

EZ C2 – BOOM

FLAG:

DF25{android_malware_case_is_just_beginning_and_this_is_second_turn_down_the_c2_and_we_have_the_flag}

Đầu tiên chúng ta có 1 file APK. Sử dụng máy thật làm thì ko chạy được app do minVersion là 31.

```
        android:versionCode="1"
        android:versionName="1.0"
        android:compileSdkVersion="33"
        android:compileSdkVersionCodename="13"
        package="com.example.pixelblackout"
        platformBuildVersionCode="33"
        platformBuildVersionName="13">
    <uses-sdk
        android:minSdkVersion="31"
        android:targetSdkVersion="33"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
```

Phải mượn điện thoại người em chạy thử thì Unsupported device với dòng Toast lên -> Cần review xem app đã làm gì vì Toast trong Android là code lập trình viên.

Phát hiện 2 vị trí sử dụng:

The screenshot shows the Android Studio search interface. The search term 'Unsupported device' is entered in the search bar. Below the search bar, there are two sections: 'Search definitions of:' and 'Search options:'. Under 'Search definitions of:', there are checkboxes for 'Class', 'Method', 'Field', 'Code', 'Resource', and 'Comments', with 'Code' being checked. Under 'Search options:', there are checkboxes for 'Case-insensitive' (checked), 'Regex' (unchecked), and 'Active tab only' (unchecked). The search results are displayed in a table with two columns: 'Node' and 'Text'. The 'Node' column shows the package and method name, and the 'Text' column shows the code with 'Unsupported device' highlighted in yellow. The results are:

Node	Text
com.example.pixelblackout.activities.MainActivity.D6668()	Toast.makeText(this, "Unsupported device", 1).show();
com.example.pixelblackout.activities.MainActivity.onCreate(Bundle)	Toast.makeText(this, "Unsupported device", 1).show();

Có vị trí check trong code java và cần bypass bằng Frida return false để chạy tiếp.

```
}

private final boolean D666b() {
    return DUUM6B6.f2539D.D66(this);
}

private final void D666b6() {
    if (!this.D666 && D6666bbb()) {
        this.D666 = true;
        DB37.D6666bBb(DB37.D6666(DUMBB.f2430D), new D666(null));
    }
}

@Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity
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.D66666(false);
        }
    }
    DataBinderMapperImpl dataBinderMapperImpl = DUM666BBB.f2342D;
    if (dataBinderMapperImpl != null) {
        dataBinderMapperImpl.onCreate(this);
    }
}
```

```
Sau khi bypass bằng Frida thì gặp D666B() và vẫn bị báo lỗi Unsupported device.  
class NativeCrypto {  
    static final void D666B() {  
        try {  
            NativeCrypto.f3858D.D6666();  
            DU66BB6 du66bb6 = new DU66BB6(this);  
            this.f2817D = du66bb6;  
            this.f2819D = new C0465f0(du66bb6);  
            D666BBB();  
        } catch (IllegalStateException unused) {  
            Toast.makeText(this, "Unsupported device", 1).show();  
            finish();  
        }  
    }  
}
```

