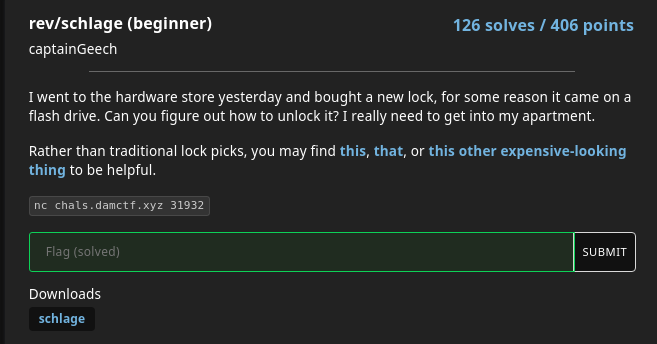
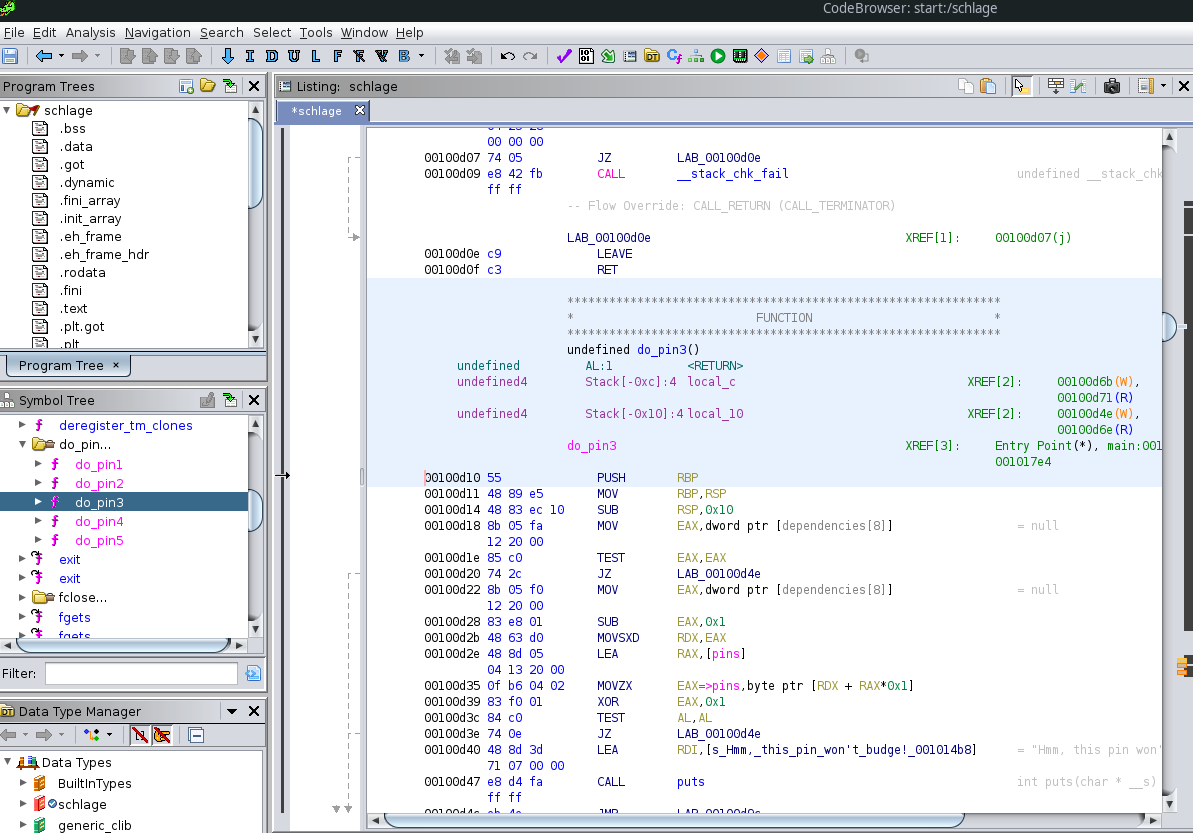
Challenge description:



The links “This”, “that” or “this other expensive-looking thing” lead to Ghidra, Binary Ninja, or IDA pro (guess which one is “this other expensive-looking thing”.. lol).

