An Empirical Study of the the Permission Gap in Android Applications

Daniel E. Krutz and xxxx
Rochester Institute of Technology
1 Lomb Memorial Drive
Rocheser, NY, USA
{dxkvse, xxxx}@rit.edu

ABSTRACT

Applications using permission based security models such as Blackberry and Android are often granted more permissions than they need. This is known as a permission gap. Developers often struggle with determining the appropriate level of permissions to grant their applications and is largely a difficult, manual task. Granting an application too many permissions will not abide by the principle of least privilege while not granting enough will lead to application crashes.

While there has been a substantial amount of research in ways to make the Android permissions model more granular, there has been little work in examining why and when developers grant these extra, unneeded permissions. In the following work, we will perform an in-depth analysis about software engineerings and why they grant these extra permissions. *[Dan says: update]*

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Keywords

xxxxxxxx

1. INTRODUCTION

[Add citations to this section]

There are over 675,000 Android applications with over 12,000 being added every month. **[find citations and update values]** Unfortunately, many of these applications also contain serious security vulnerabilities which often lead to malicious behavior ranging from learning the behavior of the user to the theft of sensitive user information.

Android uses a permission-based security model, one where every application is associated with certain permissions allowing it to access specific system resources. While this type of permission granting system has some benefits such as giving the user control of their privacy, it also has drawbacks as well. One is that developers often request more permissions for their applications than they actually need. This is known as a "permission gap" and violates the *principle of least privilege*, which states that developers should grant their applications the minimum level of security that they need to function. Extra privileges violate this principle and lead to an increased risk of security vulnerabilities including code injection or return-oriented programming [5]. Wei *et al.* found that 44.8% of Android applications were over privileged.

Research has shown that developers add to this permission gap for a variety of reasons. There is no forthright or defined process of determining the appropriate permission level, and they must walk a fine line of granting their application enough privileges so the applications do not throw exceptions, but not allow these extra, unneeded privileges [9]. Far too often, developers make the mistake of adding permissions they *think* the system needs, not ones which it actually does. This may be caused due to documentation errors, confusing permission names, and even temporary permissions added for testing which were never removed for release [4] [10] [6].

Android groups permissions into groups and forces the selection of all actions in the group if one is required by the application. For example, if the developer wants to grant their application access to a URL, they will also be giving permissions on a variety of other network resources in this generalized group [9].

In the following work, we propose the first large scale empirical analysis of why developers so often do not adhere to the rudimentary principle of least privilege. We.....

We propose.. - This work is innovative because.....

- Takl about the study here

- How do we plan on doing the study **[section will obviously nee**

The rest of the paper is organized as follows. Section **??**

1.1 Research Questions

dan

2. ANDROID PERMISSION STRUCTURE

The Android security model is a permission-based system where applications need to be granted access to various areas of functionality before they may be used. If an application attempts to perform an operation which it does not have permission, a *SecurityException* is thrown. When an Android

application is created, its developer must declare in advance what permissions the application will require [6]. These security settings are stored in the `AndroidManifest.xml` file and include a ride range of permissions including the ability to transmit data, access personal information, and charge subscription fees [3]. A few of these are `INTERNET`, `READ_CONTACTS`, and `WRITE_SETTINGS`. When an application is invoked, this manifest file is examined to determine the appropriate permissions the application should possess.

Unfortunately, developers are often forced to grant more permissions to their application than they actually need. Due to the granularity of the permission spectrum used by Android, the developer must often grant more permissions to their application than it actually requires. For example, an application which needs to send information to one site on the Internet will need to be given full permissions to the Internet, meaning that it may communicate with with all websites [8].

Each Android permission is assigned under one of several protection levels which helps to define the risk level associate with each permission [1]. These levels are:

Normal Default value with minimal risk. Grants the application access to isolated, application level features. Minimal risk is incurred by other applications and the end user.

Dangerous An elevated permission risk which provides the application access to private user information or control over the device should could negatively impact the end user. Upon the instillation of an application, the user needs to explicitly grant these permissions.

Signature Highest level of permission. Granted by the system only if the application is signed by the same certificate as was declared in the permission. The user is not notified that this permission is granted if the certificates match, and the application will automatically possess these privileges.

signatureOrSystem Application must be in the Android system image or signed with the same certificate as the application which declared the permission. The majority of developers should avoid using this protection level, as it was intended for use by applications created by multiple vendors who needs to have them built into the system image.

