CTF Writeup: Flutter APK Reverse Engineering Challenge

This challenge provided a Flutter-based Android APK (`flutter_decode.apk`) along with a hint image. The task was to recover the hidden flag in the format `CRACCON{...}`. Since Flutter apps compile Dart code into native binaries (`libapp.so`), the flag was suspected to be embedded or obfuscated in one of the native libraries.

Tools Used: 1. apktool – to unpack the APK structure. 2. jadx-gui – to inspect Java/Kotlin code (though minimal for Flutter apps). 3. strings – to extract printable strings from native libraries. 4. Python – custom script to detect and decode base64-like strings.

Step 1 – APK Inspection:
Unpacked the APK with apktool and listed contents. Noticed Flutter-related libraries inside lib/arm64-v8a/ and lib/armeabi-v7a/, such as libapp.so. **Step 2 – Initial Search:**
Ran strings on libapp.so and searched for the keyword 'CRACCON'. No direct hit was found. **Step 3 – Detect Encoded Data:**
Scanned the binary for long base64-like strings using a Python script. Identified a suspicious encoded string: Y3JhY2NvbntDcjBzc19QbDQ3ZjBybVNfNHIzJ3RfcjNhbGxZX215X1RoaW5nfQ==
Step 4 – Decoding:
Base64-decoding this string revealed: craccon{Cr0ss_Pl47f0rmS_4r3't_r3allY_my_Thing} **Step 5 – Final Flag:**
Adjusting for flag format conventions, the valid flag is:
CRACCON{Cr0ss_Pl47f0rmS_4r3't_r3allY_my_Thing}

Python Extraction Script:
```
import base64, re

data = open("libapp.so", "rb").read()
candidates = re.findall(rb"[A-Za-z0-9+/=]{20,}", data)
for c in candidates:
    try:
        dec = base64.b64decode(c)
        if b"CRACCON" in dec.upper():
            print("Found:", dec.decode())
    except:
        pass
```

Conclusion:
The challenge demonstrates that while Flutter apps obscure much of the Dart source code, the final native libraries often contain embedded assets or encoded strings. By performing static analysis of libapp.so and decoding suspicious data, the hidden flag was successfully recovered.