Vulnerabilities in Android Instant App

Yutian Tang, Xiapu Luo, and Hao Zhou

The Hong Kong Polytechnic University

1 Vulnerabilities

1.1 Overview

We found that it is possible for an attacker to hijack URLs and Instant Apps with an Instant App. To demonstrate the attacks, we build a PoC Instant App and successfully release our PoC Instant App via Google Play Console. The link of PoC Instant App is `https://play.google.com/store/apps/details?id=a.chrisyttang.instantappurlauto`. Alternatively, it can be accessed by searching name "CTInstant" from Google Play Store.

Demo We prepare two demo videos: • The demo video for Instant App hijacking attack can be found at
`https://drive.google.com/open?id=1AwXYWp8kwEu7E4EXpP3c_WeQ-yya6ylB`
• The demo video for URL hijacking attack can be found at
`https://drive.google.com/open?id=1nRKeeS6pMxv0-bruDGbTBT23DlAnWO9l`.

1.2 Environment

We test our PoC on a Google Pixel machine with the latest Android Version Android 9.

Ethics Note that our PoC Instant App is only used for demonstrating the potential attacks. We do not include any harmful code in the uploaded PoC Instant app. No data is collected from users.

1.3 Settings of the PoC Instant App

Our PoC Instant App has the following settings:

- The package name: `a.chrisyttang.instantappurlauto`;
- The hostwebsite: `https://www.chrisyttang.org`;
- Default URL to launch the Instant App :`https://www.chrisyttang.org/main`;
- The assetlinks.json is hosted on: `https://www.chrisyttang.org/.well-known/assetlinks.json`.

In addition, in the PoC Instant App, we also add an `intent-filter` to let the PoC Instant App be able to respond the URI `https://google.com/tripsapp` and `http://google.com/tripsapp` as shown in Fig. 1.

```
<activity android:name=".LoginActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:host="google.com" android:pathPrefix="/tripsapp"
android:scheme="http"/>
        <data android:host="google.com" android:pathPrefix="/tripsapp"
android:scheme="https"/>
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:host="plus.google.com" android:scheme="http"/>
        <data android:host="plus.google.com" android:scheme="https"/>
    </intent-filter>
</activity>
```

Fig. 1. The Intent-filter used in released PoC App.

1.4 Instant App Hijacking Attack

Once our PoC Instant App is downloaded to a device, the user cannot open the Google Trips Instant App with the url `https://google.com/tripsapp` link no matter whether the victim Instant App (Google Trips) has been installed or not.

1.5 URL Hijacking Attack

The URL hijacking attack tries to bind an URL to the PoC Instant App. In the demonstration, our PoC Instant App can hijack the url `https://plus.google.com`. That is, if a user clicks the link `https://plus.google.com`, she will be redirected to our Instant App rather than the browser.

1.6 Attack Surface

Evading the checking of Google Play Console Different from the typical format for declaring an app link, we remove the field "category.BROWABLE" from the Intent-filter of the PoC Instant App in order to evade the checking of Google Play Console. As a result, the PoC cannot hijack the URL from browser. That is, if the user clicks user clicks a link in a browser, the link will not be hijacked.

	browser-like apps	**non-browser apps**
attack on click	×	✓
smart text selection	✓	✓

Attack Surface However, there is still a large attack surface, including other popular apps and the widely used smart text selection scheme since Android 8.0, as shown in the above table.

For apps, this attack can be launched when the user clicks a link in the app. For example, we have successfully launched our PoC Instant App from Facebook's Message app, whatsapp, Gmail, and so forth. The only exception is that the attack could not be launched from browsers or apps with built-in browsers because the PoC Instant app does not include BROWABLE field in its Intent-filter.

Another target is smart text selection, which was introduced in Android 8.0. When a user selects the url, Android shows a popup menu for the user. Then, the user can open the url with the popup menu. For this scheme, no matter whether the app is a browser nor not, the attack can be launched successfully. Here, we record a demo on starting an attack from the browser with smart text selection. The demo can be found at
`https://drive.google.com/open?id=1V-Q1M5Dp7-Kc4yvKb8ythUteTAPOIWBv`.

1.7 Observations from Android Source Code

First, we use the source code of Android Pie 9 to illustrate how Android resolves an `Intent`. An `Intent` is resolved by `PackageManagerService`. The following introduction is based on
`http://androidxref.com/9.0.0_r3/xref/frameworks/base/services/core/java/com/android/server/pm/PackageManagerService.java`.

Android will first check whether there exist `Activities` that can resolve the `Intent` internally. Method `queryIntentActivitiesInternal` in class `PackageManagerService` is in charge of this searching. More specifically,
• In method `resolveIntentInternal` (class `PackageManagerService`) at line 5990.

❶ Line 6581: Android will first check whether a specific component is declared for handling the `Intent` with code
`ComponentName comp = intent.getComponent();;`

❷ Line 6591: As our data in the `Intent` is an URI, the `comp` must be `null`;

❸ Line 6658: If the user allows Instant App, the bool value `addInstant` should be `true`;

❹ Line 6673,6703: The if condition `intent.hasWebURI()` should be `true`; The value `sortResult` is set to be `true`;

❺ Line 6729: As `addInstant` is `true`(see ❸), Android will check whether there is a need to add an Instant App Installer to resolve the intent with method `maybeAddInstantAppInstaller` (Line 6741). Since in our attack there are already instant apps that can resolve the Intent, Android will not need to obtain the Instant App remotely. No matter whether there is an instant app that can resolve the Intent or not, Android will still build `ephemeralInstaller` and add back to `results`. Please refer to Line 6802, 6803, and 6837.

