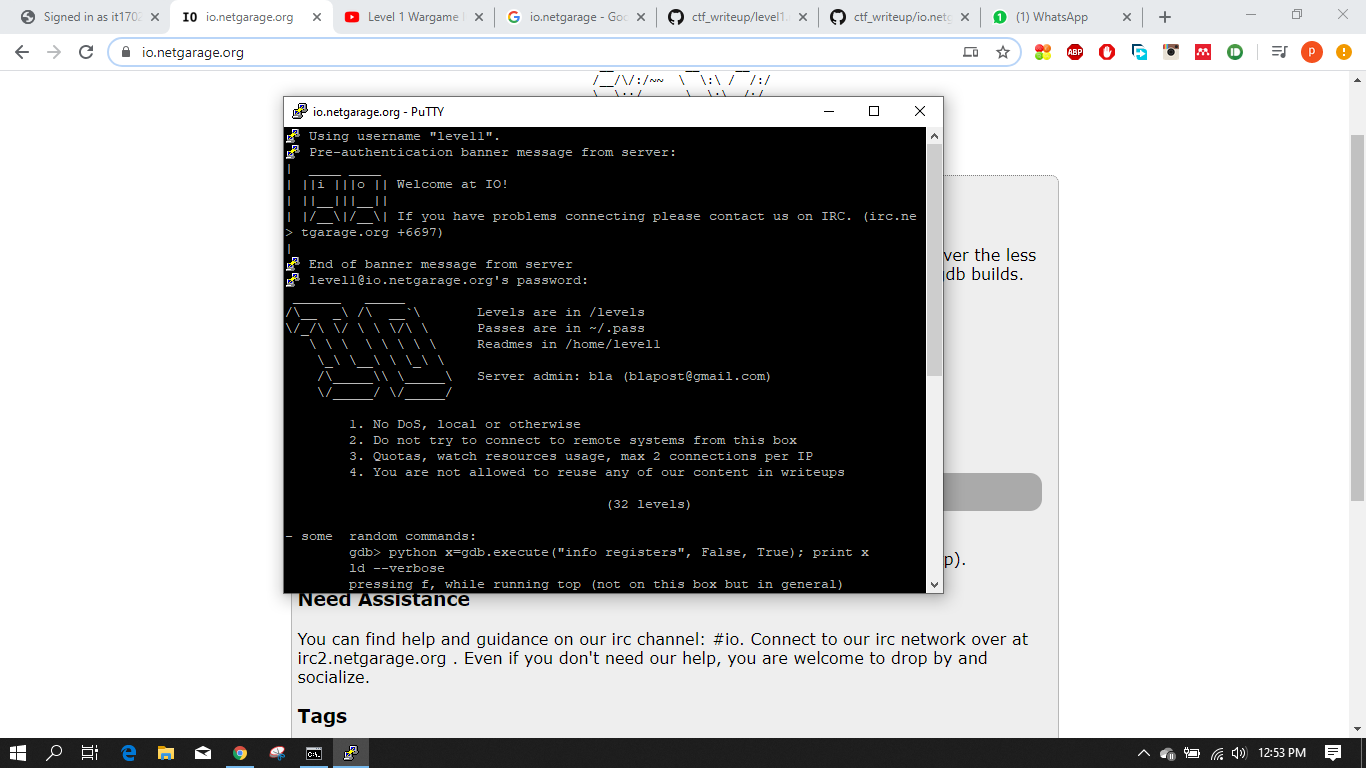
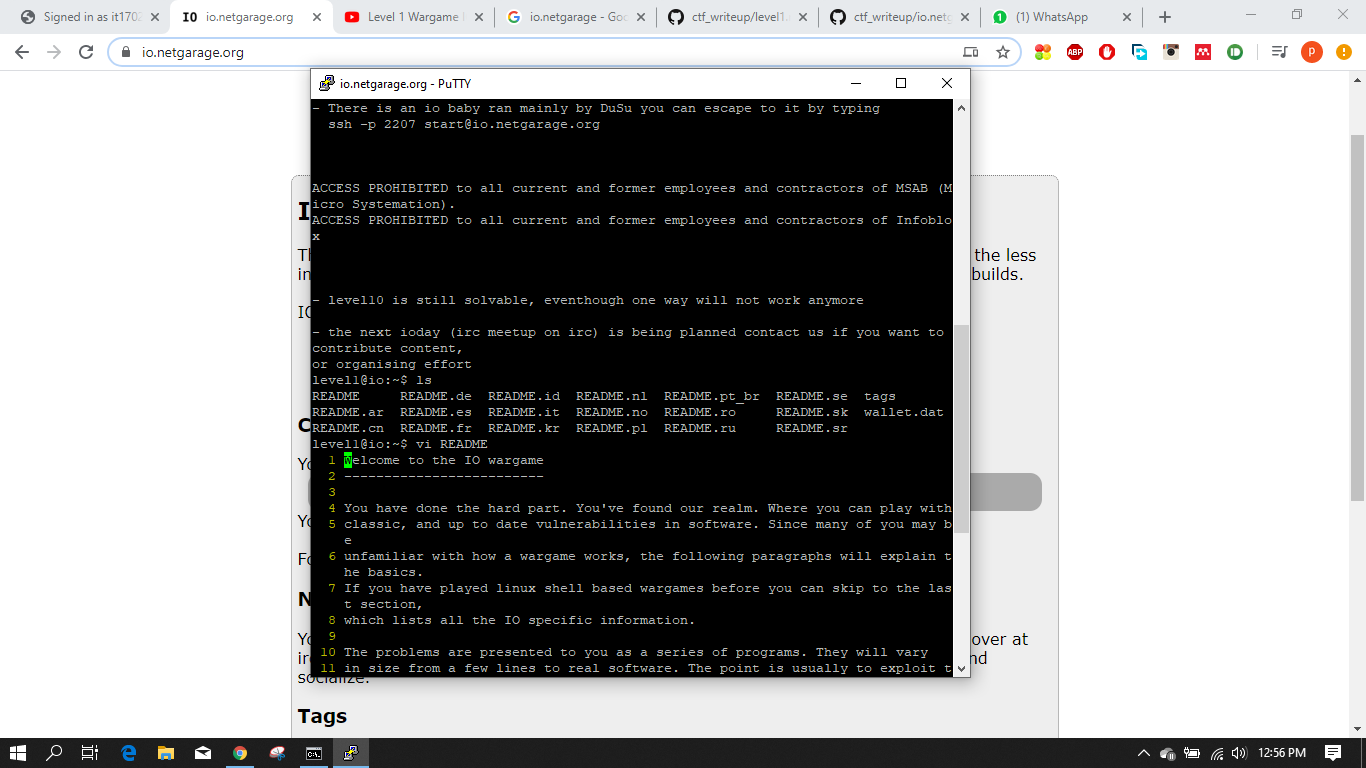
**Level 01**