Đi tiếp theo NativeCrypto phát hiện lib native với các hàm mã hóa.

```

public final class NativeCrypto {

    /* renamed from: D */
    public static final NativeCrypto f3858D = new NativeCrypto();

    /* renamed from: D */
    private static volatile boolean f3859D;

    static {
        System.loadLibrary("native-lib");
    }

    private NativeCrypto() {
    }

    private final void D666() {
        if (f3859D) {
            return;
        }
        synchronized (this) {
            if (f3859D) {
                return;
            }
            NativeCrypto nativeCrypto = f3858D;
            if (!nativeCrypto.nativeOkToRun()) {
                throw new IllegalStateException("Unsupported environment detected");
            }
            C0555D.D m828D6 = C0555D.f3852D.m828D6();
            nativeCrypto.nativeConfigure(55, m828D6.m830D6(), m828D6.m829D(), m828D6.D66());
            f3859D = true;
        }
    }

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

    private final native String nativeDecryptString(String str);

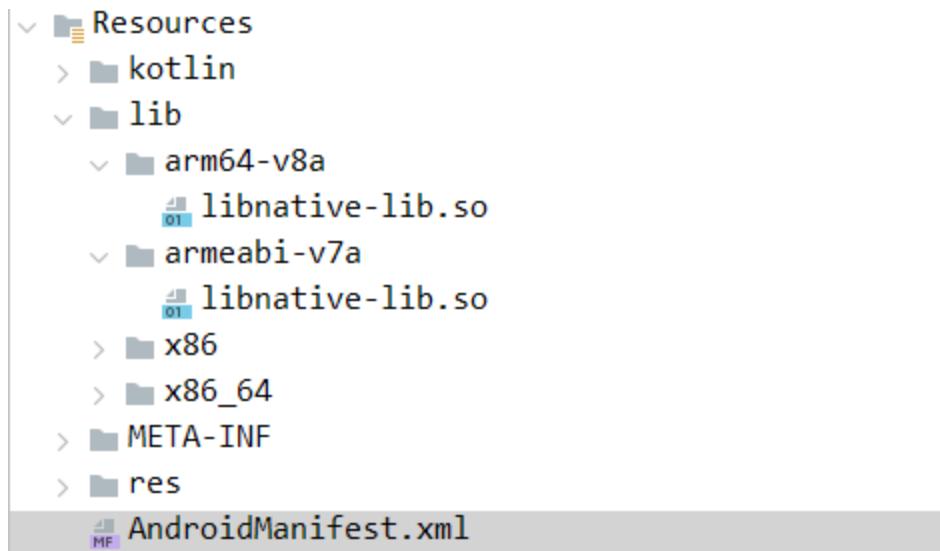
    private final native byte[] nativeEncryptArchive(byte[] bArr);

    ...
}

```

Bài lạ từng quen. Dạng bài sử dụng mã hóa trong native và call từ java xuống để làm khó ae hắc cơ đã từng được mình làm trước đây. Nên tiếp tục trace xem cơ chế mã hóa giải mã như nào. Nhưng do mình ko quá mạnh về reverse nên cần hỗ trợ từ AI.

Thực hiện reverse native-lib.so trong lib (BTC khá tốt bụng khi để x86-64 vì nhiều người chơi không có thiết bị thật minSDK 31)



Không có hàm native trong khi tìm kiếm Java -> Phần này có vẻ mới với mình. Trước đây có nghe tới ánh xạ trong lib native nhưng chưa gặp bao giờ. Nhờ AI hỗ trợ “nhẹ”.

