**📌 Now You See Me — ShaktiCTF2025**

**Category:** Android / Misc / Reversing  
**Author:** 0kay4nu  
**Flag format:** ShaktiCTF{...}  
**Solvers:** Milon Das and @sakibulalikhan

**✅ Challenge description**

*“Some secrets don’t like the spotlight. They prefer memory.”*

You get an APK. When you run it on your phone, you see two reversed strings:

"]!ffo uoy worht ot egassem a tsuj si sihT"

"drah oot ti gnikam ,eludom eht fo trap a ton si sihT"

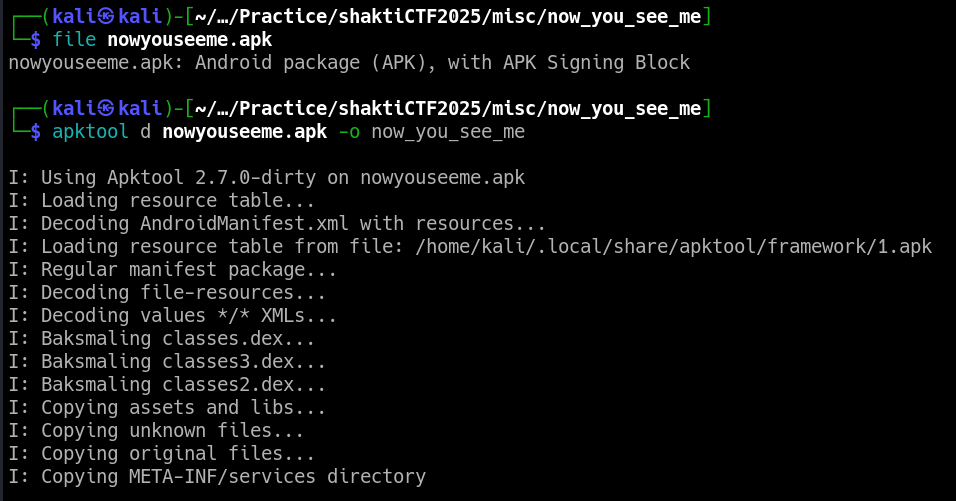
So nothing useful on screen — the flag must be hidden **in memory**.

**✅ Goal**

Find the flag **without dynamic hooking** — just static reverse engineering.

