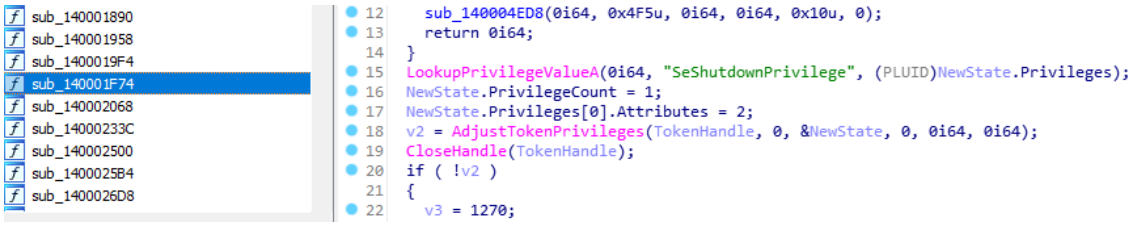
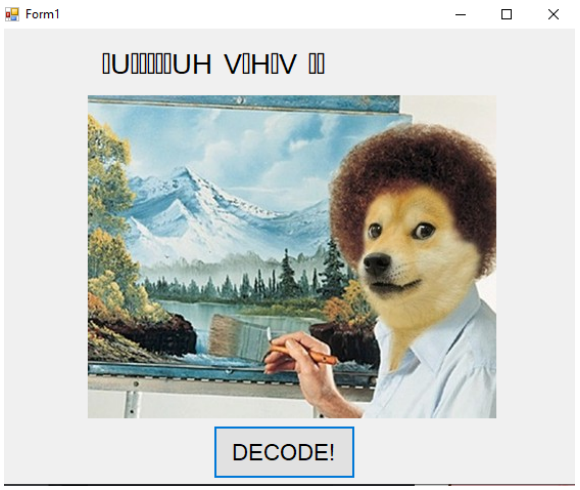
3 Lab tasks: debugging .NET and Android applications

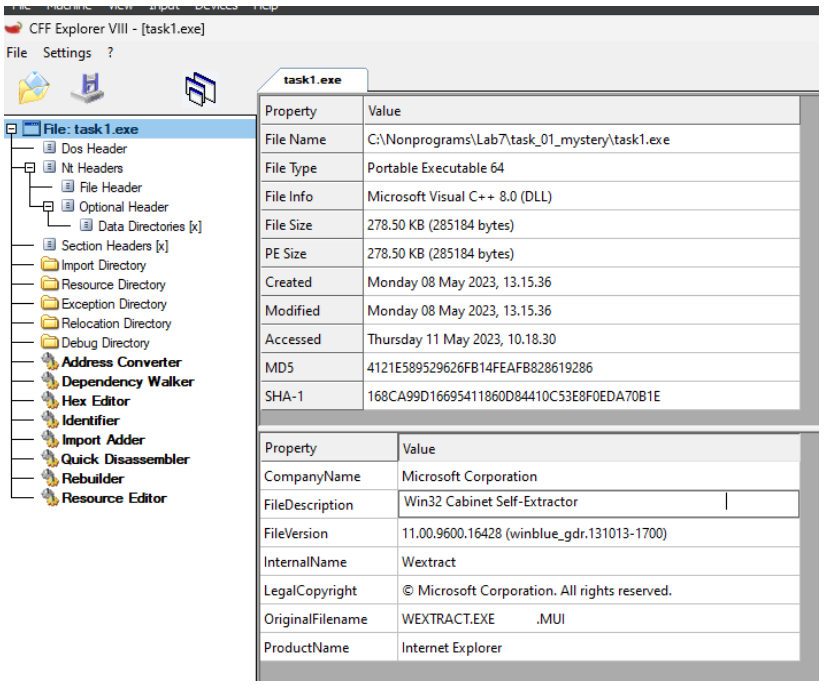
3.1 Task: reversing .NET binaries (6p)

