RUNTIME PERMISSIONS IN ANDROID 6.0

Lecture 13

COMPSCI 702

Security for Smart-Devices

Nalin Asanka Gamagedara Arachchilage

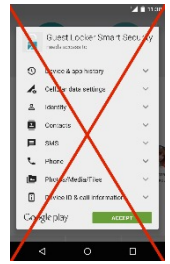
Slides from Muhammad **Rizwan** Asghar

March 30, 2021

ANDROID 6.0

- A version of the Android mobile operating system officially released in October 2015
- Named **Marshmallow**
- Android 6.0 corresponds to the SDK API level 23
- As of October 26, 2018, **21.3%** (and **49.7%**) of the devices accessing Google Play run Android 6.0 (and later versions, i.e., 7.0-8.1)
  - Source: https://developer.android.com/about/dashboards/index.html

RUNTIME PERMISSIONS



- In Android 6.0+ (API level 23+), users grant permissions at runtime
  - When the app is running
  - Not when they install the app
- Granting permissions at runtime streamlines the app installation process
- It also gives the user more control over the app's functionality
  - For example, a user could choose to give a camera app access to the camera but not to the device location

REVOKING PERMISSIONS

- The user can revoke the permissions at any time, by going to the app's Settings screen!

- It provides flexibility and more control to the user

PERMISSION PROTECTION LEVELS

- From the user point of view, we can divide permissions into two categories

- Normal permissions
  - Normal permissions do not directly risk the user's privacy
  - If your app lists a normal permission in its manifest, the system grants the permission automatically

- Dangerous permissions
  - Dangerous permissions can give the app access to the user's confidential data
  - If you list a dangerous permission, the user has to explicitly give approval to your app

CHANGE IN ANDROID 6.0

- Before API level 23, the user has to grant dangerous permissions when they install the app
  - If the user does not grant the permission, the system does not install the app at all
- With API level 23 (or later), the app has to list the permissions in the manifest, and it must request each dangerous permission it needs while the app is running
- The user can grant or deny each permission, and the app can continue to run with limited capabilities, even if the user denies a permission request

CHECK FOR PERMISSIONS

- You must check whether you have the permission every time you perform an operation that requires that permission by calling the *checkSelfPermission()* method

```
// Assume thisActivity is the current activity
int permissionCheck = ContextCompat.checkSelfPermission(thisActivity,
      Manifest.permission.WRITE_CALENDAR);
```

- If the app has the permission, the method returns *PackageManager.PERMISSION_GRANTED*, and the app can proceed with the operation

- If the app does not have the permission, the method returns *PERMISSION_DENIED*, and the app has to explicitly ask the user for permission

EXPLAIN AND REQUEST PERMISSIONS

- Android provides ways to request a permission
- You might want to help the user understand why your app needs a specific permission
- Keep in mind that you do not want to overwhelm the user with explanations
- If you provide too many explanations, the user might find the app frustrating and remove it