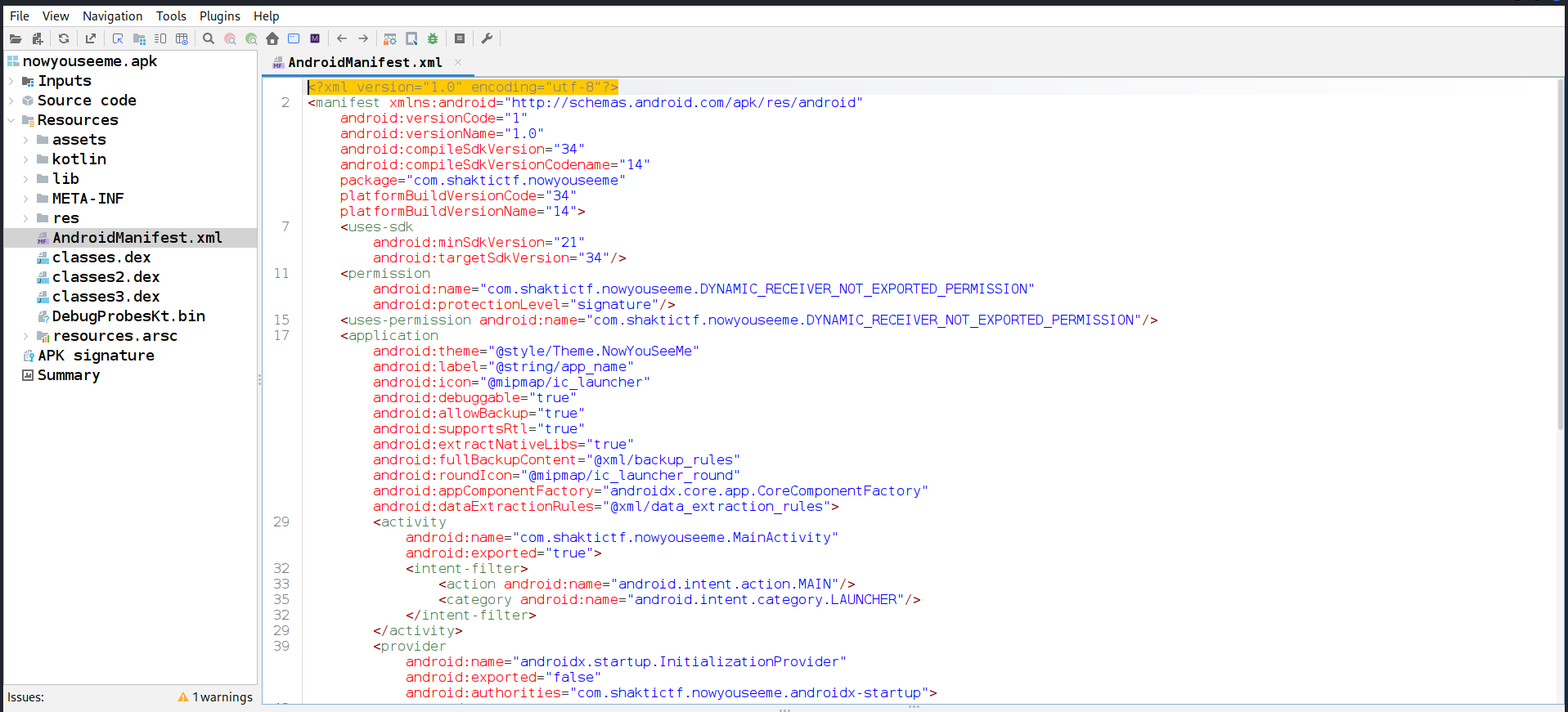
**1️. Unpack the APK**

apktool d nowyouseeme.apk -o now\_you\_see\_me



Then I ran the “jadx-gui” command and opened the given apk file in it.

**Inspect AndroidManifest.xml:**

****

<manifest package="com.shaktictf.nowyouseeme">

...

