MIT World Peace University

Wireless Devices and Mobile Security
Third Year B. Tech, Semester 5

Demonstration of Security Permissions in Android using Apps.

Lab Assignment 4

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 10

November 26, 2023

Contents

1 Aim

Study the security permissions for applications in android phones. Either demonstrates Android security permission configurations or Write the android app to demonstrate permissions usage control in android phones.

2 Objectives

1. To understand the basics of Android permissions
2. To increase user awareness and limit an app's access to sensitive data
3. Configuring permissions on Android 8.0 and lower includes allow listing, camera, storage, location permission, etc.

3 Theory

3.1 Android Security Architecture

Android security architecture is a multi-layered system designed to safeguard Android devices and user data. Key components include:

- **Linux Kernel Security:** Provides core security features like process isolation and SELinux.
- **Hardware Security:** Utilizes Trusted Execution Environments (TEEs) and hardware-backed security features.
- **Application Sandboxing:** Ensures each Android app runs in its own isolated environment.
- **Permission System:** Controls app capabilities, requiring explicit user approval for sensitive actions.
- **Secure Boot and Verified Boot:** Ensures only trusted firmware and software run during device startup.
- **Android Keystore:** A secure container for cryptographic keys used by apps.
- **Network Security:** Enforces secure communication practices, supporting protocols like HTTPS.
- **Updates and Patching:** Regular updates and security patches to address vulnerabilities.
- **Google Play Protect:** Scans apps for malware before and after installation.
- **Biometric Authentication:** Supports fingerprint and facial recognition for device security.

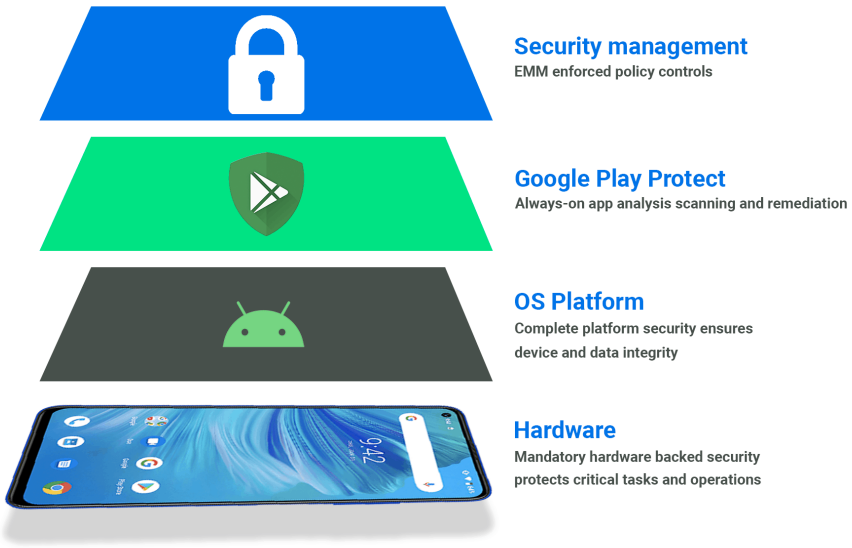

Figure 1: Android Security layers

3.2 Purpose of Permission to Protect the Privacy of an Android User

The purpose of permissions in Android is to protect the privacy and security of the user. Permissions define the actions an app can perform and the data it can access. By requiring explicit user approval for sensitive actions, Android ensures that users have control over their data and can make informed decisions about granting or denying access to specific resources. This permission model helps prevent unauthorized access, enhances user privacy, and mitigates potential security risks.

