Shop

Source:

https://play.picoctf.org/practice/challenge/134?category=3&page=1

Best Stuff - Cheap Stuff, Buy Buy Buy... Store Instance: source. The shop is open for business at `nc mercury.picoctf.net 3952`.

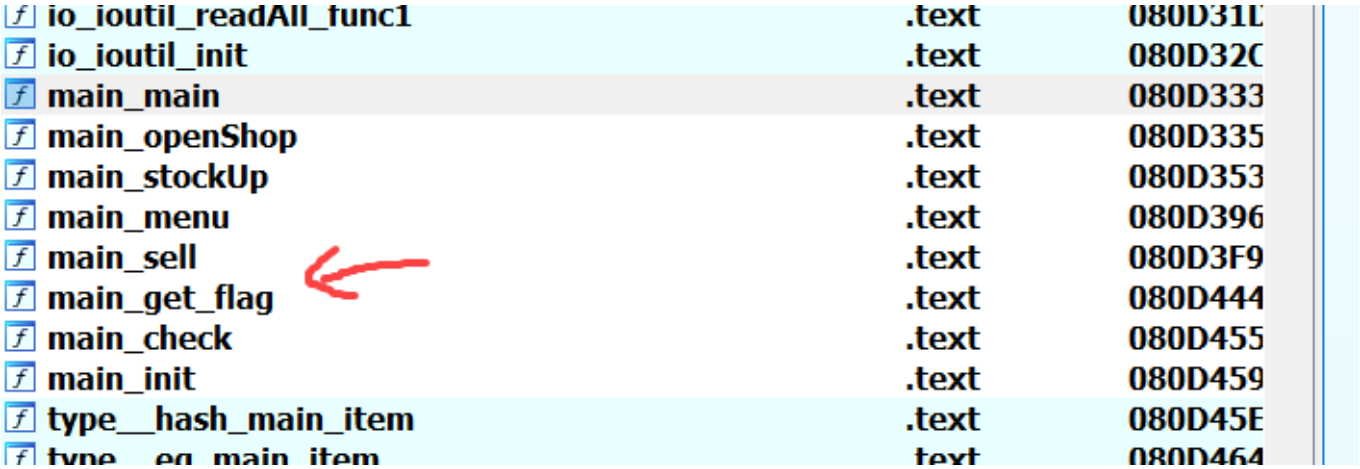
Report of findings:

First things first, let's see what file type we were provided and perform some static analysis of it.

