Crackme#02

Filovirid
filovirid@protonmail.com

December 1, 2019

1 Question

You can find the file related to the question at crackme_02.zip.

Answer the following question:

1. Find the right flag just with static analysis.

2 Objectives

The objective of this crackme is to learn how to read Java codes and how to work with Java decompilers like jadx.

In addition, in this exercise, we learn how to work with jtr, one of the most powerful hash crackers.

3 Finding the flag

The first step is to extract the zip file using the password (if you do not know what is the password, you should read the README file).

After extracting the zip file, we have an .apk file (android_cm.apk). Now, open this file using jadx as shown in Figure 1.

As it is shown in Figure 1, we are interested in the 'MainActivity' class. After clicking on 'MainActivity' on the left panel, you will see the related code in the right panel as shown in Figure 2.

In the 'MainActivity' class, on line 25-28, we can see the definition of 'onClickListener' method which is basically define the action that should be

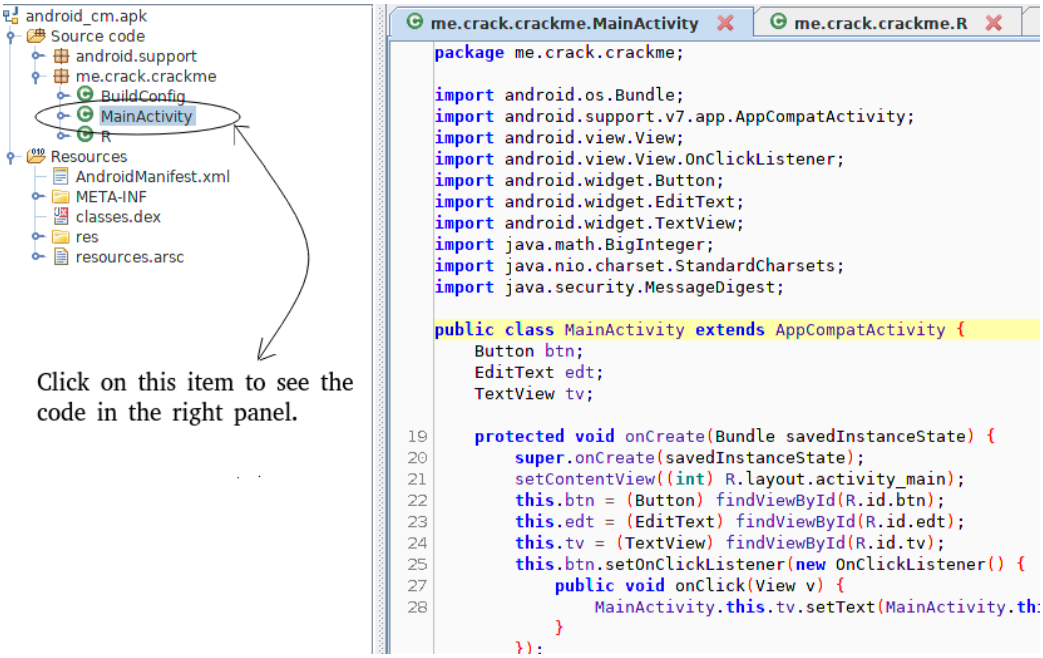

Figure 1: Opening .apk file in jadx.

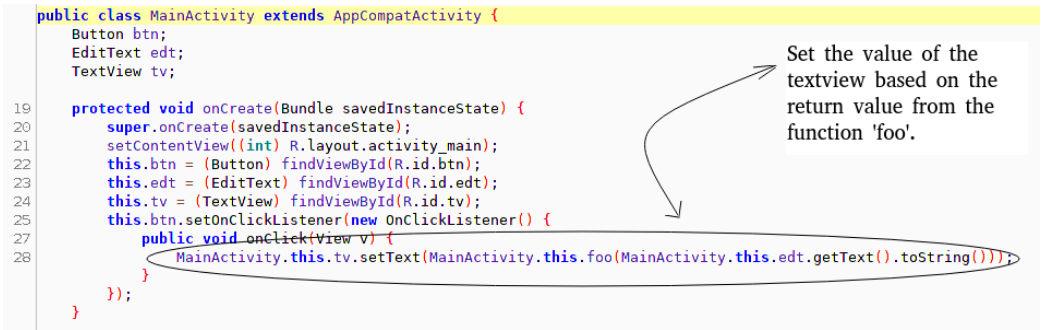

Figure 2: MainActivity class source code in Java

```
String foo(String bar) {
    try {
        String[] a49485794875498573345 = bar.split("_");
        MessageDigest.getInstance("MD5").update(StandardCharsets.UTF_8.encode(a49485794875498573345[0]));
        String a4948579487589857345 = String.format("%032x", new Object[]{new BigInteger(1, md5.digest())});
        String hint = "Weak hash...can be easily broken with the right tools.";
        long a4948579487549857345 = (long) Integer.parseInt(a49485794875498573345[1]);
        long a4248579487549857345 = a494857948754985734S;
        for (int a4948579487S498573145 = 1; ((long) a4948579487S498573145) <= a4948579487549857345; a4948579487S498573145++) {
            a4248579487549857345 += ((long) a4948579487S498573145) + a4948579487549857345;
        }
        if ("395425559298_f581a6392d04a7098098f54337f28741".equalsIgnoreCase(Long.toString(a4248579487549857345) + "_" + a4948579487589857345)) {
            return "CONGRATULATIONS!!!!!";
        }
        throw new Exception();
    } catch (Exception e) {
        return "WRONG PASSWORD";
    }
}
```