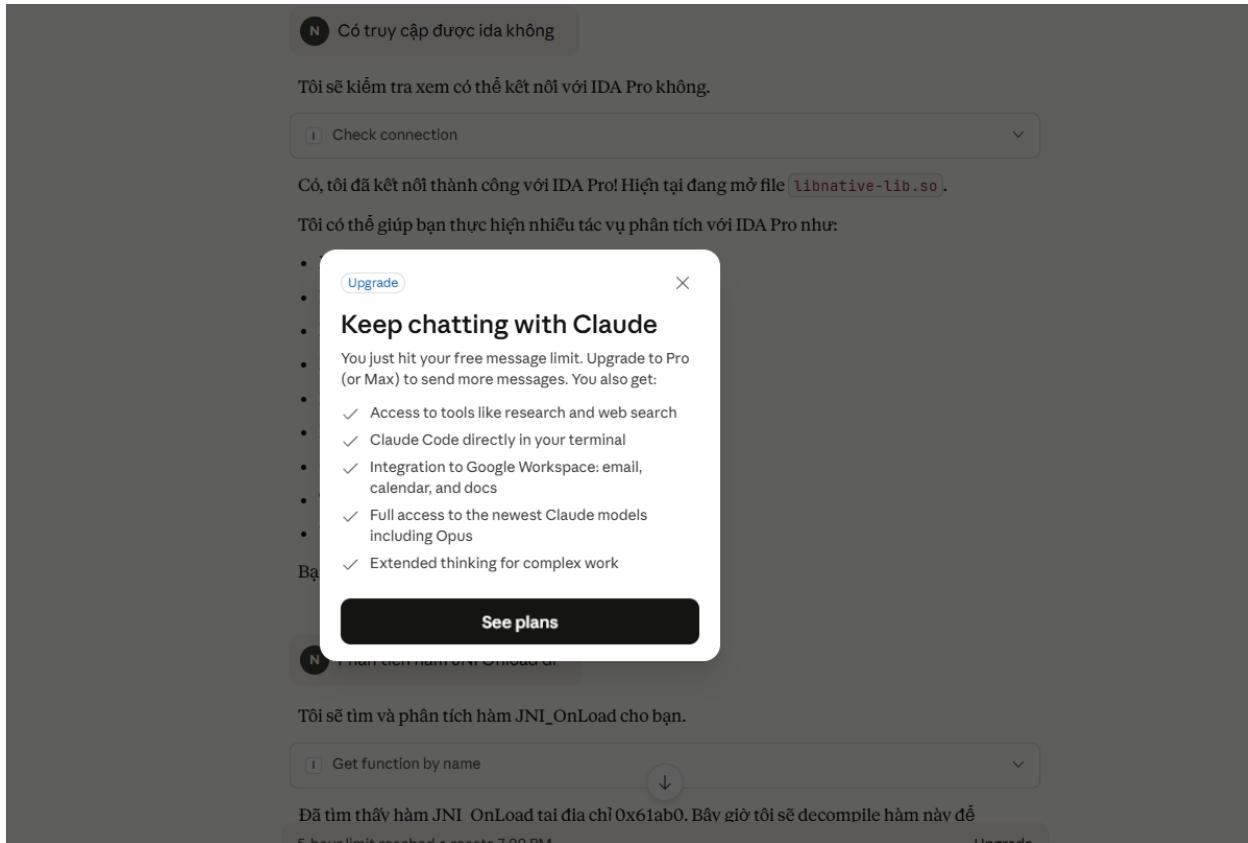
Về cơ bản khi chạy lib native sẽ load hàm JNI_Onload đầu tiên và ở đó có thấy 1 số đoạn check Frida.

```

v16 = 0;
v1 = -1;
if ( (*(unsigned int (__fastcall **)(__int64, __int64 *, __int64))(*(_QWORD *)a1 + 48LL))(a1, &v16, 65542) )
    return v1;
v19 = 28;
strcpy(v20, "/proc/net/unix");
qmemcpy(v22, "\nfrida", sizeof(v22));
sub_63840(v23, &v19, 8);
if ( !v25 )
{
    v7 = 0;
    goto LABEL_23;
}
v17 = 0;
s = 0;
LABEL_5:
while ( 1 )
{
    std::ios_base::getloc((std::ios_base *)v21);
    v3 = std::locale::use_facet((std::locale *)v21, (std::locale::id *)&std::ctype<char>::id);
    v4 = (*(__int64 (__fastcall **)(__int64, __int64))(*(_QWORD *)v3 + 56LL))(v3, 10);

    if ( (v19 & 1) != 0 )
        operator delete(*(void **)&v20[15]);
    if ( v7 )
        __android_log_print(5, "NativeCrypto", "Frida socket detected");
    byte_D5BA0 = !v7;
    v14 = (*(__int64 (__fastcall **)(__int64, const char *))(*(_QWORD *)v16 + 48LL))(
```

Bypass đoạn này không khó với AI. Khó là mình ko có AI Pro nên bị limit. Bảo nó phân tích toàn bộ dẫn đến bị limit khi đang làm và không phân tích toàn bộ được. Nhưng cũng đã giúp mình hiểu được phần nào hàm JNI_Onload.