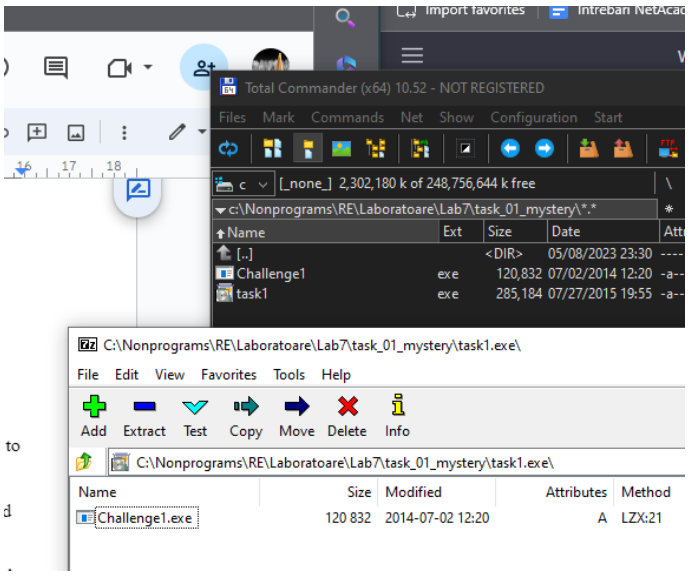
Perform the following tasks using the task1.exe binary:

- Investigate task1.exe however you see fit. Spend no more than 10-15 minutes trying to approach the binary as usual, with IDA Pro.
    - From IDA we can not deduce much, besides the fact that there are some Windows registers, some temporary files and some paths computations used by the executable
    - 

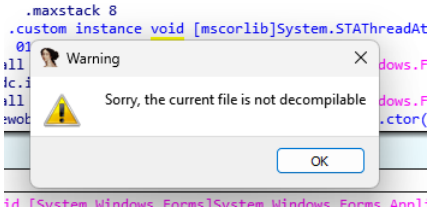
    - There are also the privileges shutdown
    - 

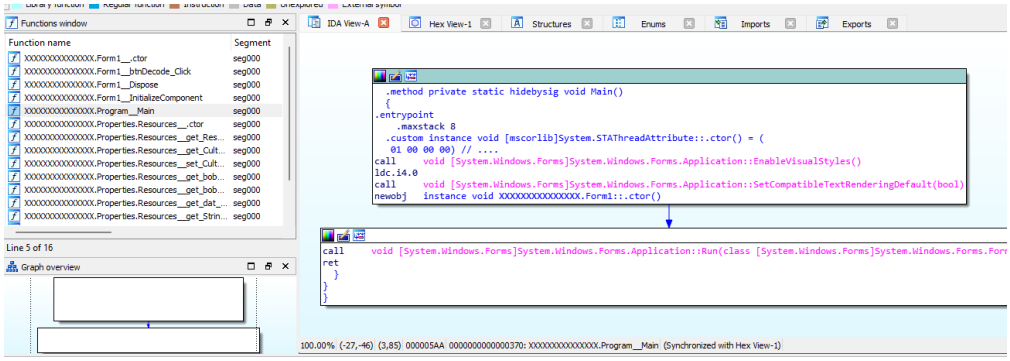
    - I also tried to unzip and run it
    - 

- Next, open the binary in the CFF Explorer and look for the “FileDescription” field. What value does the binary have and what does it mean?
  - 

    - The value is “Win32 Cabinet Self-Extractor” implies that the file is a self-extracting archive that was created in .cab file format (compress and package files). Using this format there is not need for an additional software to extract because it is “self-extracting”.
    - There can be included more file in .cab archived files, which can be extracted together.
- Open the binary in 7z or Winrar, extract the underlying executable and open it in IDA Pro. Explain why you think the extraction works. (2p)
  - 

- The extraction can be computed because of the volatile property of .cab files, which is an widely-used format for Windows archive files (especially the self-extracting ones, which means they have their own mechanism in order to auto-dearchivate). The extraction with third party software like 7-Zip or WinRAR works because the self-extracting file is based on the standard Cabinet file format and includes a compressed archive of the files to be extracted along with an executable program that can extract the files. The tool can use this program to extract the content, which means that these 2 softwares are compatible with cab format.

