Poster: Privacy Risks from Misconfigured Android Content Providers

Christopher Lenk
Zentrale Stelle für Informationstechnik im Sicherheitsbereich (ZITiS)
Munich, Germany

Johannes Kinder
Ludwig-Maximilians-Universität München (LMU Munich)
Munich, Germany

ABSTRACT

Android applications record and process personal user data, and they can share it among each other through *content providers*. While the access is protected through multiple mechanisms, unintentional misconfigurations can allow an attacker to access or modify private application data. In this work, we study how content providers protect private data in a systematic study on 14.4 million Android apps. We identify potentially vulnerable apps by using static analysis to successively reduce the set of target apps. Using a custom attack app, we can confirm data leakage in practice and successfully access privacy-sensitive information. We conclude that this points to an inherent problem in designing secure Android applications and discuss possible mitigations.

CCS CONCEPTS

• **Security and privacy** → **Mobile platform security**; *Privacy protections*.

KEYWORDS

Android, content provider, data leakage, static analysis, privacy

ACM Reference Format:
Christopher Lenk and Johannes Kinder. 2023. Poster: Privacy Risks from Misconfigured Android Content Providers. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3576915.3624389

1 INTRODUCTION

Personal data is generated and processed by mobile applications [4], and it is stored in system-wide locations and app databases [7]. The Android operating system implements isolating sandboxes to separate components and applications from each other. For interactions with other apps and components, the construction of an *intent* or the call of a *content provider* is necessary. However, these forms of inter-app communication also offer an attack surface that has been known for many years [5, 15], especially if applications do not adequately secure their endpoints. In 2012, Zhou and Jiang [15] described *content provider leakage*, a then widespread issue in which content providers were generally accessible from other apps and made data available to them. In November 2012, with the release of Android 4.2, Google then introduced a modified parameter `exported` for content provider, which had to be set explicitly in order to make it available. Despite the reduction in the attack surface, several approaches of attacks against content providers were successfully carried out in the following years [9, 11, 13].

In current versions of Android, the parameter `exported` and *custom permissions* are the protective barriers for content providers and their data. Custom permissions can be defined by the app developer and must be requested by third party apps to access the content provider. Whether a permission is allowed at installation time or at runtime depends on the selected *protection level* of the permission. The developer has full control over these processes and there are only security recommendations in the Android documentation without mandatory specifications. As a result, there is the possibility that content providers that contain similar data types are equipped with different protection levels in the app configuration, which means that data can be accessed without authorization [5, 9].

In this work, we demonstrate that content providers are vulnerable to misconfiguration and are often inadequately protected as a result. Content provider leakage still exists today and can be exploited. We examined the configurations of 3.4 million Android applications with content providers and associated custom permissions and were able to identify 18,802 app versions that were equipped with insecure parameters. We selected 179 currently available targets to assess exploitability and impact on data security and privacy of the user. We were able to access data with direct personal reference or that allow conclusions to be drawn about everyday concerns or preferences. This shows that a rethinking of how personal data is stored and shared within Android is necessary in order to protect user privacy.

2 METHODOLOGY

We use the Androzoo dataset [2] as the source for our data and process all Android packages available at the end of June 2021 to extract application properties from the `AndroidManifest.xml`. The data collection includes 14.4 million apps with 23 data fields of properties like the name and versions as well as the components of an app like permissions, activities, providers and receivers.

Since we want to search for misconfigurations in the definition of content providers as shown in Figure 1, we examine the declared custom permissions and their protection levels for anomalies. We were able to detect many differences in the configuration of apps, although their content providers provide similar functions. We can find, for example, over 700 apps where the permissions of the content provider have a protection level of zero (normal permission) and over 500 have a protection level of two (signature-based permission). We focus on applications that implement an exported

CCS '23, November 26–30, 2023, Copenhagen, Denmark

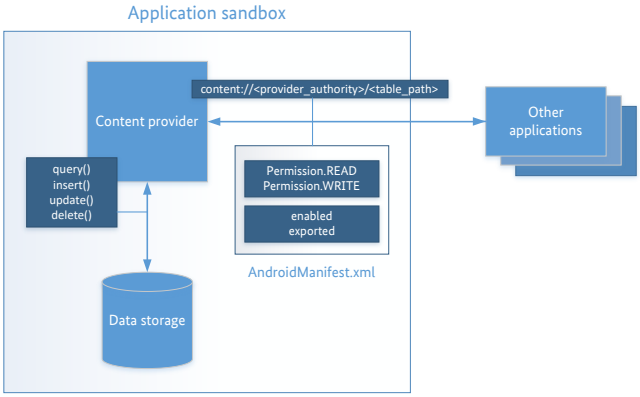

Figure 1: Schematic overview of content providers [6].

Table 1: Groups of exploitable data fields.

Data category	Objects	Count
Informational	Push notifications and log contents like warnings and errors	6
Configuration	User settings, preferences and app parameters	73
Application	User input like texts, lists and search requests	59
Personal	Names, communication details, locations and health data	55
No data	No objects accessible due to functional errors, config changes or app unavailability	72

content provider and associated permissions with protection level zero, since these can be accessible from other apps.

