Unpacking a JsonPacker-packed sample

cryptax.medium.com/unpacking-a-jsonpacker-packed-sample-4038e12119f5

@cryptax June 27, 2022

@cryptax

Jun 27

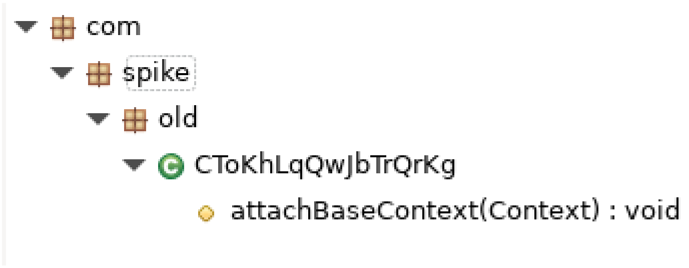
·

5 min read

With students of mine, we built a static unpacker (short of public name, I named it "JsonPacker" in APKiD). Unfortunately, the static unpacker doesn't work for the sample and the students are in for a quick patch before their defense in a few days. Lol. I'm sure they hate me 😁 The sha256 of the sample is `2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850`, and it dates back to February 2022.

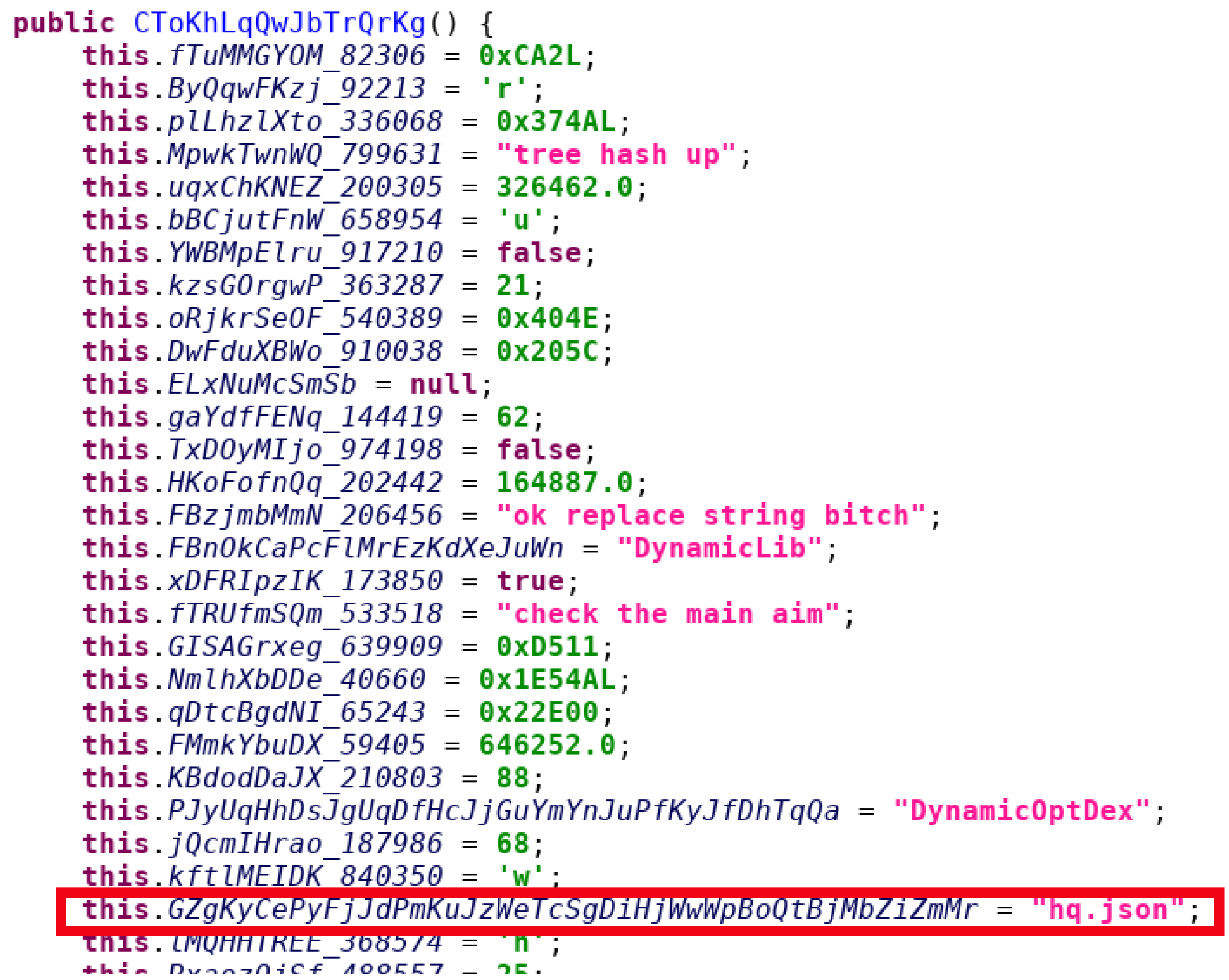
Quickly spot the encrypted json filename in the code

There are many classes, and obfuscated names, so at first it could be a bit disorientating to find the right spot. But I've unpacked such samples dozens of time: just search for the class with `attachBaseContext` (which is a method found in classes which derive from `Application` and which is called at "the very beginning").



```
▼ com
  ▼ spike
    ▼ old
      ▼ CToKhLqQwJbTrQrKg
          attachBaseContext(Context) : void
```

Head to class CToKhLqQwJbTrQrKg :)
In there, head to the object fields which get their initial value in the constructor. Spot the json file `hq.json`.