Figure 3: Source code of the foo function

```
1 String foo(String bar) {
2     try {
3         String[] inp_splt = bar.split("_");
4         MessageDigest.getInstance("MD5").update(StandardCharsets.UTF_8.encode(inp_splt[0]));
5         String final2 = String.format("%032x", new Object[]{new BigInteger(1, md5.digest())});
6         String hint = "Weak hash...can be easily broken with the right tools.";
7         long var1 = (long) Integer.parseInt(inp_splt[1]);
8         long final1 = var1;
9         for (int var0 = 1; ((long) var0) <= var1; var0++) {
10            final1 += ((long) var0) + var1;
11        }
12        if ("395425559298_f581a6392d04a7098098f54337f28741".equalsIgnoreCase(Long.toString(final1) + "_" + final2)) {
13            return "CONGRATULATIONS!!!!!";
14        }
15        throw new Exception();
16    } catch (Exception e) {
17        return "WRONG PASSWORD";
18    }
19 }
20 }
```

Figure 4: Source code of foo with more meaningful variable names

taken after users click the button in the form. Therefore, after the button clicked by the user, the textbox will be filled by the value returns from the function 'foo'.

Figure 3, shows the source code of the function 'foo'. The name of the variables and constants in this function are really ugly. So, first we change the names to more appropriate names and then try to analyse the code. Figure 4, shows the same 'foo' function but with more meaningful variable names.

The input argument of the 'foo' function is called 'bar' and is type of String. Line 3 (Figure 4), splits the input argument (of type string) based on the underscore (_) character and store the result in the array with 'inp_splt' name. Line 4 and 5, calculate the MD5 value of the first part of the input parameter and store the result in 'final2' variable. Line 12, shows the 'final2'

variable is compared to 'f581a6392d04a7098098f54337f28741' value. Therefore, from line 12, we know that the right password for this file have 2 parts in the form of 'part1_part2'. We also know that the first part of the answer should satisfy the equation below:

$$MD5(part1ofanswer) = "f581a6392d04a7098098f54337f28741" \quad (1)$$

The MD5 hash is not reversible. In other word, there is no way to find the real answer for part1 from the MD5 value. However, if the length of the 'part1' is not big, then we can try to find the answer with brute forcing. There is a hint at line 6 which is as follow:

String hint = "Weak hash...can be easily broken with the right tools.";

Well, now we know that the hash is weak and we can break it with brute force tools like John the ripper.

3.1 Breaking the hash with Jtr.

First step is to install 'john the ripper'. Download the source file from openwall[1], and then use it(I am using ubuntu and you must have openssl headers installed in order to use jtr):

```
cd src && ./configure && make
```

If everything goes fine, then you will see the binary file in 'run' directory. After installing jtr, now it's time to pass the hash to it. copy the hash value from from line 12 of the 'foo' function source code in jadx and paste it in a new file called 'myhash'. Then type the following command in bash:

```
./john --format=Raw-MD5 myhash --fork=8 --incremental
```

The above command tells the john utility that we try to break the hash that is in 'myhash' file in the 'incremental' mode and we know that the hash is of type 'md5' and also we would like to run 8 instances of john in parallel to make the process faster.

Figure 5, shows the result of running jtr. It found the password in less than 5 minutes.

So far, we know that the first part of the input should be 'w1kI'. Let's get the md5 value of this part to see if it is the same as 'f581a6392d04a7098098f54337f28741'. I use 'md5sum' tool in my Linux.

[1] https://www.openwall.com

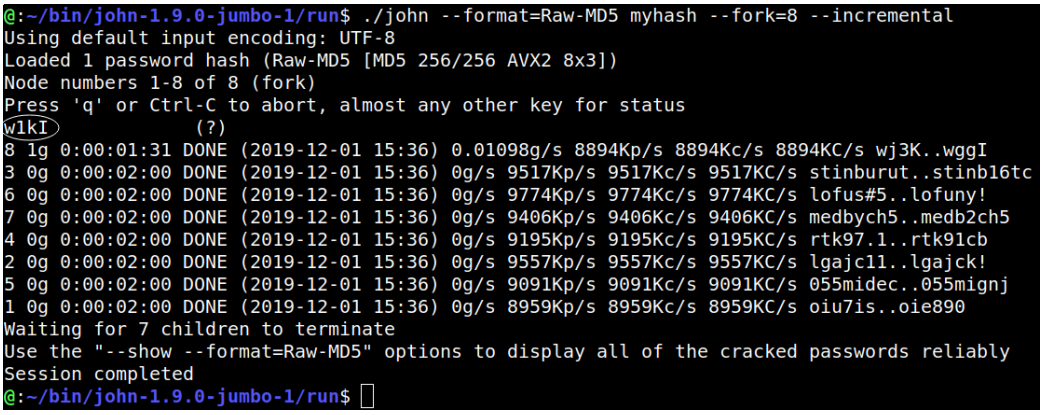

Figure 5: breaking the first part of the answer using brute forcing.

