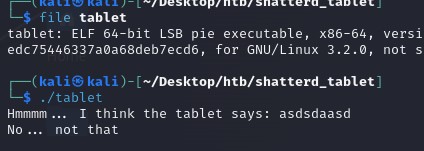
Downloaded and unzipped the files.

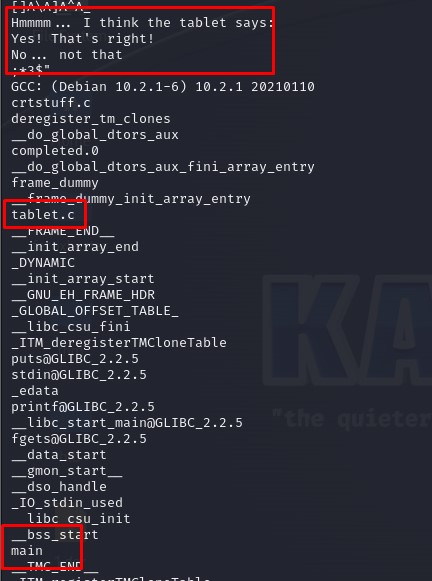


Examined it



Examined it more (strings)

---> produced nothing interesting



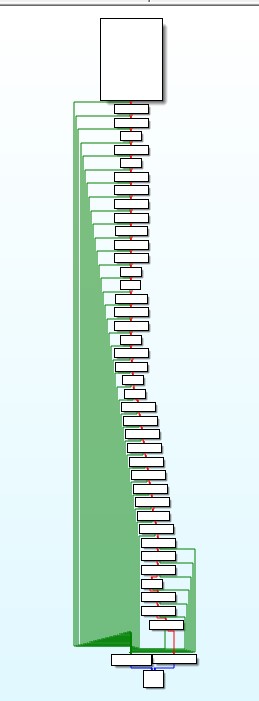


We now know :

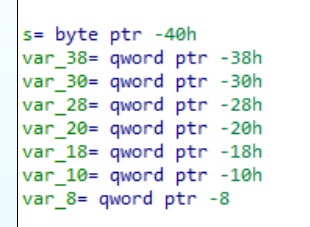
* we have 2 c file (tablet.c & crtstuff.c [which I don’t think is important])
* 1 function (main)
* the strings from before

TIME FOR IDA !!! :)

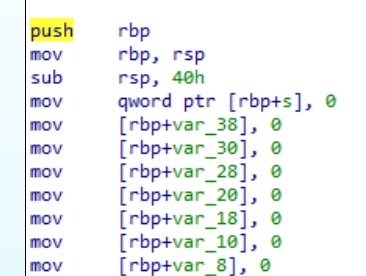
Already looks insane but by quick looks it looks like a manual string comparison



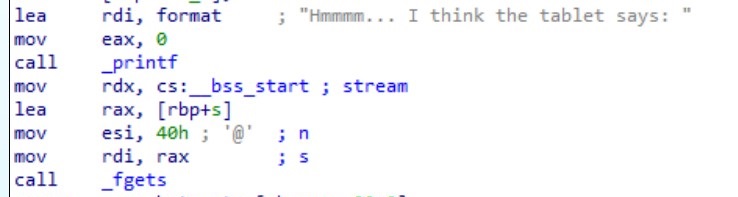
Got to remember these they have really specific values >:)



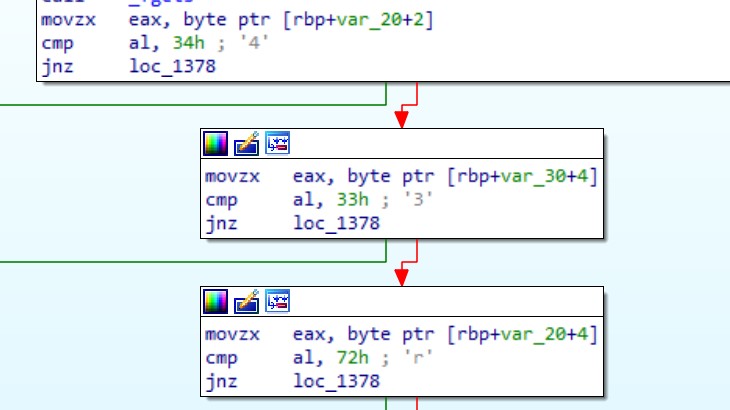
& for some reason later it tries to push 0 to the stack + values from before padding ?? (some kind of reset maybe ??? )