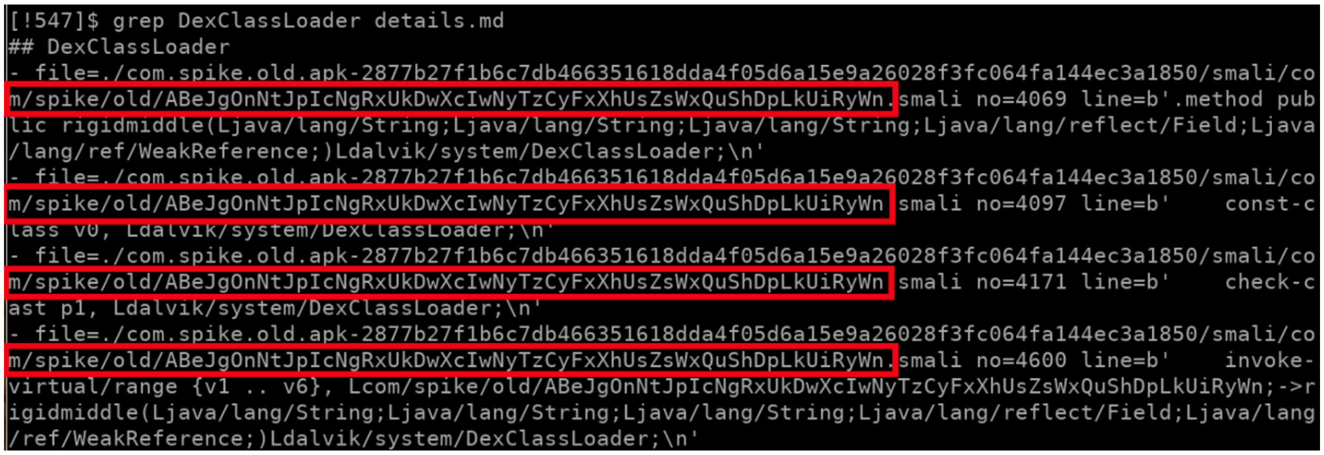


```
public CToKhLqQwJbTrQrKg() {
    this.fTuMMGYOM_82306 = 0xCA2L;
    this.ByQqwFKzj_92213 = 'r';
    this.plLhzlXto_336068 = 0x374AL;
    this.MpwkTwnWQ_799631 = "tree hash up";
    this.uqxChKNEZ_200305 = 326462.0;
    this.bBCjutFnW_658954 = 'u';
    this.YWBMpElru_917210 = false;
    this.kzsGOrgwP_363287 = 21;
    this.oRjkrSeOF_540389 = 0x404E;
    this.DwFduXBWo_910038 = 0x205C;
    this.ELxNuMcSmSb = null;
    this.gaYdfFENq_144419 = 62;
    this.TxDOyMIjo_974198 = false;
    this.HKoFofnQq_202442 = 164887.0;
    this.FBzjmbMmN_206456 = "ok replace string bitch";
    this.FBnOkCaPcFlMrEzKdXeJuWn = "DynamicLib";
    this.xDFRIpzIK_173850 = true;
    this.fTRUfmSQm_533518 = "check the main aim";
    this.GISAGrxeg_639909 = 0xD511;
    this.NmlhXbDDe_40660 = 0x1E54AL;
    this.qDtcBgdNI_65243 = 0x22E00;
    this.FMmkYbuDX_59405 = 646252.0;
    this.KBdodDaJX_210803 = 88;
    this.PJyUqHhDsJgUqDfHcJjGuYmYnJuPfKyJfDhTqQa = "DynamicOptDex";
    this.jQcmIHrao_187986 = 68;
    this.kftlMEIDK_840350 = 'w';
    this.GZgKyCePyFjJdPmKuJzWeTcSgDiHjWwWpBoQtBjMbZiZmMr = "hq.json";
    this.lMQHHTREE_368574 = 'h';
    this.Pxaaz0jSf_488557 = 25;
```

The encrypted payload is inside hq.json. I like to rename the field to something more meaningful :)

Spot the place where the file is dynamically loaded

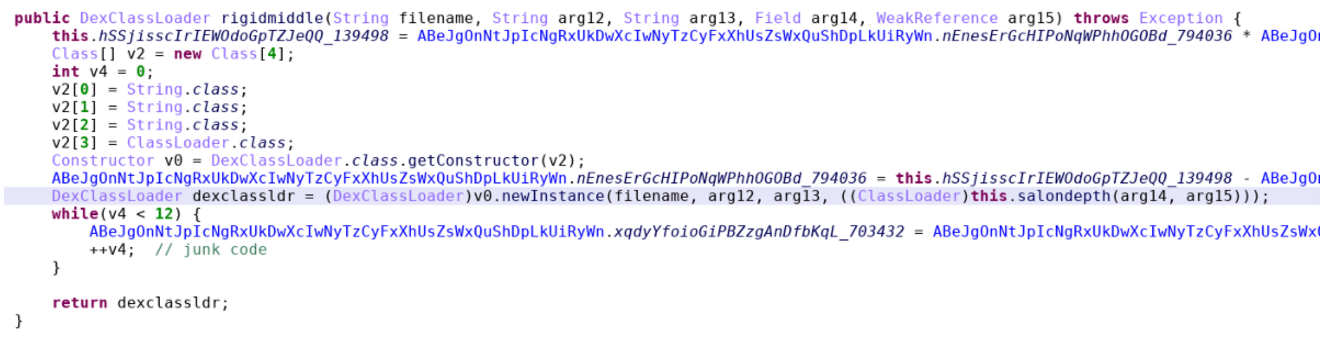
For such samples, just look where `DexClassLoader` is used. I like to use the detailed report of DroidLysis for that.



```
[!547]$ grep DexClassLoader details.md
## DexClassLoader
- file=./com.spike.old.apk-2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850/smali/co
m/spike/old/ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn.smali no=4069 line=b'.method pub
lic rigidmiddle(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/reflect/Field;Ljava
/lang/ref/WeakReference;)Ldalvik/system/DexClassLoader;\n'
- file=./com.spike.old.apk-2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850/smali/co
m/spike/old/ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn.smali no=4097 line=b'    const-c
lass v0, Ldalvik/system/DexClassLoader;\n'
- file=./com.spike.old.apk-2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850/smali/co
m/spike/old/ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn.smali no=4171 line=b'    check-c
ast p1, Ldalvik/system/DexClassLoader;\n'
- file=./com.spike.old.apk-2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850/smali/co
m/spike/old/ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn.smali no=4600 line=b'    invoke-
virtual/range {v1 .. v6}, Lcom/spike/old/ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn;->r
igidmiddle(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/reflect/Field;Ljava/lang
/ref/WeakReference;)Ldalvik/system/DexClassLoader;\n'
```

DexClassLoader is used in a single place: ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn

Go to that class, and search for `DexClassLoader` , you find method `rigidmiddle` .

