A-SIT

Zentrum für sichere Informationstechnologie - Austria

ANALYSE DER ANDROID PRIVACY SANDBOX

Analyse der Android Privacy Sandbox

Autor:
Gerald Palfinger
Tel:
Mail: gerald.palfinger@a-sit.at

Datum: 14.06.2024

Zusammenfassung:
Mit der Android Privacy Sandbox stellt Google eine neue Technologie vor, die die Privatsphäre von Nutzerinnen und Nutzern von Android-Geräten verbessern soll. Mit der Privacy Sandbox wird auch eine neue Technologie namens SDK Runtime vorgestellt. Das Ziel dieser Runtime ist es, von Applikationen eingebundene Werbebibliotheken von der Applikation selbst zu isolieren. Dadurch soll es den Werbebibliotheken nicht mehr möglich sein, auf identifizierbare Daten zuzugreifen, welche der einbindenden Applikation beispielsweise durch die Gewährung einer Berechtigung durch den Nutzer beziehungsweise die Nutzerin gewährt wurde. In diesem Bericht wird analysiert, wie die SDK Runtime technisch umgesetzt wurde. Weiters werden Experimente durchgeführt, um festzustellen, welche Änderungen an der Programmierschnittstelle dazu durchgeführt wurden. Ebenso wird untersucht, welche Informationen noch abrufbar sind, um einen Fingerabdruck eines Geräts zu erstellen.

Inhalt

1. Einleitung

Die Android Privacy Sandbox ist eine neue Technologie von Google, um den Datenschutz auf Android-Geräten zu verbessern und gleichzeitig den Anforderungen der Werbewirtschaft gerecht zu werden. Diese Initiative, die im Rahmen des Privacy Sandbox-Projekts von Google entwickelt wurde, zielt darauf ab, den Umfang der gesammelten Nutzerdaten zu minimieren und die Privatsphäre der Nutzer zu schützen, während sie personalisierte Werbung ermöglichen soll. Die Android Privacy Sandbox führt technische Ansätze ein, die darauf abzielen, das Tracking über verschiedene Applikationen hinweg einzuschränken. Insbesondere verändert die neue SDK Runtime, welche Teil der Android Privacy Sandbox ist, die Art und Weise, wie Software Development Kits (SDKs) innerhalb des Android-Ökosystems eingebunden werden. Diese Runtime hat dabei das Ziel, den Datenschutz zu verbessern, indem sie Drittanbieter-SDKs von der Haupt-Applikation isoliert. Das Ziel dabei ist, dass das SDK nur auf die Daten zugreifen kann die es zur Ausführung benötigt, ohne vollen Zugriff auf alle Daten und Berechtigungen der Applikation zu haben. In diesem Bericht wird gezeigt, wie die SDK Runtime technisch umgesetzt

wurde und wie sich diese auf Applikationen beziehungsweise die eingebundenen SDKs auswirkt. Ebenso wird untersucht, welche Informationen, die ansonsten über die Programmierschnittstelle abrufbar sind, einem solchen in der SDK Runtime laufenden Bibliothek nicht mehr zugänglich sind.

2. Hintergrund

2.1. Technische Umsetzung

Technisch gesehen werden die SDKs in einem separaten Prozess ausgeführt und so von der Haupt-Applikation isoliert. Diese Isolation bedeutet, dass die SDKs keinen direkten Zugriff auf die Daten oder Berechtigungen der Haupt-Applikation haben, sondern nur über eine vorab definierte Schnittstelle interagieren können. Eine schematische Darstellung der Aufteilung ist in Abbildung 1 ersichtlich. Durch die Aufteilung in separate Prozesse sorgt das Betriebssystem für die Isolation über bereits bekannte und getestete Mechanismen. Zudem verwaltet die SDK Runtime die Berechtigungen der SDKs, sodass diese nur auf spezifische Daten und Funktionen zugreifen können, die für ihre Aufgaben notwendig sind, wodurch unerwünschter Datenzugriff reduziert werden soll. Insbesondere erlaubt die SDK Runtime nur den Zugriff auf die Berechtigungen INTERNET, ACCESS_NETWORK_STATE, READ_BASIC_PHONE_STATE und AD_ID. Dadurch kann ein SDK, das in der SDK Runtime läuft, auf das Internet zugreifen, grundlegende Informationen über das Netzwerk abrufen, den Telefonstatus, wie zum Beispiel den Netzwerktyp auslesen und auf die Werbe-Identifikationsnummer zugreifen. Ein Zugriff auf weitere Berechtigungen ist in der derzeitigen Umsetzung der SDK Runtime nicht vorgesehen. Darüber hinaus erlaubt die Entkopplung von der Haupt-Applikation auch, dass die SDKs unabhängig von der Haupt-Applikation aktualisiert werden können.