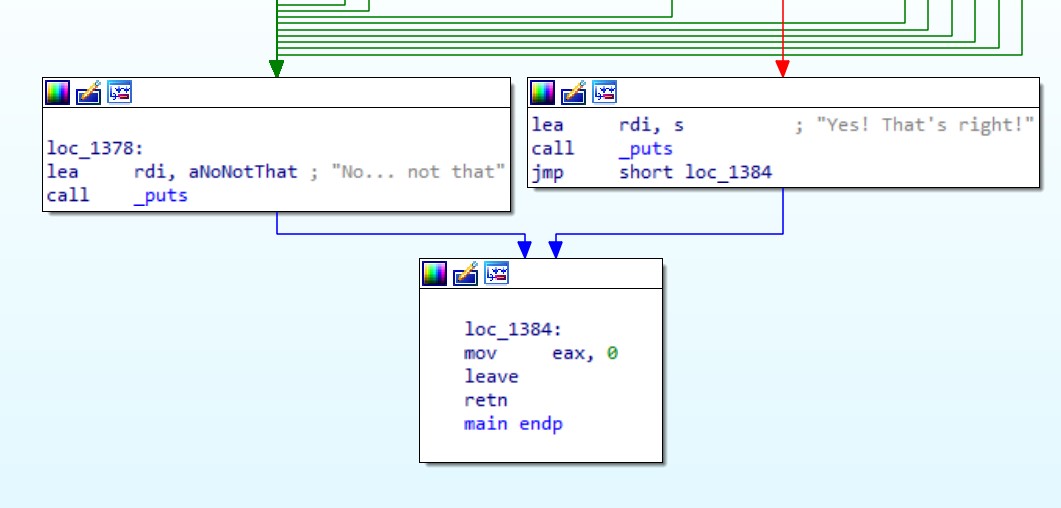
Now this is where the fun begins, we have a print / input function



From now on all we do it move a value from the stack (which was filled with our input value & controlled by a padding [which are the weird variables from before ])



And if every character stand by the correct order print say yes that’s right, else No not that. In the end exit the program.



So ill cut a bit and present the collect text but got this weird text which looks correct but messed up

43rTv0}dr33e1rH3.4{\_.bktrnt.n\_0\_prbpB\_43 which I checked and wasent the correct answer to the binary.

Which than I noticed the weird comparison watch the difference between the pointer value of the stack pointer added to rbp register (Stack base pointer) in each box. it is as if it were trying to check each char in the string array as if it was checked in random locations instead of subsequently. Example snippet in higher languages :

Check (str):

If (str[0] == “a”):

If (str[2] == “c”):

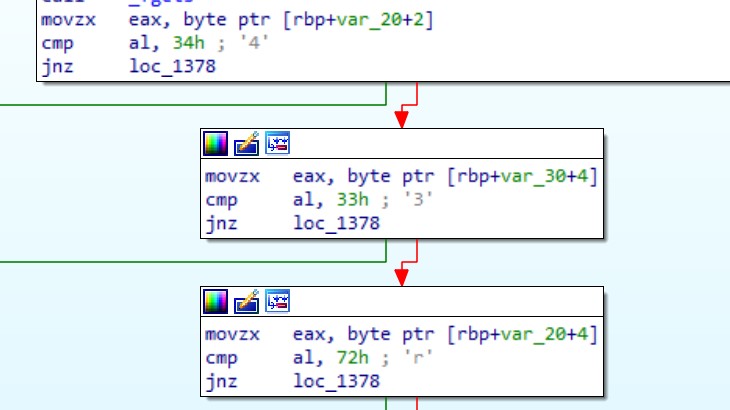
If (str[1] == “b”):

