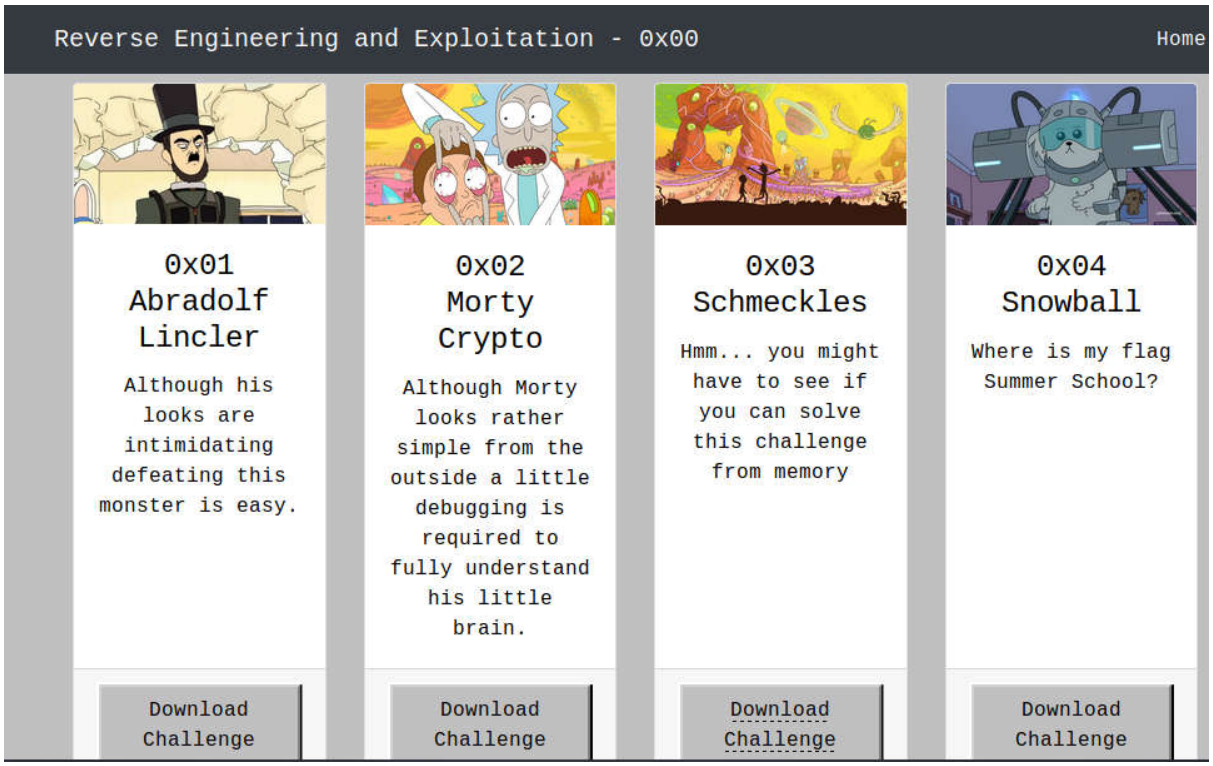
Reverse Engineering

0x01 – Abradolf Lincler
0x02 – Mortycrypto

Mitchell Luke Tuck
CYBER SECURITY – AN OFFENSIVE MINDSET 2019

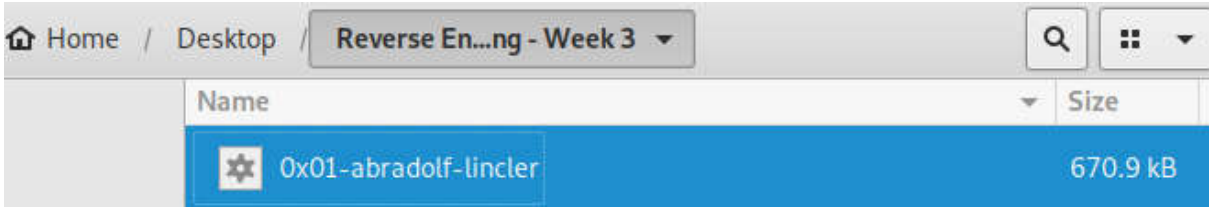
Table of Contents

Reverse Engineering Lab

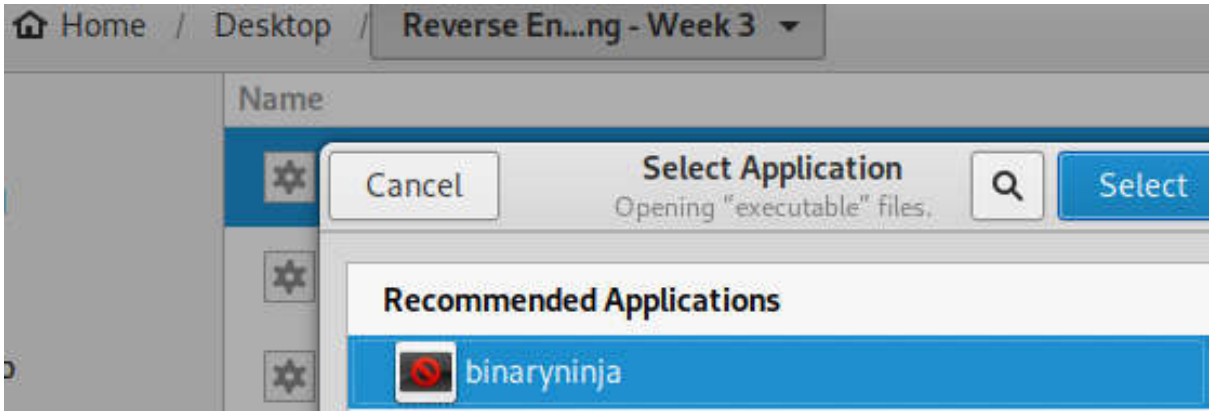