|  |  |
| --- | --- |
| **title** | **link** |
| Level 1 | ssh level1@io.netgarage.org |

Running level01 ask for a 3 digits number which we will find inside the binary.



You can find a “Readme File”. Using vi command you can see the instruction of that file.



We can see that there is no source code for level01. This means that we have to reverse engineer the logic to get to level2.

From this simple file command, we can see that it is Executable Linkable Format. 32 bits architecture and written for the intel CPU.

A screenshot of a computer screen

Description automatically generated

Running the strings utility

A screenshot of a computer screen

Description automatically generated

Launching the program under gdb:

A screenshot of a computer screen

Description automatically generated

Show the disassembly code for the main function (entry point of the program)

The first call print the question with puts. The second one asks for the user input (the password. Then the program compares a fixed value with the value of the register eax.

This value is a hexadecimal value, we can display its decimal value with p in gdb:

A screenshot of a computer screen

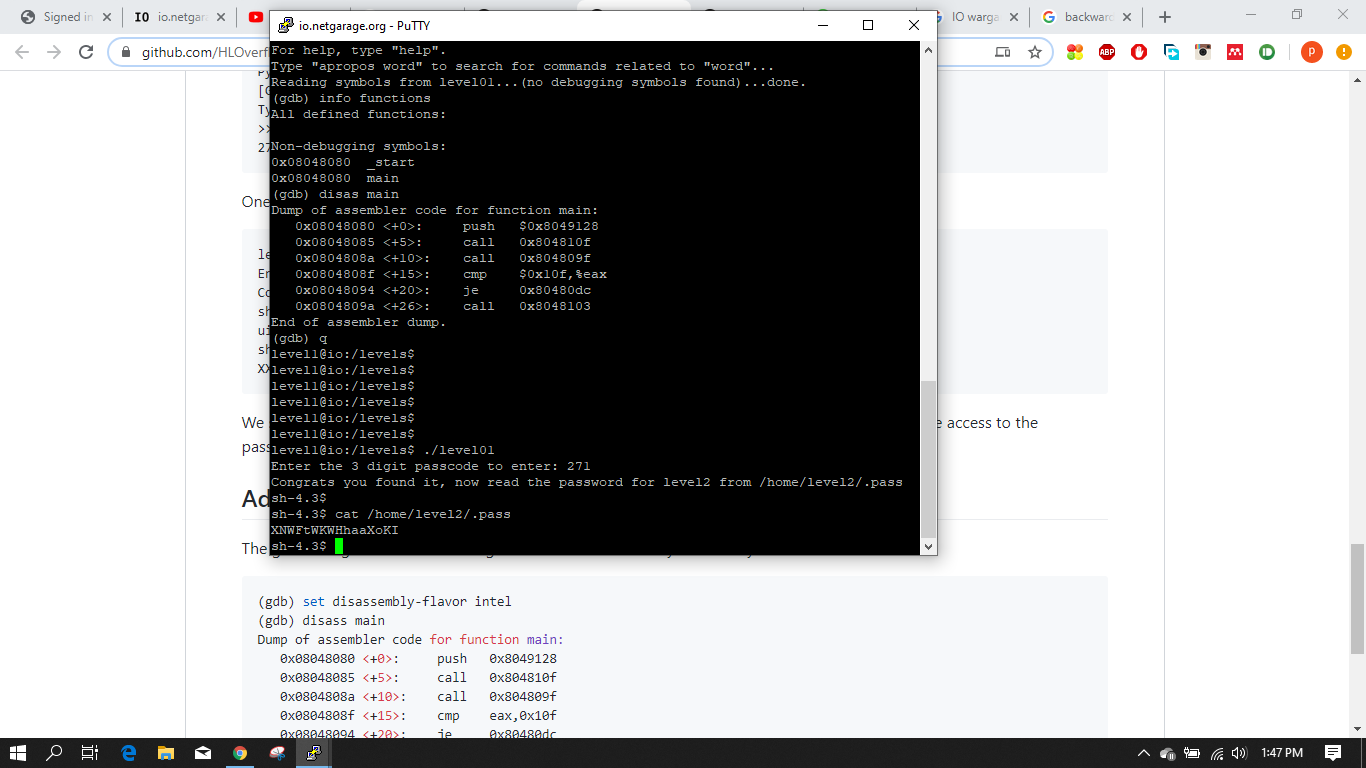
Description automatically generated

So apparently, we are comparing the entered value, which is stored in the eax register.

je will jump to the label if the values are equal, since this jump is to the YouWin section, we can assume the password is 271.

Then using cat /home/level2/.pass command, you can see the password for level 02.

The password for level2 is **XNWFtWKWHhaaXoKI** .



**Level 02**

|  |  |
| --- | --- |
| **title** | **link** |
| Level 2 | ssh level2@io.netgarage.org |

