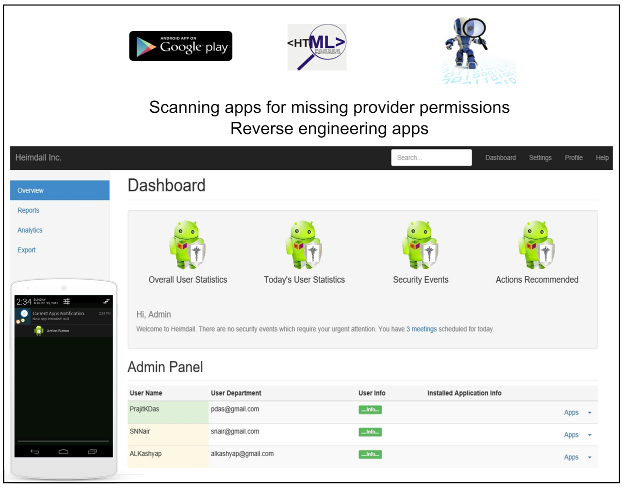

Figure 1: System Overview

update notifications when said activity occurs on the phone for any app. Upon intimation the server processes the list of heuristics that apply to the app at the moment and generates a set of actions for a system administrator. At this point the system administrator can take an appropriate action based on the detected threat level. At present we are working with the content provider heuristic to determine the impact of this vulnerability. The only solution that is possible, at the moment, is to uninstall the app and that notification is then sent back to the user's device automatically.

Heimdall server has two additional capabilities. The first is to generate reverse engineered apps that we then test on the mobile devices. The reverse engineering process removes any provider associated permission and ensures that the "exported" tag for the provider is set to true. The second capability is to scan for missing provider permissions for known apps. For this purpose we downloaded about 1500 apps from the Google Play Store and then we use the [2] package to decompile the apks and parse the manifest files to determine if any of the app's provider is missing the permission association. Naturally, as we include more heuristics, Heimdall will become capable of detecting much more such vulnerability.

3. VULNERABILITY DESCRIPTION

When it comes to content providers we observed that Google's Android documentation describes to possible scenarios. The first states that any provider that has not got an associated permission should not be accessible from another app.

- "[...]If a provider's application doesn't specify any permissions, then other applications have no access to the provider's data. However, components in the provider's application always have full read and write access, regardless of the specified permissions." [3]

- "[...]All applications can read from or write to your provider, even if the underlying data is private, be-

[2]apktool `https://ibotpeaches.github.io/Apktool/`
[3]Error in specification `http://developer.android.com/guide/topics/providers/content-provider-basics.html#Permissions`

cause by default your provider does not have permissions set. To change this, set permissions for your provider in your manifest file, using attributes or child elements of the <provider> element. You can set permissions that apply to the entire provider, or to certain tables, or even to certain records, or all three." [4]

Unfortunately, we found the first statement to be untrue. The following table shows that when a content provider app is not associating a permission to its provider then we have data leakage. This happens because Android does not verify that each provider has an associated permission. The possible solution that we can have to mitigate this issue would either require a change in how content provider access control is handled on android or a change in the app developer's code.

Content Provider app	**Content accessing app**	**Remark**
No permission associated with provider	No permission used	**Potential data leakage**
Permission associated with provider	No permission used	Permission denied
Permission associated with provider	Permission used	Ideal scenario
No permission associated with provider	Permission used	Instillation error

4. EVALUATION

5. RELATED WORK

Research being done to predict user's preferences by a number of people [1, 10, 7, 8]. Owing to that research we make an assumption that it is possible to fairly accurately create user permission choices on Android devices. However, our goal is separate from theirs in a threefold manner. Firstly we are defining policy rules for users which may allow, deny or allow with caveat specific permissions depending on the user context. Secondly we are not trying to show that it is possible to learn a user's policy from scratch rather we are agreeing with their observation that it is possible to use privacy profiles to define or group user preferences [8]. Instead we are trying to show that with user feedback it is possible to reach an individual user's "perfect" policy with a certain probability. Thirdly, we are researching ways to include app provenance information, api usage and observed mobile behavior [4] to compute a metric that will accurately measure the trustworthiness of an app. Playdrone : Crawls Playstore- how playstore evolved - source code analysis of library usage - similar app detection-secret authentication key storage (can be found by decompilation) (1) native libraries are heavily used by popular Android applications, limiting the benefits of Java portability and the ability of Android server overloading systems to run these applications, (2) 25% of Google Play is duplicative application content, and (3) Android applications contain thousands of leaked secret authentication keys which

Andradar : First, we can discover malicious applications in alternative markets, second, we can expose app distribution strategies used by malware developers, and third, we can monitor how different markets react to new malware. To identify and track malicious apps still available in a number of alternative app markets.