0X01 – ABRADOLF LINCLER

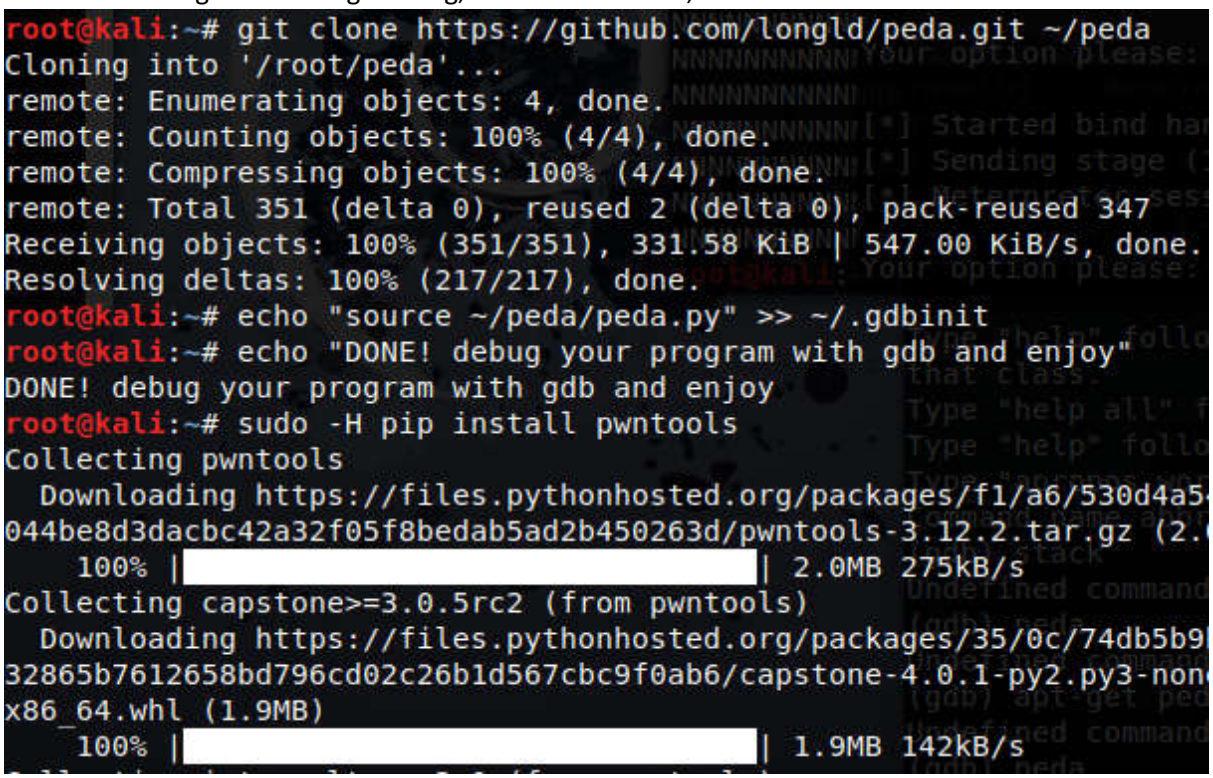
Downloaded and Decrypted Challenge to local file:
/root/Desktop/Reverse Engineering - Week 3/0x01-abradolf-lincler



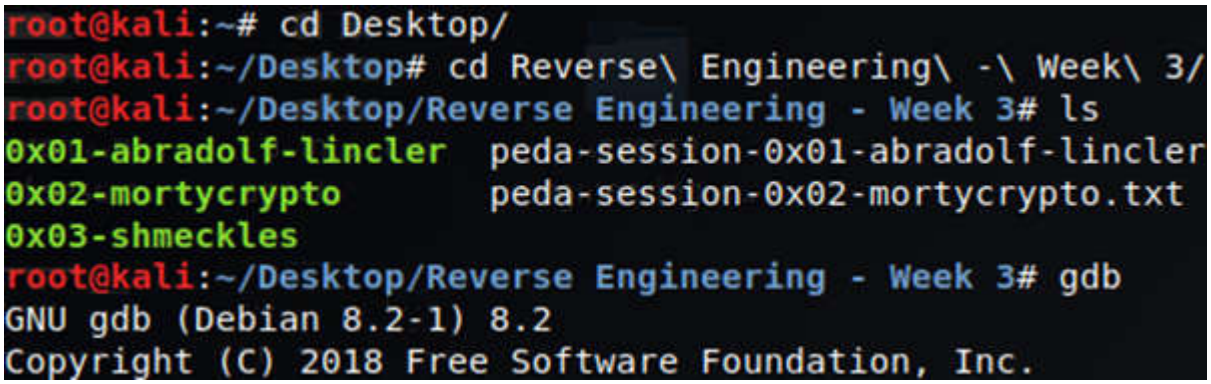
Opened file with Binary Ninja:



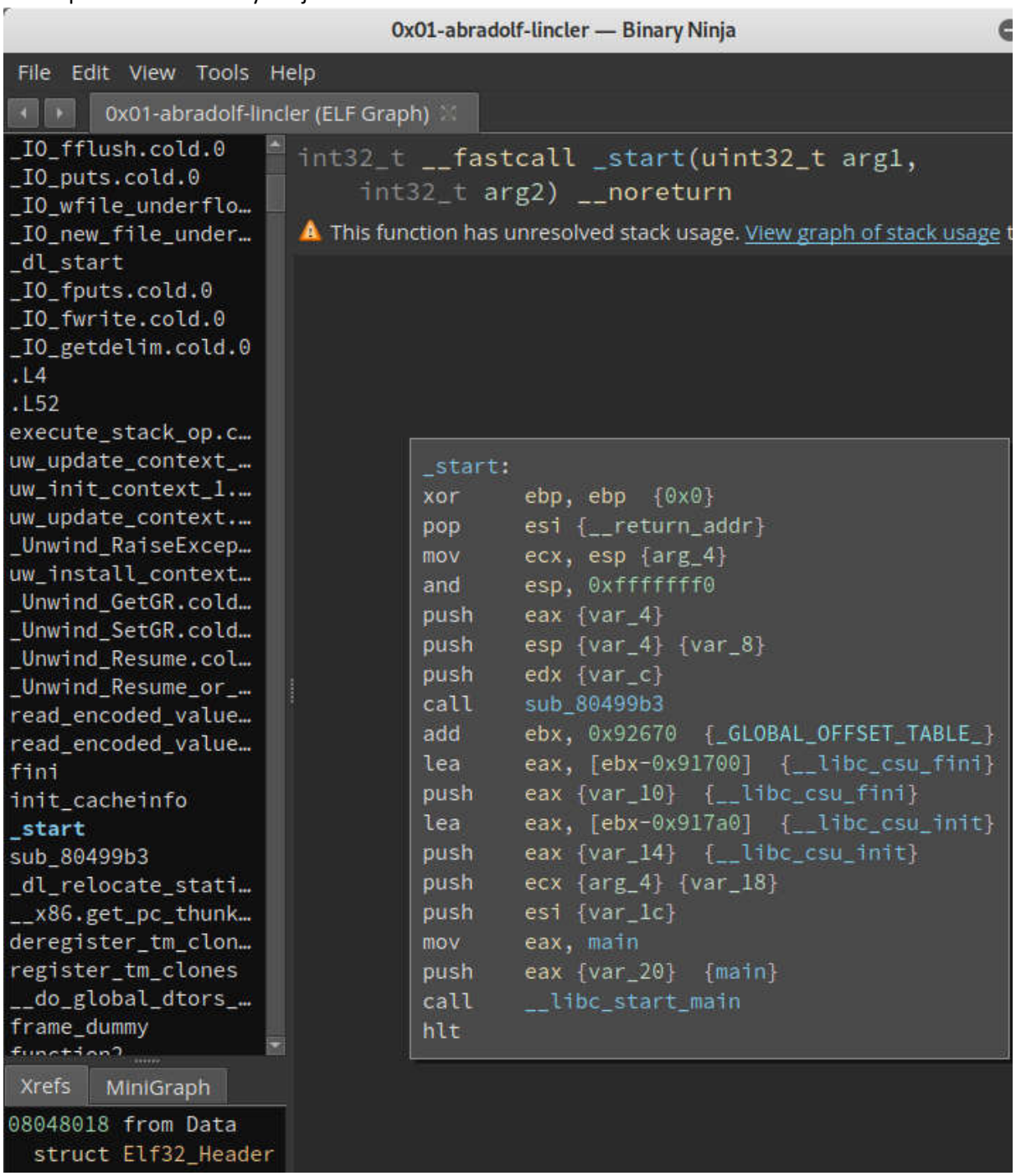
If first time doing Reverse Engineering, downloaded Peda, GDB and PwnTools:



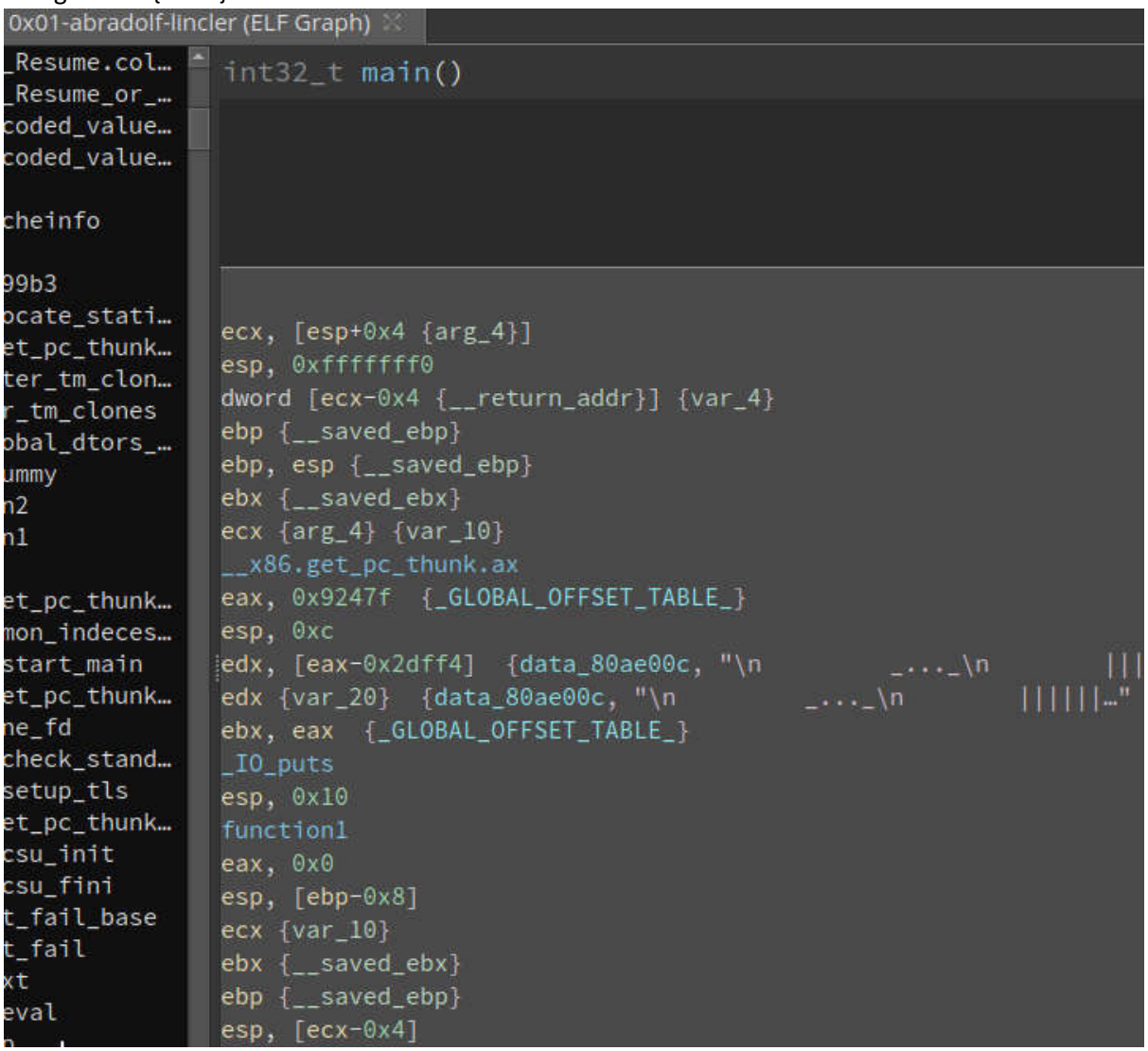
Change into the directory where the file is located and run GDB:



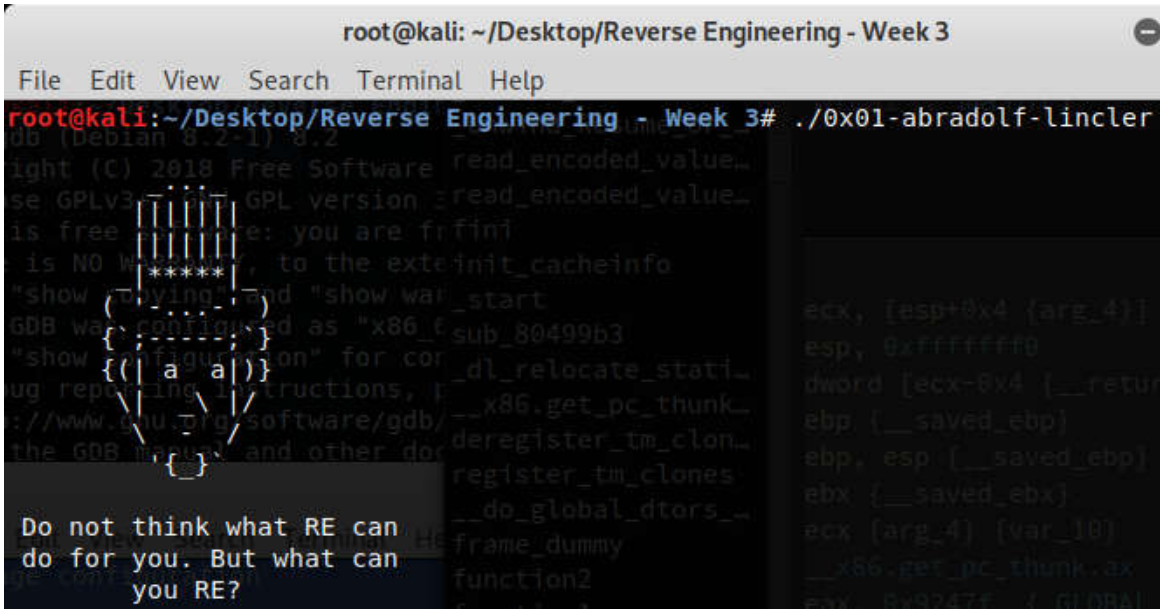
Now open file with Binary Ninja:



Navigate to “{Main}”:

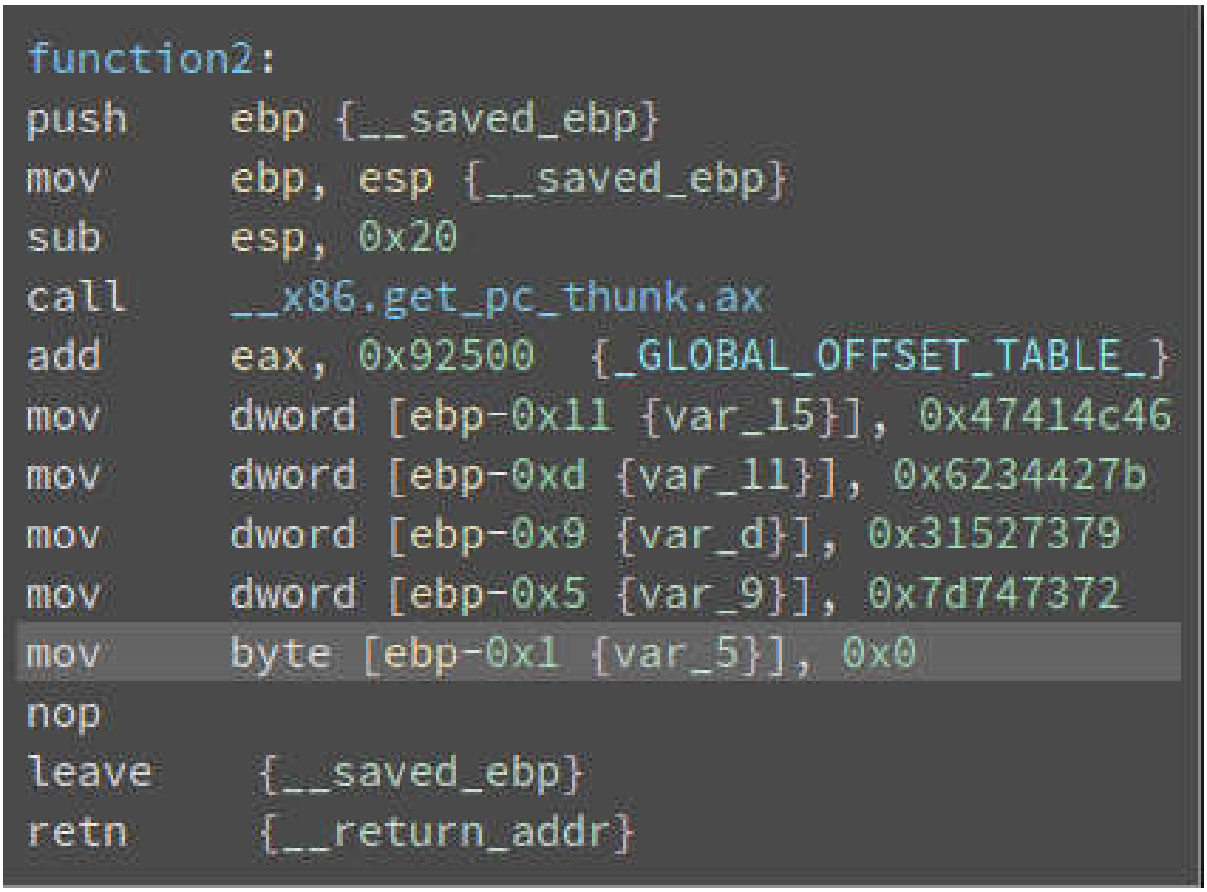


Now that we’re here, we want to see what happens when we run this code, please note that you WOULD NOT do this if you were analysing Malware:

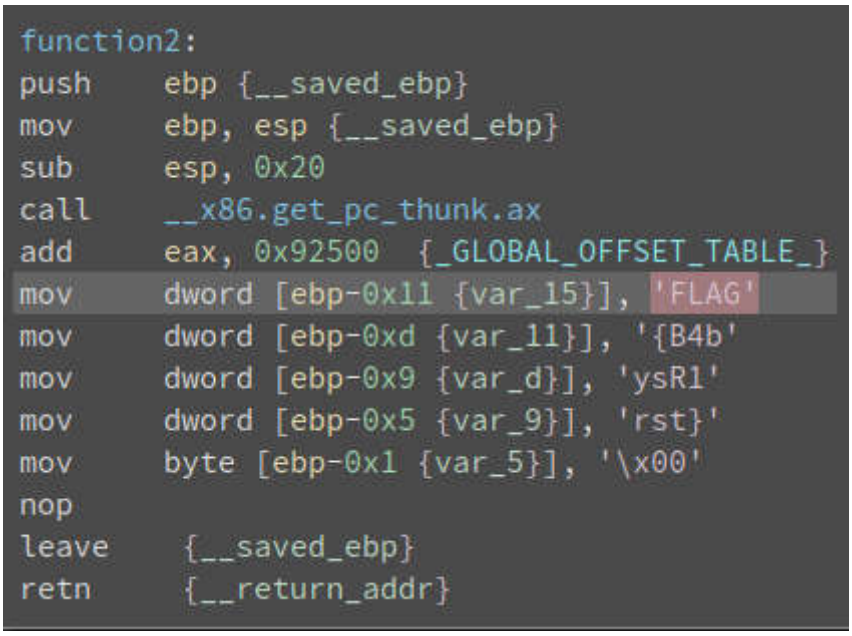


Now that we've ran it, we can see that the ASCII in the "main" forms an artwork. Now we need to find a way to access this function and find out details about this, do this by double clicking "function1" on "main" you will now arrive at the following page:

From this, we can see that there is a for loop (blue line) that adds 0x1 (1) after each iteration. But what stands out to me is the else statement (red line) that calls function two, so I navigated too it:



Here I can see that there is a string (mov) that terminates, the Function 2 is terminated by a Null byte (0x0). So, I decided to covert each line of the string to Display as a Character Constant, this revealed the flag:

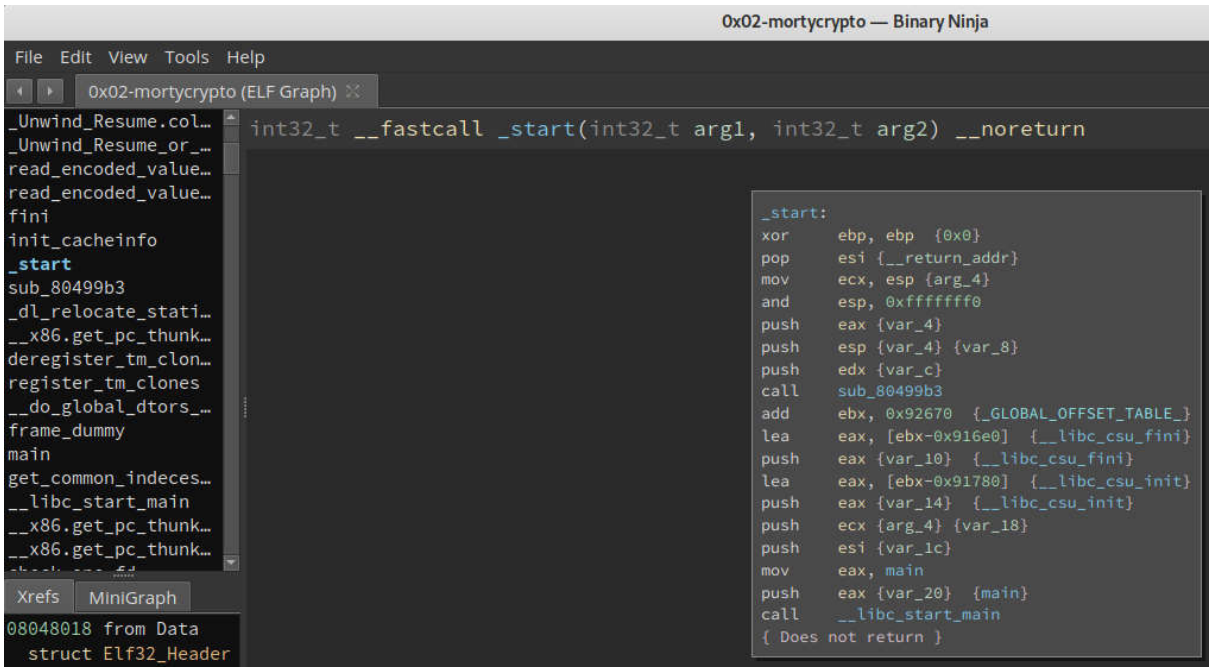


I found this to be a very good introduction into one of the most complex fields in IT, I aspire to perfect these techniques as it is a very useful skill to have as an IT Security Professional.

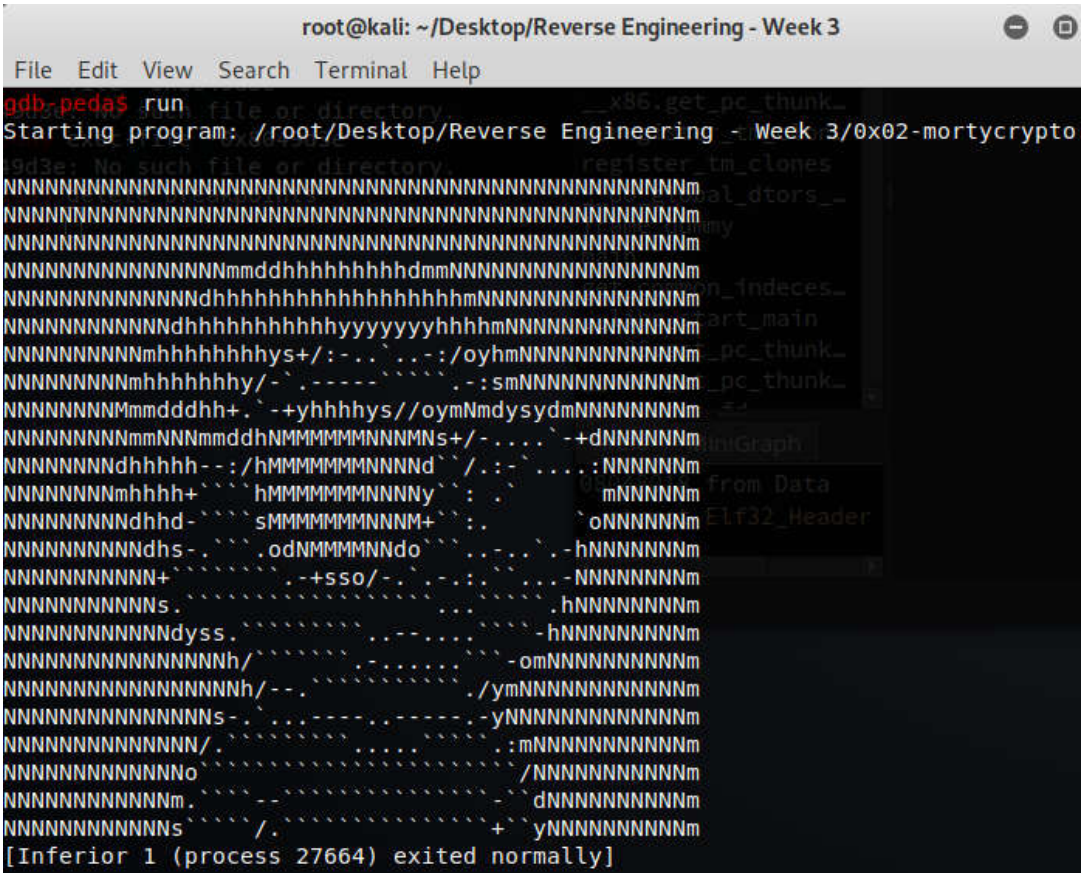
0X02 – MORTYCRYPTO

As we already have all the necessary packages installed from the previous Challenge, we can get straight into this!