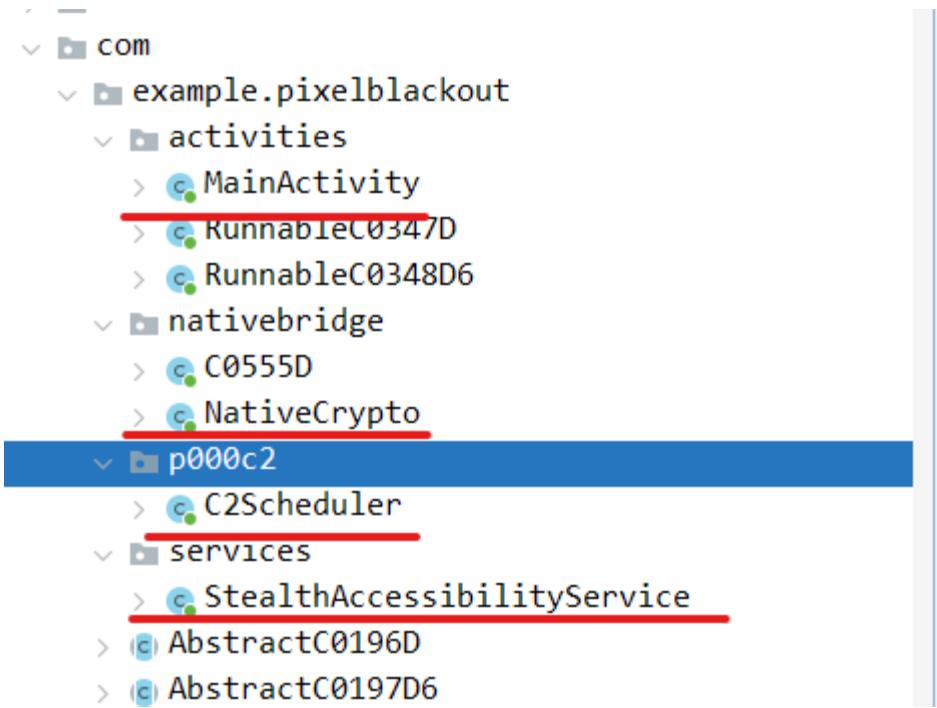
Hàm này check Frida và ánh xạ hàm java với func trong native.

```
.data.rel.ro:0000000000CD690
.data.rel.ro:0000000000CD690
.data.rel.ro:0000000000CD698
.data.rel.ro:0000000000CD6A0
.data.rel.ro:0000000000CD6A8
.data.rel.ro:0000000000CD6B0
.data.rel.ro:0000000000CD6B8
.data.rel.ro:0000000000CD6C0
.data.rel.ro:0000000000CD6C8
.data.rel.ro:0000000000CD6D0
.data.rel.ro:0000000000CD6D8
.data.rel.ro:0000000000CD6E0
.data.rel.ro:0000000000CD6E8
.data.rel.ro:0000000000CD6F0
.data.rel.ro:0000000000CD6F8
.data.rel.ro:0000000000CD700
; DATA XREF: JNI_OnLoad+291↑o
; "nativeEncryptString"
dq offset aJavaLangString_2 ; (Ljava/lang/String;)Ljava/lang/String;
dq offset sub_61E60
dq offset aNativeDecrypts ; "nativeDecryptString"
dq offset aJavaLangString_2 ; (Ljava/lang/String;)Ljava/lang/String;
dq offset sub_623A0
dq offset aNativeEncrypts ; "nativeEncryptArchive"
dq offset aBB ; "[B][B"
dq offset sub_62890
dq offset aNativeConfigure ; "nativeConfigure"
dq offset aIBBBV ; "(I[B[B[B)V"
dq offset sub_628D0
dq offset aNativeOkToRun ; "nativeOkToRun"
dq offset aZ ; "()Z"
dq offset sub_63200
```

Tiếp tục “húc húc húc”, nhưng vẫn không vào các hàm này, mình quyết định cần làm cẩn thận.

Trace ngược lại từ OnCreate và hiểu app làm gì.

Về cơ bản app có dạng 1 app giả mạo (khá trend). Với quyền yêu cầu truy cập Accessibility và C2 điều khiển



Trông khá giống app “thật” phần nào. Tiếp tục trace theo MainActivity có hàm gọi đến URL nào đó đã được mã hóa.

```

/* JADY INFO: Access modifiers changed from: private */
public static final D D66() {
    URLConnection openConnection = new URL(f3853D).openConnection();
    d628.D66(openConnection, "null cannot be cast to non-null type java.net.HttpURLConnection");
    HttpURLConnection httpURLConnection = (HttpURLConnection) openConnection;
    httpURLConnection.setConnectTimeout(5000);
    httpURLConnection.setReadTimeout(5000);
    try {
        int responseCode = httpURLConnection.getResponseCode();
        if (400 <= responseCode && responseCode < 600) {
            throw new IllegalStateException("Remote status " + responseCode);
        }
        InputStream inputStream = httpURLConnection.getInputStream();
        try {
            d628.D666(inputStream, "it");
            ByteArrayOutputStream byteArrayList = new ByteArrayOutputStream(Math.max(8192, inputStream.available()));
            DB37.D66666BBB(inputStream, byteArrayList);
            byte[] byteArray = byteArrayList.toByteArray();
            d628.D666(byteArray, "buffer.toByteArray()");
            DB37.D66666B(inputStream, null);
            return f3852D.D666(byteArray);
        } finally {
        }
    } finally {
        httpURLConnection.disconnect();
    }
}