<application>

<activity android:name="com.shaktictf.nowyouseeme.MainActivity" ... />

</application>

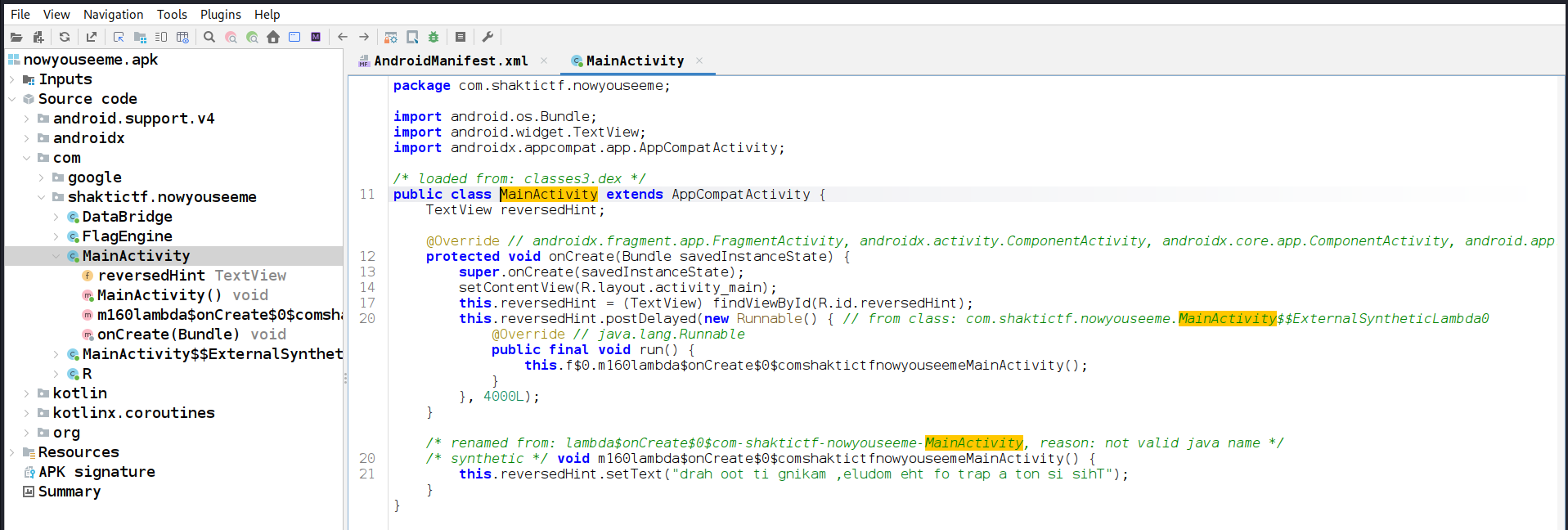
</manifest>

**2. Analyze the main activity**

Decompile with **jadx** or **apktool**:

smali/com/shaktictf/nowyouseeme/MainActivity.smali

Or view the decompiled Java:



package com.shaktictf.nowyouseeme;

import android.os.Bundle;

import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

TextView reversedHint;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

this.reversedHint = (TextView) findViewById(R.id.reversedHint);

this.reversedHint.postDelayed(new Runnable() {

public final void run() {

MainActivity.this.reversedHint.setText(

"drah oot ti gnikam ,eludom eht fo trap a ton si sihT"

);

}

}, 4000L);

}

}

So the app:

* Loads MainActivity
* Sets a TextView to a useless reversed message
* **No sign of the flag**

**3. Spot the real engine**

There’s also:

package com.shaktictf.nowyouseeme;

public class FlagEngine {

public static native String getFlag(String str);

static {

System.loadLibrary("flagengine");