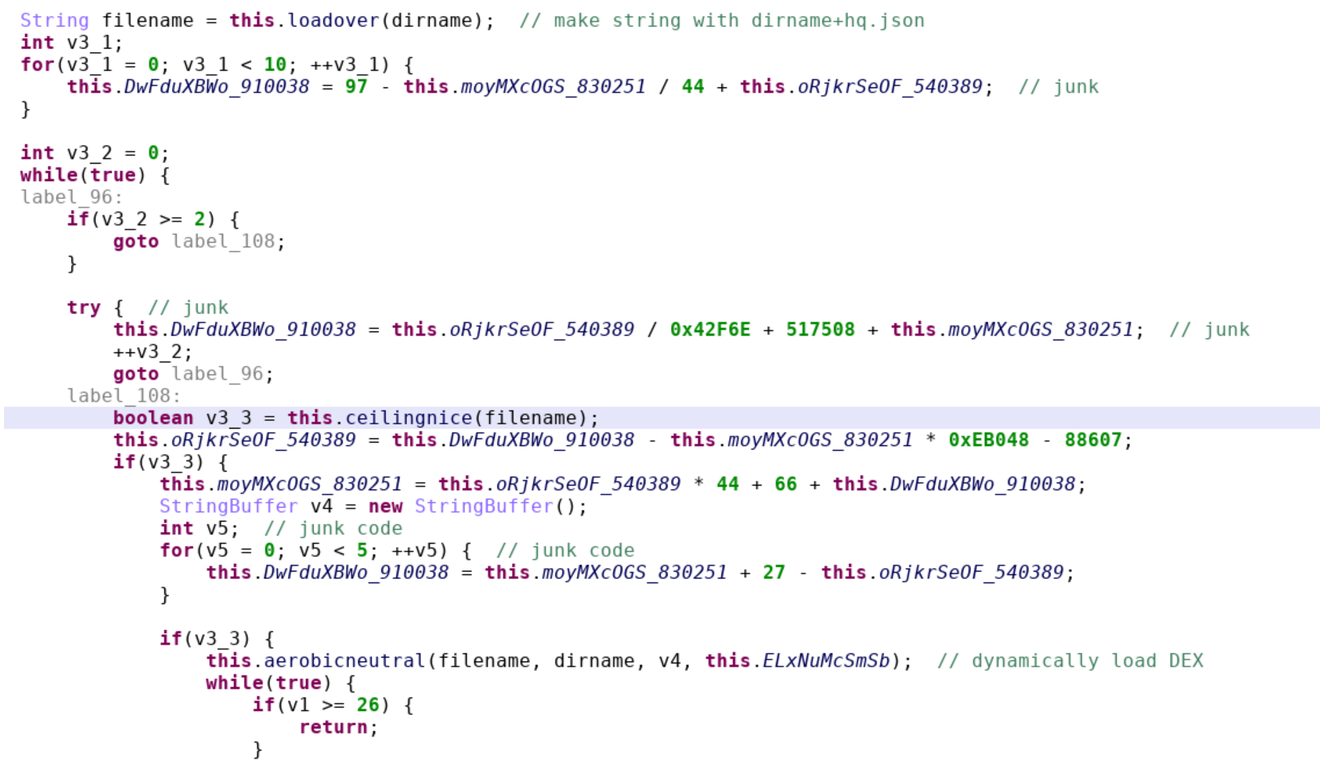


```
public DexClassLoader rigidmiddle(String filename, String arg12, String arg13, Field arg14, WeakReference arg15) throws Exception {
    this.hSSjisscIrIEWOdoGpTZJeQQ_139498 = ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn.nEnesErGcHIPoNqWPhhOGOBd_794036 * ABeJgO
    Class[] v2 = new Class[4];
    int v4 = 0;
    v2[0] = String.class;
    v2[1] = String.class;
    v2[2] = String.class;
    v2[3] = ClassLoader.class;
    Constructor v0 = DexClassLoader.class.getConstructor(v2);
    ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn.nEnesErGcHIPoNqWPhhOGOBd_794036 = this.hSSjisscIrIEWOdoGpTZJeQQ_139498 - ABeJgO
    DexClassLoader dexclassldr = (DexClassLoader)v0.newInstance(filename, arg12, arg13, ((ClassLoader)this.salondepth(arg14, arg15)));
    while(v4 < 12) {
        ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWxQuShDpLkUiRyWn.xqdyYfoioGiPBZzgAnDfbKqL_703432 = ABeJgOnNtJpIcNgRxUkDwXcIwNyTzCyFxXhUsZsWx
        ++v4;  // junk code
    }

    return dexclassldr;
}
```

This method loads dynamically the decrypted payload stored on the filesystem in “filename” Now, work your way up to who calls this method, using cross-references:

- `tailcreek`
- `guessextra`
- `aerobicneutral`
- `attachBaseContext` : the call to `aeronicneutral` is at the end of the image below.



```
String filename = this.loadover(dirname);  // make string with dirname+hq.json
int v3_1;
for(v3_1 = 0; v3_1 < 10; ++v3_1) {
    this.DwFduXBWo_910038 = 97 - this.moyMXcOGS_830251 / 44 + this.oRjkrSeOF_540389;  // junk
}

int v3_2 = 0;
while(true) {
label_96:
    if(v3_2 >= 2) {
        goto label_108;
    }

    try {  // junk
        this.DwFduXBWo_910038 = this.oRjkrSeOF_540389 / 0x42F6E + 517508 + this.moyMXcOGS_830251;  // junk
        ++v3_2;
        goto label_96;
    label_108:
        boolean v3_3 = this.ceilingnice(filename);
        this.oRjkrSeOF_540389 = this.DwFduXBWo_910038 - this.moyMXcOGS_830251 * 0xEB048 - 88607;
        if(v3_3) {
            this.moyMXcOGS_830251 = this.oRjkrSeOF_540389 * 44 + 66 + this.DwFduXBWo_910038;
            StringBuffer v4 = new StringBuffer();
            int v5;  // junk code
            for(v5 = 0; v5 < 5; ++v5) {  // junk code
                this.DwFduXBWo_910038 = this.moyMXcOGS_830251 + 27 - this.oRjkrSeOF_540389;
            }

            if(v3_3) {
                this.aerobicneutral(filename, dirname, v4, this.ELxNuMcSmSb);  // dynamically load DEX
                while(true) {
                    if(v1 >= 26) {
                        return;
                    }
```