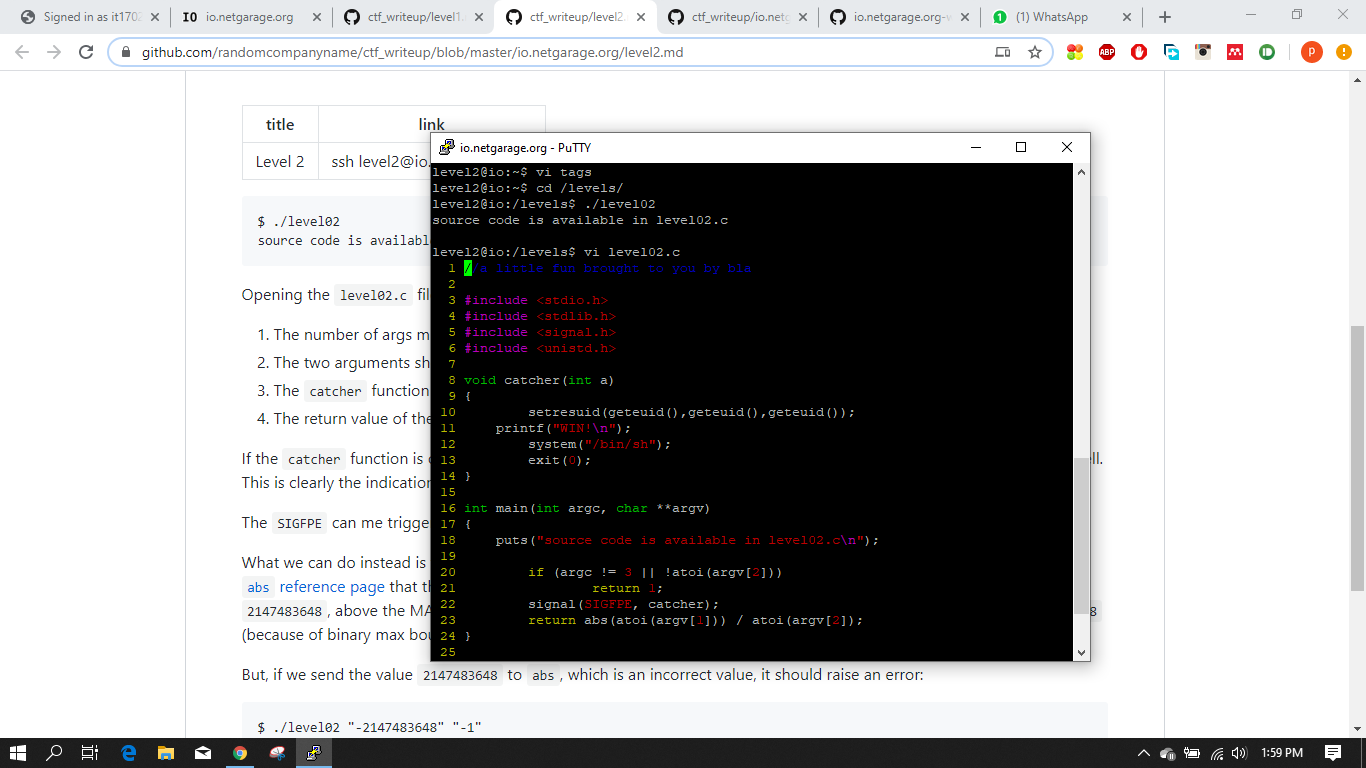
A screenshot of a computer screen

Description automatically generated

Opening the level02.c file we can see:

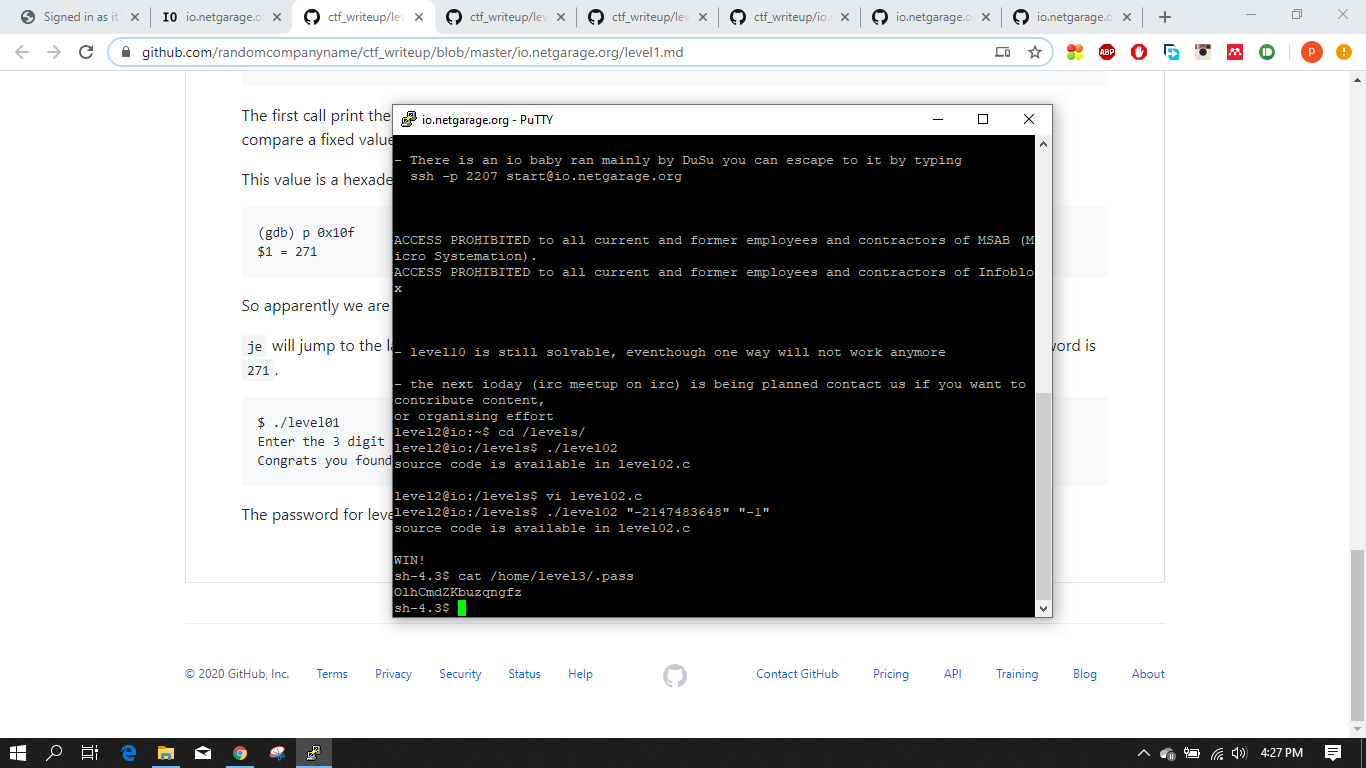
* The number of args must be 2, (argv[0] being the caller's name).
* The two arguments should be numbers
* The catcher function will be called on the event SIGFPE (launched for example for a division by 0)
* The return value of the function is argv[1]/argv[2]

If the catcher function is called, it will set the current user identity, and print a win message, before spawning a new shell. This is clearly the indication that we need to raise a SIGFPE exception.



What you can do instead is try to use an integer value outside of the bound of the integer definition. You can see on the abs reference page that the most-negative value to be out of range is -2147483648, because this will convert to 2147483648, above the MAX\_INT value.