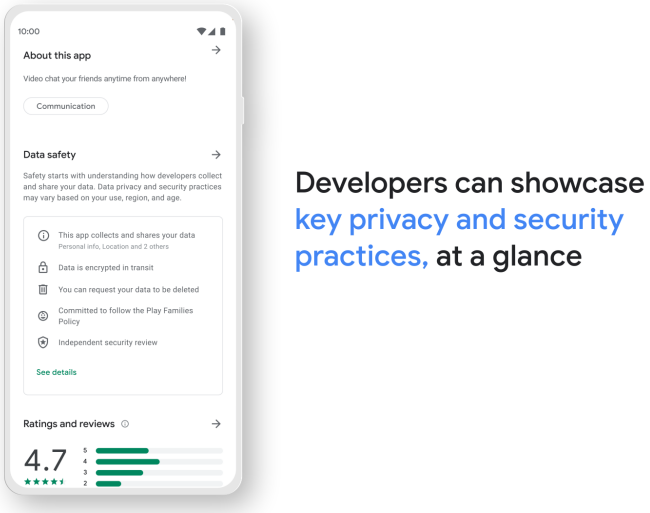

Figure 2: Policies displayed in the Play Store

3.3 Permission Approval Example

When an Android app requests access to sensitive resources, the system prompts the user for permission. For example, when a photo editing app wants to access the device's camera, a permission dialog appears. The user can then choose to grant or deny camera access. If granted, the app can utilize the camera for taking photos. If denied, the app is restricted from accessing the camera, enhancing user control over their data.

Figure 3: Approving permissions must be done judiciously

3.4 Android Security Permissions: List of Permissions, Meaning with Examples

Android permissions control app capabilities. Common permissions include:

- **CAMERA:** Allows the app to use the device camera. Example: A photo-taking app.
- **READ_CONTACTS:** Grants access to the user's contacts. Example: A messaging app needing contact information.
- **ACCESS_FINE_LOCATION:** Permits access to precise device location. Example: A navigation app.
- **READ_SMS:** Enables reading SMS messages. Example: An app for managing text messages.
- **RECORD_AUDIO:** Allows recording audio. Example: A voice recorder app.

These permissions ensure that apps have the necessary access for their intended functionality while protecting user privacy and security.

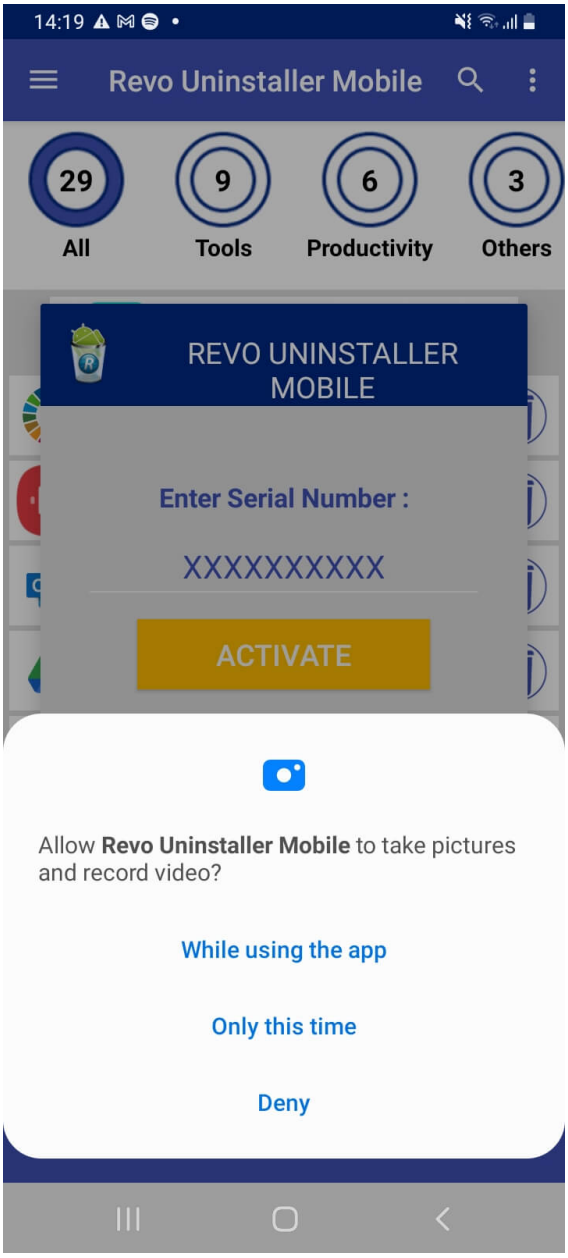

Figure 4: Apps Requesting Permissions

Figure 5: A Green Dot on the Notification bar now represents that the camera and Microphone is in use.

4 Platform

Operating System: Ubuntu 22.04 x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: NS2, NAM 1.4

5 Screenshots

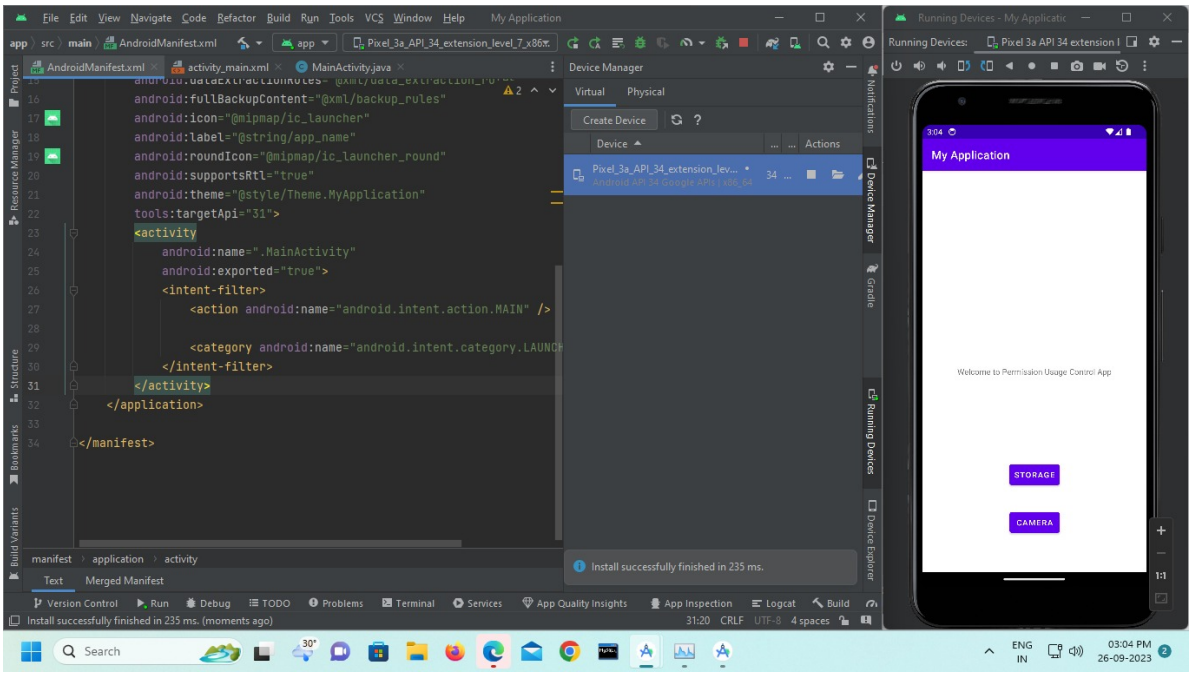

Figure 6: Application Open in the Emulator

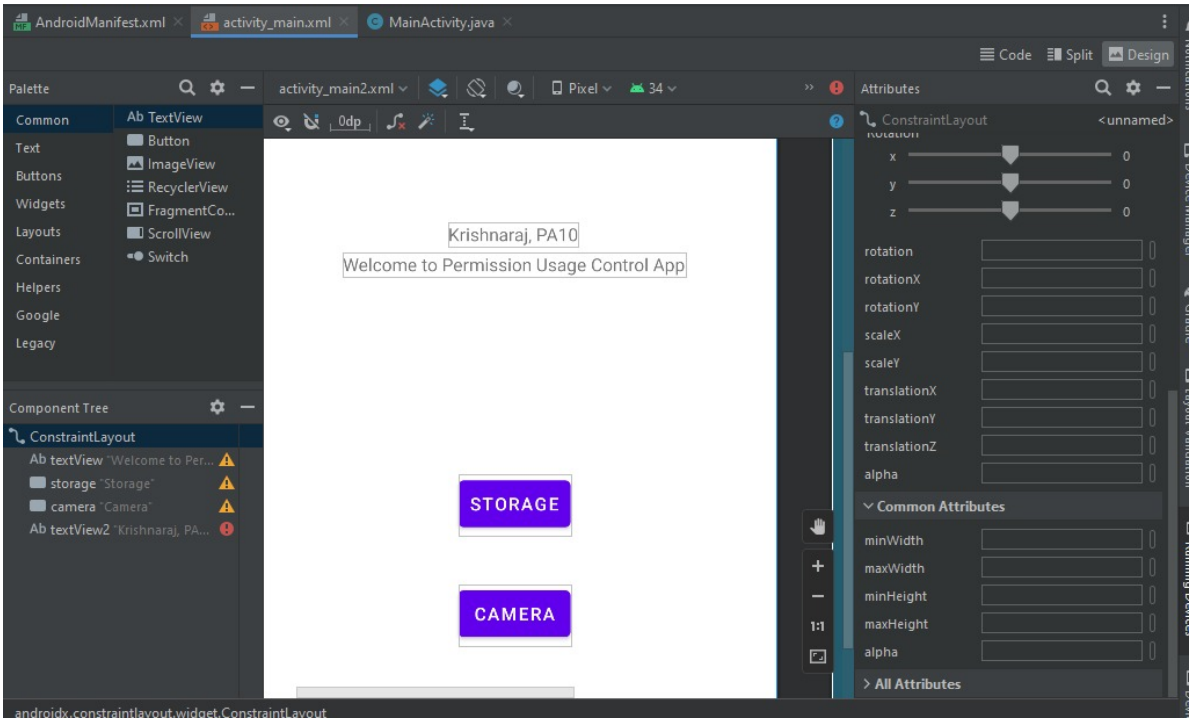

Figure 7: Designing and Adding Buttons to the App

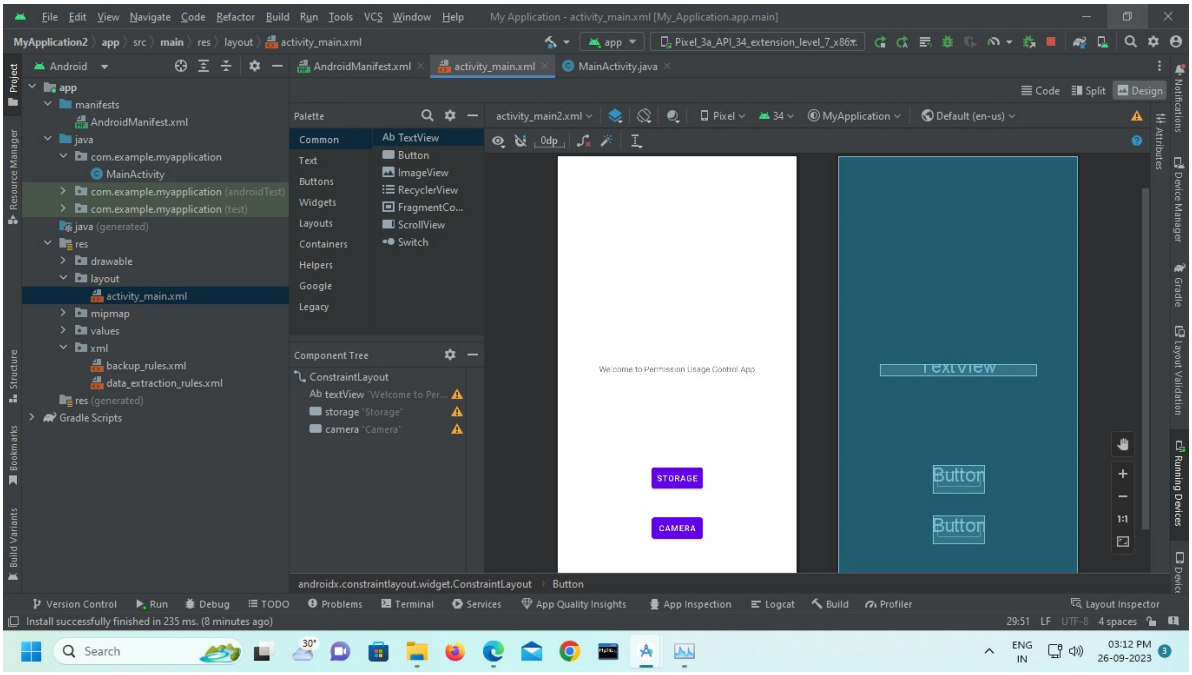

Figure 8: Designing Layout

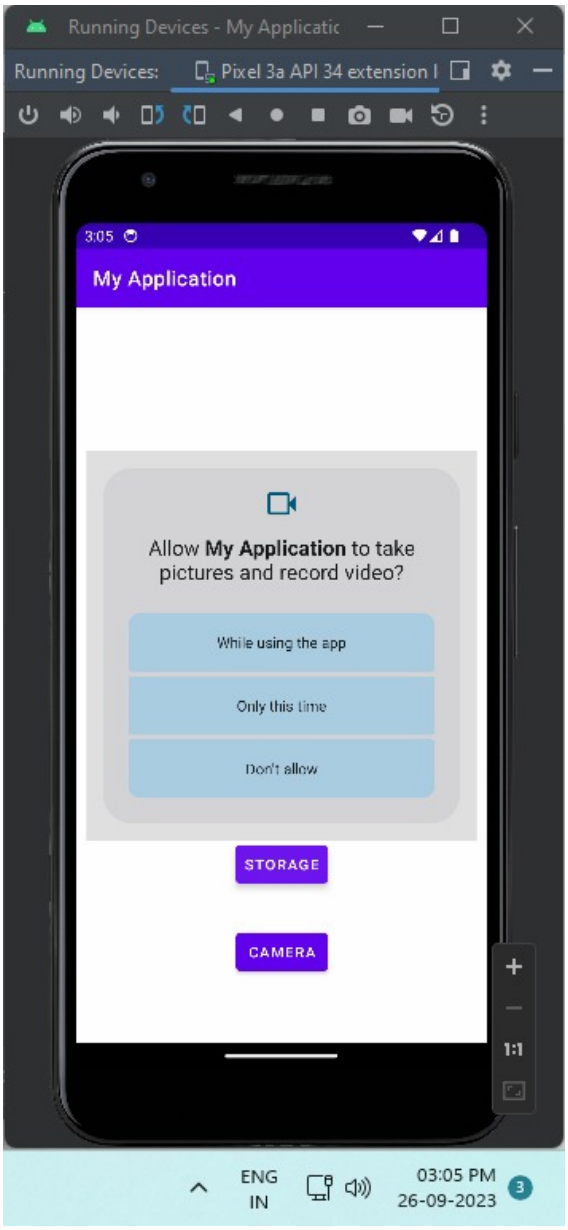

Figure 9: Accepting Permission for Camera

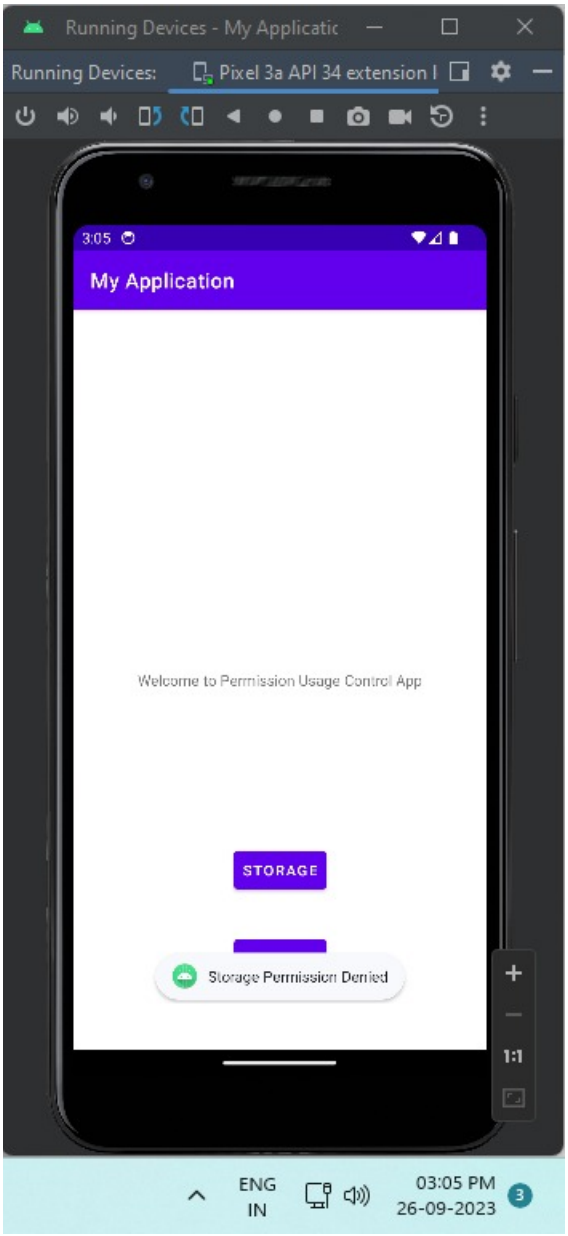

Figure 10: Denying Permission for Storage

6 Code and Algorithm

6.1 Algorithm

1. Open Android Studio and create a new project.
2. Add two buttons to the app's layout, one for invoking the camera and one for invoking storage permission.
3. Edit the app's manifest file to include the necessary permissions for camera and storage.
4. Write functions in the app's activity files to handle the button clicks and request the appropri-

ate permissions.

5. Build the app and test it on an emulator or physical device.
6. Debug and fix any issues that arise during testing.
7. Export the app as an APK file.

6.2 Code

```
package com.example.myapplication;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.os.Bundle;

import android.Manifest;
import android.content.pm.PackageManager;

import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    // Defining Buttons
    private Button storage, camera;

    // Defining Permission codes.
    // We can give any value
    // but unique for each permission.
    private static final int CAMERA_PERMISSION_CODE = 100;
    private static final int STORAGE_PERMISSION_CODE = 101;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        storage = findViewById(R.id.storage);
        camera = findViewById(R.id.camera);

        // Set Buttons on Click Listeners
        storage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                checkPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE,
    STORAGE_PERMISSION_CODE);
            }
        });

        camera.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
48              public void onClick(View v)
49              {
50                  checkPermission(Manifest.permission.CAMERA, CAMERA_PERMISSION_CODE
    );
51              }
52          });
53      }
54
55      // Function to check and request permission.
56      public void checkPermission(String permission, int requestCode)
57      {
58          if (ContextCompat.checkSelfPermission(MainActivity.this, permission) ==
    PackageManager.PERMISSION_DENIED) {
59
60              // Requesting the permission
61              ActivityCompat.requestPermissions(MainActivity.this, new String[] {
    permission }, requestCode);
62          }
63          else {
64              Toast.makeText(MainActivity.this, "Permission already granted", Toast.
    LENGTH_SHORT).show();
65          }
66      }
67
68      // This function is called when the user accepts or decline the permission.
69      // Request Code is used to check which permission called this function.
70      // This request code is provided when the user is prompt for permission.
71
72      @Override
73      public void onRequestPermissionsResult(int requestCode,
74                                             @NonNull String[] permissions,
75                                             @NonNull int[] grantResults)
76      {
77          super.onRequestPermissionsResult(requestCode,
78                  permissions,
79                  grantResults);
80
81          if (requestCode == CAMERA_PERMISSION_CODE) {
82              if (grantResults.length > 0 && grantResults[0] == PackageManager.
    PERMISSION_GRANTED) {
83                  Toast.makeText(MainActivity.this, "Camera Permission Granted",
    Toast.LENGTH_SHORT) .show();
84              }
85              else {
86                  Toast.makeText(MainActivity.this, "Camera Permission Denied",
    Toast.LENGTH_SHORT) .show();
87              }
88          }
89          else if (requestCode == STORAGE_PERMISSION_CODE) {
90              if (grantResults.length > 0
91                      && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
92                  Toast.makeText(MainActivity.this, "Storage Permission Granted",
    Toast.LENGTH_SHORT).show();
93              } else {
94                  Toast.makeText(MainActivity.this, "Storage Permission Denied",
    Toast.LENGTH_SHORT).show();
95              }
96          }
97      }
```

```
}
```

Listing 1: MainActivity.java

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.
    android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity2">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to Permission Usage Control App"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <!--Button to request storage permission-->
    <Button
        android:id="@+id/storage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:padding="8dp"
        android:text="Storage"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        tools:ignore="MissingConstraints" />

    <!--Button to request camera permission-->
    <Button
        android:id="@+id/camera"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/storage"
        android:layout_centerHorizontal="true"
        android:padding="8dp"
        android:text="Camera"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.501"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/storage"
        app:layout_constraintVertical_bias="0.361"
        tools:ignore="MissingConstraints" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 2: activitymain.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.CAMERA" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Listing 3: androidmanifest.xml

7 Conclusion

Thus, we have studied security permissions for applications in android phones also implemented android app to demonstrate permissions usage control in android phones

8 FAQ

1. **How to Toggle Permission on Android Phones:**

   To toggle permissions on Android phones:

   (a) Open the "Settings" app on your Android device.
   (b) Scroll down and select "Apps" or "Application Manager," depending on your device.
   (c) Choose the app for which you want to modify permissions.
   (d) Navigate to the "Permissions" section within the app settings.
   (e) Toggle on or off the specific permissions according to your preference.
   (f) Confirm the changes, and the app will now have the adjusted permissions.

2. **How Do I Stop an App from Accessing My Contacts:**

   To prevent an app from accessing your contacts on Android:

   (a) Open the "Settings" app on your Android device.
   (b) Go to "Apps" or "Application Manager."
   (c) Select the app you want to restrict from accessing contacts.
   (d) Look for the "Permissions" section.
   (e) Disable the "Contacts" permission for that specific app.
   (f) Confirm the changes, and the app will no longer have access to your contacts.

3. **Can Apps Steal Your Photos:**

   While reputable apps follow strict security protocols, there is a potential risk of malicious apps stealing photos. To mitigate this risk:

   - Only download apps from trusted sources like the Google Play Store.
   - Review app permissions before installation.
   - Regularly check app permissions in device settings and revoke unnecessary access.
   - Keep your device's operating system and apps up to date to benefit from security patches.

4. **What Are App Protection Policies:**

   App protection policies are security measures implemented to safeguard sensitive data within mobile applications. These policies often include:

   - **Data Encryption:** Ensuring that data stored and transmitted by the app is encrypted.
   - **Access Controls:** Defining who can access certain features or data within the app.
   - **Authentication Requirements:** Implementing secure login methods to verify user identity.
   - **Secure Communication:** Ensuring that data exchanged between the app and servers is secure.

   App protection policies are crucial for maintaining the confidentiality and integrity of app data.

5. **What Are the Types of Permissions in Android? Discuss Permission Protection Levels in Android:**

   Types of permissions in Android include:

   - **Normal Permissions:** Granted automatically when the app is installed. Examples include internet access.
   - **Dangerous Permissions:** Require explicit user consent. Examples include accessing contacts or location.
   - **Special Permissions:** Certain permissions that are particularly sensitive and need special handling.

   Permission protection levels in Android determine how permissions are granted and enforced:

   - **Normal:** Automatically granted.
   - **Dangerous:** Requested at runtime and require user approval.
   - **Signature:** Granted to apps signed with the same certificate as the system.
   - **System:** Only granted to system apps.