```

Mình thử dùng Frida hook để bypass cert (Frida code share public) thì thấy có 1 link bypass cert lạ.

```

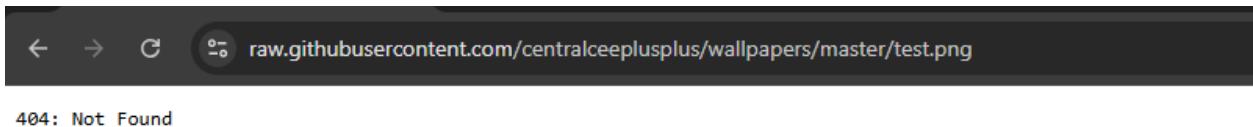
[JAVA] Forcing nativeOkToRun to return true
C0555D.m828D6 is called
[+] Bypassing TrustManagerImpl (Android > 7) checkTrustedRecursive check for: raw.githubusercontent.com
[TEST] Testing hooks by calling nativeOkToRun...

```

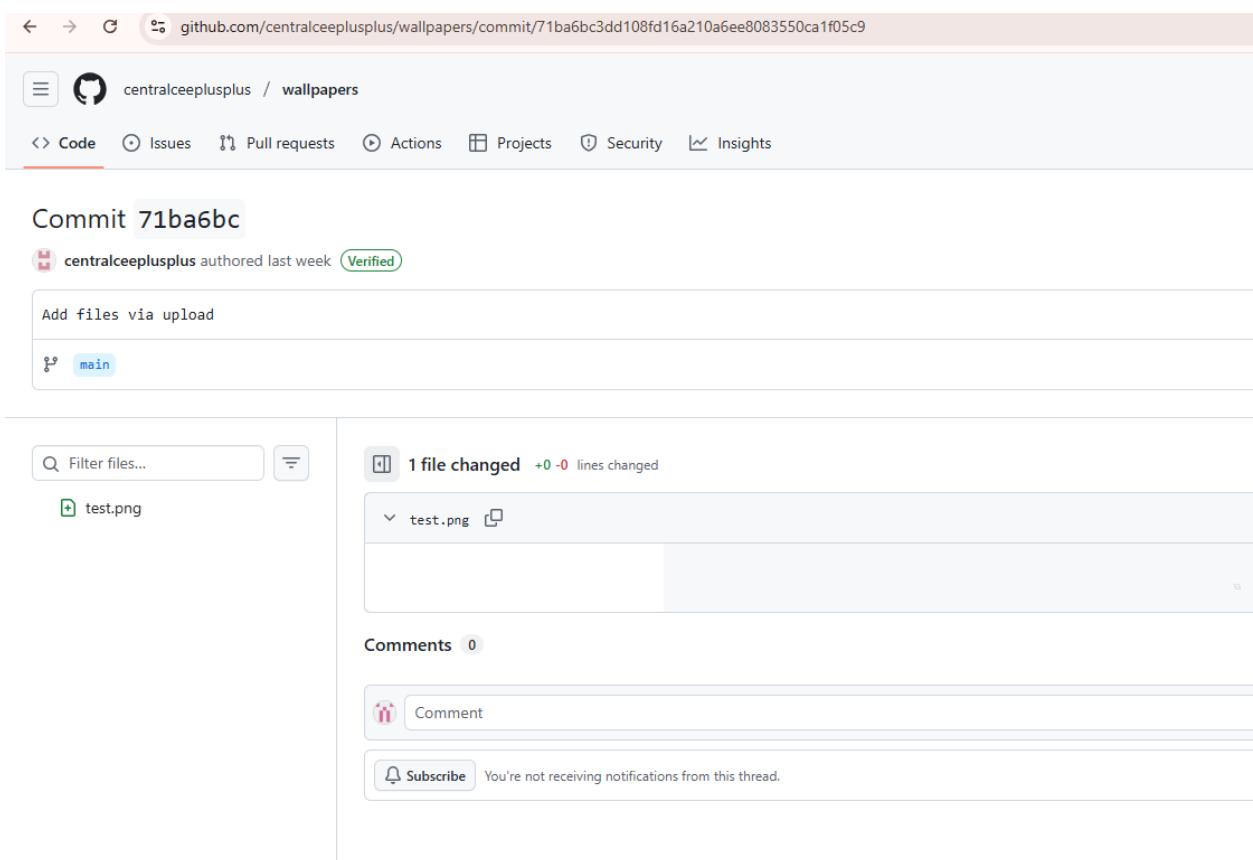
Có vẻ là link mà hàm trên đã gọi để connect. Tiếp tục theo dấu để lấy được toàn bộ link.

```
C0555D.m828D6 is called
[URL] new URL(spec): https://raw.githubusercontent.com/centralceplusplus/wallpapers/master/test.png
[URL]  params: (none)
[URL]  URL.openConnection -> https://raw.githubusercontent.com/centralceplusplus/wallpapers/master/test.png
[URL]  params: null=[]
```