Android Security : discuss the Android security enforcement mechanisms, threats to the existing security enforcements and related issues, malware growth timeline between 2010 and 2014, and stealth techniques employed by the malware authors, in addition to the existing detection methods. This review gives an insight into the strengths and shortcomings of the known research methodologies and provides a platform, to the researchers and practitioners, toward proposing the next-generation Android security, analysis, and malware detection techniques.

ANDRUBIS:, a fully automated, publicly available and comprehensive analysis system for Android apps. ANDRUBIS combines static analysis with dynamic analysis on both Dalvik VM and system level, as well as several stimulation techniques to increase code coverage.

changes in the malware threat landscape and trends amongst goodware developers. Dynamic code loading, previously used as an indicator for malicious behavior, is especially gaining popularity amongst goodware App analysis for astma!!

App behavoir against description CHABADA tool clustering apps by description topics, and identifying outliers by API usage within each cluster, our CHABADA approach effectively identifies applications whose behavior would be unexpected given their description. Recommendations for android eco system

author et.al. developed a formal android permission model to analyze the permission protocol used using Alloy. Using Alloy analyzer, they reasoned over the model to detect possible vulnerabilities in the protocol. It also generated counter examples which can possibly exploit the vulnerabilities in the protocol. Their work could detect a vulnerability in which an application with normal security level permission was able to access another application's dangerous level custom permission given the following conditions. First, both the permission names are the same and second the application with lesser security level is installed first. But we differentiate from them such that instead on just focusing on

6. CONCLUSIONS

7. ACKNOWLEDGMENTS

Support for this work was provided by NSF grants 0910838 and 1228198.

8. ADDITIONAL AUTHORS

Additional authors: Ting Zhu (University of Maryland, Baltimore County, email: `zt@umbc.edu`) and Tim Finin (University of Maryland, Baltimore County, email: `finin@umbc.edu`).

9. REFERENCES

[1] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor. Capturing location-privacy preferences: Quantifying

[4] Installing permissions `http://developer.android.com/guide/topics/providers/content-provider-creating.html#Permissions`

accuracy and user-burden tradeoffs. *Personal Ubiquitous Comput.*, 15(7):679–694, Oct. 2011.

[2] R. Chang. 10 alternative android app stores, 2014.

[3] M. Conti, V. T. N. Nguyen, and B. Crispo. Crepe: Context-related policy enforcement for android. In M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 331–345. Springer Berlin Heidelberg, 2011.

[4] 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. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–6, 2010.

[5] D. Ghosh, A. Joshi, T. Finin, and P. Jagtap. Privacy control in smart phones using semantically rich reasoning and context modeling. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 82–85, 2012.

[6] P. Jagtap, A. Joshi, T. Finin, and L. Zavala. Preserving privacy in context-aware systems. In *Fifth IEEE Int. Conf. on Semantic Computing (ICSC)*, 2011.

[7] J. Lin, B. Liu, N. Sadeh, and J. I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 199–212, Menlo Park, CA, July 2014. USENIX Association.

[8] B. Liu, J. Lin, and N. Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 201–212, New York, NY, USA, 2014. ACM.

[9] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 527–542, Berkeley, CA, USA, 2013. USENIX Association.

[10] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao. Understanding and capturing people's privacy policies in a mobile social networking application. *Personal Ubiquitous Comput.*, 13(6):401–412, Aug. 2009.

[11] Statista. Number of available applications in the google play store from december 2009 to july 2015, 2015.

[12] C. Trout. Android still the dominant mobile os with 1 billion active users, 2014.

[13] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109, May 2012.