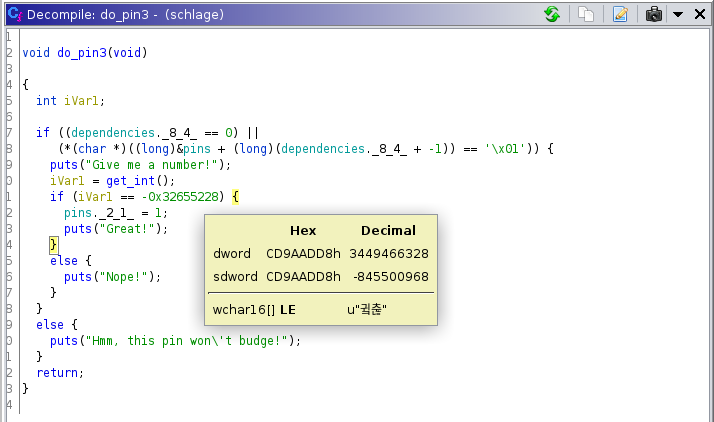
Initial look at the program with strings didn’t produce anything too useful, besides giving a general idea of what it would print out, and seeing some error or success messages, so I ran the program to see what would happen.



You can see on the left side, under “Symbol Tree” you can eventually find a group of functions named do\_pin1 – do\_pin5. It’s a safe guess these contain the code for each “pin” we need to “open”.

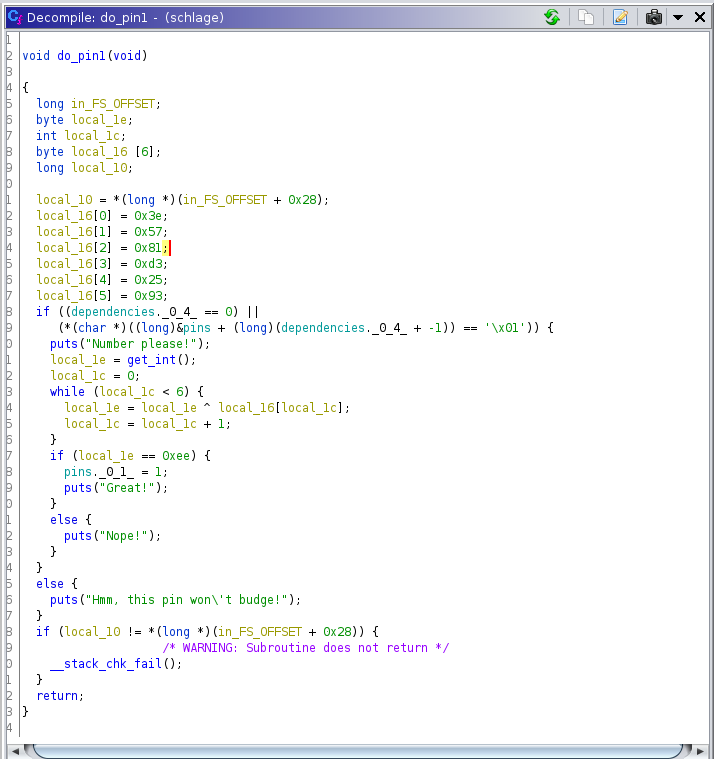
**Pin3**

After a bit of trial and error, I found that you had to “open” the pins in a specific order, starting with 3. This is where Ghidra comes in.