Parts of code of attachBaseContext. There is lots of junk code. The payload filename is used 3 times in this screenshots: loadover, ceilingnice, aerobicneutral

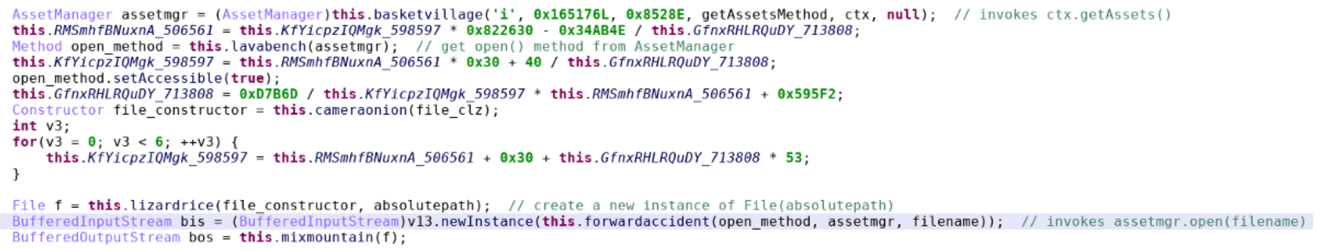
Spot where payload decryption occurs

- `loadover` constructs the absolute path of the filename.
- `ceilingnice` decrypts the file
- `aerobicneutral` loads the decrypted file.

In `ceilingnice`, let’s follow the calls to the decryption algorithm:

- `allrather`
- `orcharddecide`

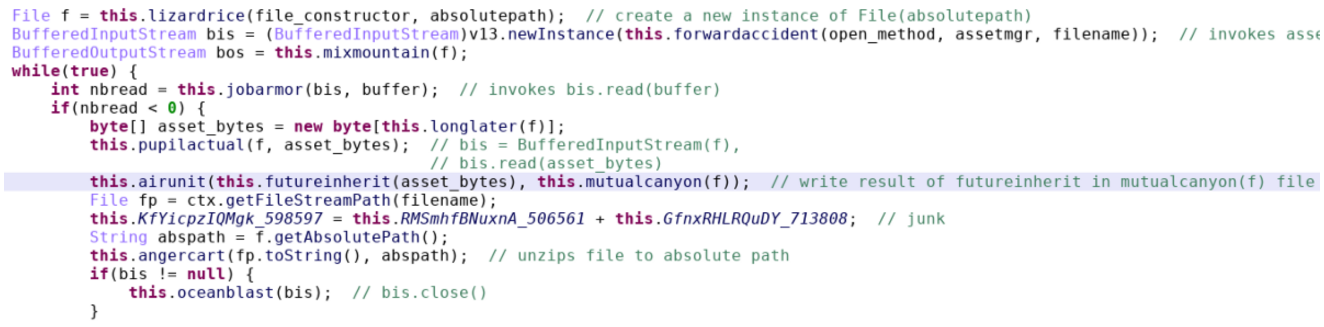
Method `orcharddecide` loads the asset:



```
AssetManager assetmgr = (AssetManager)this.basketvillage('i', 0x165176L, 0x8528E, getAssetsMethod, ctx, null);  // invokes ctx.getAssets()
this.RMSmhfBNuxnA_506561 = this.KfYicpzIQMgk_598597 * 0x822630 - 0x34AB4E / this.GfnxRHLRQuDY_713808;
Method open_method = this.lavabench(assetmgr);  // get open() method from AssetManager
this.KfYicpzIQMgk_598597 = this.RMSmhfBNuxnA_506561 * 0x30 + 40 / this.GfnxRHLRQuDY_713808;
open_method.setAccessible(true);
this.GfnxRHLRQuDY_713808 = 0xD7B6D / this.KfYicpzIQMgk_598597 * this.RMSmhfBNuxnA_506561 + 0x595F2;
Constructor file_constructor = this.cameraonion(file_clz);
int v3;
for(v3 = 0; v3 < 6; ++v3) {
    this.KfYicpzIQMgk_598597 = this.RMSmhfBNuxnA_506561 + 0x30 + this.GfnxRHLRQuDY_713808 * 53;
}

File f = this.lizardrice(file_constructor, absolutepath);  // create a new instance of File(absolutepath)
BufferedInputStream bis = (BufferedInputStream)v13.newInstance(this.forwardaccident(open_method, assetmgr, filename));  // invokes assetmgr.open(filename)
BufferedOutputStream bos = this.mixmountain(f);
```

First, the code retrieves an AssetManager. Then it opens the encrypted payload asset. The input stream is the encrypted payload, the output stream will be stored in the location of absolutepath
Then it reads the asset, decrypts it (this happens inside `futureinherit`), unzips the result and writes it to the output stream.