Open 0x02-mortycrypto in Binary Ninja:

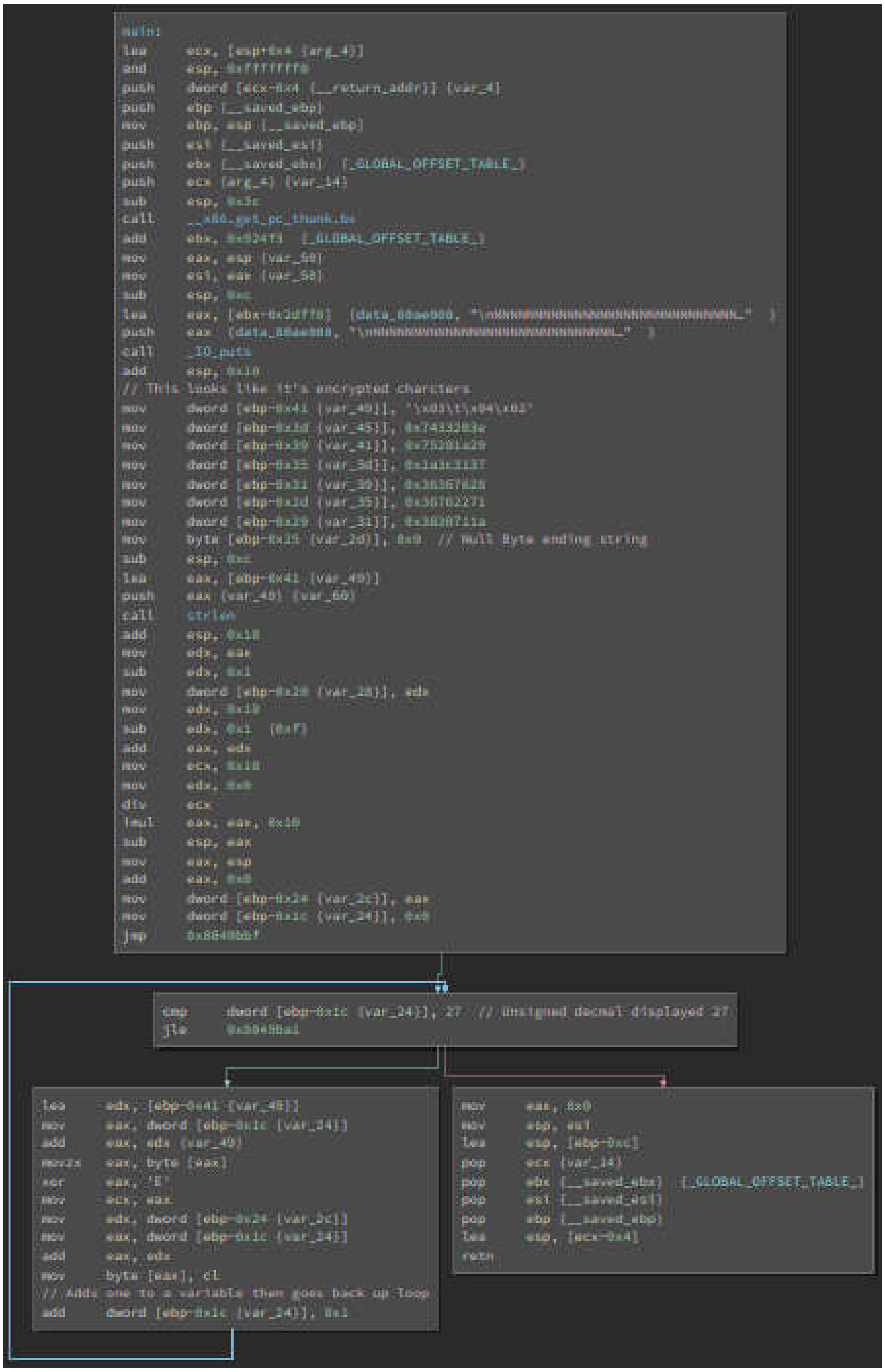


See what happens when your run the file (as mentioned in 0x01, wouldn't do this with Malware):



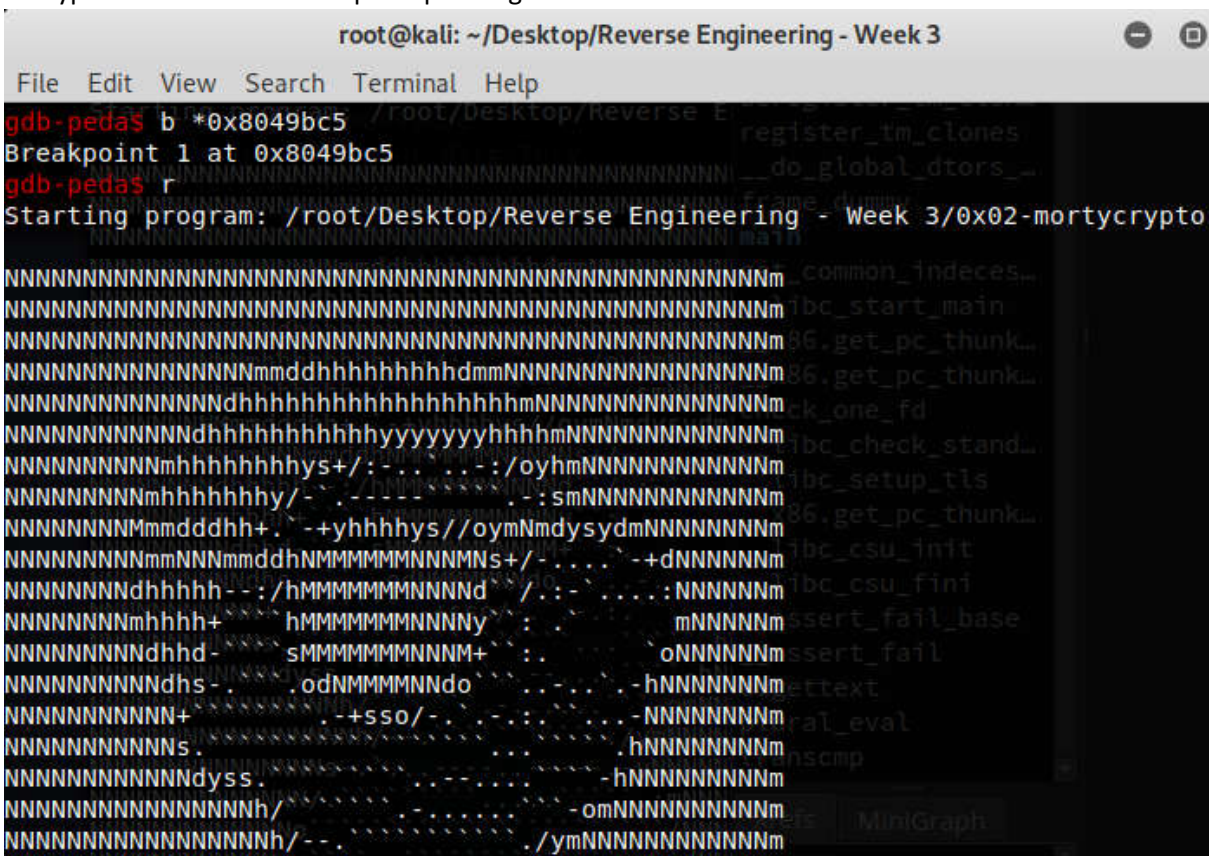
As we can see, it outputs ASCII Art.

