14) loadme

There was no text for this challenge on Moodle.

Solution

We are given only the apk file this time around. We decompile the apk with jadx via command

*jadx -d out loadme.apk*

In the usual output folders, there is more stuff to check, composed of two folders: *check* and *loadme*. Let’s inspect those one by one.

Starting from the *check* folder, the only interesting thing appears from the *Check.java* file itself, which checks for a string parameter when called and if called correctly returns the flag, which appears in plain text.

Inside the *loadme* folder, there is a *DoStuff* class which is the usual “random mess” kind of class. Specifically, we can see there is some Base64 involved, also it sets the *strict mode*.

A strict mode is a developer tool that helps to identify usage of disk and network operation on the main thread, on which app is interacting with a user. It will help to prevent accidental usage of disk and network tasks on the main thread.

Apart from a vector initialized and other functions which put Base64 strings at random, there is a *df* function, which takes a URL, opens a connection and if connection was opened OK (HTTP 200 code), checks if header is in the right format, taking a file in input and then seeting the save file path. Seems like this function only checks for a file of some sort, not interesting.

The *lc* function loads from the absolute path a *dex* file, which probably is the file called from the previous function and then via reflecion calls their own methods when loaded correctly. The *start* function instead gets the absolute path and context and then sets the strict mode for the treads called.

We can understand the .*dex* file gets downloaded via the right URL and then this way we correctly retrieve the flag.

The “good stuff” happens inside the *ds* function, which gets a cyphertext in Base64, decodes it, considers the vector and splits the key in parts as an AES encoding or similar ones and returns the string decoded.

In order to solve it, we craft our solution using also *ds* as a function:

Specifically, the first thing coming to our eyes is, from the *start* method, this one:

String path = df(gu(), ctx.getCodeCacheDir().getAbsolutePath());

The "gu()" function invokes another utility function with an encrypted string as its parameter. This is the string we want to reverse.

Also, this line:

SecretKeySpec skeySpec = new SecretKeySpec((parts[1] + parts[0] + "key!").getBytes("UTF-8"), "AES");

Specifies we are combining “mobiotsec” and “com” with “key!”, resulting in:

SecretKeySpec skeySpec = new SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"), "AES");

We can substitute that inside our code in *ds* function. Basically, the ds function returns a decrypted URL in combo with the “gu()” one, downloadcing a .dex file which invokes the methods, returning a temporary file.

The file gets downloaded from the path: https://www.math.unipd.it/~elosiouk/test.dex

Starting from the downloaded file, let’s decrypt it, even with jadx and let’s see what we get. Here, there is in particular the *LoadImage* class which is particularly interesting, which basically sets the decrypt string explictly and other methods which, when decrypted, allow to understand where the class is generated from, from which method, assets and codename.

Here we see the load and loadclass, which look both for files and try to find them, invoking their method when considering their absolute path.

When executing those methods, one can see that the Check class, which contains the folder in clear is called, then loading an asset as a .png file. Let’s look inside the *assets* folder then, which in reality is not even a .png, but actually a .dex file. Let’s inspect this one with jadx too.

Infact, looking inside the file, we see a simple boolean check which conducts us towards the flag correctly, so: FLAG{memores\_acti\_prudentes\_futuri}

A possible solver can be:

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.HttpURLConnection;

import java.net.URL;

import java.util.Base64;

import java.util.regex.Pattern;

import javax.crypto.Cipher;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

class Solver {

// The IV is the same for all the links

private static byte[] initVector = {-34, -83, -66, -17, -34, -83, -66, -17, -34, -83, -66, -17, -34, -83, -66, -17};

// Method which downloads the file from the URL and saves it to the disk from encrypted link

// gu stands for get url

private String gu() {

return ds("Bj9yLW24l0OpvkoxoPXLb+UqJGp1t1slVcl/aTlHM+iolk4i083NV8E1LNJj/6w1");

}

// ds stands for decrypt string

private String ds(String enc) {

try {

byte[] ciphertext = Base64.getDecoder().decode(enc.getBytes()); // Decoding the link

IvParameterSpec iv = new IvParameterSpec(initVector);

// Combining this from the package name and the class name

SecretKeySpec skeySpec = new SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"), "AES");

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");

cipher.init(2, skeySpec, iv);

String decoded = new String(cipher.doFinal(ciphertext));

return decoded;

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

//These functions are used to decrypt the strings

// its names are acronyms - gc/generate class name, gm/generate method name, ga/generate argument, gcn/generate code name

private static String gc() {

return decrypt\_ds("zbTHGeQeUUxj3dJ43fDwkcKmk4erD60GZXReeWl3ITA=");

}

private static String gm() {

return decrypt\_ds("LlzQOUB3opWgJZeFNI/Jsg==");

}

private static String ga() {

return decrypt\_ds("oxTrCOohrr2fAZfJZAjcNA==");

}

private static String gcn() {

return decrypt\_ds("FxojiPxNKXdtYiY65LK1CA==");

}

private static String decrypt\_ds(String s) {

try {

SecretKeySpec skeySpec = new SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"), "AES");

IvParameterSpec iv = new IvParameterSpec(initVector);

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");

cipher.init(2, skeySpec, iv);

String decoded = new String(cipher.doFinal(Base64.getDecoder().decode(s.getBytes())));

return decoded;

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

public static void main(String[] args) {

// The decrypted link is: https://www.math.unipd.it/~elosiouk/test.dex

String gu = new Solver().gu();

System.out.println("Class loaded successfully from: " + gu);

System.out.println(gc());

System.out.println(gm());

System.out.println(ga());

System.out.println(gcn());

}

}