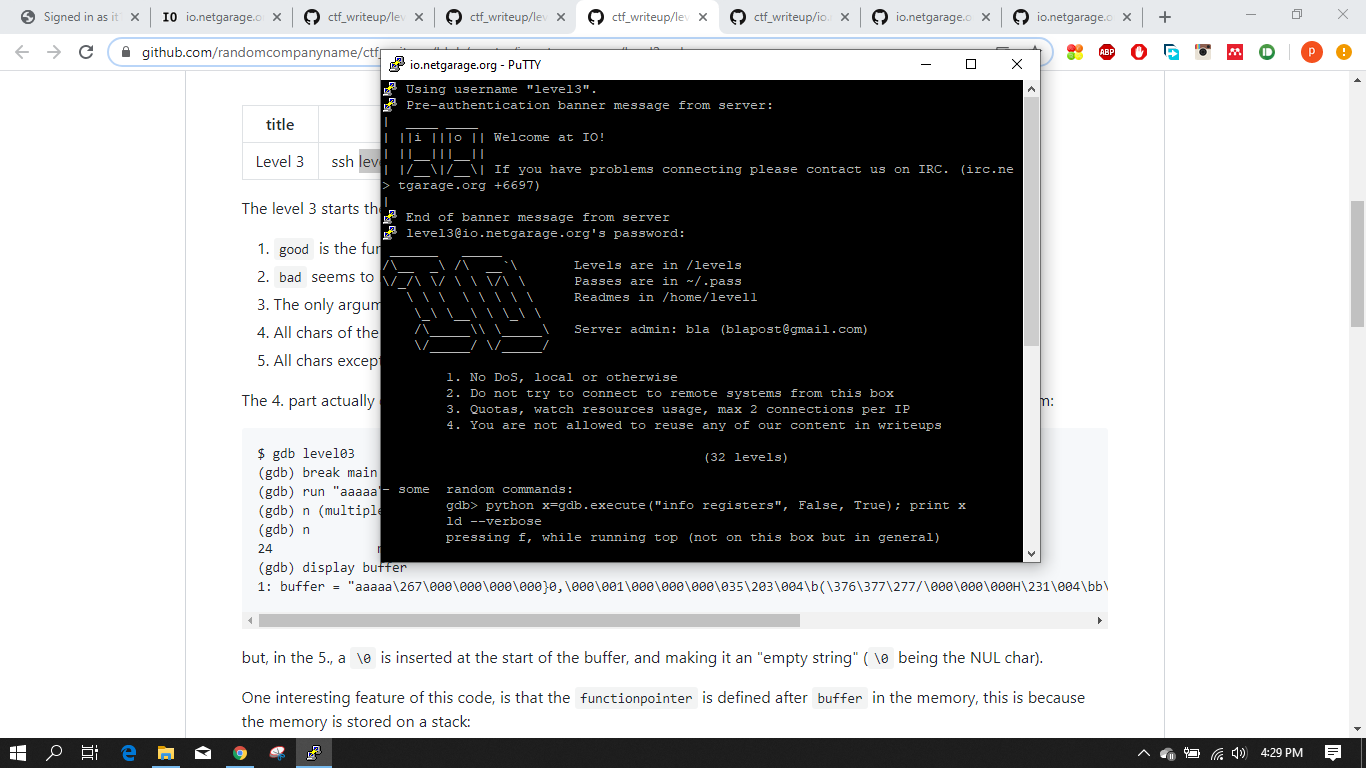
Level 3 password: **OlhCmdZKbuzqngfz**



**Level 03**

|  |  |
| --- | --- |
| **title** | **link** |