}

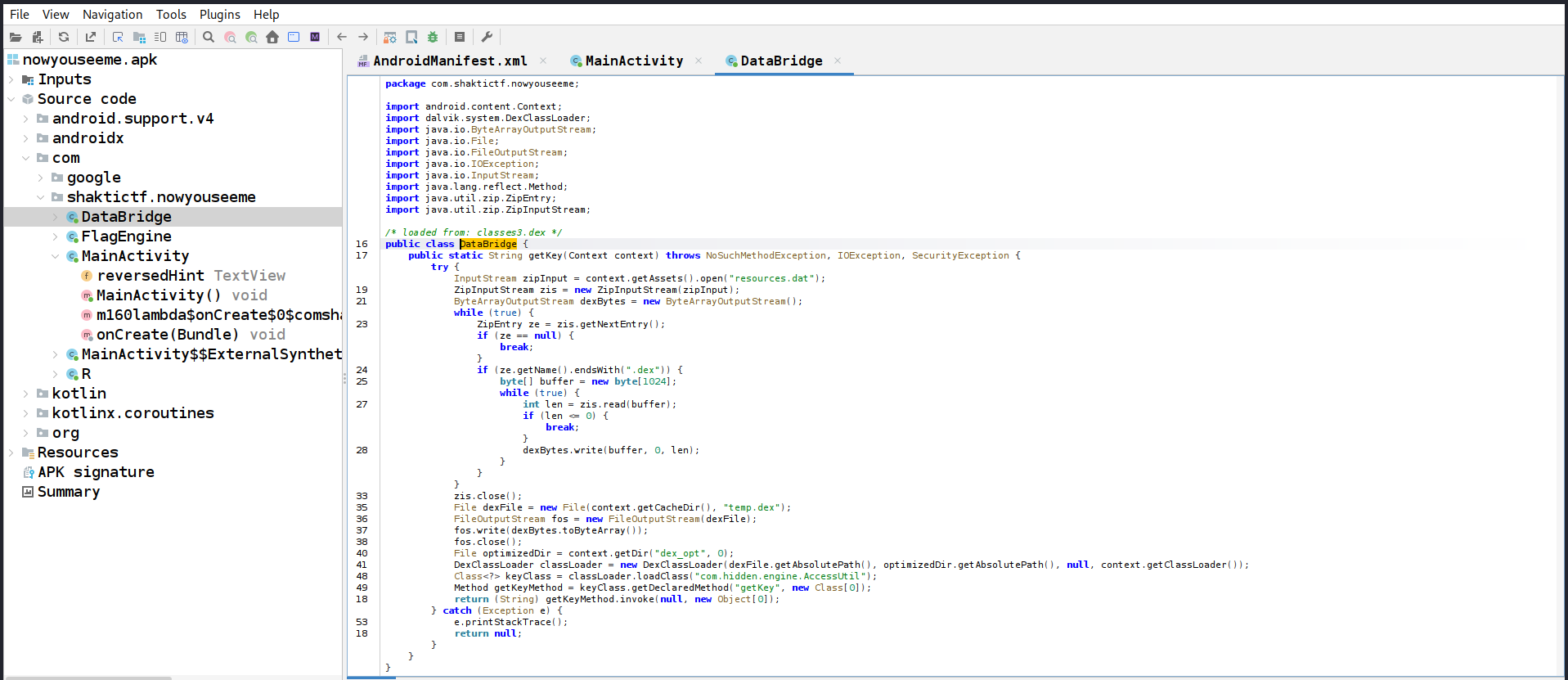
}

So:

* There’s a native .so file: libnowyouseeme.so.
* The JNI function getFlag returns the flag if you give it the **correct key**.

**4️. Where does the key come from?**

Look at:



package com.shaktictf.nowyouseeme;

import android.content.Context;

import dalvik.system.DexClassLoader;

public class DataBridge {

public static String getKey(Context context) {

try {

InputStream zipInput = context.getAssets().open("resources.dat");

ZipInputStream zis = new ZipInputStream(zipInput);

ByteArrayOutputStream dexBytes = new ByteArrayOutputStream();

while (true) {

ZipEntry ze = zis.getNextEntry();

if (ze == null) break;

if (ze.getName().endsWith(".dex")) {

byte[] buffer = new byte[1024];

int len;

while ((len = zis.read(buffer)) > 0) {

dexBytes.write(buffer, 0, len);

}

}

}

File dexFile = new File(context.getCacheDir(), "temp.dex");

FileOutputStream fos = new FileOutputStream(dexFile);

fos.write(dexBytes.toByteArray());

fos.close();

DexClassLoader classLoader = new DexClassLoader(

dexFile.getAbsolutePath(),

context.getDir("dex\_opt", 0).getAbsolutePath(),

null,

context.getClassLoader()

);

Class<?> keyClass = classLoader.loadClass("com.hidden.engine.AccessUtil");

Method getKeyMethod = keyClass.getDeclaredMethod("getKey");

return (String) getKeyMethod.invoke(null);

} catch (Exception e) {

e.printStackTrace();

return null;

}

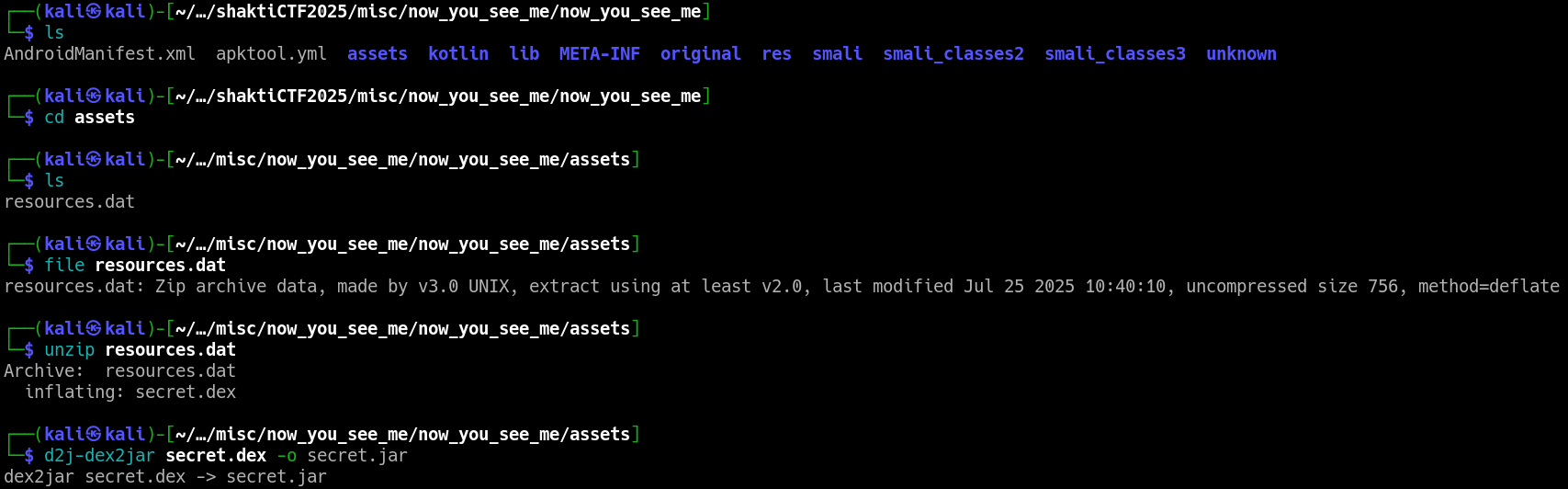
}

}

So:

* The key is hidden in a **secret.dex** inside resources.dat!
* The app unpacks resources.dat, extracts secret.dex, loads AccessUtil and calls getKey.

**5️. Unzip and inspect resources.dat**

****

Extract:

unzip -p resources.dat > secret.dex

**✅ 🔍 Here’s where we hit the jadx problem**

When trying to open secret.dex directly in jadx:

jadx-gui secret.dex

…it failed:

NullPointerException ...

**✅ 💡 Solution: Convert DEX to JAR**

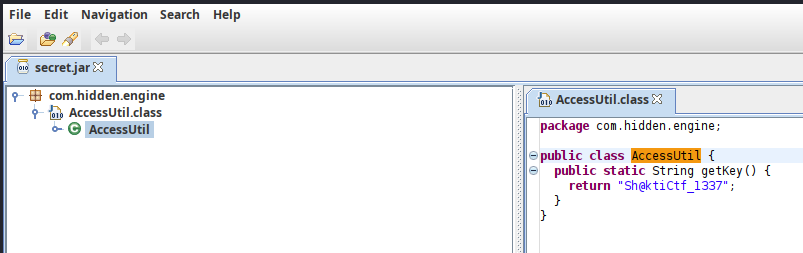
# Convert dex to jar

d2j-dex2jar secret.dex -o secret.jar

# Then open the jar

jd-gui secret.jar

Inside jd-gui, you see:



package com.hidden.engine;

public class AccessUtil {

public static String getKey() {

return "Sh@ktiCtf\_1337";