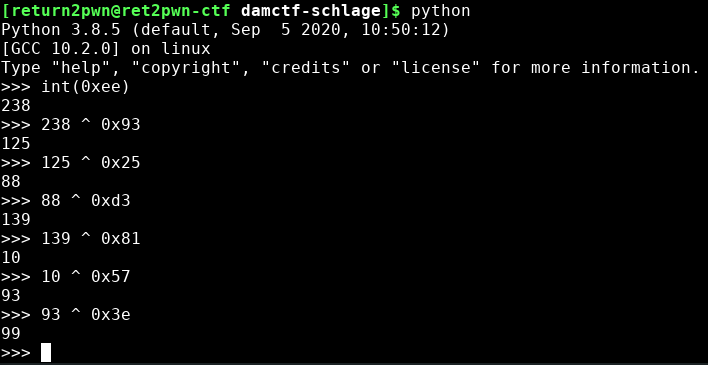


Ghidra’s disassembly of the do\_pin3 function was pretty straightforward, it takes user input and compares that input to the hex value -0x32655228. Ghidra will translate this to Decimal if you hover your cursor over that number. So we enter 3, and then 3449466328 in to the program to open pin3.

**Pin 1**

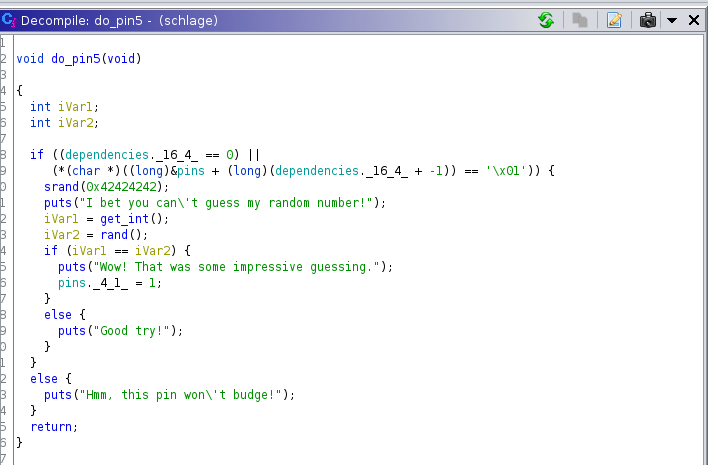


This function appears to take an integer, and it goes through a loop 6 times, xoring your input with local\_16[0] → local\_16[5]. After this, it compares the number you put in (local\_le) with 0xee (238). The problem with xor is that it can be reversed easily, if we take the end result, 238, and xor it with each number in the array local\_16, starting with the value at local\_16[5], we end up with 99, which is the input needed to open pin1. I’ll show you in a python interpreter.



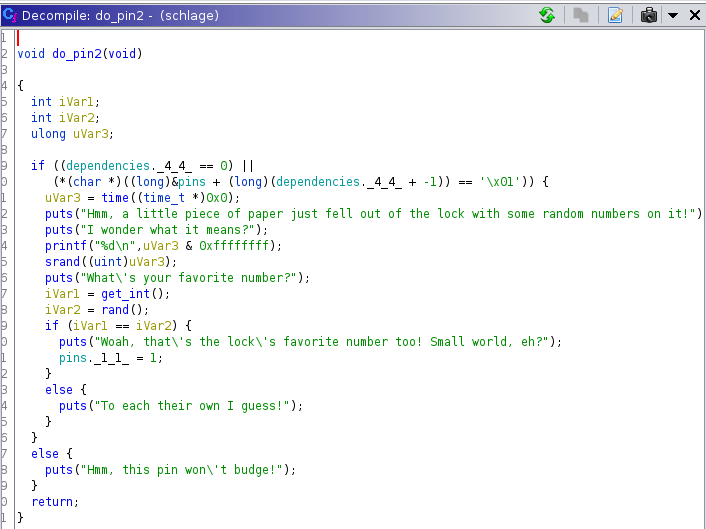
**Pin5**

After more trial and error, I found that pin5 was the next one we needed to open.