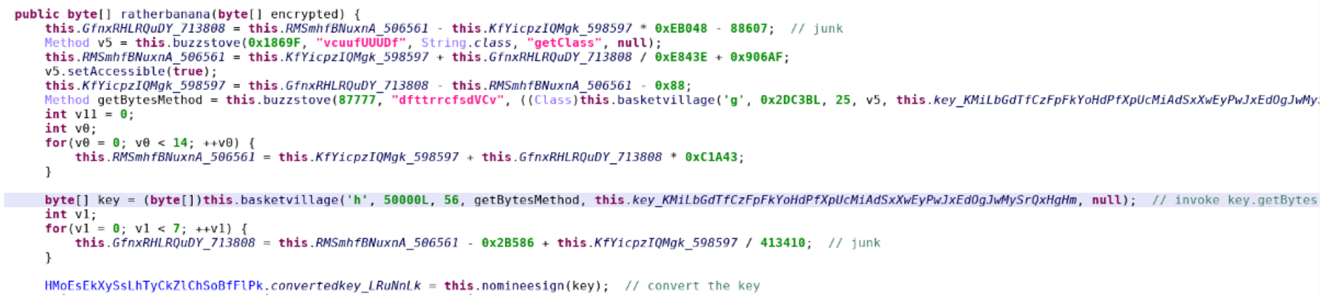


```
File f = this.lizardrice(file_constructor, absolutepath);  // create a new instance of File(absolutepath)
BufferedInputStream bis = (BufferedInputStream)v13.newInstance(this.forwardaccident(open_method, assetmgr, filename));  // invokes asse
BufferedOutputStream bos = this.mixmountain(f);
while(true) {
    int nbread = this.jobarmor(bis, buffer);  // invokes bis.read(buffer)
    if(nbread < 0) {
        byte[] asset_bytes = new byte[this.longlater(f)];
        this.pupilactual(f, asset_bytes);  // bis = BufferedInputStream(f),
                                           // bis.read(asset_bytes)
        this.airunit(this.futureinherit(asset_bytes), this.mutualcanyon(f));  // write result of futureinherit in mutualcanyon(f) file
        File fp = ctx.getFileStreamPath(filename);
        this.KfYicpzIQMgk_598597 = this.RMSmhfBNuxnA_506561 + this.GfnxRHLRQuDY_713808;  // junk
        String abspath = f.getAbsolutePath();
        this.angercart(fp.toString(), abspath);  // unzips file to absolute path
        if(bis != null) {
            this.oceanblast(bis);  // bis.close()
        }
```

This part of code (inside orcharddecide) decrypts the assets and unzips the result. `futureinherit` calls `ratherbanana`. It takes an encrypted byte array as input and returns a decrypted byte array.

Understanding the preparation of the key

`ratherbanana` reads a fixed string (“lanj” for this sample), and I assume it is the key. It converts the string to a byte array, then converts it to an integer array in `nomineesign`.



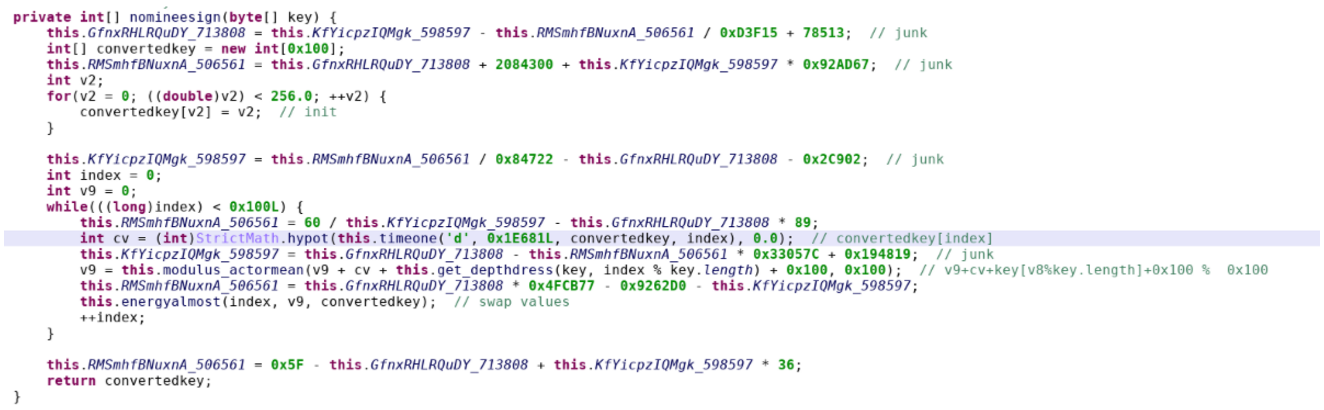
```
public byte[] ratherbanana(byte[] encrypted) {
    this.GfnxRHLRQuDY_713808 = this.RMSmhfBNuxnA_506561 - this.KfYicpzIQMgk_598597 * 0xEB048 - 88607;  // junk
    Method v5 = this.buzzstove(0x1869F, "vcuufUUUDf", String.class, "getClass", null);
    this.RMSmhfBNuxnA_506561 = this.KfYicpzIQMgk_598597 + this.GfnxRHLRQuDY_713808 / 0xE843E + 0x906AF;
    v5.setAccessible(true);
    this.KfYicpzIQMgk_598597 = this.GfnxRHLRQuDY_713808 - this.RMSmhfBNuxnA_506561 - 0x88;
    Method getBytesMethod = this.buzzstove(87777, "dfttrrcfsdVCv", ((Class)this.basketvillage('g', 0x2DC3BL, 25, v5, this.key_KMiLbGdTfCzFpFkYoHdPfXpUcMiAdSxXwEyPwJxEdOgJwMy
    int v11 = 0;
    int v0;
    for(v0 = 0; v0 < 14; ++v0) {
        this.RMSmhfBNuxnA_506561 = this.KfYicpzIQMgk_598597 + this.GfnxRHLRQuDY_713808 * 0xC1A43;
    }

    byte[] key = (byte[])this.basketvillage('h', 50000L, 56, getBytesMethod, this.key_KMiLbGdTfCzFpFkYoHdPfXpUcMiAdSxXwEyPwJxEdOgJwMySrQxHgHm, null);  // invoke key.getBytes
    int v1;
    for(v1 = 0; v1 < 7; ++v1) {
        this.GfnxRHLRQuDY_713808 = this.RMSmhfBNuxnA_506561 - 0x2B586 + this.KfYicpzIQMgk_598597 / 413410;  // junk
    }

    HMoEsEkXySsLhTyCkZlChSoBfFlPk.convertedkey_LRuNnLk = this.nomineesign(key);  // convert the key
```

Still lots of junk code. Prepare the decryption key.