Và 404 ... PNG ???



Đến đây sẽ có người bỗng dưng vì không hiểu gì. Nhưng đây là 1 link github. Từ github có thể trace ngược lại được repo của “ai đó” với khá nhiều commit và thấy được file cần tìm trong 1 commit



Để ý thì ở trên cũng app gọi đến 1 file là test.png -> khá trùng với tên file có trên git.

Tải file về và lấy được thông tin của khóa gì đó.

‰PNG

İHĐRμİDATxÜçüyöçö*İENDDB`
{"k":"BnYGTWCGUmcCZnBSU4FC3pnY1FjewJkelpBAHNeQVE=","i":"BwYFBAMCAQAPD1ZVVFSUQ==","p":"ZARUQkUEGnJPUV5bFgUHBQI="})

Trên git cũng có nhiều file liên quan đến dữ liệu mã hóa. Đề bài yêu cầu giải mã file nên đây có vẻ là file đã bị hacker mã hóa thông qua app C2 và cần chúng ta khôi phục lại.

OK LET'S GO

Với sự giúp đỡ của ChatGPT 5 Plus và Ctrl C- Ctrl V mình đã copy và hiểu được lib native được dùng như nào.

nativeConfigure là hàm được gọi mỗi lần thực hiện mã hóa giải mã:

```
private final void D666() {
    if (f3859D) {
        return;
    }
    synchronized (this) {
        if (f3859D) {
            return;
        }
        NativeCrypto nativeCrypto = f3858D;
        if (!nativeCrypto.nativeOkToRun()) {
            throw new IllegalStateException("Unsupported environment detected");
        }
        C0555D.D m828D6 = C0555D.f3852D.m828D6();
        nativeCrypto.nativeConfigure(55, m828D6.m830D6(), m828D6.m829D(), m828D6.D66());
        f3859D = true;
    }
}
```

Với 3 tham số truyền vào lấy được từ ảnh trên github:

{"k":"BnYGTWcGUmcCZnBSUV4Fc3pnY1FjewJke1pBAHNeQVE=","i":"BwYFBAMCAQAPDIZVVFNSUQ==","p":"ZARUQkUEGnJPUV5bFgUHBJl="}

Tiếp tục với AI. Thì mình báo nó giải mã với hàm native Configure code C++ mà mình đã gửi nó

Chuẩn rồi. Với lời gợi:

```
java
```

 Sao chép mã

```
nativeCrypto.nativeConfigure(55, k, i, p);
```

- `55` chính là **khóa XOR 1-byte** (`0x37`) mà hàm dùng để “giải mã”.
- `k, i, p` là 3 blob base64. Sau khi **base64 decode** rồi **XOR từng byte** với `0x37`, ta được nội dung thật và chúng được lưu vào 3 vector toàn cục, rồi bật cờ `byte_D5CE8 = 1`.