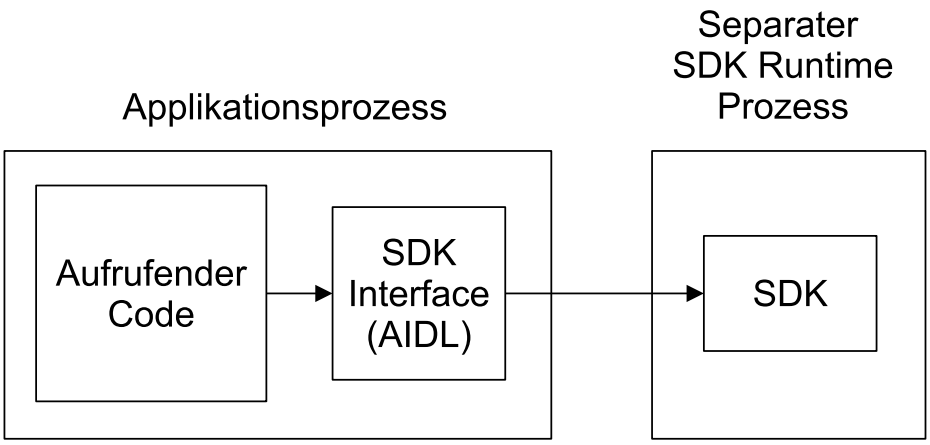

Abbildung 1 Grafische Übersicht über die Prozessisolierung der SDK Runtime (angelehnt an [1]).

2.2. Verwendung der Android Interface Definition Language

Die Android Interface Definition Language (AIDL) spielt eine zentrale Rolle in der neuen SDK Runtime der Android Privacy Sandbox. AIDL wird verwendet, um die Kommunikation zwischen der Haupt-Applikation und den isolierten SDKs zu definieren. Diese Nutzung von AIDL ist ein wichtiger Bestandteil, um sicherzustellen, dass die SDKs sicher und kontrolliert innerhalb ihrer isolierten Umgebung agieren können. In AIDL werden die Schnittstellen definiert, über die die Haupt-Applikation und das isolierte SDK miteinander kommunizieren können. Diese Schnittstelle spezifiziert die Methoden sowie die Datentypen, die ausgetauscht werden. Dies stellt sicher, dass die Interaktionen klar definiert sind. Da die Schnittstellen klar definiert sind, kann genau kontrolliert werden, welche Daten und Funktionen den SDKs zur Verfügung stehen. Dies minimiert das Risiko, dass die ausgeführten SDKs mehr Daten sammeln oder auf mehr Funktionen zugreifen als unbedingt notwendig. Ein Beispiel einer solchen AIDL-Schnittstelle ist in Abbildung 2 ersichtlich. In dem abgebildeten Beispiel werden zwei Methoden definiert, die von der Haupt-Applikation aufgerufen werden können um mit dem SDK zu interagieren.

```
package com.example.exampleaidllibrary;

interface ISdkApi {
    String createFile(int sizeInMb);

    void runMethod();
}
```

Abbildung 2 Beispiel einer AIDL-Schnittstelle zur Kommunikation zwischen Applikation und SDK.

Basierend auf der definierten AIDL-Schnittstelle wird automatisiert eine IBinder-Schnittstelle erstellt. Diese wird vom isolierten SDK implementiert um darüber die Funktionen des SDKs bereitzustellen. Die AIDL-Schnittstelle basiert auf dem Java-Interface Syntax und definiert nur die Methoden-Signaturen, welche vom SDK bereitgestellt werden. Da es sich um eine IPC-Schnittstelle handelt, werden nur eine definierte Anzahl an Datentypen unterstützt. Diese beinhalten alle primitiven Datentypen, die es in Java gibt, sowie Strings, CharSequences, Listen, Maps und Arrays, welche primitive Datentypen beinhalten. Zusätzlich werden auch Klassen unterstützt, welche die Parcelable-Schnittstelle implementieren.