EXPLAINING AND ASKING PERMISSION

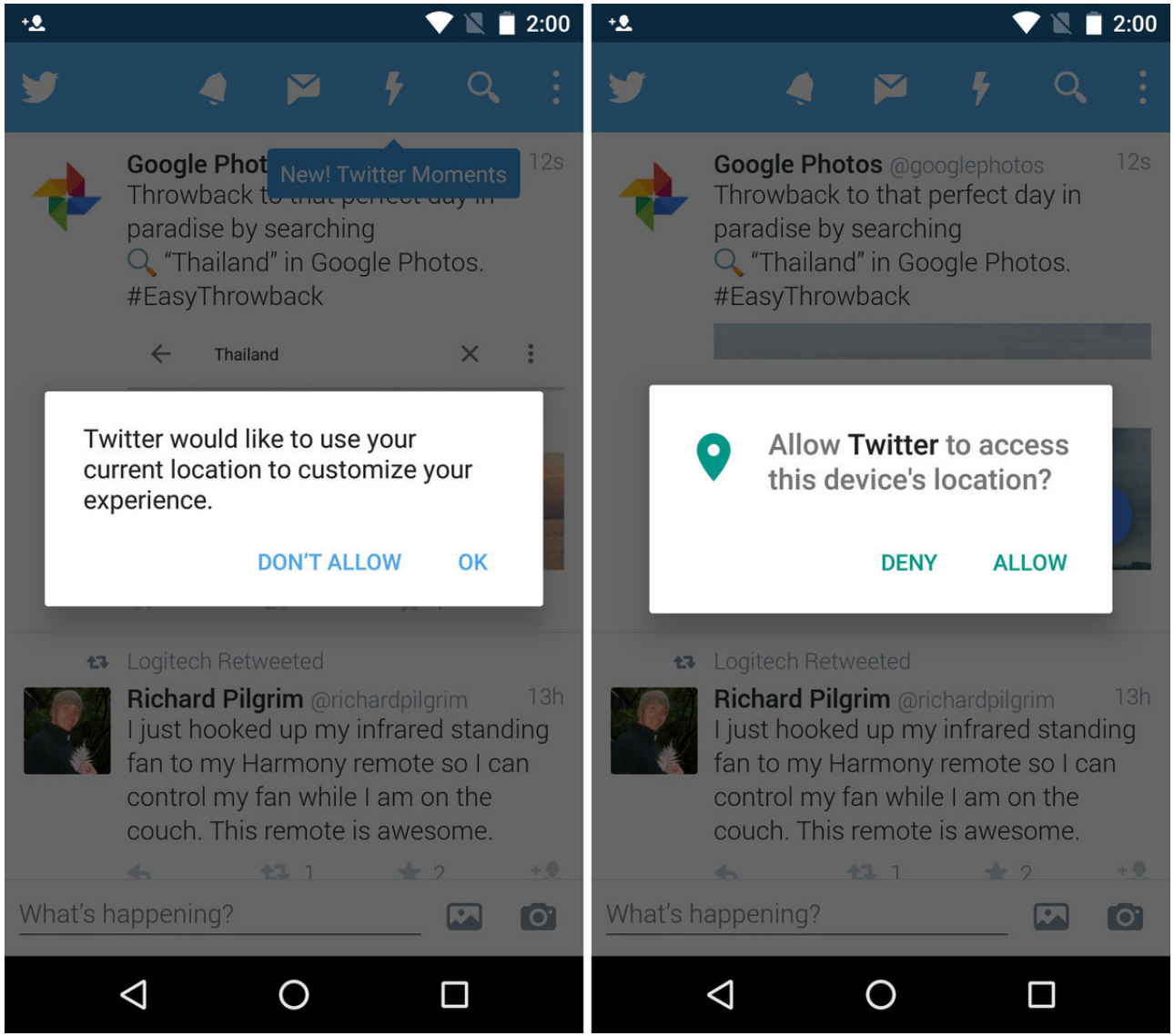
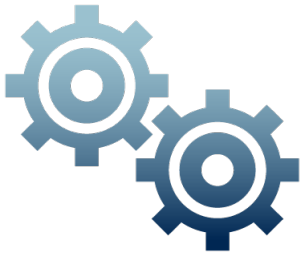


Image Source: androidcentral.com

EXPLAINING AND ASKING PERMISSION

```
if (ContextCompat.checkSelfPermission(thisActivity,
            Manifest.permission.READ_CONTACTS)
      != PackageManager.PERMISSION_GRANTED) {

   // Should we show an explanation?
   if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
         Manifest.permission.READ_CONTACTS)) {

      // Show an expanation to the user *asynchronously* -- don't block
      // this thread waiting for the user's response! After the user
      // sees the explanation, try again to request the permission.

   } else {

      // No explanation needed, we can request the permission.

      ActivityCompat.requestPermissions(thisActivity,
            new String[]{Manifest.permission.READ_CONTACTS},
            MY_PERMISSIONS_REQUEST_READ_CONTACTS);

      // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
      // app-defined int constant. The callback method gets the
      // result of the request.
   }
}
```

HANDLING REQUESTS



- When your app requests permissions, the system presents a dialog box to the user
- Your app cannot configure or alter that dialog box
- When the user responds, the system invokes *onRequestPermissionsResult()*
- Your app has to override that method to find out whether the permission has been granted

HANDLING REQUESTS

```
Override
public void onRequestPermissionsResult(int requestCode,
      String permissions[], int[] grantResults) {
   switch (requestCode) {
      case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
         // If request is cancelled, the result arrays are empty.
         if (grantResults.length > 0
            && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

            // permission was granted, yay! Do the
            // contacts-related task you need to do.

         } else {

            // permission denied, boo! Disable the
            // functionality that depends on this permission.
         }
         return;
      }

      // other 'case' lines to check for other
      // permissions this app might request
   }
}
```