- What type of file is recognized in this executable by IDA Pro? Notice that decompilation does not work and reading the assembly is pretty hard.
  - 

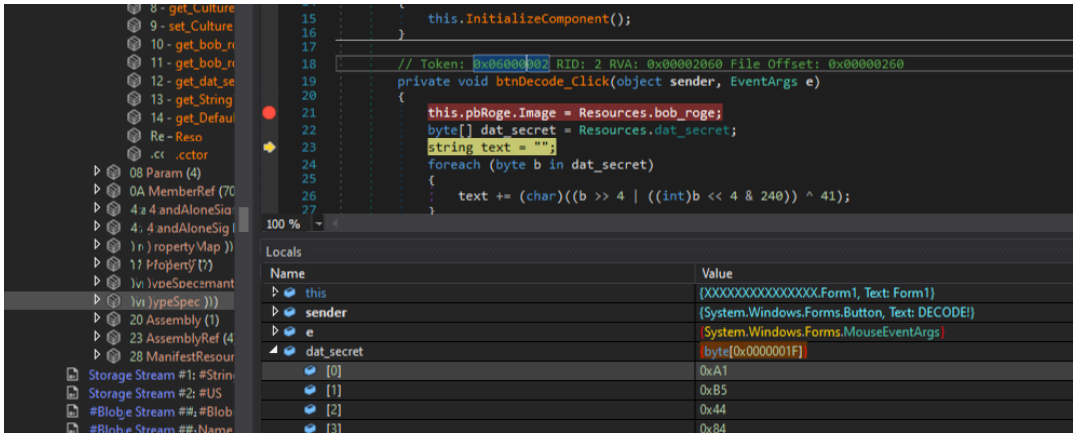
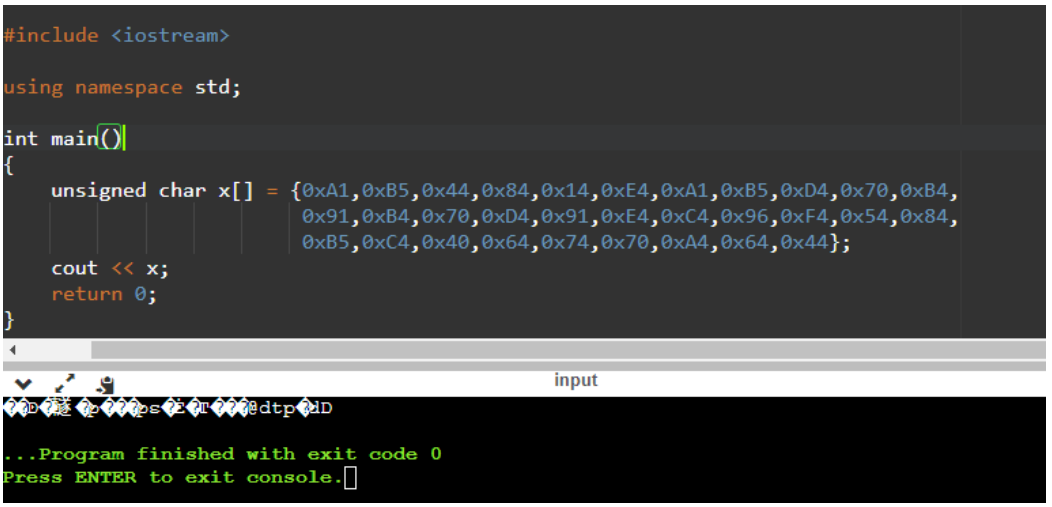
  - 