Since the dataset also contains apps that are no longer supported or are of low quality, we use a crawler that collects current information from the most widespread market, the Google Play Store, to focus on applications that are currently distributed and used. We examined the methods of the content providers of these apps with static analysis to extract the structure of the app database or the implemented content URIs (Uniform Resource Identifier).

We test the attack path with a custom Android app. After checking the targeted provider and if the associated permissions have been granted, we are able to receive the column names via the content URI and to access the data fields via the provider's query function. If permission for write access is also granted, data fields can be changed and deleted.

3 RESULTS

Overall, we gradually filtered 417 current and available variants from the 18,802 app versions. After completing the static analysis, 179 applications can be included in the detailed case study. These have hundreds to millions of downloads and come from various functional categories of the Google Play Store such as education, personalization and tools. A Samsung Galaxy S10+ with Android version 9 was used to carry out the test runs. In our experiments, we were able to access four different categories of information in 107 cases. As can be seen in Table 1, this includes informational data, configurations, application content and personal data. Excerpts from the case studies with download numbers are presented in §3. In 72 cases, no data could be accessed because there were functional problems with the app or the provider, the configuration had changed in the meantime, or the app could not be installed.

Informational data are notifications and messages from the log system. We were able to read IDs and notification data from an app that manages home cameras (5M+ installations). A telephony app (5K+ installations) shares verification data such as hash values and the content and times of push messages. An alternative app for SMS and MMS messages (over 1M installations) allows access to created log entries. This data can be used to draw conclusions about the behavior of the user and notifications can also have personal references.

Configuration data includes settings and parameters made by the user. For launcher apps (with up to 100M+ installations) used to customize the Android home screen, we were able to get screen positions and modification times. License information including the country, device and checksums could also be read out. A banking app (1M+) shared the user's login preferences. For other apps (up to 10M+), data on the status of accounts and application components such as widgets was accessible. A note app (500K+) released the scroll position within a list via a provider. Configuration data provides a detailed insight into processes and behavior and can be used by malicious actors to prepare attacks.

In the category of application data, which includes content entered by the user such as texts, calendar entries, task reminders and search queries, we were able to read out various amounts of data, starting with individual data objects up to the entire app content. For a note app (100K+), the title and content were extractable. A shopping list app (1M+) made all lists available with details on purchase objects, planned locations and prices. Several applications like a flight tracker (10M+) share search suggestions and queries, for example on travel connections. We were able to get added points of interest with coordinates and descriptions for various map apps (up to 1K+). A radio app (1M+) allowed access to favorites and information on how often they were played. In an application for managing customer relations (1K+), we can read out data on conversations. The unprotected release of internal app content and search queries violates the data security of the applications. An attacker can steal these directly and exploit them in various harmful ways depending on the app and the value of its data.

Personal data is the category with the highest security relevance. In contrast to application data, this is information that has direct personal reference, such as names, communication details or financial and health data. A group of government education apps (up to 1M+) allow access to a complete profile of students with names, institutions, courses and learning status. A network function can also be used to obtain information on networked participants. Different variants of a mobile online game (up to 10M+) share login data such as magic numbers and login tokens with each other, but also with

all other apps due to the configuration. An app for rating and ordering food deliveries (1M+) releases names and contact details. With a weight loss app (100K+), we were able to read out the entered weight with the associated time stamp. An app for farm management (5K+) allows access to detailed information about responsible persons, the location of the farm and the resident animals. We were able to identify a transport logistics app (10K+) that releases data from the linked company account, such as names and roles, via a provider. With an app for blood sugar measurement (100K+) using a connected additional device, the glucose value and date could be accessed without protection.

The results of the experiments show the critical consequences that an insufficiently configured content provider can have. Accessing the data, which can be performed by any other app, violates the user's privacy. This circumvents basic security mechanisms of the Android system and damages the user's trust in the secure processing of data. Functions for exchanging data with direct and indirect personal reference must therefore be protected accordingly. In the special case that health or financial information needs to be shared, the highest level of protection is necessary. Due to the value of personal data, the option of deactivating the provider function should be considered if this can also be enabled in another way.

4 DISCUSSION

The identified misconfigurations allow malicious actors to access and modify app-specific data. The case studies prove the effects of these attacks. Done over a longer period of time, this can allow for the creation of profiles that include daily concerns, personal preferences, and appointments, and provide stirrups for potential attacks and social engineering. Especially in the case of health data, a leak can have serious consequences such as attempts at extortion, payment fraud or unauthorized disclosure. But even log entries, usernames and notifications can also be used to prepare specific attacks and construct threats.

The issue of misconfiguration points to a general problem in the concept of the Android system. Although various security improvements have been implemented after uncovering attack vectors [5, 15], there is still potential to exploit inter-app communication. The security of applications and their components is based on recommendations and not on standardized and mandatory regulations. Due to the flexibility in the definition of parameters, it can happen that these are set inadequately because the concept has not been understood or misjudgments occur. Since there are no checks if a parameter or component has been defined sufficiently secure, these are implemented without protection against data attacks.

