Insecure Logging

In this section, we'll analyze the concept of insecure logging in Android applications and its potential impact. Insecure logging occurs when sensitive information is unintentionally recorded through the application's logging mechanism. This may include user credentials, payment details, or other personal data handled by the app.

The primary tool used to exploit this vulnerability is Logcat. Logcat is a command-line utility included in the Android SDK that captures system message streams, including logs generated by the application via the `Log` class. These logs are invaluable for developers during debugging, but they can also be a goldmine for penetration testers if they contain sensitive information. When an application logs sensitive data, it may become accessible to anyone with physical access to the device or to other applications running on the same device. Such exposure can significantly increase the risk of a security breach.

The Android logging system consists of a series of structured circular buffers managed by the system process `logd`. The number and types of buffers are fixed and defined by the system. Below are some of the most relevant buffers.

Buffer	Description
`main`	Stores most application logs.
`system`	Stores messages originating from the Android OS.
`crash`	Stores crash logs.

In the upcoming example, we'll examine an application that logs sensitive information and walk through how to identify these log entries using static analysis and Logcat. Before diving into the analysis, let's review a table of commonly used Logcat filters, which can help refine log output based on specific criteria:

Filter	Description	Code	Command
`Verbose`	Shows all log messages (default).	`Log.v(TAG, "Your verbose message");`	`adb logcat '*:V'`
`Debug`	Displays log messages that are only useful during development.	`Log.d(TAG, "Your debug message");`	`adb logcat *:D`
`Info`	Shows general log messages useful for understanding the application's state.	`Log.i(TAG, "Your info message");`	`adb logcat '*:I'`
`Warn`	Displays possible issues that are not yet errors.	`Log.w(TAG, "Your warning message");`	`adb logcat '*:W'`
`Error`	Shows issues that have caused errors.	`Log.e(TAG, "Your error message");`	`adb logcat '*:E'`
`Fatal`	Displays severe error messages that have caused the process to abort.	`Log.wtf(TAG, "Your fatal error message");`	`adb logcat '*:F'`
`Silent`	Shows no log messages. This filter completely silences the log output.	-	`adb logcat '*:S'`

Reading Application Logs

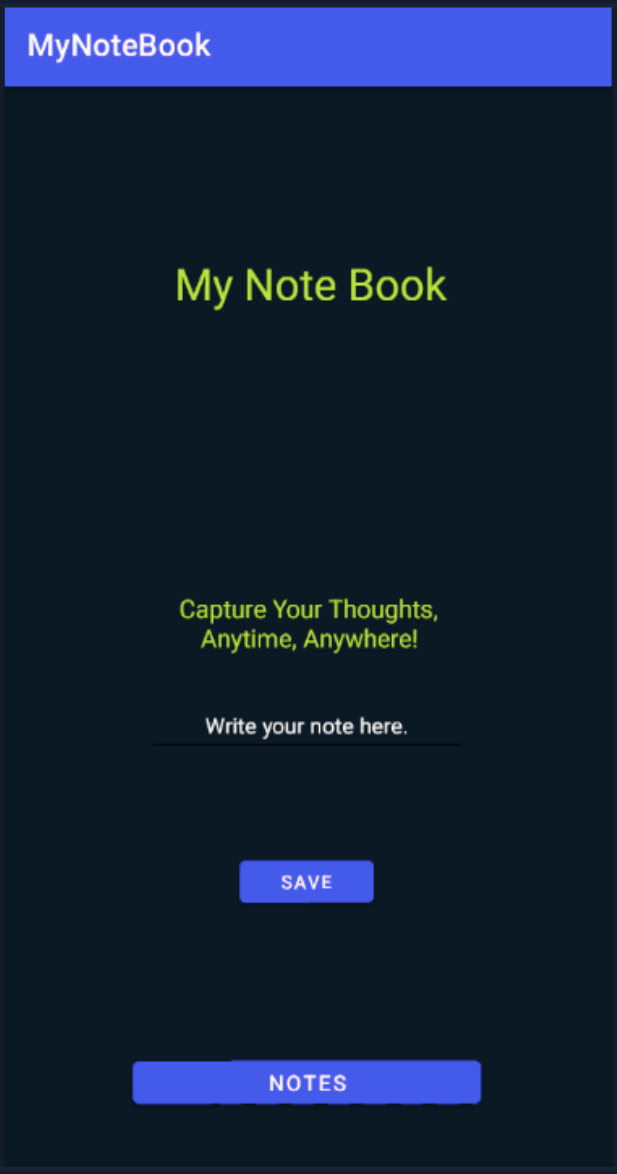
In this example, we'll use an Android Virtual Device (AVD), though the same approach applies to any rooted or emulated Android device. First, let's connect to the device via ADB and install the target application:

Insecure Logging

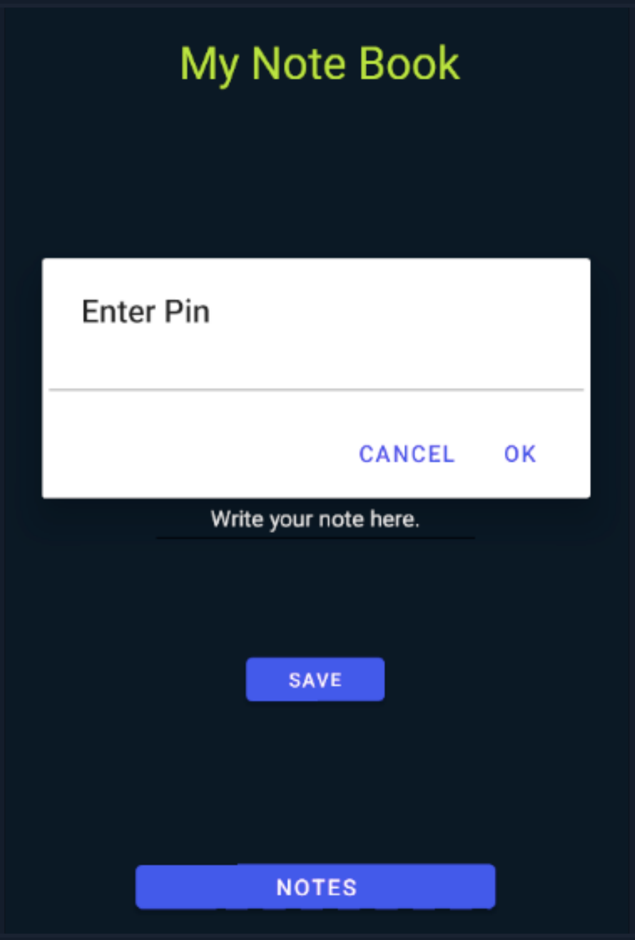
```
rl1k@htb[/htb]$ adb connect
rl1k@htb[/htb]$ adb install myapp.apk
```

```
rl1k@htb[/htb]$ adb install myapp.apk

Performing Streamed Install
Success
```

Running the application, we see it can generate tokens for any usage.



Similar to previous cases, the app requires a PIN to access saved notes:



Let's get the application's package name using the command below while the app is running, so we can further enumerate it.

Insecure Logging

```
rl1k@htb[/htb]$ adb root
rl1k@htb[/htb]$ adb shell
rl1k@htb[/htb]$ adb shell dumpsys activity activities | grep VisibleActivityProcess

VisibleActivityProcess:[ ProcessRecord{8f86b89 6255:com.hackthebox.myapp/u0a120}]
```

Listing the content of the app-specific external storage reveals the following files.

Insecure Logging

```
rl1k@htb[/htb]$ adb shell ls -l /sdcard/Android/data/com.hackthebox.myapp/files/MyPersonalNotes/

total 12
-rw-rw---- 1 u0_a121 ext_data_rw  45 2024-01-16 13:30 Note_-2074205319.txt
-rw-rw---- 1 u0_a121 ext_data_rw 230 2024-01-16 13:30 Note_309943672.txt
-rw-rw---- 1 u0_a121 ext_data_rw 157 2024-01-16 13:30 Note_311366182.txt
```

Reading the content of any of these files reveals only encrypted data:

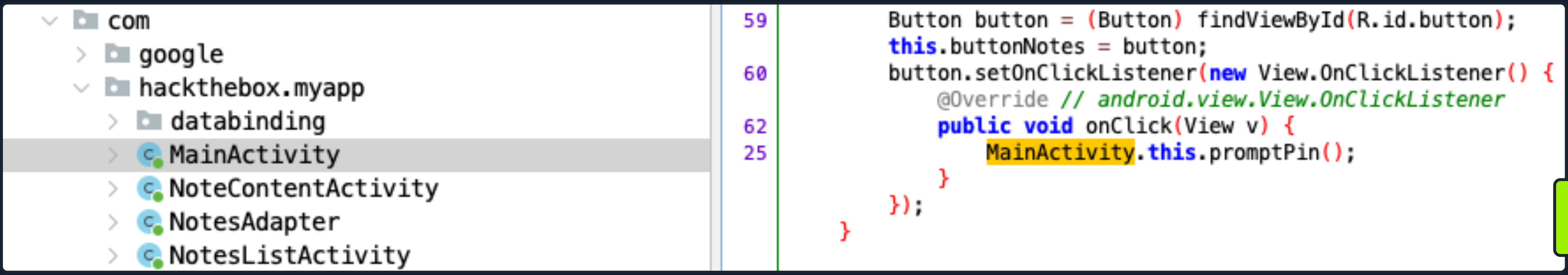
Insecure Logging

```
rl1k@htb[/htb]$ adb shell cat /sdcard/Android/data/com.hackthebox.myapp/files/MyPersonalNotes/Note_-2074205319.txt

NVtNyfHvEvK+Hg2wDJi9AYRckvLnjr19ClYVG7svSx0=
```

To better understand the app's behavior, we'll reverse-engineer it using JADX:.