| Level 3 | ssh level3@io.netgarage.org |

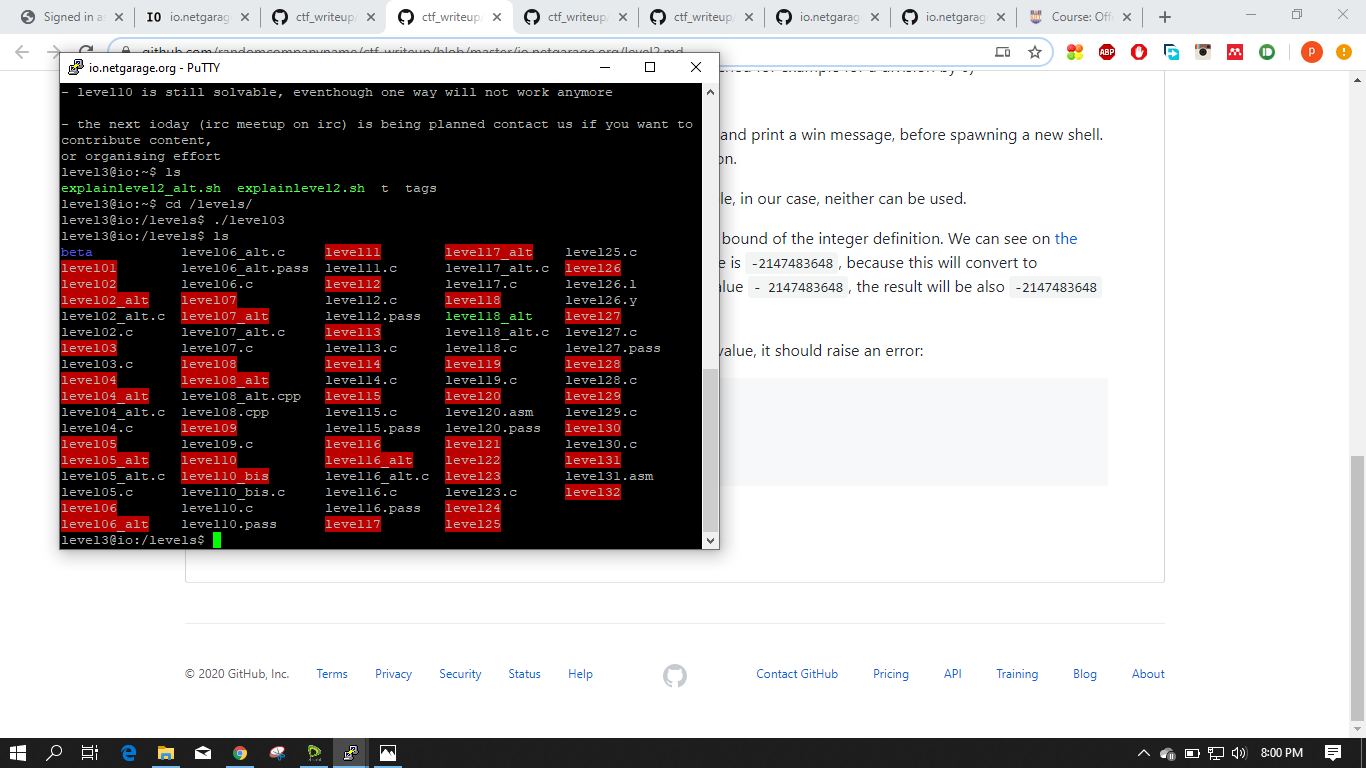
The level 3 starts the same way the level2 does.