MULTIPLE PERMISSIONS

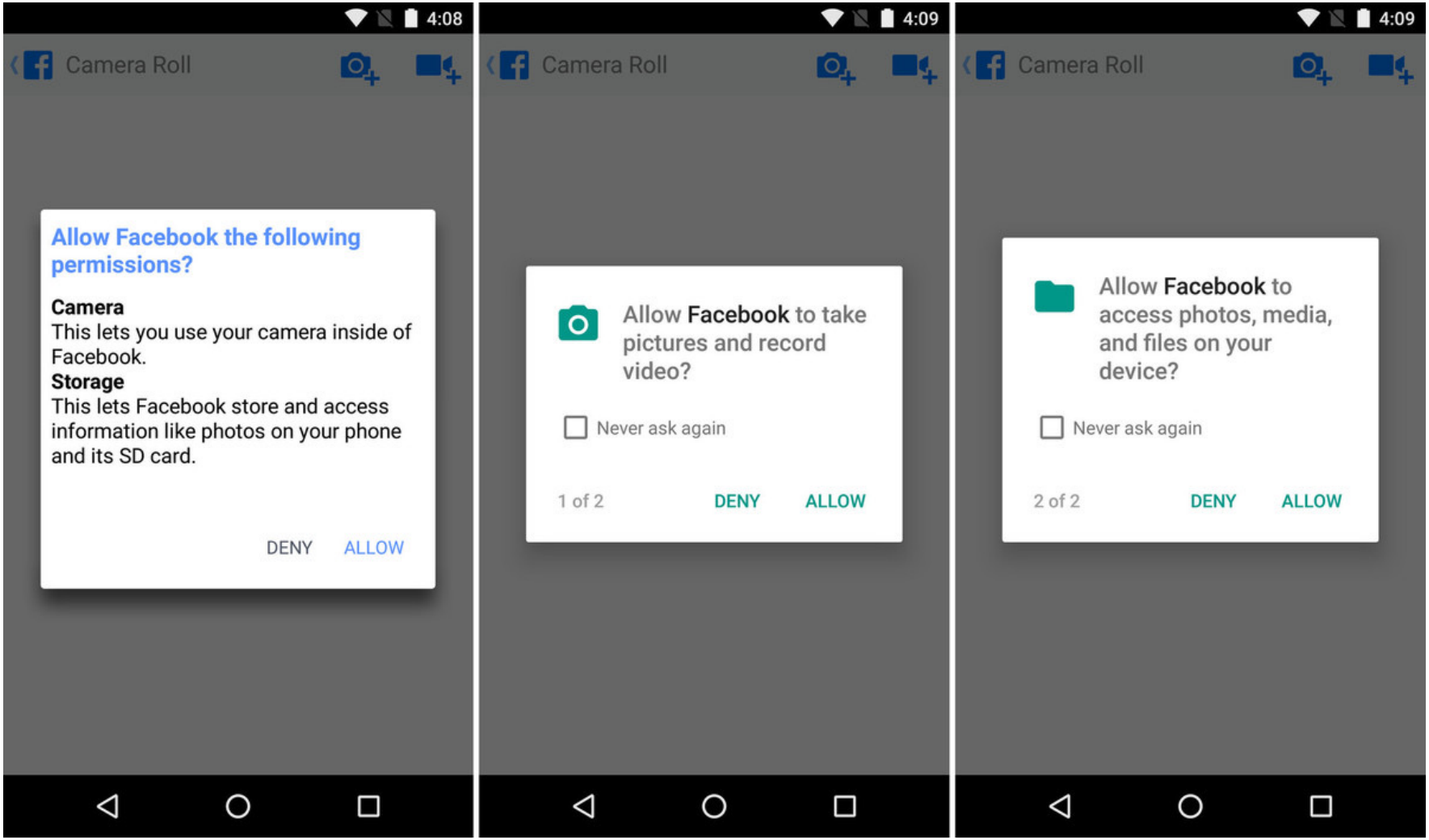


Image Source: androidcentral.com

USING AN INTENT

- In many cases, you can choose one of two ways for your app to perform a task
- Your app can ask for the permission to perform the operation
- Alternatively, the app could use an intent to have another app perform the task
- Example
  - If you need to make a phone call and access the user's contacts, you can do that by creating an appropriate intent