❻ Line 6733: As the value `sortResult` is set to `true`, all Activities that can resolve the Intent will be ranked with `mResolvePrioritySorter`. The implementation of the ranking is implemented with Line 13500 to Line 13537. The comparison is based on the following rules:

- First, Android will compare them by `priority` field. One with the higher priority will be ranked higher than another;
- Second, Android will compare them by `preferredOrder` (user's preference) field;
- Third, Android will compare them by `isDefault` field. (This filter has specified the Intent.CATEGORY_DEFAULT, meaning it would like to be considered a default action that the user can perform on this data) The default one will have a higher rank (See `isDefault` field in ResolveInfo);
- Third, Android will compare them by `match` field. The system's evaluation of how well the activity matches the IntentFilter (See `match` field in ResolveInfo);
- Forth, Android will compare them by `system` field. For short, the one from system process will be higher ranked;
- Fifth, Android will compare them by package names if activityInfo, serviceInfo or providerInfo is not null. Moreover, Android will find both our PoC Instant App and Google Trip Instant App have activity to resolve the URL. Therefore, the activityInfo is not null. When comparing our PoC Instant App and the victim Instant App (i.e., Google Trip), our PoC Instant App will be ranked higher according to the string comparison of package names (a.chrisyttang.instantappurlauto < com.google.android.app.travel.onthego). We will elaborate more in Sec. 1.3 Q4.

For all these fields mentioned, please also refer to `/frameworks/base/core/java/android/content/pm/ResolveInfo.java` for detailed descriptions.

7 Line 6747: Android will filter out ephemeral activities with method `applyPostResolutionFilter`. Here, only activities that (1) are defined in the ephemeral app or (2) marked with `visibleToEphemeral` are returned. Here, `visibleToEphemeral` is an attribute to describe whether an Intent is visible to instant app.

• Return back to method `resolveIntentInternal` at line 5991.

8 Line 5994: All results returned by method `queryIntentActivitiesInternal` will be passed to method `chooseBestActivity`

9 Method `chooseBestActivity`: If there are more than one activity that can resolve the Intent, Android does the following checking: (1) check whether the first activity is the default activity to resolve the Intent or has a higher priority comparing to others. If yes, the first activity will respond the Intent; (2) If the checking returns no, Android will check whether there is a saved preference for a preferred activity for this Intent. If there is a saved preference, Android uses the preferred activity to respond; (3) It will loop all activities to find whether there is an Instant App that can respond the Intent. If there are more than one Instant Apps that can resolve this Intent, the top-ranked one will be returned directly; (4) If there is no instant app, and no saved preferred activity to respond the Intent, it will show a list of apps to let users select one to respond.

10 Line 5996: Return the best activity;

• Return back to method `resolveIntent` at line 5969.

1.8 Attack Analysis

Furthermore, we will introduce why the attack can be launched by answering the following questions.

Q1: Why the PoC Instant App can be download successfully?

It is because our Instant App is hosted on our own website `https://www.chrisyttang.org` and users can launch the instant app with the URL `https://www.chrisyttang.org/main`. Users can download and launch our PoC Instant App successfully as it is obtained from our own URI rather than other URIs.

Q2: Why the PoC Instant App can be considered as a candidate for resolving the link?

We let our PoC instant app have an intent-filter as shown in Fig. 1 so that it can respond the URI `https://google.com/tripsapp`.

Q3: Can ❼ filter out the PoC Instant App?

No, as our activity to resolve the link `https://google.com/trip` is defined in an instant app (our PoC instant app), the method `applyPostResolutionFilter` will not filter out our PoC instant app.

Q4: Why the PoC Instant App is launched rather than the right one (Instant App Hijacking)?

First, base on the analysis ❻ and ❼, we know that all candidates for resolving the link `https://google.com/tripsapp` will be ranked according to the package name. Our PoC Instant App has the package name `a.chrisyttang.instantappurlauto`, which is *lower* than `com.google.android.apps.travel.onthego` (the package name of Google Trips Instant App) under the string comparison. As a result, our PoC Instant App will be ranked higher than the victim Instant App (i.e., Google Trips) (see ❻).

Second, when Android tries to resolve the Intent with the `chooseBestActivity` (see ❾) method, it finds two possible Instant Apps that resolve the Intent and our PoC Instant App is ranked higher than the victim Instant App(i.e., Google Trips). From ❾(3), we know that our PoC Instant App will be selected to respond the Intent.

Therefore, the attack is launched successfully.

Q4': Why the PoC Instant App is launched rather than the right one (URL Hijacking)?

Base on the analysis ❻ and ❼, when Android tries to resolve the Intent with the `chooseBestActivity` (see ❾) method, it finds an Instant App to resolve the Intent. From ❾(3), we know that our PoC Instant App will be selected to respond the Intent.

Therefore, the attack is launched successfully.

1.9 How to Fix

One possible solution to solve the Instant App Hijacking vulnerability is that if there are multiple instant apps that can resolve the link, Android needs to fetch the assetlink.json from the web server in order to obtain the list of apps that are

defined in the assetlinks.json. Alternatively, Android can list all the candidates and ask the user to select one.

Recall our Google Plus example, Android allows our PoC Instant app to respond the link `https://plus.google.com` even if the link is not bind with any app. To address URL Hijacking, Android should check the assetlink.json to verify whether an existing Instant App can respond the URL.