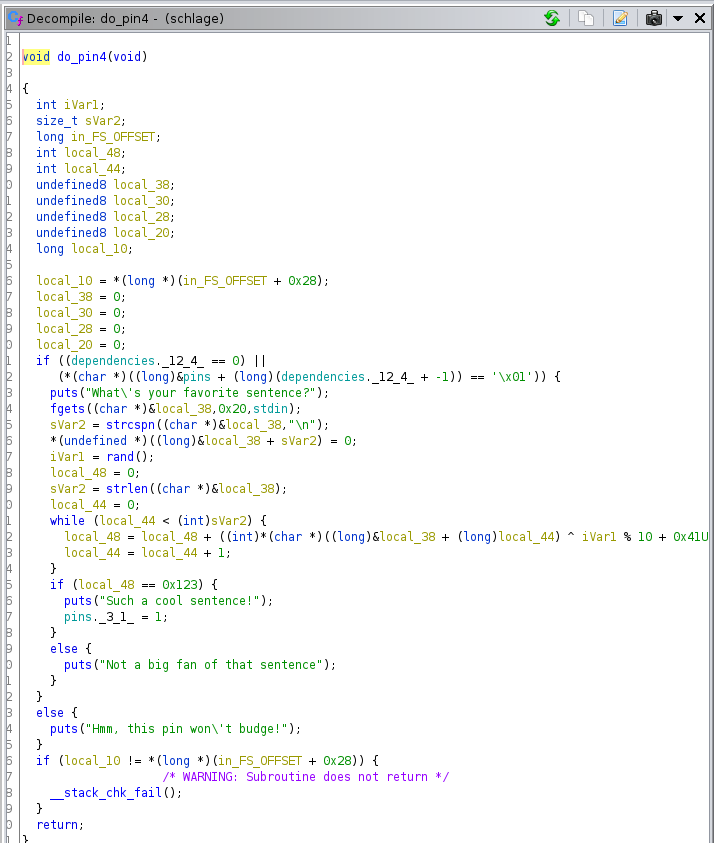
This function takes an integer from the user, generates a “random” integer with rand(), and compares the two. srand() sets a seed value that rand() will use to generate a “random” number. We can make our own rand() program with this seed value, and the result of rand() in our program will be the same as rand() in the challenge binary. Best practice for rand() is to use the system time as the seed value. After running your own binary with rand(), you’ll see that the result is 1413036362, so we just enter that in for pin5.

**Pin2**



This level appears to set a seed value, print that value out (uVar3), generate a random number, and get an int from the user. If the random number and the user input(iVar1) match, the pin will “open”. The solution is pretty similar to pin5. This seed value actually seems to use the system time, so it will change each time the program is run. Using the given seed value as input in your own c program will produce the random number needed, just like it does for pin5. Input the random number that our program prints out to “open” the pin.

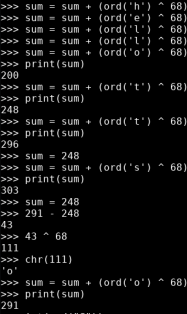
**Pin4**



This function takes an input from the user (local\_38), produces a random number (iVar1), and goes through a while loop, xoring each letter of your input with the result of (iVar1 % 10 + 0x41). After the while loop, the sum (local\_48) is compared with 0x123 (decimal 291), and if it’s equal, you get the flag.

So one important note for this is that the seed value from srand() is global, and the last time a seed value was assigned was in pin2. So a second random number is generated with that seed value, which is used for pin4 (I modified my C program to print out a second random number using that seed value for this part of the challenge). Through trial and error I found that the result of (iVar1 % 10 + 0x41) often fell between the numberd 65-75, hitting 68 pretty often. I refer to this value in my python script as xor\_var.

My solution is probably a little unconventional, I decided to focus exclusively on the scenario where xor\_var == 68. From here, I started with the word “hello” and found that if you run the loop with that word, you end up with 200. I figured I could get a little closer to 291 so I added a ‘t’, running the loop with “hellot”. This resulted in 248. Heres a screenshot of a python interpreter where I put this together.



From here, I needed one more letter from 248 to get my word to come out to 291. So going back to how we could reverse xor in pin1, I did the same here. First I subtracted 248 from 291 to get the difference. I took that difference and xored it with 68 (xor\_var) to see what I would need to xor with 68 to add 43 to my current sum of 248, getting me to 291. That result was 111, or the letter ‘o’. So, if xor\_var == 68, “helloto” will get us the flag.