Now in Binary Ninja we want to analyse the _start to get an idea of how this code is functioning. The first thing I'd note is that there is an "xor" that forms part of this, xor's are known to be utilised primarily in encryption methods, and with the name "mortycrypto" we can almost guarantee that this code is using encryption of some form. We will now navigate to "{Main}":

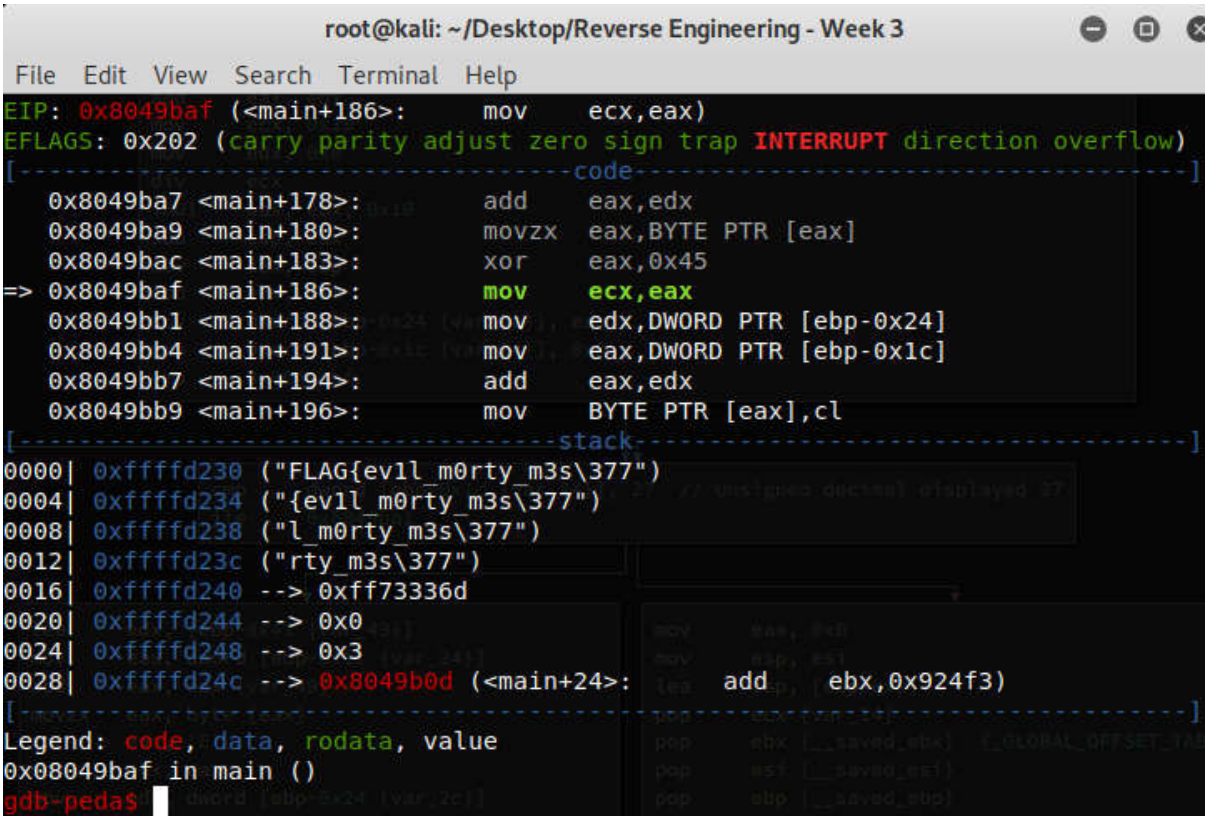


We can see again that in main, ASCII is utilised to create the art, and that there are quite a few strings being inserted in and a strlen being called. It then jumps to an if/else statement with a for loop on the if statement. In the bottom left box, we can see this is where the encryption is being utilised (xor) which I converted to find it to be "E", and at the bottom it adds 1 to the variable before looping up to the cmp, this seems to be looping through the encrypted data. In cmp I decided I wanted to see the hex number converted to an unsigned decimal which revealed 27, this shows that the code will count up to =<27.

I identified 0x8049bc5 to be of interest as there seems to be another string that's being entered and being terminated by a null byte, though as mentioned above (converting 0x8049bc5 over shows encryption. I then set a breakpoint pointing at 0x8049bc5:



Which then displayed the following flag/strings:



REFERENCES

Binary Ninja 2019, *Binary Ninja Demo*, viewed 25 February 2019, <https://binary.ninja/>.