}

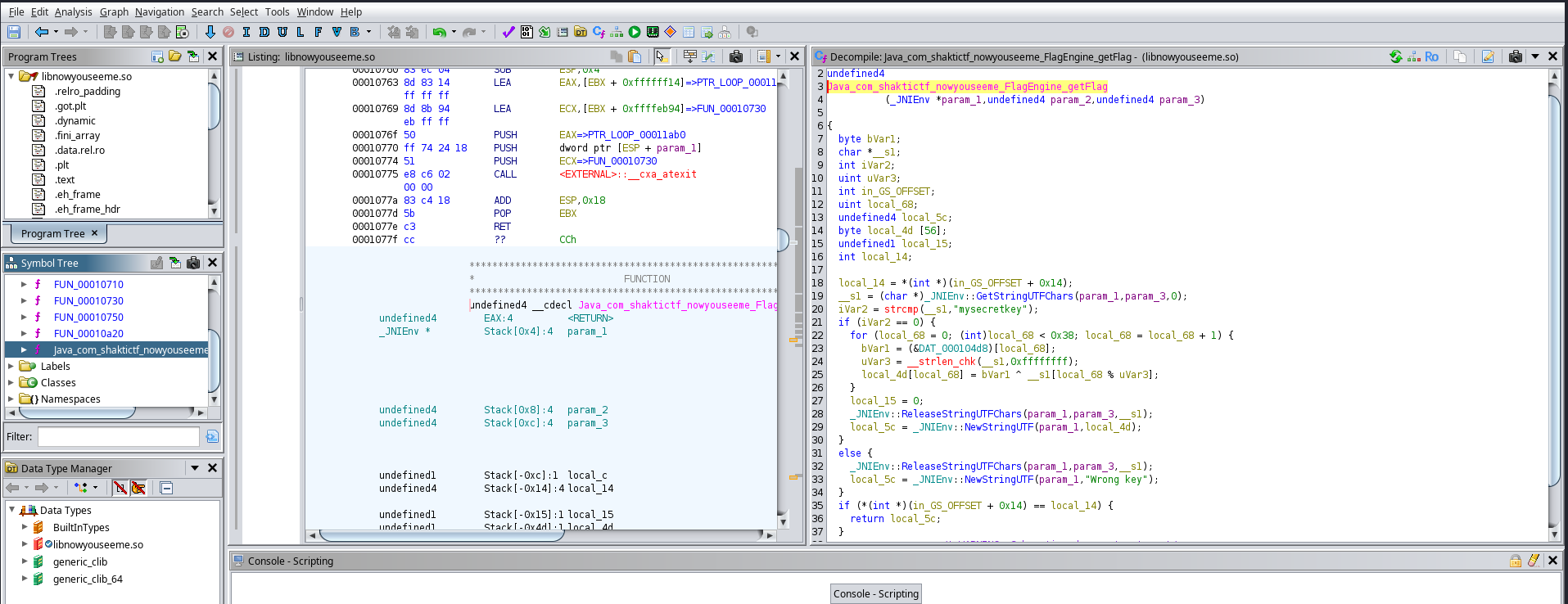
}

**Boom!** That’s the key.

**6️. What does the native lib do?**

Open libnowyouseeme.so in **Ghidra** or **radare2**:

Key disassembly:



Java\_com\_shaktictf\_nowyouseeme\_FlagEngine\_getFlag:

cmp input, "Sh@ktiCtf\_1337"

if wrong -> "Wrong key"

else:

flag[i] = T[i] XOR key[i mod keylen]

The secret flag is hidden in a **56-byte table** inside the .so.

**7️. Find the table**

Use radare2:

r2 -A libnowyouseeme.so

Search for the static data — you find:

Offset: 0x6B0

Length: 56 bytes (0x38)

**8️. Write a Python extractor**

**Final working code:**

# Team Name: Hidden Investigations

# Copyright Milon Das & @sakibulalikhan

# Solution of Now You See Me - ShaktiCTF2025

from pathlib import Path

import binascii

# Load the native .so

blob = Path("/home/sak/Downloads/now/resources/lib/x86\_64/libnowyouseeme.so").read\_bytes()

# Hardcoded offsets found via disassembly

TABLE\_OFF = 0x6B0

TABLE\_LEN = 0x38

KEY = b"Sh@ktiCtf\_1337"

# Extract table bytes

T = blob[TABLE\_OFF:TABLE\_OFF + TABLE\_LEN]

# XOR decrypt

flag = bytes(t ^ KEY[i % len(KEY)] for i, t in enumerate(T))

print("Raw bytes :", flag)

print("As string :", flag.decode('latin-1', 'replace'))

print("Hex dump :", binascii.hexlify(flag).decode())

**✅ ✅ ✅ Output**

$ python3 solve.py

Raw bytes : b'ShaktiCTF{m3m0ry\_1s\_wh3r3\_th3\_h34rt\_1s\_!!}'

As string : ShaktiCTF{m3m0ry\_1s\_wh3r3\_th3\_h34rt\_1s\_!!}

Hex dump : 5368616b74694354467b6d336d3072795f31735f77683372335f7468335f68333472745f31735f21217d

**✅ ✅ ✅ Final Flag**

**ShaktiCTF{m3m0ry\_1s\_wh3r3\_th3\_h34rt\_1s\_!!}**

**🎯 Key takeaway**

**The flag never appears in the app UI** — it lives only in **native memory**, assembled dynamically by JNI when given the secret key.

*“Some secrets don’t like the spotlight. They prefer memory.”*

You must:

* Unpack,
* Find the dynamic key,
* Reverse the native routine,
* XOR the bytes,
* Rebuild the flag!

**🏆 Clean summary**

* **APK clue:** MainActivity only misdirects with reversed hints.
* **Dex clue:** The real key is hidden in secret.dex → use dex2jar + jd-gui when jadx fails.
* **Native lib:** XOR loop with a hardcoded table → disassemble with Ghidra or radare2.
* **Result:** Reconstruct the flag statically.

**📚 Tools used**

* apktool — unpack APKs
* jadx / jd-gui — decompile Java
* d2j-dex2jar — convert .dex to .jar
* radare2 / ghidra — disassemble native .so
* python — to XOR and recover the flag

**👑 Solved by**

**Milon Das and @sakibulalikhan** — ShaktiCTF 2025