- Now use dnSpy (64 bit) to open the binary and poke around in btnDecode_Click. Find the correct output (either through static analysis or dynamic analysis, both using dnSpy). (4p)

```
// Token: 0x06000002 RID: 2 RVA: 0x00002060 File Offset: 0x00000260
private void btnDecode_Click(object sender, EventArgs e)
{
    this.pbRoge.Image = Resources.bob_roge;
    byte[] dat_secret = Resources.dat_secret;
    string text = "";
    foreach (byte b in dat_secret)
    {
        text += (char)((b >> 4 | ((int)b << 4 & 240)) ^ 41);
    }
    text += "\0";
    string text2 = "";
    for (int j = 0; j < text.Length; j += 2)
    {
        text2 += text[j + 1];
        text2 += text[j];
    }
    string text3 = "";
    for (int k = 0; k < text2.Length; k++)
    {
        char c = text2[k];
        text3 += (char)((byte)text2[k] ^ 102);
    }
    this.lbl_title.Text = text3;
}
```

```
// Token: 0x17000005 RID: 5
// (get) Token: 0x0600000C RID: 12 RVA: 0x00002468 File Offset: 0x00000668
internal static byte[] dat_secret
{
    get
    {
        object @object = Resources.ResourceManager.GetObject("dat_secret", Resources.resourceCulture);
        return (byte[])@object;
    }
}
```

```
[EditorBrowsable(EditorBrowsableState.Advanced)]
internal static CultureInfo Culture
{
    get
    {
        return Resources.resourceCulture;
    }
    set
    {
        Resources.resourceCulture = value;
    }
}
```

- 
- I extracted the secret
  - 0xA1 0xB5 0x44 0x84 0x14 0xE4 0xA1 0xB5 0xD4 0x70 0xB4 0x91 0xB4 0x70 0xD4 0x91 0xE4 0xC4 0x96 0xF4 0x54 0x84 0xB5 0xC4 0x40 0x64 0x74 0x70 0xA4 0x64 0x44
  - 

-
```
using namespace std;

string btnDecode_Click_decryption ( unsigned char dat_secret[32])
{
    string text = "";
    unsigned char b;
    for (int i = 0; i < 32; i++)
    {
        b = dat_secret[i];
        text += (char)((b >> 4 | ((int)b << 4 & 240)) ^ 41);
    }
    text += "\0";
    string text2 = "";
    for (int j = 0; j < text.size(); j += 2)
    {
        text2 += text[j + 1];
        text2 += text[j];
    }
    string text3 = "";
    for (int k = 0; k < text2.size(); k++)
    {
        char c = text2[k];
        text3 += (char)(text2[k] ^ 102);
    }
    return (text3);
}

int main()
{
    unsigned char x[32] = {0xA1,0xB5,0x44,0x84,0x14,0xE4,0xA1,0xB5,0xD4,0x70,0xB4,
                           0x91,0xB4,0x70,0xD4,0x91,0xE4,0xC4,0x96,0xF4,0x54,0x84,
                           0xB5,0xC4,0x40,0x64,0x74,0x70,0xA4,0x64,0x44};
    cout << btnDecode_Click_decryption(x);
    return 0;
}
```

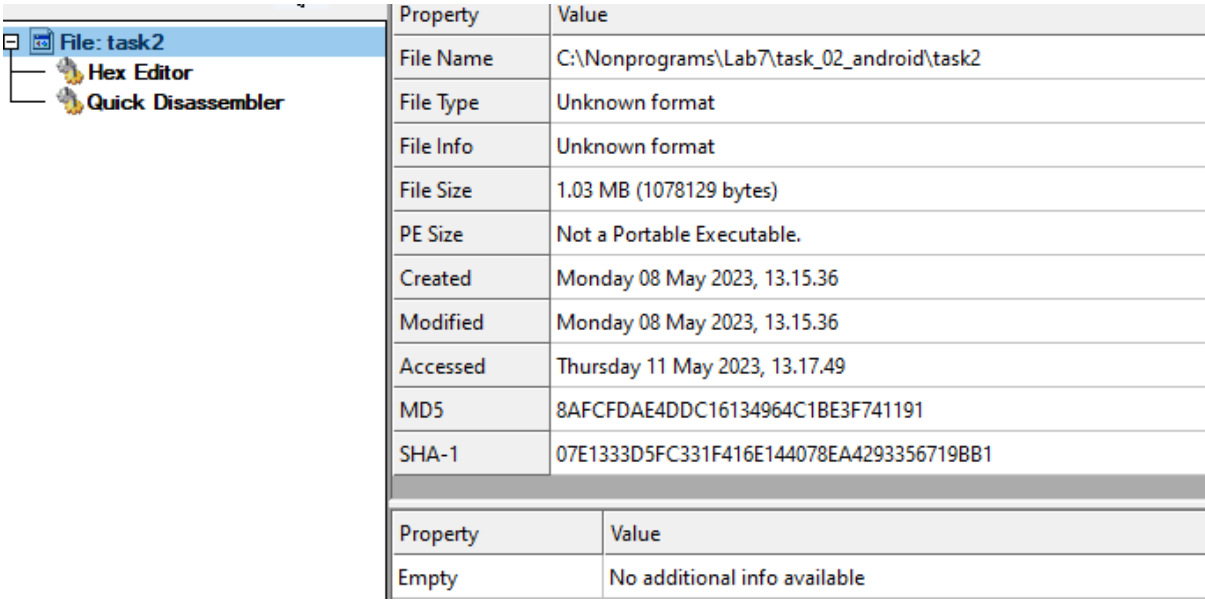
```
input
U
 UHVHV&
        KH      o


...Program finished with exit code 0
Press ENTER to exit console.
```