`echo -n 'w1kI' | md5sum`

After running the above command, you see that the answer is correct. So we just found the first part of the answer. Let's find the second part now.

3.2 First part of the answer

Looking at line 12 of the code, we know that the final result for the second part of the answer should be '395425559298'. line 7-11, do some math operation on the second part of the input and store the result in 'final1' variable and then compares it to '395425559298'. 'var1' is the second part of the input (line 7). Then, we have the following equation in line 8-11:

$$final1 = var1 + \sum_{var0=1}^{var1} (var0 + var1) \tag{2}$$

Which equals to:

$$final1 = var1 + (1 + var1) + (2 + var1) + ... + (var1 + var1) \tag{3}$$

Simplifying the equation, we have:

$$final1 = var1 + (1 + 2 + 3 + 4 + ... + var1) + var1(var1) \tag{4}$$

And finally:

$$final1 = var1 + var1^2 + \frac{var1(var1+1)}{2} \tag{5}$$

Replacing the value of 'final1' with '395425559298', we have a quadratic equation as below:

$$var1^2 + var1 - 263617039532 = 0 \tag{6}$$

And solving this equation, the value of 'var1' is : 513436

4 Final answer

So far, we solved the first and the second part of the answer. By concatenating these two parts, we have the final answer as **w1kI_513436**.

If you enter 'w1kI_513436' as the answer, you will see the 'CONGRATULATIONS!!!!!' message (line 13). Remember that to solve the second part of the answer, you need to completely understand the logic behind the code.

Figure 6, 8, and 7, show the right and wrong input for this crackme problem.

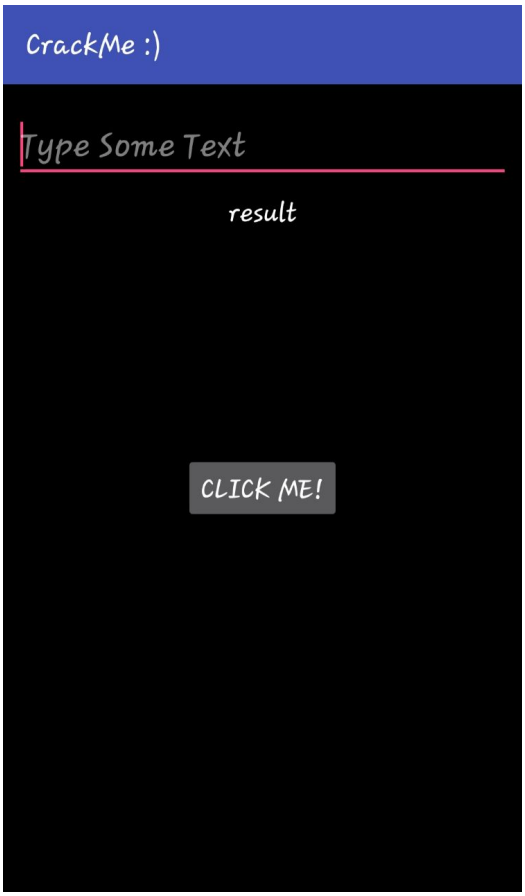

Figure 6: GUI of the android program.

Figure 7: Entering wrong password.

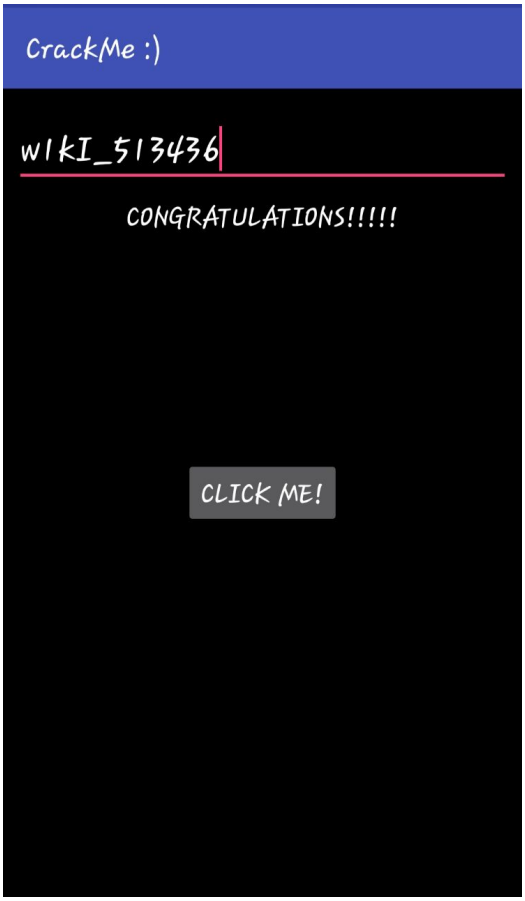

Figure 8: Entering right password.