The number of Android permissions has risen from 103 in API Level 3 (Cupcake), in 2009 to over xxx in API 19 (KITKAT) **[find number and citation]**. This list is expected to continue to grow, with the permissions in the *Dangerous* group the largest, and fastest growing. Permissions are added over time as device functionality grows and changes [10].

[Dan says: Make sure I clearly described everything and laid the appropriate groundwork for the rest of the paper]

3. COLLECTION PROCESS

BLah

4. RESULTS

BLah

5. DISCUSSION

BLah

6. THREATS TO VALIDITY

Blah

7. FUTURE WORK

Future Work

8. RELATED WORK

The topic of reducing the permission gap in Android applications has received a considerable amount of attention recent years. Much of the existing work on this area has dealt with ways of reducing these unneeded permissions and the security vulnerabilities they may lead to. Jeon *et al.* introduced a framework for creating finer-grained permissions in Android. They believe that the course grained permissions currently used by Android limit developers by forcing them to choose all of the permissions located in each bucketẅhen they really only want to add a few of them. This leads to applications having many more permissions than they actually require. The authors believe that finer-grained permissions would lead to only having the needed permissions used by an application, and thus lead to few vulnerability possibilities [8].

Wei *et al.* studied the evolution of Android to determine if the platform was allowing the system become more secure. They found that the privacy and overall security in the overall Android system is not improving over time and that the principle of least privilege is not being adequately addressed [10].

There have been several tools which have been developed to assist in the decision making, permission process for developers. Felt *et al.* created a tool known as *Stowaway* which uses a permissions-to-API calls maps in order to statically analyze request permissions in Android applications [6]. This tool notes the extra, unneeded permissions requested by the application, along with permissions that should have been requested, but were not. One criticism of this tool is that it may be difficult to determine if a permission is actually used through the use of static analysis.

Permlyzer is another tool which was built to determine where permissions are utilized in Android applications by using a mixture of static and runtime analysis [11]. This is a recently published tool, so it has not yet been discussed or used in a substantial amount of subsequent research. However, the authors were able to achieve promising results and this may be a powerful tool for assisting in the permissions granting decision process for developers. *PScout* was another tool developed to extract permission specifications from Android applications using static analysis [2]. While the authors of this tool were able to achieve promising results, subsequent work has criticized this tool for not being accurate enough since Android's permissions could be different at runtime, which is something the tool is not capable of discovering [12].

While this work represents the largest known, empirical analysis of developers allowing over privilege to occur in Android applications, it is not the first research into developers not following the principle of least privilege. Felt *et al.* described some common developer errors they found using their tool Stowaway, including confusing permission names,

the use of depreciated permissions and errors due to copy and pasting existing code [6]. In another work, Felt *et al.* very briefly described some inclinations they had for developers gave too many permissions to applications, but this was largely based on assumptions and not data [7].

Bartel *et al.* and Wei *et al.* also discussed some basic, high level discoveries about why developers make these mistakes [4] [10]. While these works were beneficial for numerous reasons, none to date have explored the question of why developers do not adhere to the principle of least security nearly as much as they should. **[Make sure to tweak this section with our actual findings]**

9. CONCLUSION

Conclusion

10. REFERENCES

[1] Android developers.

[2] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: Analyzing the android permission specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 217–228, New York, NY, USA, 2012. ACM.

[3] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM.

[4] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus. Automatically securing permission-based software by reducing the attack surface: An application to android. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 274–277, New York, NY, USA, 2012. ACM.

[5] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on android. In *Proceedings of the 13th International Conference on Information Security*, ISC'10, pages 346–360, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] 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*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[7] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

[8] J. Jeon, K. K. Micinski, J. A. Vaughan, N. Reddy, Y. Zhu, J. S. Foster, and T. Millstein. Dr. android and mr. hide: Fine-grained security policies on unmodified android. 2011.

[9] T. Vidas, N. Christin, and L. F. Cranor. Curbing android permission creep. In *In W2SP*, 2011.

[10] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 31–40, New York, NY, USA, 2012. ACM.

[11] W. Xu, F. Zhang, and S. Zhu. Permlyzer: Analyzing permission usage in android applications. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 400–410, 2013.

[12] 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 2013 ACM SIGSAC conference on Computer & communications security*, pages 611–622. ACM, 2013.