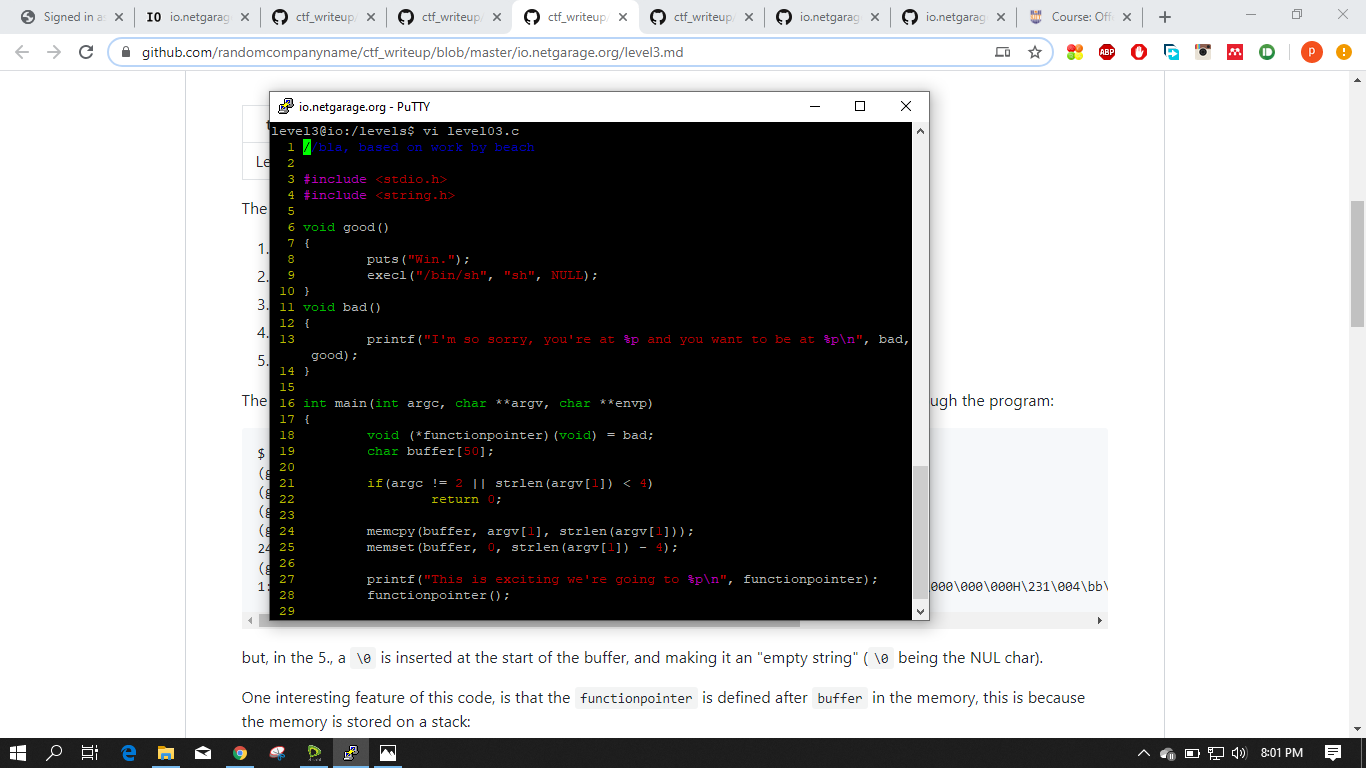
looking at the source code.