PERMISSIONS VS INTENT

- Using **permission**, your app has full control over the user experience
- However, such broad control adds to the complexity of your task, since you need to design an appropriate UI

- Using **intent**, you do not have to design the UI for the operation
- The app that handles the intent provides the UI
- However, this means you have no control over the user experience

BEST PRACTICES

- Consider using an intent
- Only ask for permissions your app needs
- Do not overwhelm the user
- Explain why you need the permissions

PERMISSION WORKFLOW

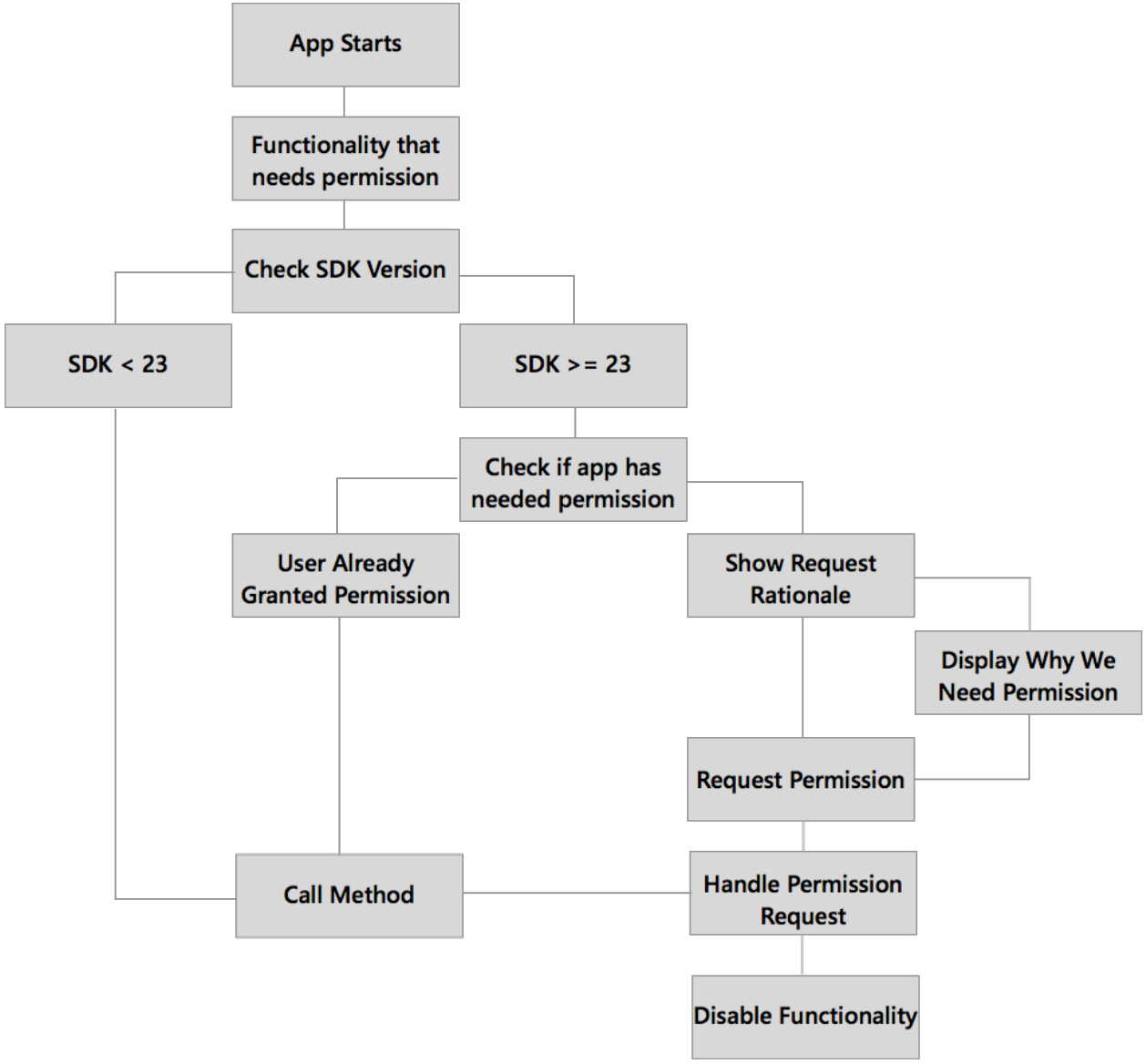
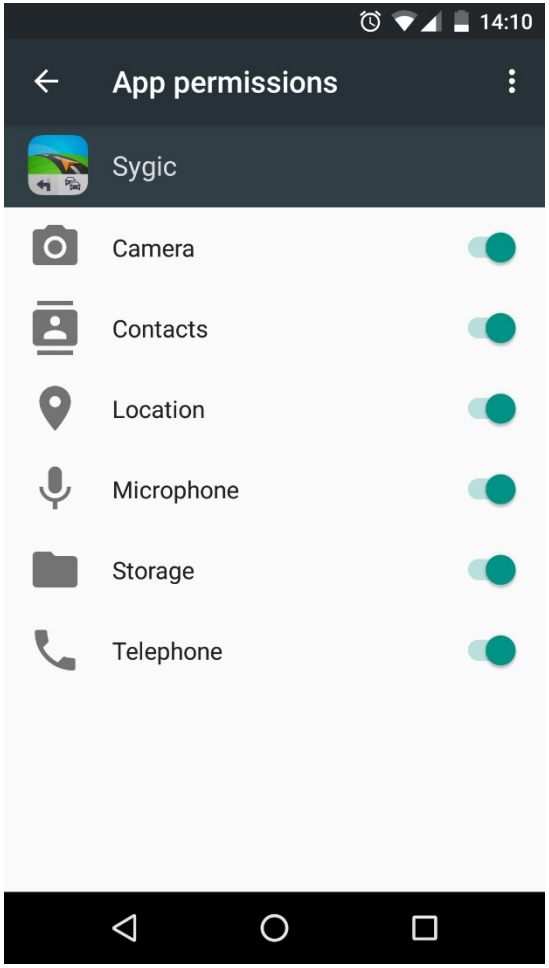
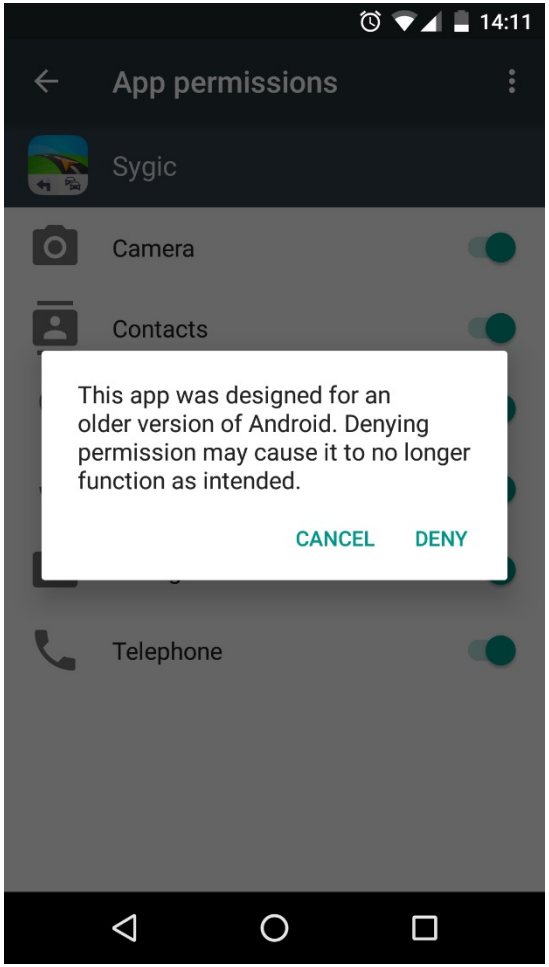


Image Source: xamarin.com

REVOKING PERMISSIONS



REVOKING PERMISSIONS OF OLD APPS



RESOURCES

- **Requesting Permissions at Run Time**
  http://developer.android.com/training/permissions/requesting.html

- **Permissions Best Practices**
  http://developer.android.com/training/permissions/best-practices.html

Questions?

Thanks for your attention!