The code of `nomineesign` is not very long but requires close attention to remove junk code (but not too much: the loop initializing the `convertedkey` table is *not* junk!), and de-obfuscate code.



```
private int[] nomineesign(byte[] key) {
    this.GfnxRHLRQuDY_713808 = this.KfYicpzIQMgk_598597 - this.RMSmhfBNuxnA_506561 / 0xD3F15 + 78513;  // junk
    int[] convertedkey = new int[0x100];
    this.RMSmhfBNuxnA_506561 = this.GfnxRHLRQuDY_713808 + 2084300 + this.KfYicpzIQMgk_598597 * 0x92AD67;  // junk
    int v2;
    for(v2 = 0; ((double)v2) < 256.0; ++v2) {
        convertedkey[v2] = v2;  // init
    }

    this.KfYicpzIQMgk_598597 = this.RMSmhfBNuxnA_506561 / 0x84722 - this.GfnxRHLRQuDY_713808 - 0x2C902;  // junk
    int index = 0;
    int v9 = 0;
    while(((long)index) < 0x100L) {
        this.RMSmhfBNuxnA_506561 = 60 / this.KfYicpzIQMgk_598597 - this.GfnxRHLRQuDY_713808 * 89;
        int cv = (int)StrictMath.hypot(this.timeone('d', 0x1E681L, convertedkey, index), 0.0);  // convertedkey[index]
        this.KfYicpzIQMgk_598597 = this.GfnxRHLRQuDY_713808 - this.RMSmhfBNuxnA_506561 * 0x33057C + 0x194819;  // junk
        v9 = this.modulus_actormean(v9 + cv + this.get_depthdress(key, index % key.length) + 0x100, 0x100);  // v9+cv+key[v8%key.length]+0x100 %  0x100
        this.RMSmhfBNuxnA_506561 = this.GfnxRHLRQuDY_713808 * 0x4FCB77 - 0x9262D0 - this.KfYicpzIQMgk_598597;
        this.energyalmost(index, v9, convertedkey);  // swap values
        ++index;
    }

    this.RMSmhfBNuxnA_506561 = 0x5F - this.GfnxRHLRQuDY_713808 + this.KfYicpzIQMgk_598597 * 36;
    return convertedkey;
}
```

The line with StrictMath.hypot is obfuscated.
For example, the lign with `hypot` is interesting:

```
int cv = (int)StrictMath.hypot(this.timeone('d', 0x1E681L, convertedkey, index),
0.0);
```

`timeone` actually returns the index-th byte of `convertedkey`, i.e `convertedkey[index]`.

`hypot` computes square root of (convertedkey[index]² + 0²). As 0² = 0, the variable `cv` will simply receive the value of `convertedkey[index]`.

In the end, the algorithm boils down to this:

```
private void swap(int a, int b, int[] array) {
        int tmp = array[a];
        array[a]=array[b];
        array[b]=tmp;
}

    private int[] convert_key(byte[] key) {        int[] convertedkey = new
int[0x100];        int i;        for(i = 0; i < 256; ++i) {
convertedkey[i] = i;  // init        }        int j = 0;        int k = 0;
while(j < 0x100) {            int cv = convertedkey[j];            k =
(k+cv+key[j%key.length]+0x100) %  0x100;            swap(j, k, convertedkey);  //
swap values            ++j;        }       return convertedkey;}
```

Decryption algorithm

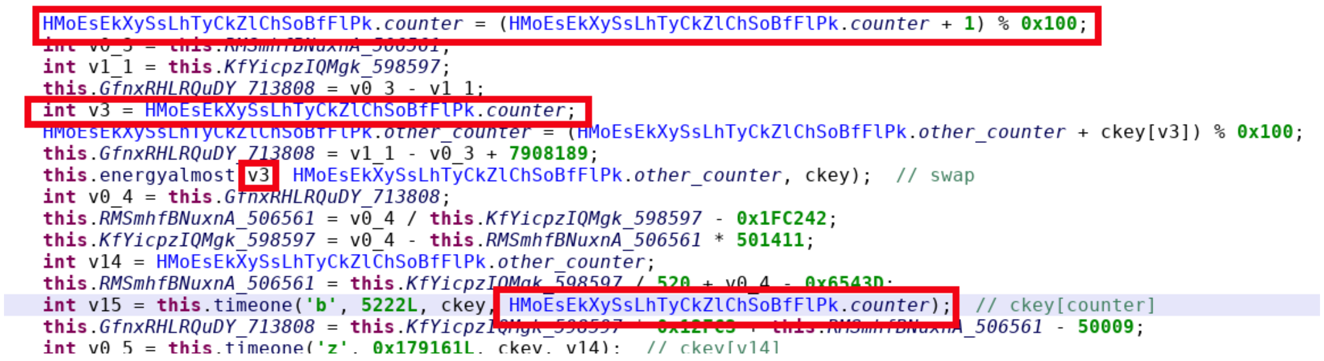
The next step is to understand the decryption algorithm in itself. Actually, there is lots of junk code that can be removed. To start, I focus on where the encrypted input byte array is used.

```
decrypted[i] = this.motionavoid(Math.round(v0_6) ^ encrypted[i]);
```

The method motionavoid is there just for obfuscation: it merely returns its argument. Also, obviously, we only have integers, so Math.round is useless. So, we have `decrypted[i] = v0_6 ^ encrypted[i];`. A few lines above, we have `v0_6`: `int v0_6 = ckey[(v15 + v0_5) % 0x100];`. A few lines above, we have `v15` and `v0_5`:

```
int v15 = this.timeone('b', 5222L, ckey, HMoEsEkXySsLhTyCkZlChSoBfFlPk.counter);  // ckey[counter]this.GfnxRHLRQuDY_713808 = this.KfYicpzIQMgk_598597 * 0x12FC3 + this.RMSmhfBNuxnA_506561 - 50009; // junkint v0_5 = this.timeone('z', 0x179161L, ckey, v14);  // ckey[v14]
```