  - So, the secret is: ��D�轂�p���pεÈ�T���@dtp�dD

3.2 Task: reversing Android binaries (9p)

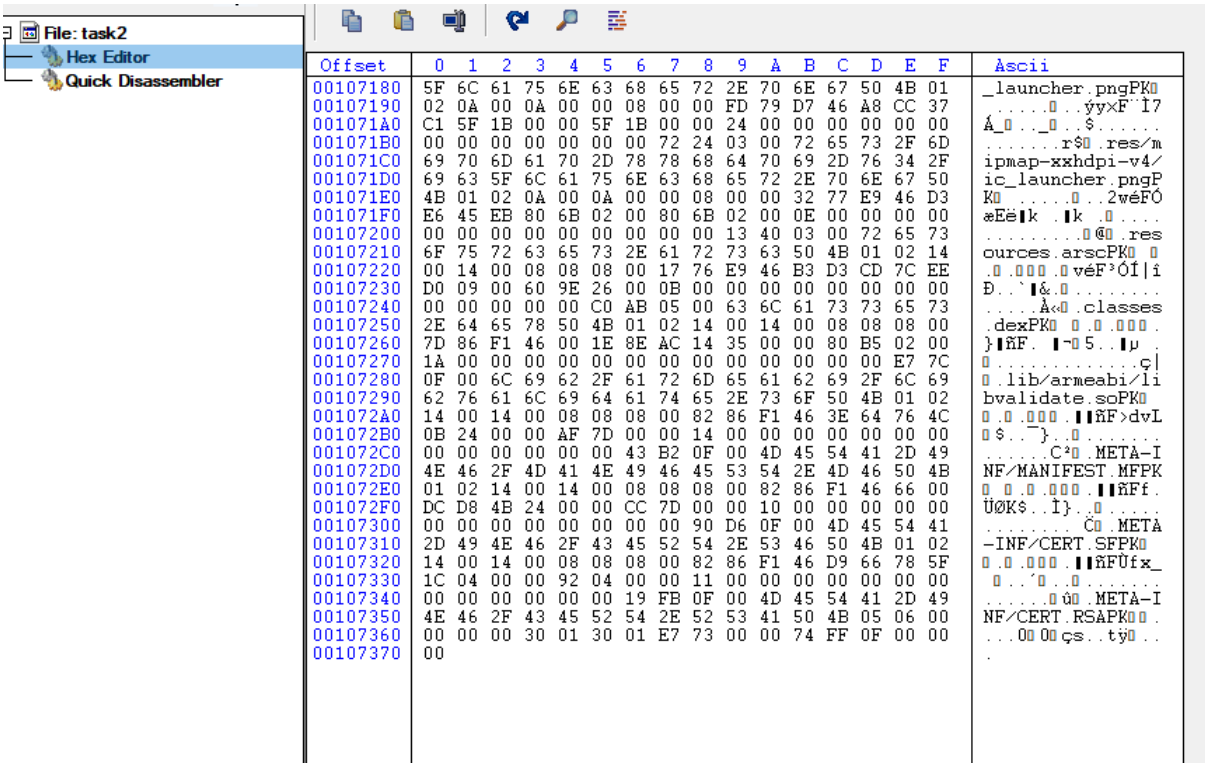
Perform the following tasks using the task2 binary:

- What type of file is it? How can you unpack it? Run the application in an emulator. (2p)
  - We can not deduce the format from CFF Explorer

- 

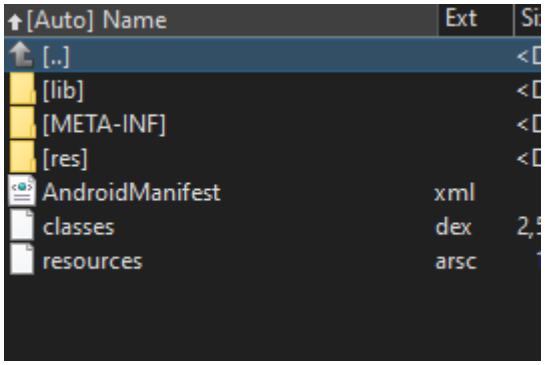
Property	Value
File Name	C:\Nonprograms\Lab7\task_02_android\task2
File Type	Unknown format
File Info	Unknown format
File Size	1.03 MB (1078129 bytes)
PE Size	Not a Portable Executable.
Created	Monday 08 May 2023, 13.15.36
Modified	Monday 08 May 2023, 13.15.36
Accessed	Thursday 11 May 2023, 13.17.49
MD5	8AFCFDAE4DDC16134964C1BE3F741191
SHA-1	07E1333D5FC331F416E144078EA4293356719BB1

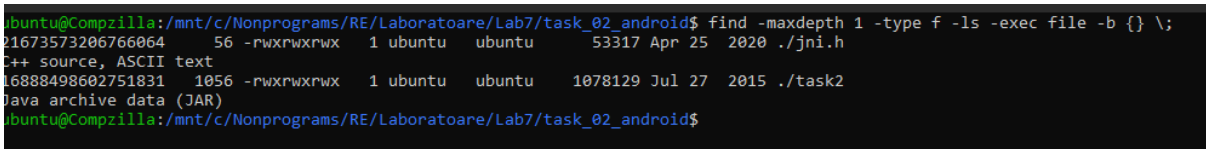
Property	Value
Empty	No additional info available

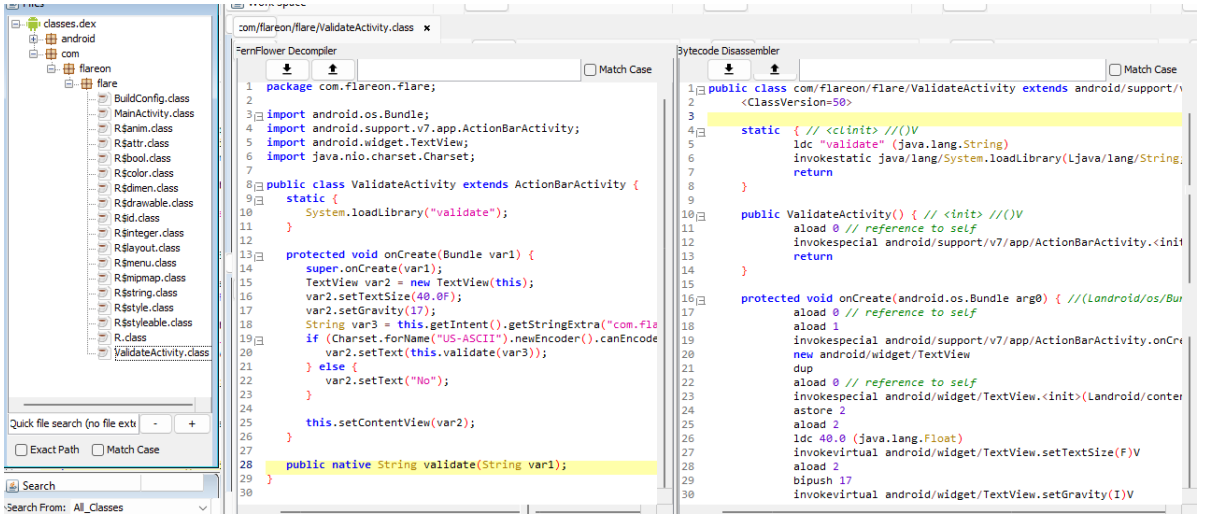
  - The only information we can extract is that the file is not a PE
  - If we see the hex we can see more metadata
- 