Print (“incorrect”);

Return ;

Print (“incorrect”);

check (‘abc’);

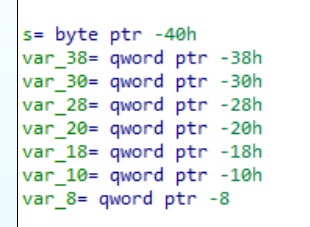




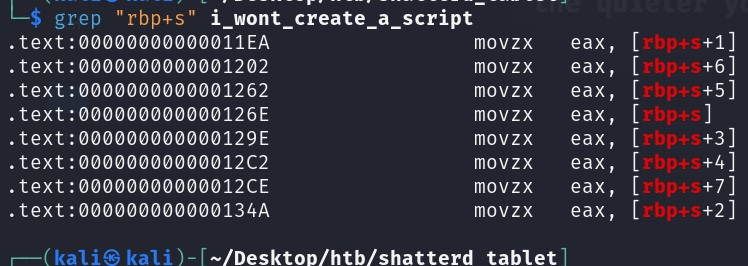
What I did to fix the the messed up string is change to text mode (instead of graph mode in ida)

Copy all lines of code involding the boxes seen above (or in other words the string comparison).

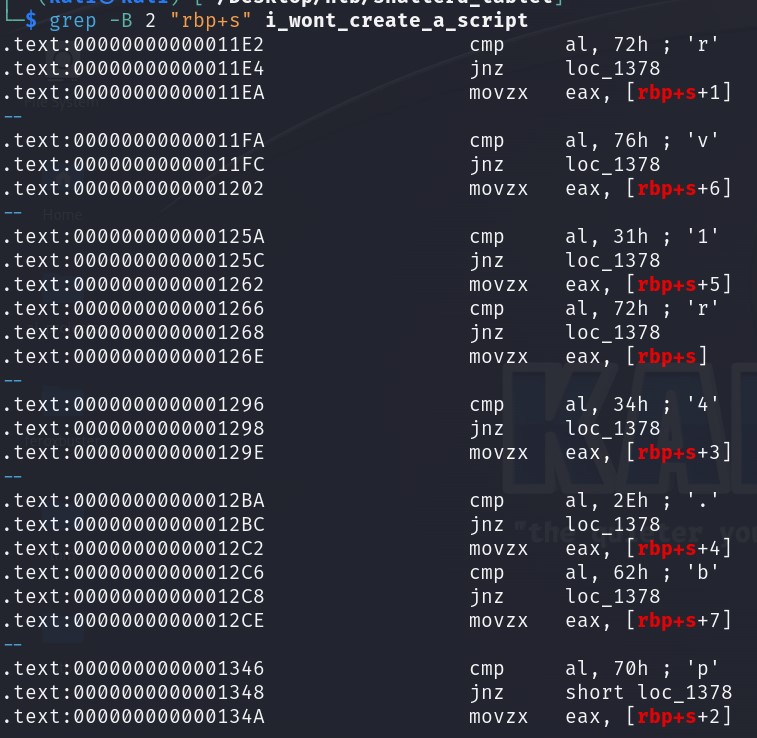
To start with marking an order out of this mess I started by following the login of the stack address flow. What I mean is that from the stack pointer (rbp register) I would always jump a fixed amount of addresses which are DRUM ROLLLLL the weird variables from before. So I just started with s since it has the highest value of all of the variables.



As you can see in the grep you have a fixed amount after the special variable padding to the stack pointer. Which if we too the rbp + s value will lead us to DRUM ROLLLLL the letter H which means Im in the right direction to the flag since all flags look like this 🡪 HTB{ … }



From now on its only ants work since I\_wont\_create\_a\_script for this…



And the flag is 🡪



Walkthrough by Daniel Fligel aka Namx :)