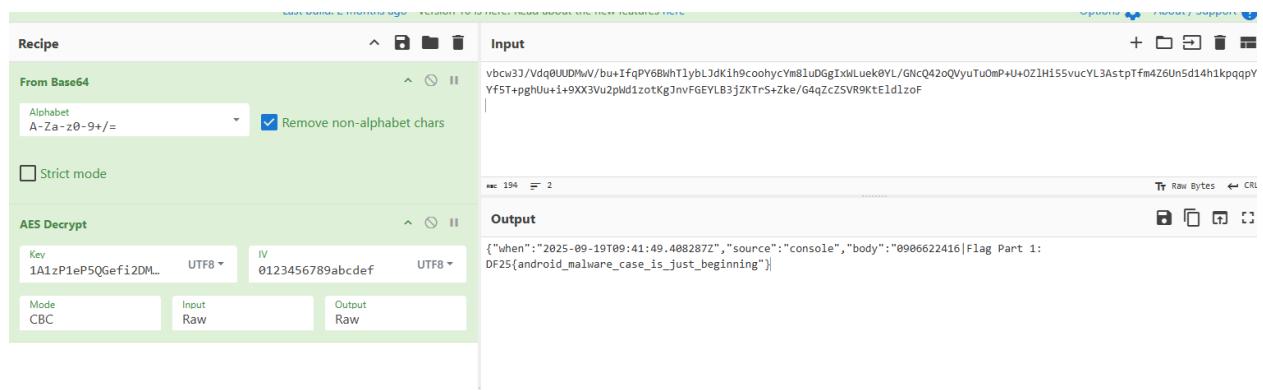
Mình đã giải ra 3 giá trị plaintext của bạn:

- `k` → `1A1zP1eP5QGefi2DMPTfTL5SLmv7Divf` (32 bytes — *khả năng là key*)
- `i` → `0123456789abcdef` (16 bytes — *chuẩn IV 128-bit*)
- `p` → `S3cur3-Exfil!2025` (16 bytes — *có thể là pass/pepper*)

=> Tức là hàm đang nạp 3 tham số (key/iv/pass) được “che” bằng XOR `0x37` và đặt vào biến toàn cục để các chỗ khác dùng.

Tiếp tục với hàm mã hóa và không có điều gì đặc biệt ở hàm mã hóa này vẫn là truyền thống AES/CBC.

Từ các file trên github và key đã có ở đây có thể giải mã được file của người dùng.



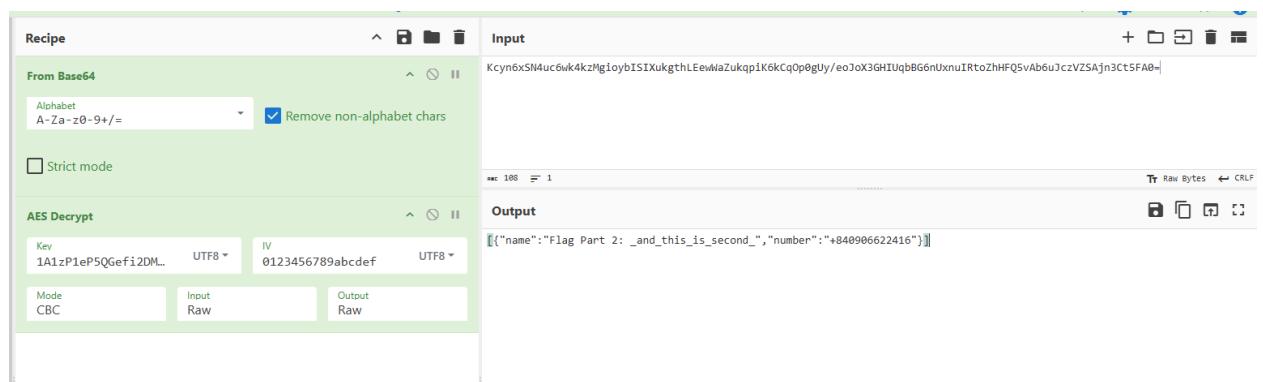
Input

```
vbcw3J/Vdq0UUDMwV/bu+IfqPY6BWhTlybLJdKih9coohycYm8luD6gIxwLuek@YL/GNcQ42oQVuTu0mP+U+O21H155vucYL3AstpTfm4Z6Un5d14h1kppqYyf5T+pghUu+i+9XX3Vu2pWd1zotKgJnvFGEYLB3jZKTrS+2ke/G4q2c25VR9KtEld1zoF
```

Output

```
{"when": "2025-09-19T09:41:49.408287Z", "source": "console", "body": "0906622416|Flag Part 1: DF25{android_malware_case_is_just_beginning}"}
```

First blood



Input

```
Kcyn6xSN4uc6wk4kzMgiobISIXukgthLEewMaZukqp1K6kCq0p0gUy/eoJoX3GHUqbBG6nUxnuIRtoZhhFQ5vAb6uJczV5Ajin3ct5FA0=
```