The security of Android's inter-app communication needs to be improved from both the developer and system side. The developer must be made aware of the impact of mistakes in the configuration of developed apps and the influence of parameters on the Android system and data. In addition, the concept of protecting content providers should be improved. One possibility would be the introduction of a component at database level or between the provider and the data structure, which checks queries based on defined rules and, if necessary, adjusts or blocks them [1, 3, 12]. At the app level, it would be possible to check the Android manifest for incorrect or missing configurations [8, 10], for example with existing tools like CodeQL. The standardization of provider parameters would also be conceivable. A specified manifest scheme based on security recommendations could be used to check and adjust configurations against a secure model [14].

5 CONCLUSION

We developed a methodology to specifically search for configuration parameters within Android applications that allow access to sensitive data from content providers and applied it to a dataset of 14.4 million apps. We were able to carry out a detailed study with 179 current applications of different categories, which shows that errors in the configuration of components for data sharing can expose private user data in practice. In detailed case studies we proved the successful exploitation of the vulnerability that can have a significant impact on the privacy and the everyday life of users of Android devices. While no easy solutions to this problem are in sight, developers need more assistance to make the right choices and build secure applications that protect the personal data of their users.

REFERENCES

[1] Aisha I. Ali-Gombe, Golden G. Richard III, Irfan Ahmed, and Vassil Roussev. 2016. Don't Touch that Column: Portable, Fine-Grained Access Control for Android's Native Content Providers. In *Proc. ACM Conference on Security & Privacy in Wireless and Mobile Networks (WISEC)*. ACM, 79–90.

[2] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: collecting millions of Android apps for the research community. In *Proc. 13th Int. Conf. Mining Software Repositories (MSR)*. ACM, 468–471.

[3] Sven Bugiel, Stephen Heuser, and Ahmad-Reza Sadeghi. 2013. Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies. In *USENIX Security Symp.* 131–146.

[4] Kai Chih Chang, Razieh Nokhbeh Zaeem, and K. Suzanne Barber. 2020. Is Your Phone You? How Privacy Policies of Mobile Apps Allow the Use of Your Personally Identifiable Information. In *2nd IEEE Int. Conf. Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, 256–262.

[5] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David A. Wagner. 2011. Analyzing inter-application communication in Android. In *Proc. 9th Int. Conf. Mobile Systems, Applications, and Services (MobiSys)*. ACM, 239–252.

[6] Google. 2023. Content providers | Android Developers. https://developer.android.com/guide/topics/providers/content-providers Accessed: July 18, 2023.

[7] Google. 2023. Review how your app collects and shares user data | Android Developers. https://developer.android.com/guide/topics/data/collect-share Accessed: July 21, 2023.

[8] Zhihui Han, Liang Cheng, Yang Zhang, Shuke Zeng, Yi Deng, and Xiaoshan Sun. 2014. Systematic Analysis and Detection of Misconfiguration Vulnerabilities in Android Smartphones. In *13th IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE Computer Society, 432–439.

[9] Behnaz Hassanshahi and Roland H. C. Yap. 2017. Android Database Attacks Revisited. In *Proc. ACM Asia Conf. Computer and Communications Security (AsiaCCS)*. ACM, 625–639.

[10] Ajay Kumar Jha, Sunghee Lee, and Woo Jin Lee. 2017. Developer mistakes in writing Android manifests: an empirical study of configuration errors. In *Proc. 14th Int. Conf. Mining Software Repositories (MSR)*. IEEE Computer Society, 25–36.

[11] Ryan Johnson, Mohamed Elsabagh, Angelos Stavrou, and Jeff Offutt. 2018. Dazed Droids: A Longitudinal Study of Android Inter-App Vulnerabilities. In *Proc. Asia Conf. Computer and Communications Security (AsiaCCS)*. ACM, 777–791.

[12] Simone Mutti, Enrico Bacis, and Stefano Paraboschi. 2015. SeSQLite: Security enhanced sqlite: Mandatory access control for Android databases. In *Proc. 31st Annu. Computer Security Applications Conf. (ACSAC)*. 411–420.

[13] Hossain Shahriar and Hisham M. Haddad. 2014. Content Provider Leakage Vulnerability Detection in Android Applications. In *Proc. Int. Conf. Security of Information and Networks (SIN)*. ACM, 359.

[14] Yuqing Yang, Mohamed Elsabagh, Chaoshun Zuo, Ryan Johnson, Angelos Stavrou, and Zhiqiang Lin. 2022. Detecting and Measuring Misconfigured Manifests in Android Apps. In *Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS)*. ACM, 3063–3077.

[15] Yajin Zhou and Xuxian Jiang. 2013. Detecting Passive Content Leaks and Pollution in Android Applications. In *Annu. Network and Distributed System Security Symposium (NDSS)*. The Internet Society.