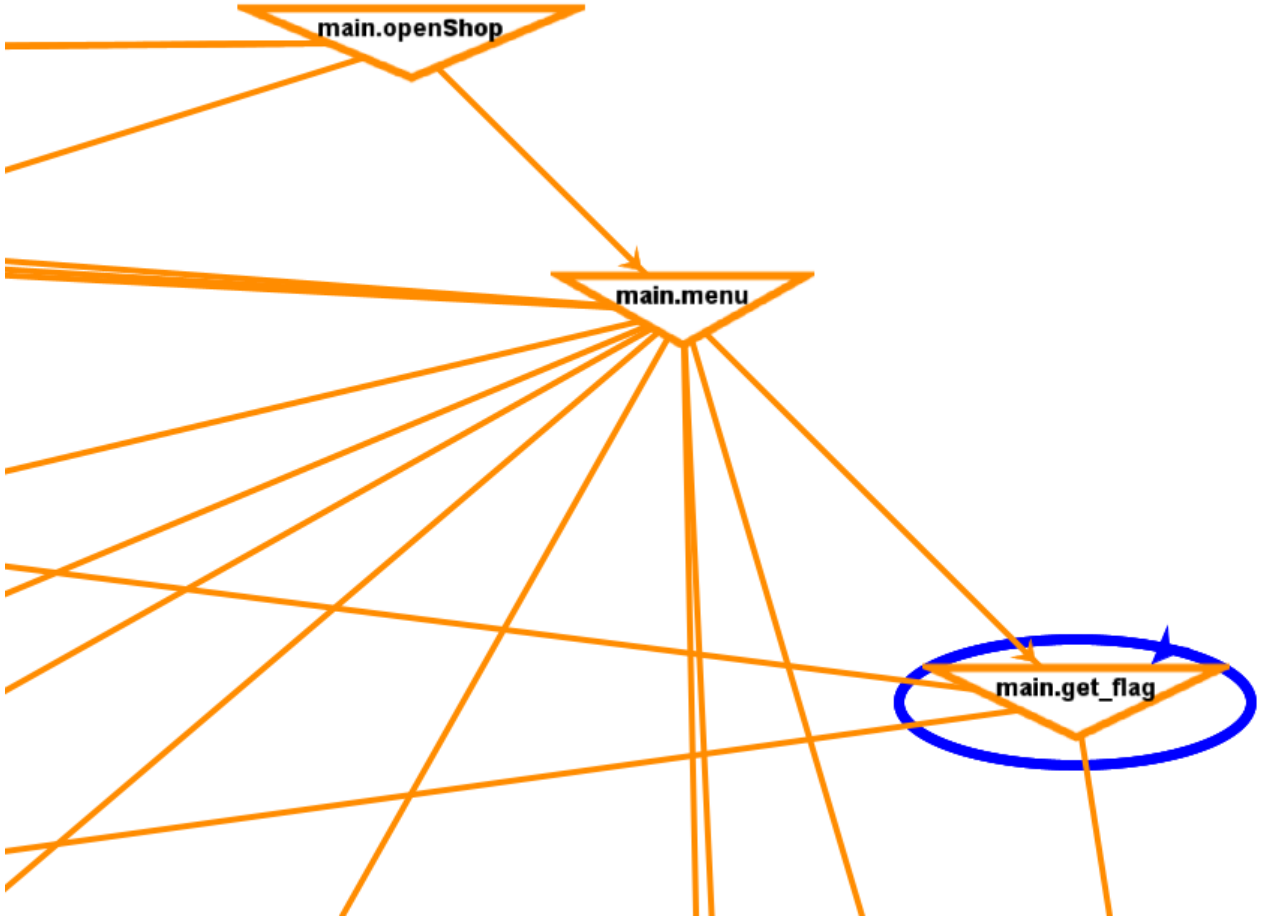
The Linux `file <filename>` command says its an x86 ELF binary developed with golang

```
./source: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically
linked, Go
BuildID=Wq2z6hkBrrAovu6w8dMb/chyUPt3_NgRB5zVfMpf8/99tYvdYHy3xNdHp4wuxA/ClEGFX9e3WU6