Output

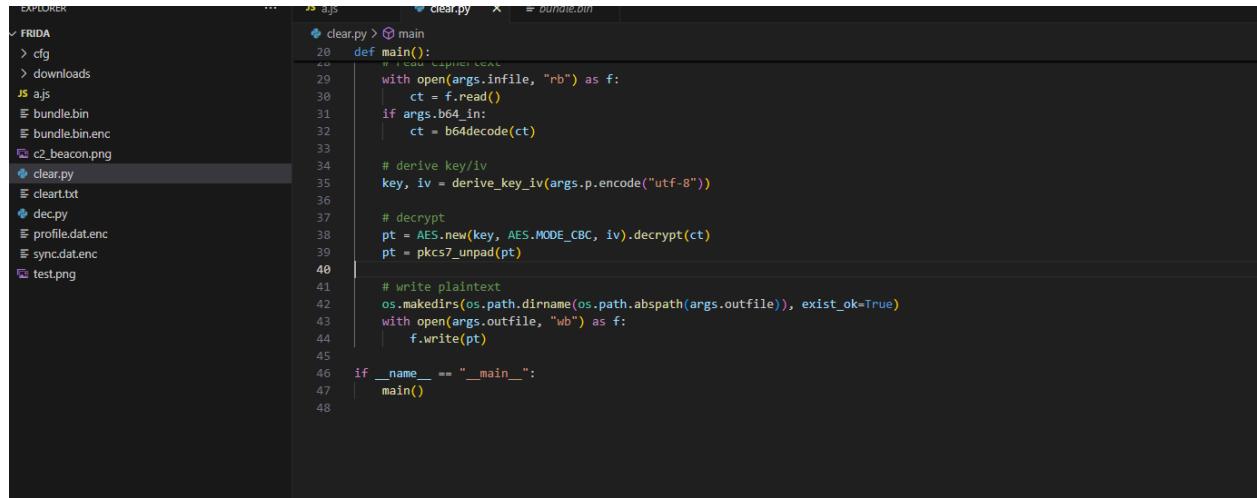
```
[{"name": "Flag Part 2: _and_this_is_second_", "number": "+840906622416"}]
```

Double kill

Triple kill???

Tất nhiên để ý kỹ trong native-lib vẫn còn 1 hàm mã hóa chưa dùng, trên git còn 1 file bundle.bin.enc chưa dùng đến 😊

Tiếp tục với Ulti AI. Chúng ta thấy được hàm mã hóa file zip cuối. Sử dụng AI gen luôn hàm giải mã và kết quả giải mã được file bin.



```
clear.py
1  #!/usr/bin/python
2
3  import os
4  import sys
5  import argparse
6
7  from Crypto.Cipher import AES
8
9  def main():
10    """Read cipher text
11    with open(args.infile, "rb") as f:
12      ct = f.read()
13    if args.b64_in:
14      ct = b64decode(ct)
15
16    # derive key/iv
17    key, iv = derive_key_iv(args.p.encode("utf-8"))
18
19    # decrypt
20    pt = AES.new(key, AES.MODE_CBC, iv).decrypt(ct)
21    pt = pkcs7_unpad(pt)
22
23    # write plaintext
24    os.makedirs(os.path.dirname(os.path.abspath(args.outfile)), exist_ok=True)
25    with open(args.outfile, "wb") as f:
26      f.write(pt)
27
28  if __name__ == "__main__":
29    main()
```

Tiếp tục tìm hiểu về file bundle.bin thì đây là file của Android và chỉ cần giải nén để lấy thông tin.

PK...3[cfg/profile.dat0N@13<<3)M6+Í6É@òMÍÍ~LòöŒ(ÍN/ÉðqM-Œ*Í.,Èô6Œv.ô/0<
vÌÈ3v.1us4°PK;uÛ@n1PK...3[downloads/flag.pngi½wP"[×7œD;"ŠtP@,ôN éŠ€"Ð{}]
¤Jo_Ðã} ÍóÍ|3Íùã{ç=Í8dïµ@µxú²÷^!ZKC...'Œ'P@*jär'. fÆs:æófÔCÕÐ zfý|z
¤3ivšžššiçòvT>cðMàíBB ~ ~ > ç>w@{~ðTTT~!~!^--: \.¥~\F:Ntð—nñf±.f»:vñl i

Và kết quả flag 3 nằm trong file đó:

