IOActive Security Advisory

Title	Android (AOSP) Download Provider SQL Injection in Query Sort Parameter (CVE-2019-2196)
Severity	High
Discovered by	Daniel Kachakil, Senior Security Consultant
Advisory Date	January 17, 2020

Affected Products

Android Open Source Project (AOSP)
Android versions: 5.1, 6.0.1, 7.0, 7.1.1, 7.1.2, 8.0, 8.1, 9, 10

Impact

A malicious application with the `INTERNET` permission granted could retrieve all entries from the Download Provider internal database, bypassing all currently implemented access control mechanisms, by exploiting an SQL injection in the `sort` parameter (`ORDER BY` clause) and appending a `LIMIT` clause, which allows expressions, including subqueries.

The information retrieved from this provider may include potentially sensitive information such as file names, descriptions, titles, paths, URLs (which may contain sensitive parameters in the query strings), cookies, custom HTTP headers, etc., for applications such as Gmail, Google Chrome, the Google Play Store, etc.

Background

The Download Provider is used to handle OTA updates and the basic download needs of relevant applications such as Gmail, Google Chrome, and the Google Play Store, amongst many others.

By design, all of this information should be restricted to the application that requested the download or applications with explicit permission to access all downloads. This is why custom permissions and different URI paths exist for this provider.

Technical Details

Access to the Download Content Provider requires different permissions, such as `INTERNET` or `ACCESS_ALL_DOWNLOADS`, depending on the requested URI, as shown in the `AndroidManifest.xml`[1] file:

[1] https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/master/AndroidManifest.xml


```
<provider android:name=".DownloadProvider"
          android:authorities="downloads" android:exported="true">
  <!-- Anyone can access /my_downloads, the provider internally restricts
       access by UID for these URIs -->
  <path-permission android:pathPrefix="/my_downloads"
                   android:permission="android.permission.INTERNET"/>
  <!-- to access /all_downloads, ACCESS_ALL_DOWNLOADS permission is
       required -->
  <path-permission android:pathPrefix="/all_downloads"
      android:permission="android.permission.ACCESS_ALL_DOWNLOADS"/>
  <!-- Temporary, for backwards compatibility -->
  <path-permission android:pathPrefix="/download"
      android:permission="android.permission.INTERNET"/>

  <!-- Apps with access to /all_downloads/... can grant permissions,
       allowing them to share downloaded files with other viewers -->
  <grant-uri-permission android:pathPrefix="/all_downloads/"/>
  <!-- Apps with access to /my_downloads/... can grant permissions,
       allowing them to share downloaded files with other viewers -->
  <grant-uri-permission android:pathPrefix="/my_downloads/"/>
</provider>
```

Note that this attack vector is different from those affecting the selection clauses. To the best of our knowledge, it affects all versions of Android. This vulnerability allows a malicious application to efficiently retrieve all existing data from the internal `downloads.db` database.

Proof of Concept

In order to exploit the issue, a malicious application granted the `INTERNET` permission can request a legitimate download in order to differentiate between `true` and `false` conditions, based on the number of returned rows from the legitimate query to the `content://downloads/my_downloads` URI, with a sort order like the following:

```
column LIMIT CASE WHEN (condition) THEN 1 ELSE 0 END
```

For instance:

```
_id LIMIT CASE WHEN ((SELECT COUNT(*) FROM downloads WHERE _id=123 AND
title LIKE 'a%') > 0) THEN 1 ELSE 0 END
```

When the condition is evaluated to `true`, the executed query will return information, while it will not return any rows when it is evaluated to `false`. This is enough to perform a classic blind SQL injection attack and retrieve all of the information from the database.

Note that, in addition to the default `downloads` table, the injection also allows access to the `request_headers` table and all private columns (such as `UID`, `ETag`, or

`CookieData`), which should be restricted by the projection explicit limitations, as mentioned in the internal documentation[2]:

> ***Reducing the list of visible columns***
>
> *Security in the download provider is primarily enforced with two separate mechanisms:*
>
> - *Column restrictions, such that only a small number of the download provider's columns can be read or queried by applications.*
> - *UID restrictions, such that only the application that initiated a download can access information about that download.*
>
> *The first mechanism is expected to be fairly robust (the implementation is quite simple, based on projection maps, which are highly structured), but the second one relies on arbitrary strings (URIs and SQL fragments) passed by applications and is therefore at a higher risk of being compromised. Therefore, sensitive information stored in unrestricted columns (for which the first mechanism doesn't apply) is at a greater risk than other information.*

A PoC app accompanies this advisory[3] which implements a faster and more efficient extraction, retrieving one byte per query. This is achieved by inserting 256 arbitrary downloads (in the first execution only), and then just reading the number of returned rows, rather than retrieving a single bit per query.

This PoC app will retrieve several columns from all existing downloads in the table, such as `URI`, `title`, or `description`, including some of the restricted ones, such as `ETag` or `CookieData`, and all custom headers (if any) as well.

If the output is empty, make sure that the provider contains some data, by downloading any file (i.e. a PDF) from Google Chrome or any attachment from Gmail, for instance.

Suggested Fixes

Make sure that the `sort` parameter of the `query` method is properly validated before executing the underlying request to the database.

Since the `ORDER BY` clause in SQLite does not allow expressions, the risk can be fully mitigated by ensuring that the `sort` parameter does not contain any malicious payload injecting a `LIMIT` clause with a potentially malicious subquery.

Stricter validations may also be performed, such as requiring the parameter to only contain a comma-separated list of existing columns and the string literals “`asc`” or “`desc`”.

---

[2] https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/master/docs/index.html
[3] https://github.com/IOActive/AOSP-DownloadProviderDbDumperSQLiLimit

For instance, a simple fix that should mitigate the issue would be adding the following condition to the `DownloadProvider.java` file[4]:

```
@Override
public query(final Uri uri, String[] projection,
         final String selection, final String[] selectionArgs,
         final String sort) {
    ...
    if (shouldRestrictVisibility()) {
        if (sort != null &&
                sort.toLowerCase(Locale.ENGLISH).contains("limit"))
            throw new IllegalArgumentException("invalid sort");
    ...
```

Mitigation

The vulnerability has been fixed in the official repository. Specifically, in the following commits, also affecting the Android's Base Framework and Media Provider:

https://android.googlesource.com/platform/frameworks/base/+/07d6f1fe094b6dbde854fb82ada06e85d7a97ecd

https://android.googlesource.com/platform/frameworks/base/+/36a5c576f0d379b0be3716fe5b8b9ae8bb3952f5

https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/ef25600f187f5372ad89645e6e6e7b4204bf0676

https://android.googlesource.com/platform/packages/providers/MediaProvider/+/4a827429765fa167571ec21611ee057d43c8e336

Google had released security patches for this vulnerability in November 2019. IOActive recommends applying the latest security patches from your vendor. If for any reason it is not possible to apply such updates, make sure that your Android device only contains trusted applications before attempting to download any files, particularly if they contain confidential information.

Timeline

- 2019-06-05 IOActive discovers vulnerability
- 2019-06-13 IOActive reports vulnerability to Google
- 2019-11-05 Google publishes the fix for the vulnerability
- 2020-01-17 IOActive advisory published

4 https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/refs/heads/master/src/com/android/providers/downloads/DownloadProvider.java

