Challenge description:  
  
*The career of every reverser starts with a babyrev chall. Here is yours.*

Solution

You can install many tools here, like *jadx.*

To install JADX and have it accessible as a command in the terminal on Ubuntu, follow these steps:

First, you'll need to download JADX. You can download it from the official GitHub repository: <https://github.com/skylot/jadx/releases>

After downloading, extract the archive to a suitable location on your system. You can use the command line for this. Replace `/path/to/destination` with your desired destination folder:

* *tar -xvzf jadx-<version>.tar.gz -C /path/to/destination*

- You'll want to set up environment variables for JADX. You can do this by modifying your `~/.bashrc` file. Add the following line at the end of the file, replacing `/path/to/jadx` with the actual path to the JADX directory:

* *export PATH=$PATH:/path/to/jadx/bin*

Make sure you save the file after editing. You can do this with a text editor, such as `nano`:

* *nano ~/.bashrc*

Then, add the line, save the file and run:

* *source ~/.bashrc*

To verify that JADX is installed correctly and accessible as a command, open a new terminal window and type:

* *jadx*

You should see the JADX command-line interface (same procedure for *jadx-gui*).

Also, you can install *apktool* via *sudo snap install apktool*

And simply run:

* *apktool d app.apk -o output*
* *apktool b output -o patched.apk*
  + Used for repackaging attacks

Here, if you use *jadx-gui*, you will see a lot of files, classes and stuff organized, while *jadx-cli* will simply create an output folder for you to inspect; *apktool* will do the same as the cli.

You can see, for example launching the command

* *jadx -d out babyrev.apk*

(don’t mind if it says there are errors).

You will find some classes which are interesting:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

The *r.java* file is not interesting, while the *MainActivity* only checks the flag, giving a simple widget to see if text changed, otherwise it simply prints “Invalid flag” or “Valid flag”. Given *BuildConfig* does nothing apart from setting the application ID let’s inspect the *FlagChecker.java*, I’d say.

The FlagChecker class has three functions:

1. The CheckFlag, in which we understand:
   * it starts with FLAG{scientia
   * it is alphanumeric, at least combining both uppercase and lowercase chars with a regex at the end
   * it’s 27 chars long
   * 12th character (a) is equal to 21st character (a)
   * the 13th and 22nd characters must be underscores ('\_')
   * the last character of the flag must be '}'.
   * there is a check equal to cBgRaGv
2. The CheckFlag, in which we understand: the getX, getY, getZ which are used inside last of FlagChecker to make some mess

This part makes something like:  
*getX() -> 2*

*getY() -> 3*

*getZ() -> 4*

*charAt((int) pow(2,2) + pow(2,3)) == charAt( pow(4,2) + 5.0d)*

*4 + 8 == 16 + 6*

*12 == 22*

*a == a*

*&& flag.toLowerCase().charAt((int) (Math.pow((double) getX(), (double) getX()) + Math.pow((double) getX(), (double) getY()))) == flag.toLowerCase().charAt((int) (Math.pow((double) getZ(), (double) getX()) + 5.0d))*

*/r == char(13)*

*/n == char(9)*

*bam(flag.substring((int) (Math.pow((double) getZ(), (double) getX()) - 2.0d), ((int) Math.pow((double) getX(), (double) (getX() + getY()))) - 10)).equals("cBgRaGvN")*

*cBgRaGvN*

*pOtEnTiA*

*0 1 2 3 4 5 6 7 8 9 10 11 12*

*a b c d e f g h i j k l m*

*n o p q r s t u v w x*

1. The third method creating the regex of upper/lowercase chars

On the last *checkFlag* code part, it simply makes some random calculation over powers, but don’t get distracted; it simply says that the part from 14th character up to the 22nd will be substituted with the combined lower/upper case regex transformation.

Now, for the last part:  
the bam function in the provided code performs character transformations based on the position of characters in the alphabet. Here's an explanation of how it works:

1. The function processes each character in the input string s one by one.
2. If the character is in the range 'a' to 'm' (lowercase letters from 'a' to 'm'), it increments its Unicode value by 13, effectively shifting it to the second half of the lowercase alphabet.
3. If the character is in the range 'A' to 'M' (uppercase letters from 'A' to 'M'), it also increments its Unicode value by 13, shifting it to the second half of the uppercase alphabet.
4. If the character is in the range 'n' to 'z' (lowercase letters from 'n' to 'z'), it decrements its Unicode value by 13, moving it to the first half of the lowercase alphabet.
5. If the character is in the range 'N' to 'Z' (uppercase letters from 'N' to 'Z'), it decrements its Unicode value by 13, moving it to the first half of the uppercase alphabet.

Here’s how it works here:

* 'c' becomes 'p':
  + 'c' (ASCII value 99) + 13 = 'p' (ASCII value 112)

1. 'B' becomes 'O':
   * 'B' (ASCII value 66) + 13 = 'O' (ASCII value 79)
2. 'g' becomes 't':
   * 'g' (ASCII value 103) + 13 = 't' (ASCII value 116)
3. 'R' becomes 'E':
   * 'R' (ASCII value 82) + 13 = 'E' (ASCII value 69)
4. 'a' becomes 'n':
   * 'a' (ASCII value 97) + 13 = 'n' (ASCII value 110)
5. 'G' becomes 'T':
   * 'G' (ASCII value 71) + 13 = 'T' (ASCII value 84)
6. 'v' becomes 'i':
   * 'v' (ASCII value 118) - 13 = 'i' (ASCII value 105)
7. 'N' becomes 'A':
   * 'N' (ASCII value 78) - 13 = 'A' (ASCII value 65)

Up to now, we have:  
*FLAG{scientia\_pOtEnTiA*

We know it ends with a curly bracket, anyway, we still have to analyze:

* *new StringBuilder(flag).reverse().toString().toLowerCase().substring(1).startsWith(ctx.getString(R.string.last\_part))*

It’s in reverse order and lowercase. Note the first substring starts with a value located on the resources and it must be a last string, it seems.

If we navigate into "babyrev.apk->Resources->resources.arsc->res->values->strings.xml".

We know the location thanks to "R.string", which retrieves it there. We then locate the string under the keyword "last\_part" which has the following value "tse", so its in reverse and it’s gotta be “est”.

The part:

* *flag.substring(5, flag.length() - 1).matches(getR()))*

simply says the flag must have alternating upper/lower case, starting with uppercase (so, first a uppercase and then a lowercase char, up to the end from the 5th char).

And so the flag is: *FLAG{ScIeNtIa\_pOtEnTiA\_EsT}*