2.3. Aktivierung der SDK Runtime

Bei der Privacy Sandbox handelt es sich derzeit um eine Developer Preview. Die dazugehörende SDK Runtime muss am Gerät, auf dem der Entwicklermodus aktiviert wurde, zuerst eingerichtet werden. Dazu werden über die Android Debug Bridge (adb) zuerst die dazugehörigen Dienste aktiviert. Die Aktivierung der Dienste erfolgt über die in Abbildung 3 dargestellten Befehle. Schlussendlich wird die SDK Runtime mit dem letzten Befehl persistent gesetzt, damit diese nicht nach wenigen Minuten neu gestartet wird.

```
adb shell device_config put adservices ppapi_app_allow_list
\"\*\" && \
adb shell device_config put adservices
ppapi_app_signature_allow_list \"\*\" && \
adb shell device_config put adservices
msmt_api_app_allow_list \"\*\" && \
adb shell device_config put adservices
adservice_system_service_enabled true && \
adb shell device_config put adservices adservice_enabled true
&& \
adb shell device_config put adservices
adservice_enable_status true && \
adb shell device_config put adservices global_kill_switch
false && \
adb shell device_config put adservices disable_sdk_sandbox
false && \
adb shell device_config set_sync_disabled_for_tests
persistent
```

Abbildung 3 Aktivierung der SDK Runtime auf dem Gerät.

3. Untersuchung

3.1. Methodik

Für die Datenerfassung von Geräten wird ein automatisiertes Framework verwendet, das Rückgabewerte von Methodenaufrufen, Inhalte von Feldern und Content Providern abfragt. Ziel ist es, alle wesentlichen Informationsquellen, die einem Drittprogramm zugänglich sind, zu erfassen. Das Framework basiert auf einer Liste aller bekannten Methoden und Konstruktoren der Android API, welche aus der API-Dokumentation [2] geparst wurde.

3.1.1. Erstellung und Analyse von Objekten

Ausgehend von der geparsten Liste an Klassen werden Objekte der Klassen erstellt und deren Methoden sowie Felder mittels Java Reflection aufgerufen und ausgewertet. Dazu wurden auch mögliche Konstanten aus der Dokumentation geparst, welche bei Methoden- und Konstruktorenaufrufen verwendet werden. Für Parameter, für die keine Konstanten vorliegen, werden entweder manuell definierte Werte oder Standardwerte verwendet. Die manuell vordefinierten Werte werden anhand des Parameternamens zugewiesen, während die Standardwerte anhand des Parametertyps bestimmt werden.

3.1.2. Minimierung von Seiteneffekten

Um Seiteneffekte wie Änderungen an der Applikation oder dem System sowie Applikationsabstürze zu minimieren, werden nur Methoden aufgerufen, die typischerweise Informationen bereitstellen. Diese werden anhand des Präfixes des Methodennamens ausgewählt. Für diese Studie werden die Methoden auf die Präfixe get, has, is, query, check, support, available, total, max und scan eingeschränkt. Gibt eine Methode einen primitiven Wert (z.B. Integer oder String) zurück, wird dieser für die weitere Analyse gespeichert. Gibt die Methode jedoch ein Objekt einer Klasse zurück, so werden darin enthaltene Methoden, sofern sie die genannten Präfixe im Methodennamen haben, ebenfalls aufgerufen. Das Resultat der toString() Methode wird ebenfalls für die Analyse herangezogen.

3.1.3. Auswertung von Feldern und Content Providern

Zusätzlich zu den Rückgabewerten der Methoden werden auch alle Inhalte von Feldern der erstellten bzw. erhaltenen Objekte abgerufen und für die Analyse gespeichert. Ebenso werden Inhalte von Content Providern ausgelesen. Die Content Provider Architektur [3] ermöglicht den Zugriff auf Daten in Systemdiensten. Auch Applikationen können Content Provider implementieren, um Daten über eine einheitliche Schnittstelle anderen Applikationen zur Verfügung zu stellen. Für diese Studie konzentrieren wir uns jedoch auf die Content Provider, die vom System bzw. von vorinstallierten Diensten bereitgestellt werden. Alle Content Provider URIs (also Strings, die mit content:// starten) wurden zuvor gesammelt. In diesem Schritt werden nur jene Content Provider ausgelesen, die ohne erhöhte Berechtigungen zugänglich sind. Die zugreifbaren Daten werden für die weitere Analyse gespeichert.

3.1.4. Datenbereinigung und -auswertung

Im letzten Schritt werden alle gesammelten Daten bereinigt, wobei Duplikate entfernt werden. Das bedeutet, dass Aufrufe von Methoden, die trotz unterschiedlicher Parameter oder Aufrufobjekte denselben Wert zurückgeben, ausgeschlossen werden. Danach wird analysiert, welche Methoden, Felder sowie Content Provider in der SDK Runtime nicht aufgerufen beziehungsweise ausgelesen werden können.

3.2. Ergebnisse

In diesem Abschnitt wird auf die Unterschiede zwischen der Ausführung des Frameworks in der Privacy Sandbox und direkt auf dem Gerät ohne Privacy Sandbox eingegangen. Dazu wurde nach fingerprintbaren Informationsquellen auf einem Pixel 7a mit Android 14 einmal in der SDK Runtime und einmal ohne SDK Runtime gesucht. Als Vergleichsgerät wurde ein Pixel 6 Pro gewählt, auf dem ebenfalls nach fingerprintbaren Informationsquellen gesucht wurde. Bei der Untersuchung werden nur Methoden, Felder und Content Provider auf- beziehungsweise abgerufen, welche keine gesonderte Berechtigung oder nur eine sogenannte „normale" Berechtigung benötigen, die automatisch bei der Installation der Applikation vom System gewährt wird. In der SDK Runtime können nur jene normalen Berechtigungen angefordert werden, welche dafür freigeschaltet wurden (siehe Abschnitt - 2 -2.1).

3.2.1. Content Provider

Insgesamt waren in der SDK Runtime drei Content Provider zugänglich, die fingerprintbare Informationen bereitstellen. Ohne die SDK Runtime wurden zwei weitere Content Provider gefunden, die fingerprintbare Informationen beinhalten, also insgesamt fünf. Bei den drei in der SDK Runtime zugreifbaren Content Providern handelt es sich um Content Provider, die den Zugriff auf durch den Nutzer beziehungsweise die Nutzerin beeinflussbare Einstellungen sowie Informationen über das System ermöglichen. Diese enthalten die mit Abstand meisten fingerprintbaren Einträge von den fünf fingerprintbaren Content Providern. Neben einigen Konstanten enthalten diese Anbieter hauptsächlich Systemeinstellungen, die vom Benutzer beziehungsweise der Benutzerin des Geräts angepasst werden können. Der Einstellungs-Provider ist dabei in drei Unter-Provider unterteilt. Im Einzelnen sind dies content://settings/global, content://settings/secure und content://settings/system. Die meisten Einträge enthält der Content Provider content://settings/global mit 225 Einträgen. Dieser enthält zum Beispiel den vom Benutzer bzw. der Benutzerin einstellbaren Gerätenamen, die Bootanzahl und die Mobilfunkeinstellungen. Der Content Provider content://settings/secure erlaubt den Zugriff auf 151 verschiedene Einträge. Dazu gehören unter anderem die Einstellungen für die Barrierefreiheit, die Eingabemethode und der Autofill-Dienst sowie die Einstellungen für den Sperrbildschirm. Der Content Provider content://settings/system enthält schließlich 46 Einträge. Dieser Anbieter enthält in erster Linie Informationen über Medien- und Benachrichtigungseinstellungen, beispielsweise über eingestellte Benachrichtigungstöne oder Vibrationseinstellungen.

Die SDK Runtime verhindert den Zugriff auf die beiden Content Provider content://media/internal/audio/media und content://service-state. Der Content Provider content://media/internal/audio/media enthält dabei Informationen über Klingel-, Benachrichtigungs- und Alarmtöne. Dieser enthält auf dem Testgerät 42 verschiedene Einträge. Neben verschiedenen Daten über die Töne enthält diese Tabelle auch eine Spalte date_modified. Bei nicht gepatchten Geräten entspricht diese Spalte dem Zeitpunkt des ersten Starts des Geräts mit den vorinstallierten Sounds. Der Service-State-Provider enthält schließlich Informationen über das Mobilfunknetz, wie zum Beispiel die Art des Datennetzes oder die Identifikationsnummer des Netzbetreibers. Da in dem verwendeten Testgerät eine einzige SIM-Karte installiert war, enthält dieser Provider einen Eintrag.

3.2.2. Methoden und Felder

Im Vergleich zwischen Pixel 6 Pro und Pixel 7a ohne SDK Runtime wurden 852 Methoden und Felder gefunden, welche sich zwischen den Geräten unterscheiden und so potentiell fingerprintbare Informationen liefern. Im Vergleich dazu wurden zwischen Pixel 6 Pro und Pixel 7a mit SDK Runtime 823 Methoden und Felder gefunden, die Informationen zum Erstellen eines Fingerabdrucks liefern können. In den folgenden beiden Abschnitten wird darauf eingegangen, wie sich die Ergebnisse zwischen der Ausführung innerhalb und außerhalb der SDK Runtime unterscheiden. Zum einen wurde festgestellt, dass manche Methoden und Felder in der SDK Runtime nicht auf- beziehungsweise abrufbar sind. Dies kann zum Beispiel daran liegen, dass in der SDK Runtime eine „normale" Berechtigung nicht gewährt wurde. Weiters konnte festgestellt werden, dass manche Methoden beziehungsweise Felder unterschiedliche Werte liefern. Dies liegt beispielsweise daran, dass bestimmte Dateisystempfade aus der SDK Runtime nicht zugreifbar sind.

3.2.2.1. Nicht zugreifbare Methoden und Felder

Insgesamt konnten in der SDK Runtime 35 Methoden und Felder, die fingerprintbare Informationen liefern, nicht auf- beziehungsweise abgerufen werden. Darüber hinaus konnten weitere 235 Methoden und Felder in der SDK Runtime nicht auf- bzw. abgerufen werden. Diese lieferten jedoch im Vergleich mit dem Pixel 6 Pro keine fingerprintbaren Informationen. Dabei konnten Methoden der Klasse KeyguardManager nicht aufgerufen werden. Diese erlauben den Zugriff auf Informationen über den Sperrmechanismus des Geräts. Ebenso kann auf den Systemdienst WallpaperManager nicht zugegriffen werden. Dieser ermöglicht ohne SDK Runtime den Zugriff auf Informationen über den gesetzten Bildschirmhintergrund, wie zum Beispiel die darin vorkommenden Farben oder die erforderliche Größe eines Bildschirmhintergrunds. Darüber hinaus können Objekte der Klasse DevicePolicyManager nicht instanziiert werden. Dieser erlaubt auf verschieden Einstellungen und Eigenschaften des Geräts zuzugreifen, wie beispielsweise den Verschlüsselungsstatus des Speichers oder die Unterstützung der Geräteattestierung. In der SDK Runtime war es darüber hinaus nicht möglich, auf andere am Gerät installierte Nutzerprofile sowie die im PackageManager hinterlegten gemeinsamen Bibliotheken zuzugreifen. Ebenso konnte nicht auf den gesetzten Klingelton sowie Informationen des MediaControllers über die gesetzte Lautstärke sowie Medieninformationen zugegriffen werden. Auch der Zugriff auf den WifiManager ist aus der SDK Runtime heraus nicht möglich. Dieser erlaubt den Zugriff auf Informationen über unterstützte beziehungsweise aktivierte Funktionen der WLAN-Hardware. In ähnlicher Weise ist auch der Zugriff auf Informationen über die NFC-Hardware über die NfcAntennaInfo-Klasse nicht möglich.

3.2.2.2. Unterschiede zwischen SDK Runtime und Gerät

Neben den nicht auf- beziehungsweise abrufbaren Methoden und Feldern gab es auch Methoden und Felder, deren Werte sich zwischen der Ausführung mit und ohne SDK Runtime unterschieden. Dies betrifft den Package-Namen, der über verschiedene Methoden abgerufen werden kann. Dieser lautet derzeit immer com.google.android.sdksandbox, unabhängig vom gewählten Package-Namen der in der Runtime laufenden Bibliothek. Dadurch unterscheiden sich auch manche der erhaltenen

Dateisystempfade, wie zum Beispiel der des Cache-Ordners, welcher über Context.getPreloadsFileCache() abgerufen werden kann. Darüber hinaus verbietet die SDK Runtime den Zugriff auf manche Pfade des Dateisystems, die für eine Applikation außerhalb der Runtime zugänglich sind. Das inkludiert die Datei- und Medienordner der Applikation auf dem integrierten Speicher. Dadurch geben beispielsweise die Methoden Context.getExternalFilesDirs(...) und Context.getExternalMediaDir() null zurück, anstatt eines File-Objektes mit einem gültigen Dateisytempfad wie /storage/emulated/0/Android/media/<package-name>/. Ebenso gibt die Methode android.os.Environment.getExternalStorageState() in der SDK Runtime den Status ausgehängt (unmounted) zurück, während dieselbe Methode ohne SDK Runtime eingehängt (mounted) zurückgibt.

In der SDK Runtime ist kein Zugriff auf das DRM-Framework möglich. Die Methode android.drm.DrmManagerClient.getAvailableDrmEngines() gibt ein leeres Array zurück. Über das DRM-Framework ist es ansonsten prinzipiell möglich, zu DRM-Zwecken eine (APK-spezifische) Widevine ID [4] zu erstellen. Ein Zugriff auf den gesetzten Klingelton über die Methode android.media.Ringtone.getTitle(Context) ist aus der SDK Runtime ebenso nicht möglich. Darüber hinaus unterscheidet sich die Eingabemethode, welche über den InputManager abgerufen werden kann. Schlussendlich ist auch ein Zugriff auf die verfügbaren Benachrichtigungskanäle über die Methode android.app.NotificationManager.getNotificationChannels() nicht möglich. Diese gibt in der SDK Runtime anstatt der Liste an verfügbaren Nachrichtenkanälen ein leeres Array zurück.

4. Fazit

In diesem Bericht wurde die Android Privacy Sandbox und insbesondere die dazugehörende SDK Runtime analysiert. Neben einer Analyse der technischen Umsetzung der Privacy Sandbox wurde ein Experiment durchgeführt, um die Unterschiede zwischen einer Ausführung einer Applikation in der SDK Runtime und auf dem Gerät zu untersuchen. Dabei wurde festgestellt, dass die SDK Runtime den Zugriff auf manche Methoden, Felder und Content Provider verhindert. Ebenso unterscheiden sich die erhaltenen Informationen von manchen Methoden und Feldern zwischen der Ausführung in der SDK Runtime und direkt am Gerät. Es sind jedoch noch immer viele Methoden, Felder und Content Provider zugreifbar, welche Informationen bereitstellen, die zur Erstellung eines Fingerabdrucks genutzt werden können. Ein großer Vorteil der SDK Runtime ist jedoch, dass die darin ausgeführten Bibliotheken keinen Zugriff auf Informationen in der Applikation selbst und auf Berechtigungen bekommen, welche der Nutzer bzw. die Nutzerin der Applikation erteilt hat.

Referenzen

[1] Google Inc., „SDK Runtime Overview - Process Isolation," 10 06 2024. [Online]. Available: https://developers.google.com/privacy-sandbox/relevance/sdk-runtime#process-isolation. [Zugriff am 12 06 2024].

[2] Google Inc., „Android API reference | Android Developers," 02 03 2024. [Online]. Available: https://developer.android.com/reference. [Zugriff am 14 06 2024].

[3] Google Inc., „Content Providers | Android Developers," 23 01 2024. [Online]. Available: https://developer.android.com/guide/topics/providers/content-providers. [Zugriff am 14 06 2024].

[4] „Best practices for unique identifiers | Android Developers," 23 04 2024. [Online]. Available: https://developer.android.com/identity/user-data-ids. [Zugriff am 14 06 2024].