The method `timeone` only uses the last two arguments: a table and an index, and returns `table[index]` value. Quite strangely, v15 uses a static integer that I have renamed `counter`. I search where this counter is used:



```
HMoEsEkXySsLhTyCkZlChSoBfFlPk.counter = (HMoEsEkXySsLhTyCkZlChSoBfFlPk.counter + 1) % 0x100;
int v0_3 = this.RMSmhfBNuxnA_506561;
int v1_1 = this.KfYicpzIQMgk_598597;
this.GfnxRHLRQuDY_713808 = v0_3 - v1_1;
int v3 = HMoEsEkXySsLhTyCkZlChSoBfFlPk.counter;
HMoEsEkXySsLhTyCkZlChSoBfFlPk.other_counter = (HMoEsEkXySsLhTyCkZlChSoBfFlPk.other_counter + ckey[v3]) % 0x100;
this.GfnxRHLRQuDY_713808 = v1_1 - v0_3 + 7908189;
this.energyalmost(v3, HMoEsEkXySsLhTyCkZlChSoBfFlPk.other_counter, ckey);  // swap
int v0_4 = this.GfnxRHLRQuDY_713808;
this.RMSmhfBNuxnA_506561 = v0_4 / this.KfYicpzIQMgk_598597 - 0x1FC242;
this.KfYicpzIQMgk_598597 = v0_4 - this.RMSmhfBNuxnA_506561 * 501411;
int v14 = HMoEsEkXySsLhTyCkZlChSoBfFlPk.other_counter;
this.RMSmhfBNuxnA_506561 = this.KfYicpzIQMgk_598597 / 520 + v0_4 - 0x6543D;
int v15 = this.timeone('b', 5222L, ckey, HMoEsEkXySsLhTyCkZlChSoBfFlPk.counter);  // ckey[counter]
this.GfnxRHLRQuDY_713808 = this.KfYicpzIQMgk_598597 * 0x12FC3 + this.RMSmhfBNuxnA_506561 - 50009;
int v0_5 = this.timeone('z', 0x179161L, ckey, v14);  // ckey[v14]
```

The first use basically increments the counter, making sure it remains below 0x100. Then, counter is put in v2 and swapped with another value (energyalmost is a method that performs byte swap). Finally, v15 gets the value of the ckey[counter]
I work out that `int v15 = ckey[counter];`.

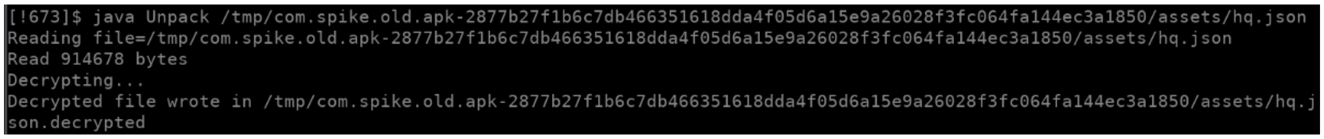
As for `v0_5`, we have `v0_5 = ckey[v14]` and `v14` is yet another static counter: `int v14 = HMoEsEkXySsLhTyCkZlChSoBfFlPk.other_counter;`. Same, I search where this other counter is used, and it's basically the same: an increment modulus 0x100, a swap and `ckey[other_counter]`. That's it! We have all elements to decrypt! The algorithm boils down to this:

```
public byte [] decrypt(byte [] encrypted, int len) {
    byte [] key = this.key.getBytes();
    int [] ckey = convert_key(key);
    byte [] decrypted = new byte[len];
    int i;
    for (i=0; i<len; i++) {
        counter = (counter+1) % 0x100;
        other_counter = (other_counter + ckey[counter]) % 0x100;
        swap(counter, other_counter, ckey);
        decrypted[i] = (byte) (ckey[(ckey[counter] + ckey[other_counter]) % 0x100] ^ encrypted[i]);
    }
    return decrypted;
}
```

Simplified decryption method. For this sample, the initial key is “lanj”. The encrypted byte array is the contents of hq.json. I added a length argument because actually in my code the hq.json is read into a bigger array, and we only need to decrypt up to the length of hq.json file.
Note: the code above uses static variables counter and other_counter, but actually it works fine with local variables, and probably would be easier to read with local ones.

Decrypt the payload

To the key + decryption algorithm, we just need to add something to read `hq.json` and write to another file. Then, we can unpack!

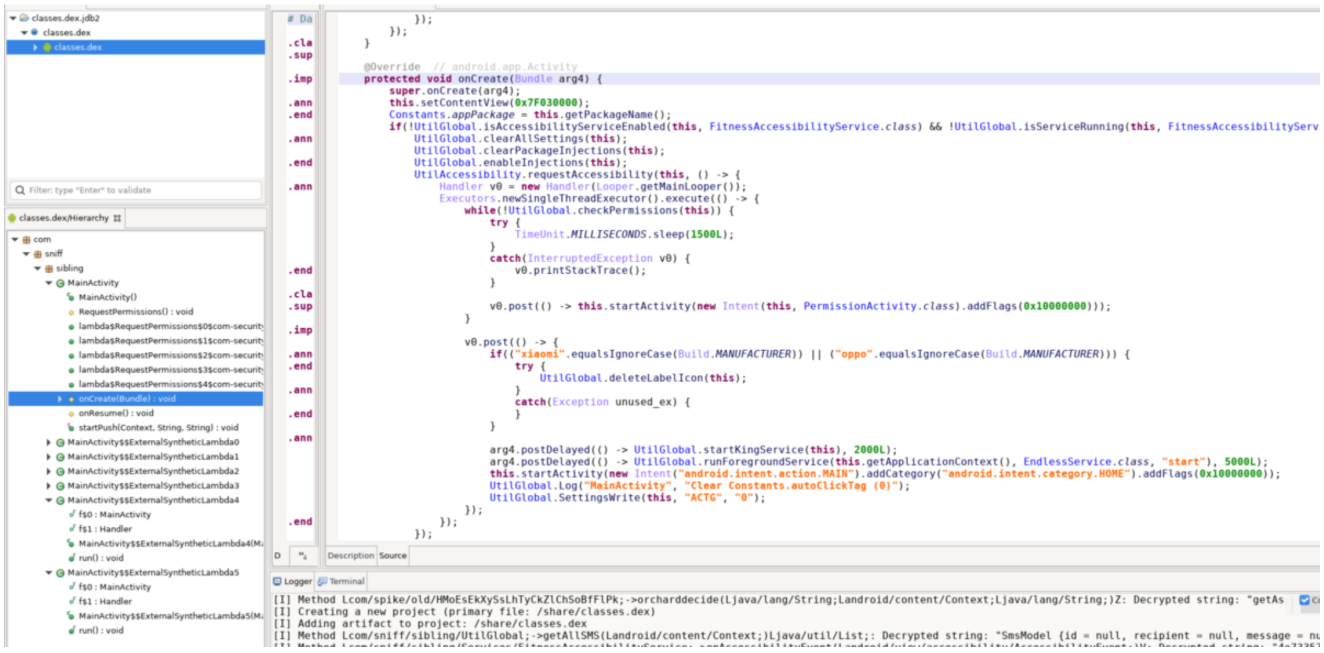


```
[!673]$ java Unpack /tmp/com.spike.old.apk-2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850/assets/hq.json
Reading file=/tmp/com.spike.old.apk-2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850/assets/hq.json
Read 914678 bytes
Decrypting...
Decrypted file wrote in /tmp/com.spike.old.apk-2877b27f1b6c7db466351618dda4f05d6a15e9a26028f3fc064fa144ec3a1850/assets/hq.j
son.decrypted
```