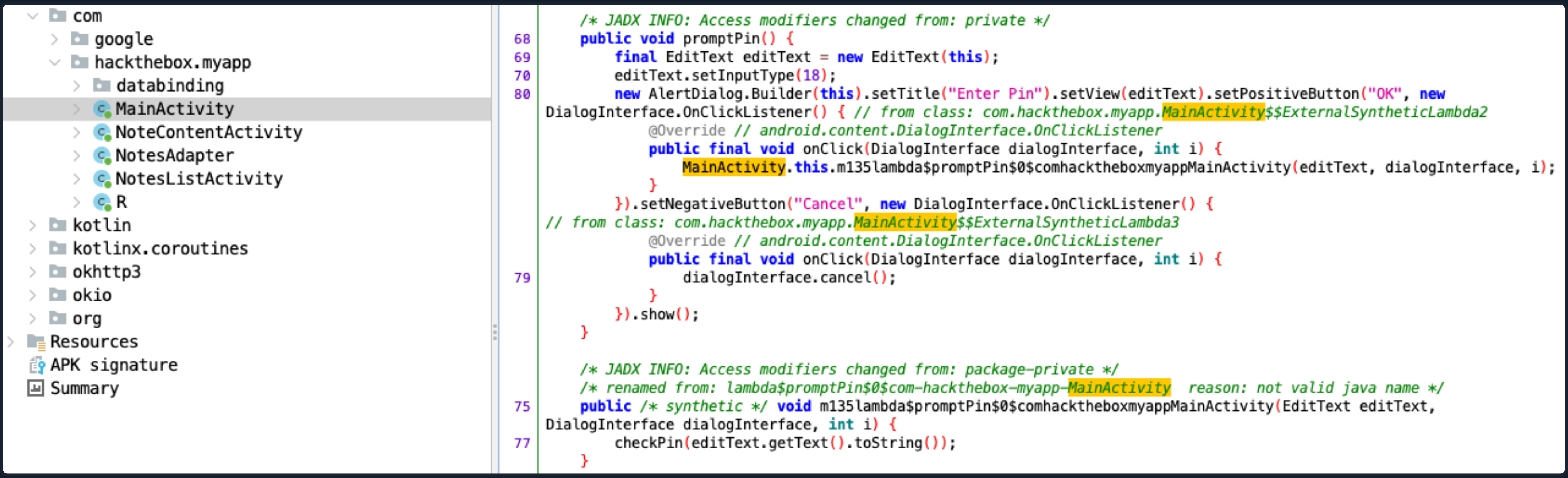
Insecure Logging

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```



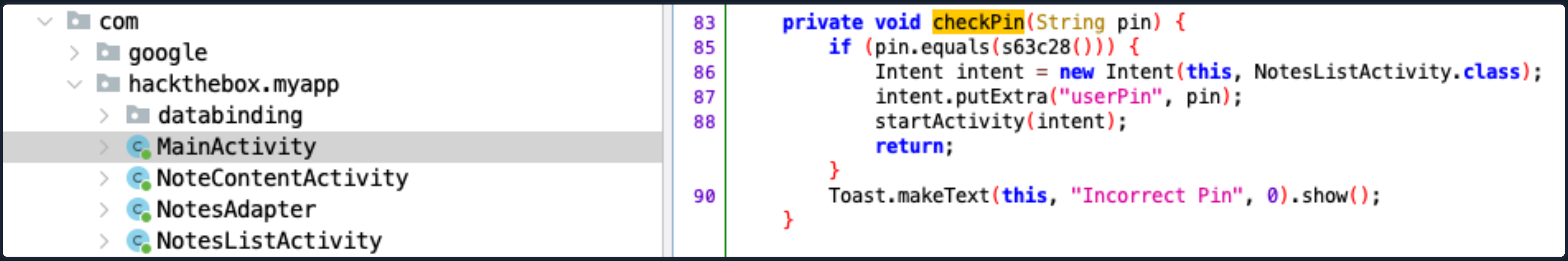
```
        Button button = (Button) findViewById(R.id.button);
        this.buttonNotes = button;
        button.setOnClickListener(new View.OnClickListener() {
            @Override // android.view.View.OnClickListener
            public void onClick(View v) {
                MainActivity.this.promptPin();
            }
        });
    }
```

Reviewing the MainActivity code reveals that tapping the buttonNotes UI element calls the promptPin() method:



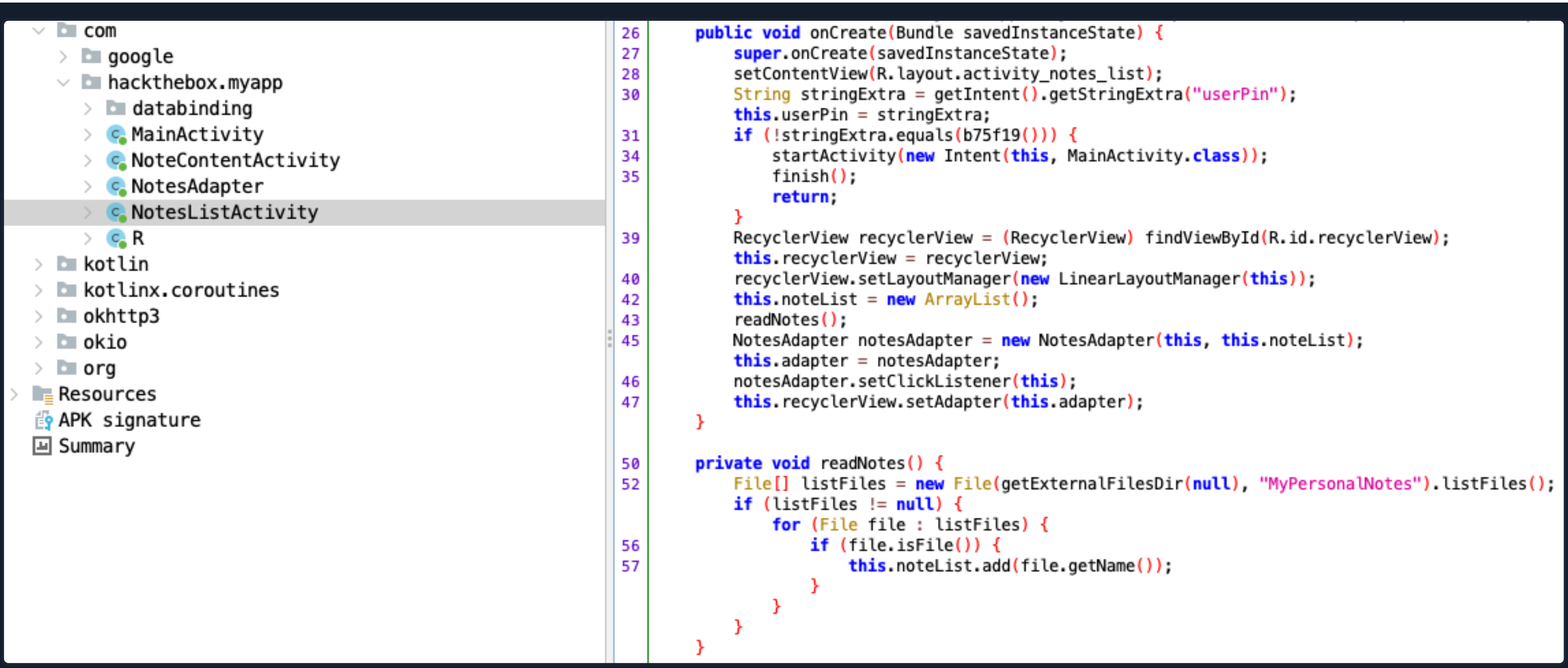
```
    /* JADX INFO: Access modifiers changed from: private */
    public void promptPin() {
        final EditText editText = new EditText(this);
        editText.setInputType(18);
        new AlertDialog.Builder(this).setTitle("Enter Pin").setView(editText).setPositiveButton("OK", new
DialogInterface.OnClickListener() { // from class: com.hackthebox.myapp.MainActivity$$ExternalSyntheticLambda2
            @Override // android.content.DialogInterface.OnClickListener
            public final void onClick(DialogInterface dialogInterface, int i) {
                MainActivity.this.m135lambda$promptPin$0$comhacktheboxmyappMainActivity(editText, dialogInterface, i);
            }
        }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
// from class: com.hackthebox.myapp.MainActivity$$ExternalSyntheticLambda3
            @Override // android.content.DialogInterface.OnClickListener
            public final void onClick(DialogInterface dialogInterface, int i) {
                dialogInterface.cancel();
            }
        }).show();
    }

    /* JADX INFO: Access modifiers changed from: package-private */
    /* renamed from: lambda$promptPin$0$com-hackthebox-myapp-MainActivity  reason: not valid java name */
    public /* synthetic */ void m135lambda$promptPin$0$comhacktheboxmyappMainActivity(EditText editText,
DialogInterface dialogInterface, int i) {
        checkPin(editText.getText().toString());
    }
```

If a PIN is given, the method m135lambda$promptPin$0$comhacktheboxmyappMainActivity() will be called, which in turn will call the method checkPin().



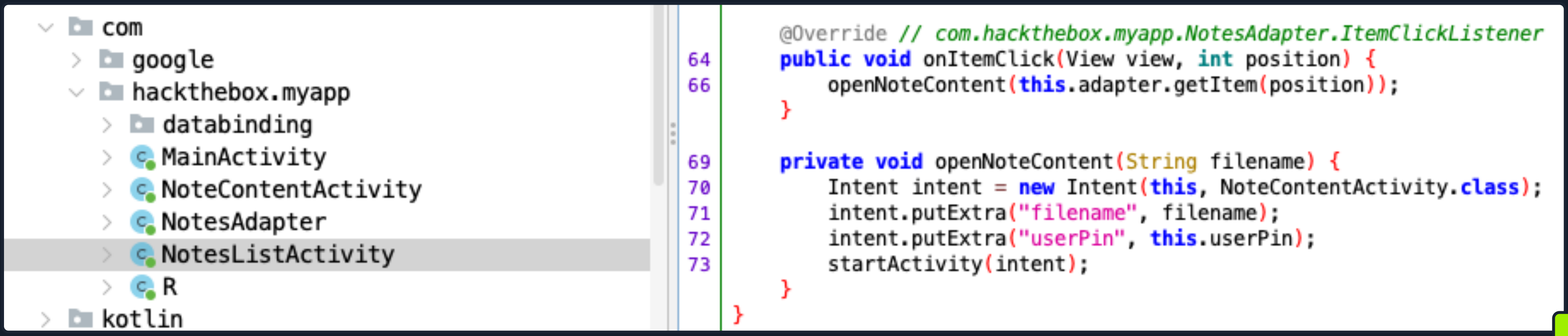
```
    private void checkPin(String pin) {
        if (pin.equals(s63c28())) {
            Intent intent = new Intent(this, NotesListActivity.class);
            intent.putExtra("userPin", pin);
            startActivity(intent);
            return;
        }
        Toast.makeText(this, "Incorrect Pin", 0).show();
    }
```

As shown in the picture above, the method checkPin() will check if the PIN is correct, and if it is, the NotesListActivity will be called with the userPin variable passed as a parameter.



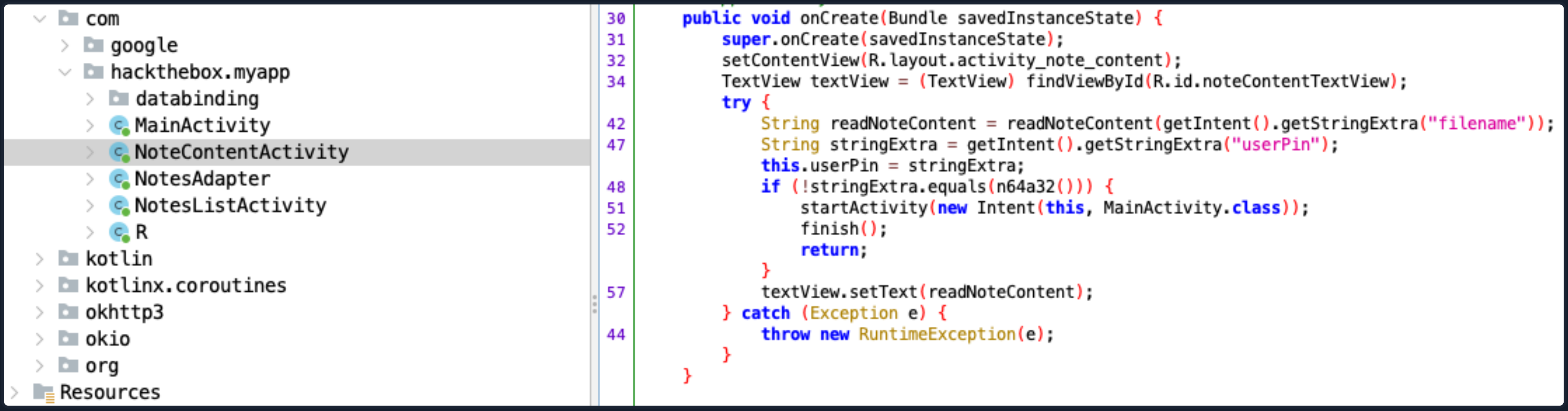
```java
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_notes_list);
        String stringExtra = getIntent().getStringExtra("userPin");
        this.userPin = stringExtra;
        if (!stringExtra.equals(b75f19())) {
            startActivity(new Intent(this, MainActivity.class));
            finish();
            return;
        }
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        this.recyclerView = recyclerView;
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        this.noteList = new ArrayList();
        readNotes();
        NotesAdapter notesAdapter = new NotesAdapter(this, this.noteList);
        this.adapter = notesAdapter;
        notesAdapter.setClickListener(this);
        this.recyclerView.setAdapter(this.adapter);
    }

    private void readNotes() {
        File[] listFiles = new File(getExternalFilesDir(null), "MyPersonalNotes").listFiles();
        if (listFiles != null) {
            for (File file : listFiles) {
                if (file.isFile()) {
                    this.noteList.add(file.getName());
                }
            }
        }
    }
```

Within `NotesListActivity`, the PIN is validated again, and if successful, the available note files are displayed. When the user selects a note, the app starts `NoteContentActivity`, passing both the filename and the PIN:



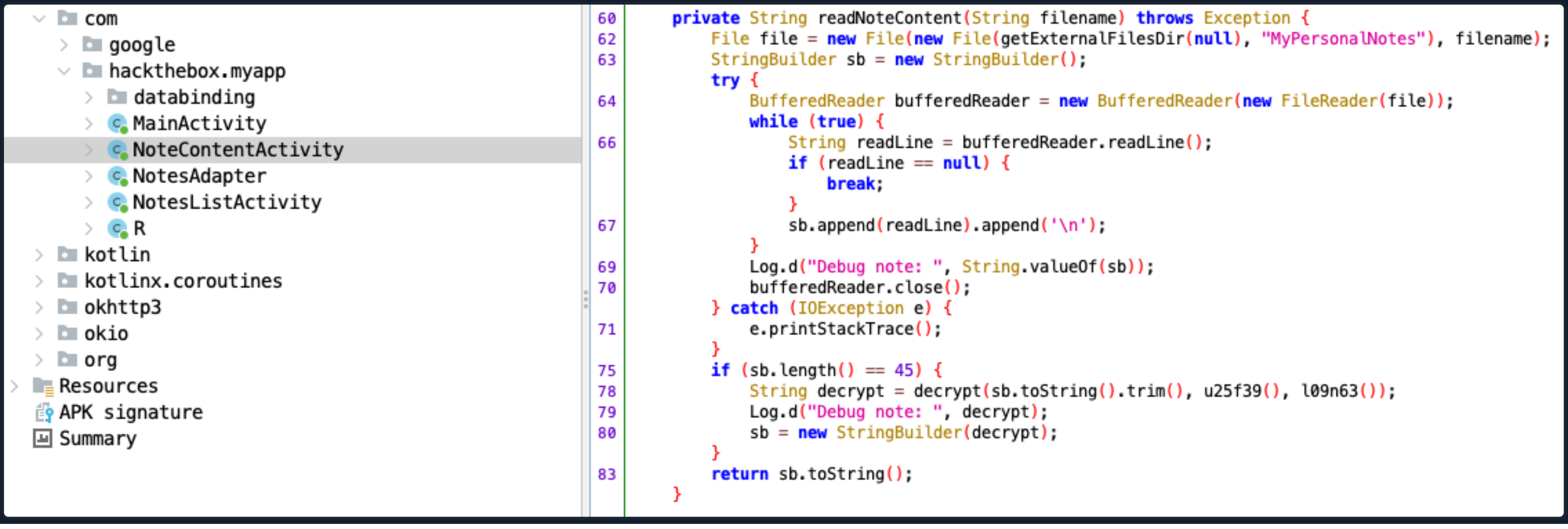
```java
    @Override // com.hackthebox.myapp.NotesAdapter.ItemClickListener
    public void onItemClick(View view, int position) {
        openNoteContent(this.adapter.getItem(position));
    }

    private void openNoteContent(String filename) {
        Intent intent = new Intent(this, NoteContentActivity.class);
        intent.putExtra("filename", filename);
        intent.putExtra("userPin", this.userPin);
        startActivity(intent);
    }
}
```

Examining `NoteContentActivity`, the `onCreate()` method reveals that the `checkPin()` method is invoked after the note content is read:



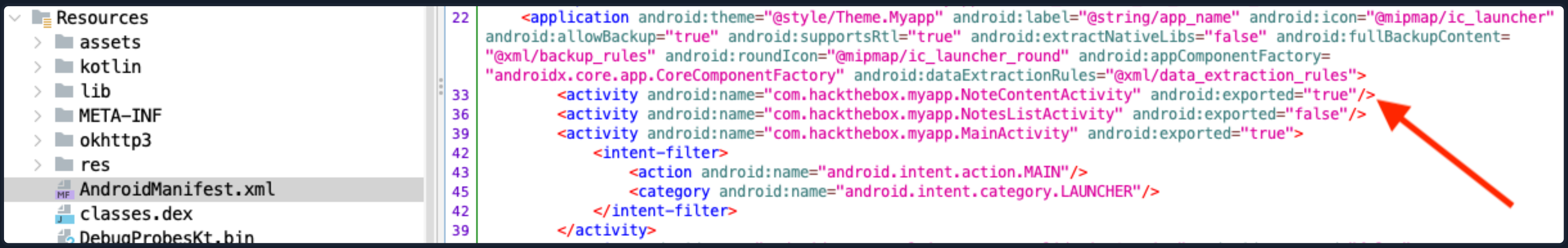
```java
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_note_content);
        TextView textView = (TextView) findViewById(R.id.noteContentTextView);
        try {
            String readNoteContent = readNoteContent(getIntent().getStringExtra("filename"));
            String stringExtra = getIntent().getStringExtra("userPin");
            this.userPin = stringExtra;
            if (!stringExtra.equals(n64a32())) {
                startActivity(new Intent(this, MainActivity.class));
                finish();
                return;
            }
            textView.setText(readNoteContent);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

This means the content is processed before the PIN is validated. However, the user is redirected back to the main screen before they can view the notes. Further inspection of `readNoteContent()` shows that the app logs the decrypted content using `Log.d()`:



```java
    private String readNoteContent(String filename) throws Exception {
        File file = new File(new File(getExternalFilesDir(null), "MyPersonalNotes"), filename);
        StringBuilder sb = new StringBuilder();
        try {
            BufferedReader bufferedReader = new BufferedReader(new FileReader(file));
            while (true) {
                String readLine = bufferedReader.readLine();
                if (readLine == null) {
                    break;
                }
                sb.append(readLine).append('\n');
            }
            Log.d("Debug note: ", String.valueOf(sb));
            bufferedReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (sb.length() == 45) {
            String decrypt = decrypt(sb.toString().trim(), u25f39(), l09n63());
            Log.d("Debug note: ", decrypt);
            sb = new StringBuilder(decrypt);
        }
        return sb.toString();
    }
```

This line logs the `decrypt` variable, which likely contains the plaintext content of the note. Simply tapping the NOTES button won't reach this point in the code unless the correct PIN is provided. But because `NoteContentActivity` is marked as exported in the `AndroidManifest.xml`, we can access it directly:



```
Resources
  assets
  kotlin
  lib
  META-INF
  okhttp3
  res
  AndroidManifest.xml
  classes.dex
  DebugProbesKt.bin
```

```
    <application android:theme="@style/Theme.Myapp" android:label="@string/app_name" android:icon="@mipmap/ic_launcher"
android:allowBackup="true" android:supportsRtl="true" android:extractNativeLibs="false" android:fullBackupContent=
"@xml/backup_rules" android:roundIcon="@mipmap/ic_launcher_round" android:appComponentFactory=
"androidx.core.app.CoreComponentFactory" android:dataExtractionRules="@xml/data_extraction_rules">
        <activity android:name="com.hackthebox.myapp.NoteContentActivity" android:exported="true"/>
        <activity android:name="com.hackthebox.myapp.NotesListActivity" android:exported="false"/>
        <activity android:name="com.hackthebox.myapp.MainActivity" android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
```

Since the activity is exported, it can be started externally using ADB. To do this, we'll need two parameters: the filename, which we found earlier during enumeration, and the PIN. Before launching the activity, let's start Logcat to capture the log output. We could use a broad filter like adb `logcat '*:D'`, but since we already know the tag is `Debug note:`, we can use `grep` for more focused output:

Insecure Logging

```
rl1k@htb[/htb]$ adb logcat '*:D' | grep 'Debug note: '
```

Now let's launch the `NoteContentActivity` directly via ADB:

Insecure Logging

```
rl1k@htb[/htb]$ adb shell am start -n com.hackthebox.myapp/.NoteContentActivity --es filename "Note_-2074205319.txt" --es use

Starting: Intent { cmp=com.hackthebox.myapp/.NoteContentActivity (has extras) }
```

Back in the Logcat output, we can now see the logged content of the note.

Insecure Logging

```
rl1k@htb[/htb]$ adb logcat '*:D' | grep 'Debug note: '

01-17 10:34:13.598  8392  8392 D Debug note: : NVtNyfHvEvK+Hg2wDJi9AYRckvLnjr19ClYVG7svSx0=
01-17 10:34:13.598  8392  8392 D Debug note: : HTB{N0t3s_4r3_l0gg3d!!}
```

Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK 36ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Cheat Sheet

Answer the question(s) below to complete this Section and earn cubes!

+ 5 What is the content of the file "Note_-2074205319.txt" ?

Submit your answer here...

+10 Streak pts Submit insecure_logging.zip

Previous Next

Cheat Sheet

? Go to Questions

Table of Contents

IDOR Attack

SSL/TLS Certificate Pinning Bypass

Skills Assessments

Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left