qjzPxg9K, with debug_info, not stripped
```

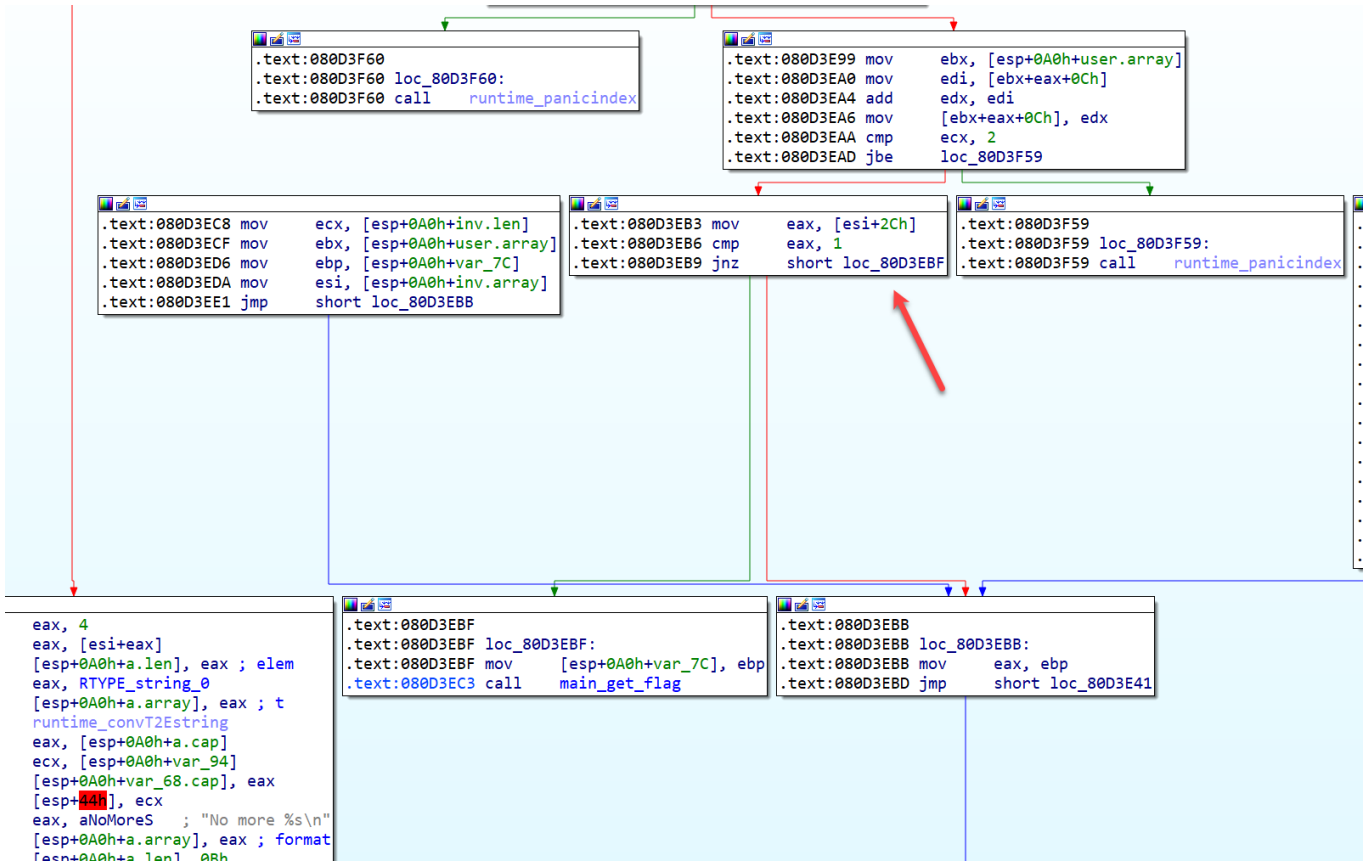
loaded the program on IDA Pro to check for all the different components. Looking at the functions, one of them (`main_get_flag`) seems more interesting than the others:



However, once I build the function call path to get to this function, I though at first that it would take an special case to reach to this code branch.



Anyways, that's was just a quick thought which was later confirmed looking at the graph view of the `main_menu` function in IDA Pro. In particular, right away I noticed that this variable needed to not 1 in order to satisfy the conditional jump and therefore, get to our function of interest.



Now the next step is to debug the program and figure out what this variable is. The way I setup this debugging project is using IDA Pro remote Linux debugger. I uploaded the binary to the target Linux VM system and also have a copy of the Linux debug server available on `C:\Program Files\IDA Pro 8.x\dbgsrv`. On the target system where this binary will execute, I just run:

```
# ./linux_server
IDA Linux 32-bit remote debug server(ST) v8.3.28. Hex-Rays (c) 2004-2023
2023-12-15 12:12:42 Listening on 0.0.0.0:23946...
```

Once this is running, I go back to my analysis VM, load the same binary, select the debugger to `Remote Linux Debbuger`, go to the menu Debugger -> Process Options and in the new windows, configure the Application, Input file and directory using the target location of the binary file; then write the target system IP address and the port that linux_server is listing on.

Assuming the above is properly setup, I placed a few brake points along the execution path to see how the program behaves and identify the areas of importance.

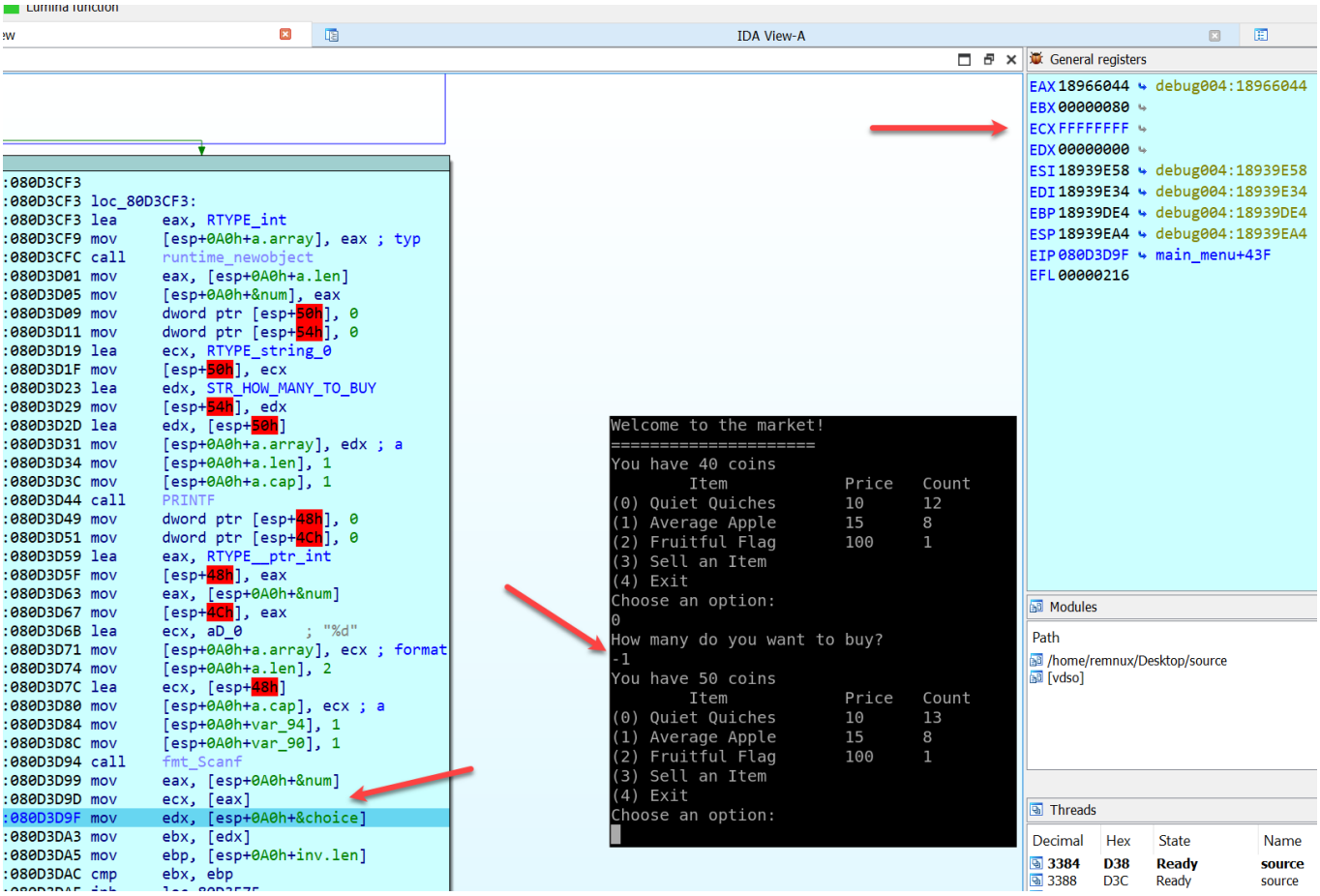
The program is a very simple 3 items cart that you can buy and sell at the same price. From the start of the program, you are given only 40 coins

```
Welcome to the market!
======================
You have 40 coins
        Item            Price   Count
(0) Quiet Quiches       10      12
(1) Average Apple       15      8
(2) Fruitful Flag       100     1
(3) Sell an Item
(4) Exit
Choose an option:
```

from which you can only buy item 0 or item 1, Quiches and Apples respectively. However, not enough coins to ever buy Fruitful Flag, I wonder why that is? Also, I see that there is only 1 Fruitful Flag so maybe this value has to do with the branch check we spotted earlier to get into the `main_get_flag` function. However, we have a problem, how can we ever reach to the amount of coins we need in order to buy a Fruitful Flag, at the end, it costs 100 coins and we are only given 40 coins. Is it possible that we can buy items that gives us coins, meaning, if we buy an 1 item @ 10 coins, we then have 30 coins remaining but, if we buy -1 item @ 10 coins, maybe that gives us 10 coins and then we will have 50 coins? Ok, let's test those boundaries:

In IDA Pro, I start debugging, hit the brake point right before deciding which item to buy, make the selection (0) on target system.