Static unpacker works fine :) Hurray!
The decrypted file is a Zip file (this was expected: remember that `orcharddecide` unzips the result): inside, there is a `classes.dex` (sha256: `dae52bbee7f709fae9d91e06229c35b46d4559677f26152d4327fc1601d181be`). It is the payload of the **Xenomorph malware**.

Which class/method does the malware load dynamically?

Before we decompile this payload, we need to know which method is called. The manifest shows the main activity is `com.sniff.sibling.MainActivity`. This class is not present in the wrapping apk… so it must be in the payload! This will be automatically called by Android as soon as it’s time to launch the main activity.


classes.dex.jdb2
classes.dex
classes.dex
Filter: type "Enter" to validate
classes.dex/Hierarchy
com
sniff
sibling
MainActivity
MainActivity()
RequestPermissions() : void
lambda$RequestPermissions$0$com-security
lambda$RequestPermissions$1$com-security
lambda$RequestPermissions$2$com-security
lambda$RequestPermissions$3$com-security
lambda$RequestPermissions$4$com-security
onCreate(Bundle) : void
onResume() : void
startPush(Context, String, String) : void
MainActivity$$ExternalSyntheticLambda0
MainActivity$$ExternalSyntheticLambda1
MainActivity$$ExternalSyntheticLambda2
MainActivity$$ExternalSyntheticLambda3
MainActivity$$ExternalSyntheticLambda4
f$0 : MainActivity
f$1 : Handler
MainActivity$$ExternalSyntheticLambda4(M
run() : void
MainActivity$$ExternalSyntheticLambda5
f$0 : MainActivity
f$1 : Handler
MainActivity$$ExternalSyntheticLambda5(M
run() : void
Description
Source
Logger
Terminal

```
                });
            });
        }

        @Override  // android.app.Activity
        protected void onCreate(Bundle arg4) {
            super.onCreate(arg4);
            this.setContentView(0x7F030000);
            Constants.appPackage = this.getPackageName();
            if(!UtilGlobal.isAccessibilityServiceEnabled(this, FitnessAccessibilityService.class) && !UtilGlobal.isServiceRunning(this, FitnessAccessibilityServ
                UtilGlobal.clearAllSettings(this);
                UtilGlobal.clearPackageInjections(this);
                UtilGlobal.enableInjections(this);
                UtilAccessibility.requestAccessibility(this, () -> {
                    Handler v0 = new Handler(Looper.getMainLooper());
                    Executors.newSingleThreadExecutor().execute(() -> {
                        while(!UtilGlobal.checkPermissions(this)) {
                            try {
                                TimeUnit.MILLISECONDS.sleep(1500L);
                            }
                            catch(InterruptedException v0) {
                                v0.printStackTrace();
                            }

                            v0.post(() -> this.startActivity(new Intent(this, PermissionActivity.class).addFlags(0x10000000)));
                        }

                        v0.post(() -> {
                            if(("xiaomi".equalsIgnoreCase(Build.MANUFACTURER)) || ("oppo".equalsIgnoreCase(Build.MANUFACTURER))) {
                                try {
                                    UtilGlobal.deleteLabelIcon(this);
                                }
                                catch(Exception unused_ex) {
                                }
                            }

                            arg4.postDelayed(() -> UtilGlobal.startKingService(this), 2000L);
                            arg4.postDelayed(() -> UtilGlobal.runForegroundService(this.getApplicationContext(), EndlessService.class, "start"), 5000L);
                            this.startActivity(new Intent("android.intent.action.MAIN").addCategory("android.intent.category.HOME").addFlags(0x10000000));
                            UtilGlobal.Log("MainActivity", "Clear Constants.autoClickTag (0)");
                            UtilGlobal.SettingsWrite(this, "ACTG", "0");
                        });
                    });
                });
```

```
[I] Method Lcom/spike/old/HMoEsEkXySsLhTyCkZlChSoBfFlPk;->orcharddecide(Ljava/lang/String;Landroid/content/Context;Ljava/lang/String;)Z: Decrypted string: "getAs
[I] Creating a new project (primary file: /share/classes.dex)
[I] Adding artifact to project: /share/classes.dex
[I] Method Lcom/sniff/sibling/UtilGlobal;->getAllSMS(Landroid/content/Context;)Ljava/util/List;: Decrypted string: "SmsModel {id = null, recipient = null, message = nu
```

The main activity is indeed found in the payload.

We’ve had enough for a single blog post, but the payload, similarly to many Android botnets, uses the Accessibility Services API to overlay windows of given applications.

— Cryptax