A-SIT

Zentrum für sichere Informationstechnologie - Austria

A-SIT

Untersuchung der Content Provider-Schnittstelle unter Android

Autor:
Gerald Palfinger
Tel: +43 316 873 -
Mail: gerald.palfinger@a-sit.at

Datum: 30.06.2023

Abstract/Zusammenfassung: Content Provider werden auf Android eingesetzt, um Daten anderen Komponenten und Anwendungen in definierter Weise zur Verfügung zu stellen. Sie fungieren als Schicht zwischen dem zugrunde liegenden Datenspeicher und den Anwendungskomponenten, die Zugriff auf die Daten benötigen. In neueren Android-Versionen nimmt dieses System jedoch zusätzlich dazu einen immer wichtigeren Platz in der Sicherheitsarchitektur des Systems ein. In diesem Bericht wird darauf eingegangen, wie das System der Content Provider funktioniert und wie es technisch umgesetzt ist. Weiters wird darauf eingegangen, welche sicherheitsrelevanten Schwachstellen in Bezug auf Content Provider gefunden wurden.

Inhalt

1. Einleitung

Auf der Android-Plattform spielen die Bereitstellung und gemeinsame Nutzung von Daten eine entscheidende Rolle. Ein Content Provider ist dabei ein zentraler Bestandteil des Android-Frameworks, der den Zugriff auf strukturierte Daten für verschiedene Anwendungen ermöglicht. Dabei ist ein Content Provider eine Schnittstelle, die den Zugriff auf Datenbanken, Dateien oder andere persistente Datenspeicher in Android ermöglicht. Content Provider sind hierbei ein Teil der von Android bereitgestellten Interkomponentenkommunikation (Inter-Component Communication, ICC). Die Content Provider-Schnittstelle ermöglicht dabei lesenden und schreibenden Zugriff auf Daten. Insbesondere ermöglichen Content Provider auch den gemeinsamen Zugriff auf Daten zwischen verschiedenen Applikationen. Dadurch können Anwendungen auf eine einheitliche und koordinierte Weise auf dieselben Daten zugreifen und es abstrahiert dabei die Komplexität des dahinterliegenden Speicherkonzepts. Content Provider ermöglichen die Definition von Berechtigungen, um den Zugriff auf sensible Daten einzuschränken und unautorisierte Zugriffe zu verhindern. Sie stellen Daten über eine einheitliche Schnittstelle auf Basis eines Content-URI (Uniform Resource Identifier) zur Verfügung. Anwendungen können diesen URI verwenden, um über die Programmierschnittstelle auf die vom Content Provider bereitgestellten Daten zuzugreifen. In neueren Android-Versionen nehmen Content Provider jedoch auch einen immer wichtigeren Platz in der Sicherheitsarchitektur des Systems ein. So wird beispielsweise der Zugriff auf das Dateisystem über das sogenannte Storage Access Framework abstrahiert, das wiederum auf Content Providern basiert. Damit soll verhindert werden, dass jede Anwendung, die Zugriff auf

einzelne Dateien benötigt, direkt die Berechtigung für den Zugriff auf alle am Gerät gespeicherten Dateien erhält. Die Verantwortung für die Rechteverwaltung wird dadurch jedoch vollständig auf den jeweiligen Content Provider übertragen. In diesem Bericht wird darauf eingegangen, wie Content Provider vom Android-System verwendet werden, wie sie von Drittanbieteranwendungen bereitgestellt werden können und welche Schwachstellen in der Literatur in Bezug auf Content Provider gefunden wurden.

2. Hintergrund

Content Provider können von jeder Anwendung bereitgestellt werden, um Daten mit anderen Applikationen zu teilen. Es gibt jedoch auch Content Provider, die durch das System selbst bereitgestellt werden [1]. Dadurch sind diese Content Provider auf jedem Android-Gerät vorhanden. Viele dieser Content Provider bieten den Zugriff auf vertrauenswürdige Informationen, wie beispielsweise Kontakte, Kalendereinträge, Nachrichten und weitere persönliche Informationen. Deswegen sind diese durch eine eindeutig identifizierbare Berechtigung gesichert [2].

In älteren Android-Versionen konnten Applikationen Dateien über den gemeinsamen internen Speicher teilen. Dazu musste die Applikation die Berechtigung READ_EXTERNAL_STORAGE zum Lesen beziehungsweise die Berechtigung WRITE_EXTERNAL_STORAGE zum Schreiben von Dateien in diesem Speicherbereich anfordern. Mit Android 6 [3] musste diese Berechtigung nunmehr zwar durch den Benutzer beziehungsweise die Benutzerin erteilt werden, jedoch erlaubte eine solche einmalige Zustimmung den dauerhaften Zugriff auf alle Daten, die im gemeinsamen internen Speicherbereich gespeichert sind. Dies inkludiert neben den von anderen Applikationen gespeicherten Daten beispielsweise, Fotos, Videos, Sprachaufnahmen, Downloads und andere Mediendateien. Wie sich mit der Zeit gezeigt hat, wurde diese umfassende Berechtigung durch Applikationen auch ausgenutzt.

So wurde der gemeinsame Speicher ausgenutzt, um das Berechtigungsmodell von Android durch die Absprache zweier oder mehrerer konspirierenden Applikationen zu umgehen [4]. Jede Applikation für sich genommen sollte hierbei harmlos erscheinen, aber wenn sie gemeinsam agieren, können sie eine Bedrohung darstellen. Die beteiligten Applikationen können ihre Aktivitäten so koordinieren, dass sie weniger auffällig sind und nicht sofort Verdacht erregen. Durch die Aufteilung auf mehrere Applikationen ist es also schwieriger, dieses Verhalten zu erkennen. Eine der beteiligten Applikationen kann beispielsweise über eine erhaltene Berechtigung vertrauenswürdige Daten auslesen und am internen Speicher ablegen. Die zweite Applikation überträgt die gesammelten Daten dann weiter. Der interne Speicher dient hierbei als Kommunikationsmittel [5].

Ebenso konnte der Zugriff auf Fotos missbraucht werden, um den Standort des Nutzers bzw. der Nutzerin herauszufinden. Dies deshalb, weil viele Kameraanwendungen den Standort der Aufnahme in die Metadaten der Bilddatei schreiben. Durch Auslesen dieser Metadaten konnten Applikationen mit Speicherzugriff also feststellen, wo sich der Nutzer beziehungsweise die Nutzerin während der Aufnahme befand. Dadurch konnte effektiv die gesonderte Berechtigung, die für den Zugriff auf den Standort nötig ist, umgangen werden. Um diese Probleme zu beheben, wurde an einem neuen Zugriffssystem, den auf Content Providern basierendem Scoped Storage gearbeitet. Mit Android 10 wurde Scoped Storage als noch nicht verpflichtende Vorschau eingeführt, um den Applikationsentwicklern Zeit für die Umstellung zu geben. Mit Android 11 wurde Scoped Storage verbessert und ist seitdem verpflichtend. Für den Zugriff auf Mediendateien wie Fotos oder Videos gibt es hierbei die MediaStore Programmierschnittstelle. Diese erlaubt den Zugriff auf Mediendateien, welche durch die Applikation erstellt wurden. Will die Applikation zusätzlich auf Mediendateien zugreifen, die durch andere Applikationen am gemeinsamen internen Speicher erstellt wurden, so muss die Applikation die Berechtigung READ_EXTERNAL_STORAGE anfordern. Für den Zugriff auf alle anderen Daten wie beispielsweise Dokumente oder Downloads muss

das Storage Access Framework verwendet werden. Will eine Applikation Zugriff auf Dateien oder Verzeichnisse, die von anderen Applikationen erstellt wurden, so muss der Nutzer beziehungsweise die Nutzerin diesem konkreten Zugriff zustimmen. Ein Zugriff auf die applikationsspezifischen Ordner unter Android/data/ ist nicht mehr möglich. Dadurch wird erschwert, dass Applikationen über den internen Speicher unerkannt zusammenarbeiten.

3. Content Provider

Content Provider ermöglichen den einheitlichen Zugriff auf Daten einer Applikation. Content Provider ermöglichen sowohl einen lesenden als auch einen schreibenden Zugriff auf die Daten. Sie sind prädestiniert dafür, Daten anderen Applikationen zur Verfügung zu stellen, da sie die Implementationsdetails der darunter liegenden Speicherschicht abstrahieren, wie in Abbildung 1 gezeigt wird. Content Provider können jedoch auch von anderen Komponenten derselben Applikation aufgerufen werden. Dadurch kann der Zugriff auf Daten auch innerhalb einer Applikation abstrahiert werden, was den Austausch beziehungsweise Änderungen an dem darunter liegenden Datenspeicher vereinfacht. Wird ein Content Provider nur von Komponenten derselben Applikation verwendet, so sollte dieser auch nur der Applikation selbst zur Verfügung gestellt werden. Beim Definieren des Content Providers kann dies durch Setzen des android:exported Flags im Manifest der Applikation auf false realisiert werden. In neueren Android-Versionen ist dies aus Sicherheitsgründen gleichzeitig auch der Standardwert, sollte das Flag durch den Entwickler nicht explizit gesetzt werden. Dadurch soll verhindert werden, dass Daten versehentlich anderen möglicherweise nicht vertrauenswürdigen Applikationen auf dem Gerät zur Verfügung gestellt werden.

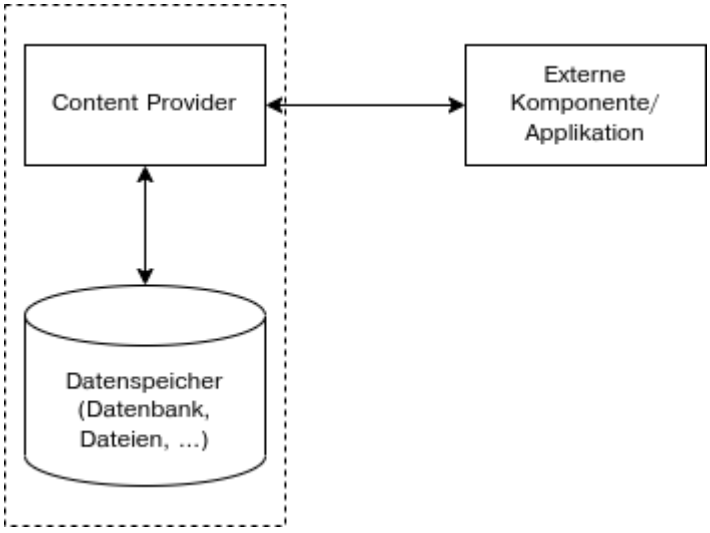

Abbildung 1 Content Provider abstrahieren den Zugriff auf Daten für externe Komponenten und Applikationen.

Ist das Teilen der Daten mit anderen Applikationen erwünscht, so muss das android:exported Flag im Manifest der Applikation auf true gesetzt werden. Dadurch kann der Content Provider von anderen Applikationen aufgerufen werden. Um Daten vor dem Zugriff durch unberechtigte Applikationen zu schützen, sollte der Content Provider jedoch durch eine Berechtigung geschützt werden. Hierbei ist es

möglich, getrennte Berechtigung für den lesenden oder schreibenden Zugriff zu erstellen oder eine Berechtigung, die beides ermöglicht. Wird der Content Provider nur verwendet, um Daten mit Applikationen desselben Entwicklers zu teilen, so kann der android:protectionLevel auf signature gesetzt werden. Dadurch haben nur Applikationen Zugriff, welche bei der Erstellung mit dem gleichen Schlüssel signiert wurden.

Temporärer bzw. partieller Zugriff

Darüber hinaus kann der Content Provider noch feinere Berechtigungen erteilen. Insbesondere ist es für den Provider möglich, Lese- oder Schreibzugriff auf einzelne Elemente zu erteilen [6]. Dazu gibt der Entwickler des Content Providers im Manifest der Applikation das Element grant-uri-permission an. Dieses kann auf true gesetzt werden, um den temporären Zugriff auf alle Elemente des Content Providers zu ermöglichen, oder es kann einzelne Pfade enthalten, für die ein temporärer Zugriff ermöglicht werden kann. Dies wird auch von vielen Content Providern des Systems unterstützt. So kann auf ein einzelnes Element eines Content Providers wie beispielsweise einem Kontakt oder einer Datei zugegriffen werden. Dabei wählt der Nutzer beziehungsweise die Nutzerin das jeweilige Element aus, das an die aufrufende Applikation freigegeben werden soll. Dadurch muss eine Applikation, die nur Zugriff auf beispielsweise einzelne Kontakte benötigt, nicht die umfassende und durch den Benutzer beziehungsweise die Benutzerin zu erteilende READ_CONTACTS Berechtigung anfordern.

Um Zugriff auf einzelne Elemente eines Content Providers zu erhalten, muss eine Applikation dabei folgendermaßen vorgehen:

- Die Applikation erstellt einen Intent und startet diesen. Bei einem Intent handelt es sich generell um den Aufruf einer anderen Applikation oder Komponente. Dieser Intent beinhaltet entweder eine konkrete Content URI, oder, sofern diese der Applikation nicht bekannt ist, einen Content Typ mit der Aktion einen Auswahldialog zu starten.
- Der Content Provider erhält den Intent und startet gegebenenfalls einen Auswahldialog oder fragt gegebenenfalls den Benutzer beziehungsweise die Benutzerin, ob der angeforderte Inhalt freigegeben werden soll.
- Der Nutzer beziehungsweise die Nutzerin wählt ein Element im Auswahldialog aus oder bestätigt die Anfrage.
- Der Content Provider erstellt wiederum einen Intent mit der URI zum ausgewählten Element und fügt für lesenden Zugriff FLAG_GRANT_READ_URI_PERMISSION und für schreibenden Zugriff FLAG_GRANT_WRITE_URI_PERMISSION hinzu. Dieser Intent wird der aufrufenden Applikation zurückgegeben.
- Über die erhaltende Content URI kann die aufrufende Applikation nun lesend und/oder schreibend auf das Element zugreifen.

4. Literatur

In diesem Kapitel wird darauf eingegangen, welche Schwachstellen in Bezug auf Content Provider erkannt und wie diese behoben wurden beziehungsweise, wie diese in Applikationen gefunden werden können. So wurden in [7] zwei verschiedene Arten von Schwachstellen untersucht, die bei der Verwendung von Content Providern auftreten können. Zum einen wurde untersucht, ob vertrauenswürdige Daten fälschlicherweise für andere Applikationen zur Verfügung gestellt werden. Weiters wurde untersucht, ob durch den schreibenden Zugriff auf Content Provider Daten der Applikationen missbräuchlich verändert werden können. In der Analyse im Rahmen dieser Arbeit wurden mehr als 62.000 Applikationen analysiert. Es konnte festgestellt werden, dass 2% dieser Applikationen möglicherweise vertrauenswürdige Daten an andere Applikationen freigeben. Diese Schwachstelle wurde als passives Datenleck definiert. Darüber hinaus wurde gezeigt, dass 1,4% der untersuchten Applikationen nicht nur einen lesenden, sondern auch einen schreibenden Zugriff auf mindestens einen der durch die Applikation definierten Content Provider ermöglichen. Dadurch war es den Autoren mitunter sogar möglich, das Verhalten der Applikationen zu beeinflussen. Haben diese Applikationen erhöhte Berechtigungen, so konnte mitunter sogar Einfluss auf Systemfunktionen genommen werden. Beispielsweise erlaubte das Ändern von Daten eines Content Providers einer Applikation es Telefonanrufe oder SMS-Nachrichten von bestimmten, von Angreifern ausgewählten Telefonnummern, zu verbieten. Um die Prävalenz dieser Schwachstelle zu verringern, sind Content Provider in neueren Android-Versionen nicht mehr standardmäßig exportiert. Das heißt, dass Content Provider, die durch den Entwickler definiert werden, nicht mehr für andere Drittanbieteranwendungen zugänglich sind. Ein Entwickler muss sich nun aktiv dafür entscheiden, den Inhalt eines Content Providers freizugeben beziehungsweise die Möglichkeit zur Änderung des Inhalts eines Content Providers zu ermöglichen. Diese Maßnahme soll gegen passive Datenlecks helfen. In [8] wurde jedoch gezeigt, dass manche Applikationen oder Bibliotheken vertrauenswürdige Daten absichtlich an andere Applikationen freigeben. Im Vergleich zur vorherigen Arbeit wurden hier also aktive Datenlecks gefunden. Besonders im Fokus standen dabei Werbebibliotheken, welche Informationen vom System abrufen und daraufhin weiterleiten. Die abgerufenen Daten stammten dabei auch aus Content Providern, welche vom Android-System zur Verfügung gestellt wurden.

Um den möglicherweise ungerechtfertigten Abruf von Daten aus Content Providern zu erkennen, müssen die jeweiligen Applikationen sowie deren Bibliotheken untersucht werden. Dabei wird in der Regel auf eine statische Analyse der Applikation zurückgegriffen. Bei dieser wird eine Untersuchung des Quellcodes, der Ressourcen und anderer Artefakte einer Android-Anwendung durchgeführt, ohne die Applikation tatsächlich auszuführen. Um Probleme mit Content Providern mit Hilfe von statischer Analyse zu entdecken, müssen diese als Quelle (Source) beziehungsweise Senke (Sink) unterstützt werden. Dies hat sich das IC3-Framework [9] neben der Unterstützung von anderen ICC-Paradigmen (Inter-Component Communication) zum Ziel gesetzt. Dabei werden die für Content Provider üblichen Uniform Resource Identifiers (URIs) unterstützt. Ebenso werden Intents unterstützt, welche für andere ICC-Paradigmen verwendet werden. Dabei werden die Content Provider-Aufrufe als auch die Intents als COAL-Spezifikation definiert. Bei COAL handelt es sich um eine von den Autoren definierte deklarative Sprache, welche es ermöglicht, ICC-Aufrufe zu deklarieren. Im Rahmen der Arbeit wurden dabei alle unter Android möglichen ICC-Nachrichten in 750 Zeilen COAL-Code definiert. Dadurch ist es möglich, einen interprozeduralen Kontrollflussgraphen zu erstellen, der für die statische Analyse benötigt wird. Auf Basis des Kontrollflussgraphen wurde weiters ein String-Parser entwickelt, welcher es ermöglicht, die Werte von ICC-Aufrufen aus Applikationen zu extrahieren. Damit war es möglich, etwa 85% der ICC-Aufrufe aus den 460 untersuchten Applikationen zu extrahieren. Bezogen auf URIs, welche für Content Provider verwendet werden, konnten 374 der 511 gefundenen URIs (72%) zu einem konkreten Wert aufgelöst werden. Ein Großteil der nicht aufgelösten URIs stammt aus Callbacks, beziehungsweise aus Aufrufen aus anderen Applikationen, welche zum Teil nur während der Ausführung erkannt werden können. Eine

kleinere Anzahl an URIs konnte nicht aufgelöst werden, da diese aus komplexeren Datentypen wie Listen oder Sets stammten welche von IC3 nicht unterstützt wurden.

Ein weiteres statisches Analyseframework, welches Content Provider unterstützt ist DroidRista [10]. Es basiert auf einer modifizierten Version von IC3 und verbessert es im Hinblick auf die Auswertung und Extrahierung von ICC-Links. Darüber hinaus wurde ein sogenannter Instrumentor entwickelt, welcher die Verbindung des Kontrollflussgraphen über ICC-Methoden hinweg verbessert. Dadurch können Applikationen, welche prozessübergreifende Kommunikationsmittel wie Content Provider verwenden, noch besser analysiert werden. Insbesondere wurde die Unterstützung für Callbacks verbessert. Dabei wird mittels statischer Analyse der Kontrollflussgraph erstellt, um daraufhin schrittweise die gefundenen Callbacks zu analysieren. Im Rahmen dieses Prozesses wird mit jedem Durchgang der Kontrollflussgraph um die gefundenen Callbacks erweitert. Dies resultiert in einer besseren Erkennung von ICC-Aufrufen. Weiters wurde die Analyse der APK verbessert. Dadurch findet DroidRista in der Evaluierung mehr ICC-Aufrufe und benötigt weniger Zeit als IC3. Bei der Analyse von Applikationen aus dem PlayStore konnten mehrere potenzielle Datenlecks gefunden werden.

APPLADroid [11] untersucht die Datenflüsse über mehrere Applikationen hinweg. So kann eine Applikation, die Zugriff auf eine Content Provider hat, der private Daten wie E-Mails oder Kontakte speichert, diese über einen eigenen Content Provider an weitere Applikationen weitergeben. Der ursprüngliche Content Provider kann dabei nicht erkennen, dass die Daten möglicherweise an nicht berechtigte Applikationen weitergegeben werden. Um solche Datenflüsse zu finden, führt APPLADroid eine applikationsübergreifende statische Analyse durch. Es basiert auf SniffDroid [12] und erweitert dieses unter anderem durch die Unterstützung von Content Providern. Ähnlich wie APPLADroid versucht auch COVERT [13] Applikationen zu finden, die mit anderen Applikationen zusammenarbeiten, um Berechtigungen zu umgehen. Dazu wird mittels statischer Analyse ein Modell der untersuchten Applikationen erstellt. Das erstellte Modell bildet die Interaktionen zwischen den Applikationen ab. Auf Basis dieses Modells wird dann untersucht, ob es potenzielle Schwachstellen durch die Interaktion der Applikationen gibt. In der Evaluierung wurden über 500 Applikationen auf Schwachstellen untersucht. Es wurde festgestellt, dass viele der Applikationen Zugriff auf durch Berechtigungen geschützte Informationen und Funktionen bieten und dadurch potenziell durch andere Applikationen angreifbar sind.

In [14] wurde die Programmierschnittstelle von Android nach fingerprintbaren Informationsquellen untersucht. Dabei wurden neben den Methoden und Feldern der Programmierschnittstelle auch die Content Provider untersucht, welche vom System bereitgestellt wurden. Zum Abruf der Content Provider ist eine URI notwendig. Um diese zu erhalten, wurden die Rückgabewerte der Methoden und der Inhalt der Felder auf Content-URIs untersucht. Dabei wurden alle Werte, welche mit content:// beginnen, extrahiert. Daraufhin wurde versucht, auf diese Content Provider zuzugreifen. Alle Provider, welche keine Berechtigung benötigten und aufrufbar waren, wurden daraufhin analysiert. Dazu wurde der Inhalt dieser Content Provider extrahiert, das heißt alle Zeilen und Spalten der Content Provider wurde ausgelesen. Dieser Prozess wurde auf verschiedenen Geräten wiederholt. Die erhaltenen Werte wurden daraufhin unter den Geräten verglichen. Unterschieden sich die Werte auf den untersuchten Geräten, so wurden diese genauer analysiert. Dabei stellte sich heraus, dass vor allem durch Herstelleranpassungen potenziell geheime Daten für alle Applikationen verfügbar waren. Dies inkludierte beispielsweise eindeutige Identifikationsmerkmale, E-Mail-Adressen und kryptographische Schlüssel.

In [15] werden vorinstallierte Applikationen auf Android-Smartphones auf potenzielle Schwachstellen überprüft. Diese können für Angreifer besonders interessant sein, da sie teilweise Zugriff auf Content Provider und Berechtigungen haben können, die durch Systemberechtigungen geschützt sind. Dazu wurden die vorinstallierten Applikationen von mehreren Android-Smartphones extrahiert. Auf diesen wurde dann eine statische Analyse ausgeführt. Dabei wurde ein Kontrollflussgraph von möglichen

externen Aufrufen der Applikation zur Nutzung der Berechtigung gesucht. Daraufhin wurde der Datenfluss analysiert, um zu erkennen, ob die Nutzung auch durch eine Drittanbieteranwendung ausgelöst werden kann. Dabei wurde festgestellt, dass viele der vorinstallierten Anwendungen die Nutzung einer Berechtigung ermöglichen. Ähnlich dazu wurden in [16] die vorinstallierten Applikationen von Custom ROMs analysiert. Auch hier konnte festgestellt werden, dass viele der vorinstallierten Applikationen vertrauliche Daten potenziell freigeben können.

Durch die Umstellung auf das auf Content Provider basierende Storage Access Framework kam es auch zu ungewollten Sicherheitslücken. So bietet das eingebaute Screenshot-Tool der Google Pixel-Geräte die Funktion, Screenshots direkt nach deren Aufnahme zu beschneiden, um beispielsweise private Informationen aus den Screenshots zu entfernen. Dazu öffnet das Screenshot-Tool die aufgenommene Datei und überschreibt die vorhandene Aufnahme mit der Editierten. Beim Überschreiben der Datei wird vom Screenshot-Tool der „write"-Modus der Programmierschnittstelle verwendet. In Android 9, also vor der Einführung des Storage Access Framework, wurde durch die Verwendung dieses Modus die vorhandene Datei geleert, bevor der geänderte Inhalt geschrieben wurde. Dadurch wurde die Bildschirmaufnahme korrekt beschnitten. Durch die Einführung des Storage Access Framework veränderte sich jedoch das Verhalten des „write"-Modus. Die vorhandene Datei wird nun nicht mehr vor dem Schreiben vom System geleert (dazu muss nunmehr der „write-truncate" Modus gewählt werden). Da die editierte Version der Bildschirmaufnahme kleiner ist als das Original, bleibt dadurch ungewollt ein Teil der Originalaufnahme in der resultierenden Datei über [17]. Wird diese Datei geteilt, so kann der übrig gebliebene Teil der Originalaufnahme rekonstruiert werden, wodurch potenziell geheim geglaubte Informationen wiederhergestellt werden können.

5. Fazit

In diesem Bericht wurde gezeigt, wofür die Content Provider-Schnittstelle unter Android verwendet wird. Es wurde näher darauf eingegangen, wie die darin gespeicherten Daten über Berechtigungen gesichert werden können. Ebenso wurde gezeigt, wie durch temporäre Berechtigungen der Zugriff durch externe Applikationen auf einen Teil der gespeicherten Daten beschränkt werden kann. Weiters wurde darauf eingegangen, wie Content Provider durch das System selbst eingesetzt werden und wie sich dies in neueren Android-Versionen verändert hat. Abschließend wurde gezeigt, welche Schwachstellen es in Bezug auf Content Provider gab und wie diese gefunden werden können.

Referenzen

[1] Android Developers, „android.provider," Google Inc., [Online]. Available: https://developer.android.com/reference/android/provider/package-summary. [Zugriff am 26 06 2023].

[2] Android Developers, „Personal Information - Application Security," Google Inc., [Online]. Available: https://source.android.com/docs/security/overview/app-security#personal-information. [Zugriff am 13 06 2023].

[3] Android Developers, „Android 6.0 Changes," Google Inc., [Online]. Available: https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-runtime-permissions. [Zugriff am 15 06 2023].

[4] I. Khokhlov und L. Reznik, „Colluded Applications Vulnerabilities in Android Devices," *IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC),* 2017.

[5] S. Bhandari, W. B. Jaballah, V. Jain, V. Laxmi, A. Zemmari, M. S. Gaur, M. Mosbah und M. Conti, „Android inter-app communication threats and detection techniques," *Computers & Security,* Nr. 70, pp. 392-421, 2017.

[6] A. Developers, „Get access with temporary permissions," Google Inc, [Online]. Available: https://developer.android.com/guide/topics/providers/content-provider-basics#get-access-with-temporary-permissions. [Zugriff am 15 06 2023].

[7] Y. Zhou und X. Jiang, „Detecting Passive Content Leaks and Pollution in Android Applications," *20th Annual Network & Distributed System Security Symposium,* 2013.

[8] M. Grace, W. Zhou, X. Jiang und A.-R. Sadeghi, „Unsafe Exposure Analysis of Mobile In-App Advertisements," *ACM Conference on Security and Privacy in Wireless and Mobile Networks,* pp. 101-112, 2012.

[9] D. Octeau, D. Luchaup, M. Dering, S. Jha und P. McDaniel, „Composite Constant Propagation: Application to Android Inter-Component Communication Analysis," *IEEE/ACM International Conference on Software Engineering,* pp. 77-88, 2015.

[10] A. Alzaidi, S. Alshehri und S. M. Buhari, „DroidRista: a highly precise static data flow analysis framework for android applications," *International Journal of Information Security,* p. 523–536, 2019.

[11] V. Jain, V. Laxmi, M. S. Gaur und M. Mosbah, „APPLADroid: Automaton Based Inter-app Privacy Leak Analysis for Android," *International Conference on Security & Privacy,* p. 219–233, 2019.

[12] V. Jain, S. Bhandari, V. Laxmi, M. S. Gaur und M. Mosbah, „SniffDroid: Detection of Inter-App Privacy Leaks in Android," *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom),* 2017.

[13] H. Bagheri, A. Sadeghi, J. Garcia und S. Malek, „COVERT: Compositional Analysis of Android Inter-App Permission Leakage," *IEEE Transactions on Software Engineering,* Bd. 41, Nr. 1, pp. 866-886, 2015.

[14] G. Palfinger und B. Prünster, „AndroPRINT: Analysing the Fingerprintability of the Android API," *Availability, Reliability and Security - ARES,* pp. 94:1 - 94:10, 2020.

[15] M. C. Grace, Y. Zhou, Z. Wang und X. Jiang, „Systematic Detection of Capability Leaks in Stock Android Smartphones," *Network and Distributed System Security Symposium,* 2012.

[16] N. T. Cam, V.-H. Pham und T. A. Nguyen, „Sensitive Data Leakage Detection in Pre-Installed Applications of Custom Android Firmware," *IEEE International Conference on Mobile Data Management,* pp. 340-343, 2017.

[17] D. Buchanan, „Exploiting aCropalypse: Recovering Truncated PNGs," 18 03 2023. [Online]. Available: https://www.da.vidbuchanan.co.uk/blog/exploiting-acropalypse.html. [Zugriff am 16 06 2023].

[18] K. Kollnig, A. Shuba, R. Binns, M. V. Kleek und N. Shadbolt, „Are iPhones Really Better for Privacy? A Comparative Study of iOS and Android Apps," *Proc. Priv. Enhancing Technol.,* pp. 6--24, 2022.

[19] I. Shklovski, S. D. Mainwaring, H. H. Skúladóttir und H. Borgthorsson, „Leakiness and creepiness in app space: perceptions of privacy and mobile app use," *Conference on Human Factors in Computing Systems,* pp. 2347-2356, 2014.

[20] C. F. Torres und H. Jonker, „Investigating Fingerprinters and Fingerprinting-Alike Behaviour of Android Applications," *European Symposium on Research in Computer Security - ESORICS,* pp. 60-80, 2018.

[21] K. Yang, J. Zhuge, Y. Wang, L. Zhou und H. Duan, „IntentFuzzer: detecting capability leaks of android applications," *ASIA CCS '14: Proceedings of the 9th ACM symposium on Information, computer and communications security,* p. 531–536, 2014.

[22] M. Grace, Y. Zhou, Z. Wang und X. Jiang, „Systematic Detection of Capability Leaks in Stock Android Smartphones," *19th Annual Network & Distributed System Security Symposium,* 2012.

[23] N. T. Cam, V.-H. Pham und T. A. Nguyen, „Sensitive Data Leakage Detection in Pre-Installed Applications of Custom Android Firmware," *18th IEEE International Conference on Mobile Data Management (MDM),* 2017.