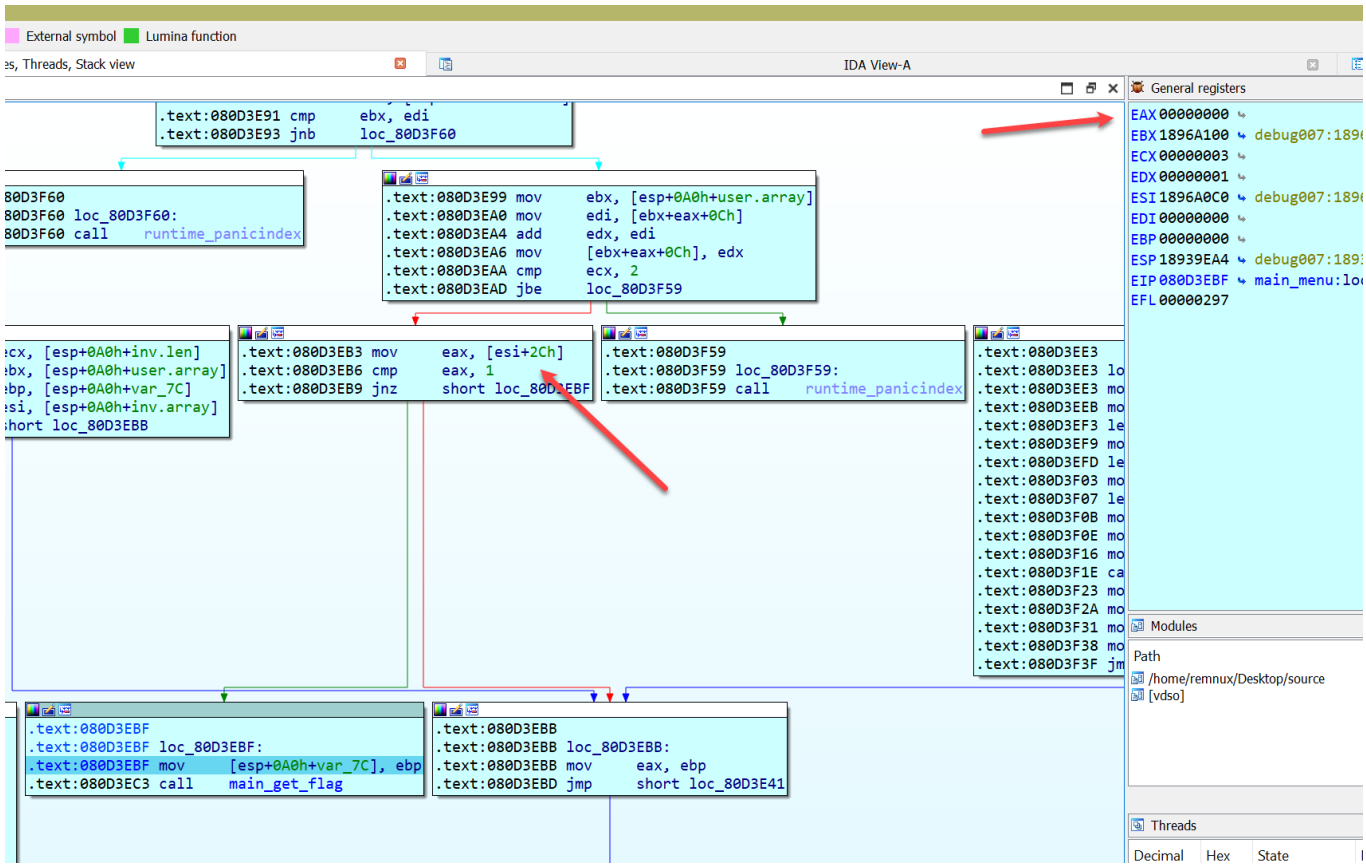
How many items do I want to buy? how about -1, let's see what happens. That should give me +10 coins for a total of 50 coins, right?



Well, indeed. I can now buy -5 or less to give me just enough to buy my Fruitful Flag. Lets do that and maybe that gives us our flag ?

```
You have 50 coins
        Item                Price   Count
(0) Quiet Quiches           10      13
(1) Average Apple           15      8
(2) Fruitful Flag           100     1
(3) Sell an Item
(4) Exit
Choose an option:
0
How many do you want to buy?
-5
You have 100 coins
        Item                Price   Count
(0) Quiet Quiches           10      18
(1) Average Apple           15      8
(2) Fruitful Flag           100     1
(3) Sell an Item
(4) Exit
Choose an option:
```

Nice, now let's buy the item we could not buy before and see what happens.



There it is, EAX is now holding the amount of Fruitful Flag for sale, which is now 0 after the purchase and enough to satisfy the branch check and get to our `main_get_flag` function. Let's now get into that function and see if we need to do more work.

Well, looks like is reading some data from a file `flag.txt` in the same directory where the ELF binary is run from. There is another function (`main_check`) that we might need to investigate but for now, let's ignore it and continue execution and see what happens:

```
        Item            Price   Count
(0) Quiet Quiches       10      12
(1) Average Apple       15      18
(2) Fruitful Flag       100     1
(3) Sell an Item
(4) Exit
Choose an option:
2
How many do you want to buy?
1
Flag is:  [112 105 99 111 67 84 70 123 98 52 100 95 98 114 111 103 114 97 109 109 101 114 95 57 99 49 49 56 98 98 102 12
5]
```

Indeed, after buying that Fruit Flag, we got the above array of integers which we can try decode in CyberChef and see what get get:

Well, I got what I was looking for, right?
Looking forward for the next challenge!