IOActive Security Advisory

Title	Android (AOSP) Download Provider SQL Injection (CVE-2018-9493)
Severity	High
Discovered by	Daniel Kachakil
Advisory Date	April 01, 2019

Affected Products

1. Android Open Source Project (AOSP)
   Android versions: 8.0, 8.1, 9 (other versions may be also affected)

Impact

By exploiting an SQL injection vulnerability, a malicious application without any permission granted could retrieve all entries from the Download Provider, bypassing all currently implemented access control mechanisms. Also, applications that were granted limited permissions, such as `INTERNET`, can also access all database contents from a different URI.

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

Further access to the downloaded contents may be possible as well, depending on the permissions granted to the app and files.

Background

According to internal documentation, the Android Download Provider is used to handle OTA updates and the basic download needs of relevant applications such as Gmail, Android's Browser (now Google Chrome), or Market (i.e. Google Play Store).

By design, all this information should be restricted to the application that requested the download or to applications with the 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` file:


```
<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>
```

However, the following accessible URI does not require any permission:

- `content://downloads/public_downloads/#`

We can find a reference to this URI in the source code (`DownloadProvider.java`)

```
sURIMatcher.addURI("downloads",
    Downloads.Impl.PUBLICLY_ACCESSIBLE_DOWNLOADS_URI_SEGMENT + "/#",
    PUBLIC_DOWNLOAD_ID);
```

As its name implies, it is supposed to be used for downloads that are public, but nothing prevents an attacker from injecting an SQL selection clause to access any row, column, or table from the database, including the columns that are protected. Also, a URI that allows access by ID usually does not need any additional selection clause, but it is allowed in this case.

The selection clause was restricted in previous versions of Android. In the following commit[1], the protection was removed in an attempt to add searching capabilities to the Download Provider:

```
Title: Enable search for Downloads
Commit hash: b759707b80987d0cb4ad2a3a78c11702a45a36c2
Change-Id: Ide23c822b97ccab29a341184f14698dc942e8e14
Commit date: 10/05/2016 23:48:01
```

As we can see in the `diff` changes, the following highlighted line was removed in that commit, introducing the vulnerability:

```
@Override
public query(final Uri uri, String[] projection,
         final String selection, final String[] selectionArgs,
         final String sort) {

    Helpers.validateSelection(selection, sAppReadableColumnsSet);

    SQLiteDatabase db = mOpenHelper.getReadableDatabase();
    ...
```

Since the selection clause is no longer validated, an attacker would be only limited by the `setStrict(true)` further call, which is insufficient to prevent most injections.

It seems that the author of that commit tried to prevent the issue, by adding the following test to the `AbstractDownloadProviderFunctionalTest` class:

```
private boolean isDatabaseSecureAgainstBadSelection() {
    Cursor cursor = null;
    try {
        cursor = mResolver.query(Downloads.Impl.ALL_DOWNLOADS_CONTENT_URI,
              null, "('1'='1'))) ORDER BY lastmod DESC--", null, null);
    }
    catch (Exception e) {
        return true;
    } finally {
```

[1] https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/b759707b80987d0cb4ad2a3a78c11702a45a36c2


```
        if (cursor != null) {
            cursor.close();
        }
    }
    return false;
}
```

However, this test was proven not to be useful, as it will always throw an exception. The syntax of the underlying SQL statement will be invalid in all cases.

Note that the same vulnerability can also be exploited through different scenarios and URIs:

- No permission required:
  - `content://downloads/public_downloads/#`
- Requiring `android.permission.INTERNET`:
  - `content://downloads/my_downloads/`
  - `content://downloads/my_downloads/#`
  - `content://downloads/download/`
  - `content://downloads/download/#`

Proof of Concept

To reproduce the issue, run the following `adb` command:

```
adb shell content query --uri content://downloads/public_downloads/0 --
where "1=1) OR (1=1"
```

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.

Note that the injection also allows access to all private columns that 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.

[2] https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/master/docs/index.html

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 straightforward injection based on `UNION` statements could not be achieved due to the strict mode enforced in the underlying `SQLiteQueryBuilder`, but it is possible to extract all information by exploiting a blind SQL injection. For instance, querying the provider with this kind of SQL selection clauses will give access to internal and otherwise protected columns and tables:

```
adb shell content query --uri content://downloads/public_downloads/0 --
where "1=1) AND (_id=1 AND cookiedata LIKE 'a%') OR (1=1"
```

Similarly, it is also possible to dump all contents from the `request_headers` table:

```
adb shell content query --uri content://downloads/public_downloads/0 --
where "1=1) AND (SELECT header FROM request_headers WHERE _id=1) LIKE 'a%'
OR (1=1"
```

Note that it is also possible to obtain the same results from an application granted the `INTERNET` permission, by either invoking the URI above or the `/my_downloads` one.

There is a PoC app accompanying this advisory[3]. From its UI, we can dump several columns from the Download Provider database. Optionally, it is also possible to include restricted columns, such as `UID`, `ETag` or `CookieData`, exploiting a blind SQL injection (if this option is enabled, the process will be slightly slower).

[3] https://github.com/IOActive/AOSP-DownloadProviderDbDumper

Fixes

If it does not break any functionality, consider adding the line that was removed, or revert the above-referenced commit that introduced the vulnerability:

```
@Override
public query(final Uri uri, String[] projection,
         final String selection, final String[] selectionArgs,
         final String sort) {


    Helpers.validateSelection(selection, sAppReadableColumnsSet);
```

Mitigation

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

https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/e7364907439578ce5334bce20bb03fef2e88b107

https://android.googlesource.com/platform/frameworks/base/+/462aaeaa616e0bb1342e8ef7b472acc0cbc93deb

https://android.googlesource.com/platform/frameworks/base/+/ebc250d16c747f4161167b5ff58b3aea88b37acf

Several vendors integrating Android had released security patches for this vulnerability in October 2018. 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

2018-06-19	IOActive discovers vulnerability
2018-06-29	IOActive reports vulnerability to Google
2018-10-01	Google publishes the fix for the vulnerability
2019-03-30	Presented at RootedCon Security Conference (Spain)
2019-04-01	IOActive advisory published