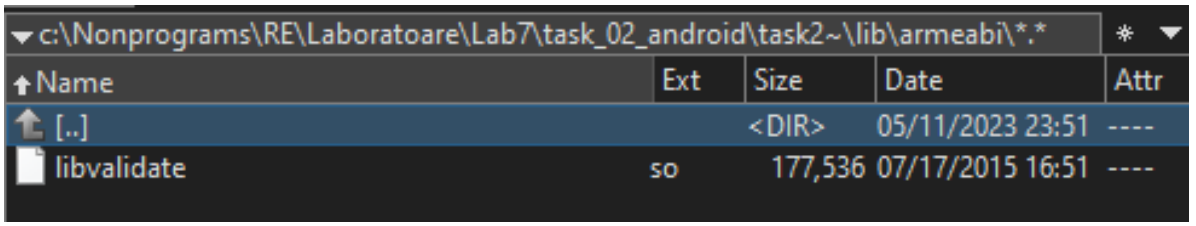
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00107180	5F	6C	61	75	6E	63	68	65	72	2E	70	6E	67	50	4B	01	_launcher.pngPK□
00107190	02	0A	00	0A	00	00	08	00	00	FD	79	D7	46	A8	CC	37	.....□..ýy×F¨Ì7
001071A0	C1	5F	1B	00	00	5F	1B	00	00	24	00	00	00	00	00	00	Á_□.._□..$.....
001071B0	00	00	00	00	00	00	00	72	24	03	00	72	65	73	2F	6D	.......r$□.res/m
001071C0	69	70	6D	61	70	2D	78	78	68	64	70	69	2D	76	34	2F	ipmap-xxhdpi-v4/
001071D0	69	63	5F	6C	61	75	6E	63	68	65	72	2E	70	6E	67	50	ic_launcher.pngP
001071E0	4B	01	02	0A	00	0A	00	00	08	00	00	32	77	E9	46	D3	K□.....□..2wéFÓ
001071F0	E6	45	EB	80	6B	02	00	80	6B	02	00	0E	00	00	00	00	æEë▌k .▌k .□....
00107200	00	00	00	00	00	00	00	00	13	40	03	00	72	65	73		.........□@□.res
00107210	6F	75	72	63	65	73	2E	61	72	73	63	50	4B	01	02	14	ources.arscPK□ □
00107220	00	14	00	08	08	08	00	17	76	E9	46	B3	D3	CD	7C	EE	.□.□□□.□véF³ÓÍ|î
00107230	D0	09	00	60	9E	26	00	0B	00	00	00	00	00	00	00	00	Ð..`▌&.□........
00107240	00	00	00	00	00	C0	AB	05	00	63	6C	61	73	73	65	73	.....À«□.classes
00107250	2E	64	65	78	50	4B	01	02	14	00	14	00	08	08	08	00	.dexPK□ □.□.□□□.
00107260	7D	86	F1	46	00	1E	8E	AC	14	35	00	00	80	B5	02	00	}▌ñF. ▌¬□5..▌µ .
00107270	1A	00	00	00	00	00	00	00	00	00	00	00	00	00	E7	7C	□.............ç|
00107280	0F	00	6C	69	62	2F	61	72	6D	65	61	62	69	2F	6C	69	□.lib/armeabi/li
00107290	62	76	61	6C	69	64	61	74	65	2E	73	6F	50	4B	01	02	bvalidate.soPK□
001072A0	14	00	14	00	08	08	08	00	82	86	F1	46	3E	64	76	4C	□.□.□□□.▌▌ñF>dvL
001072B0	0B	24	00	00	AF	7D	00	00	14	00	00	00	00	00	00	00	□$..¯}..□.......
001072C0	00	00	00	00	00	00	43	B2	0F	00	4D	45	54	41	2D	49	......C²□.META-I
001072D0	4E	46	2F	4D	41	4E	49	46	45	53	54	2E	4D	46	50	4B	NF/MANIFEST.MFPK
001072E0	01	02	14	00	14	00	08	08	08	00	82	86	F1	46	66	00	□ □.□.□□□.▌▌ñFf.
001072F0	DC	D8	4B	24	00	00	CC	7D	00	00	10	00	00	00	00	00	ÜØK$..Ì}..□.....
00107300	00	00	00	00	00	00	00	00	90	D6	0F	00	4D	45	54	41	.........Ö□.META
00107310	2D	49	4E	46	2F	43	45	52	54	2E	53	46	50	4B	01	02	-INF/CERT.SFPK□
00107320	14	00	14	00	08	08	08	00	82	86	F1	46	D9	66	78	5F	□.□.□□□.▌▌ñFÙfx_
00107330	1C	04	00	00	92	04	00	00	11	00	00	00	00	00	00	00	□..´□..□.......
00107340	00	00	00	00	00	00	19	FB	0F	00	4D	45	54	41	2D	49	.....□û□.META-I
00107350	4E	46	2F	43	45	52	54	2E	52	53	41	50	4B	05	06	00	NF/CERT.RSAPK□□.
00107360	00	00	00	30	01	30	01	E7	73	00	00	74	FF	0F	00	00	...0□0□çs..tÿ□..
00107370	00																.

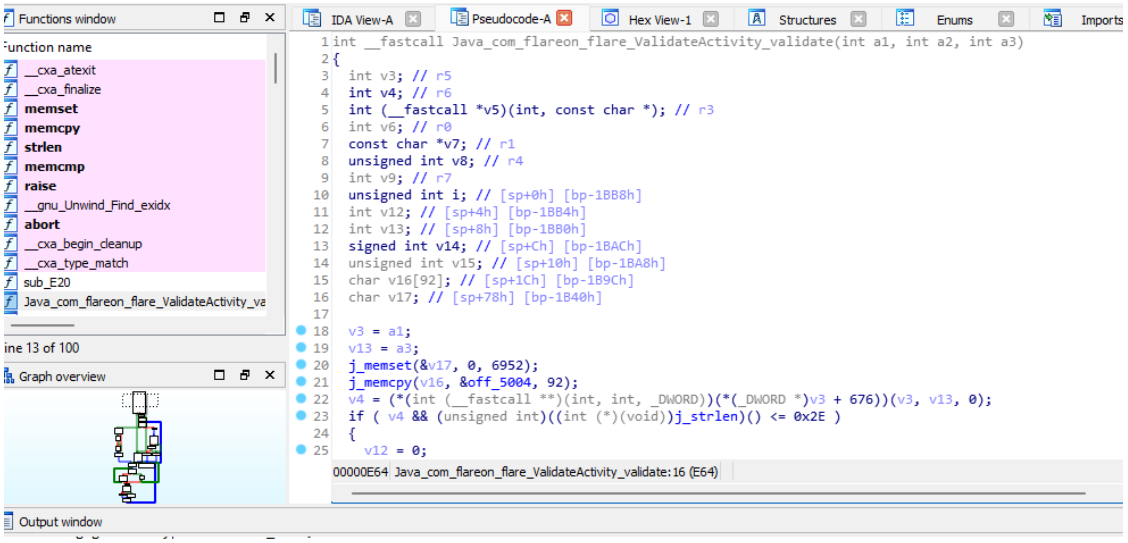
  - Using total commander we can see the content

- 

  - So it seems the file is also an archive type. If we analyze it using an ubuntu subsystem, we can conclude that the file is a JAR
  - 

- Use Bytecode Viewer to open the same file. Look under “com” through the Activity classes. Where is the password checked? What does Loadlibrary do and where is the library file?
  - password check
  - 

  - The System.loadLibrary() function in Java is used to load a native library - “validate” - in JVM memory .
  - The library can be found in project’s path
  - 

- Open it in IDA, look in the “Java_com_flareon_flare_ValidateActivity_validate” function.
  - 

- Load jni.h using File/Load File/Parse C header file and retype the function as “int __fastcall Java_com_flareon_flare_ValidateActivity_validate(JNIEnv *jnienv, int a2, jstring input)”
  - 

- Reverse the function and find the correct input. (7p)