1. good is the function to call to clear the stage
2. bad seems to be the default function, referenced in the main.
3. The only argument need to be string of length more than 4
4. All chars of the argument will be copied in the buffer (max of 50)
5. All chars except last 4 ones will be set to 0

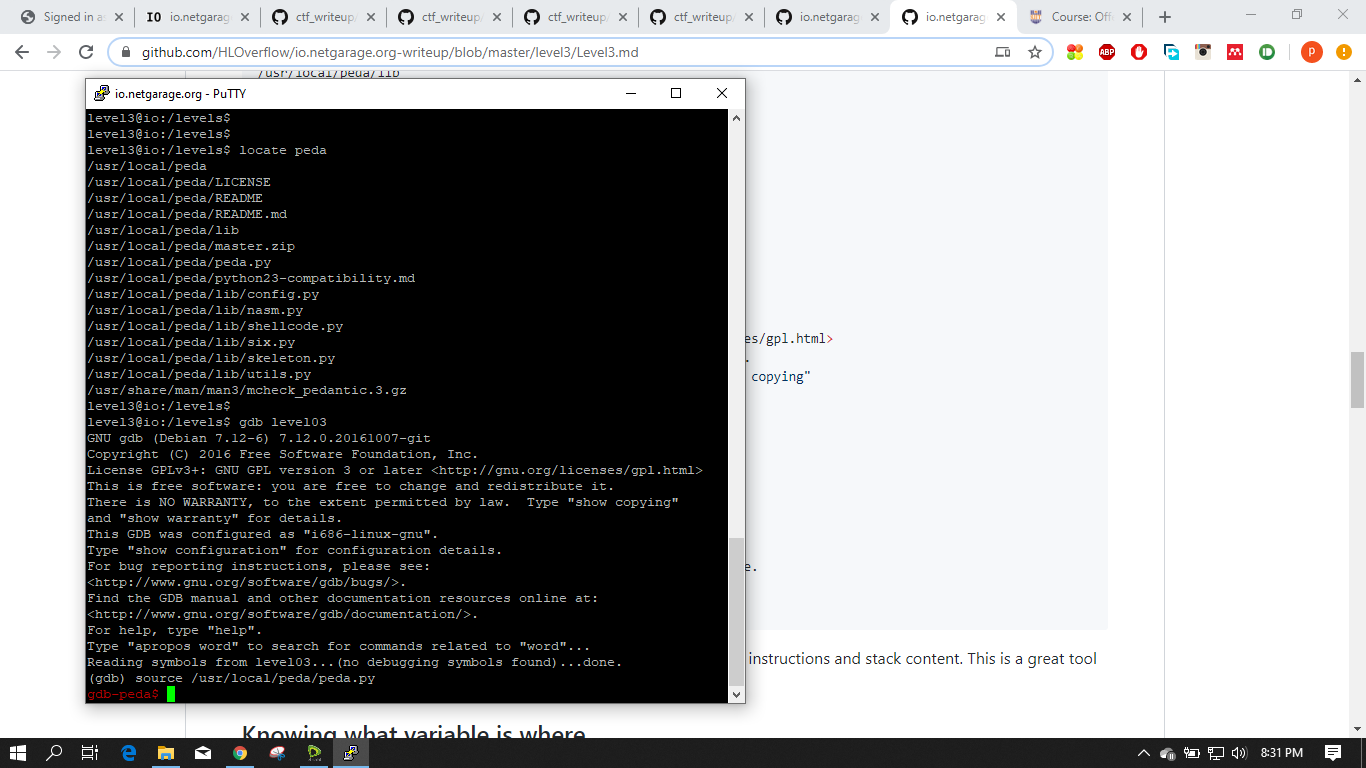
The 4. part actually duplicate the value of argv[2] into buffer.



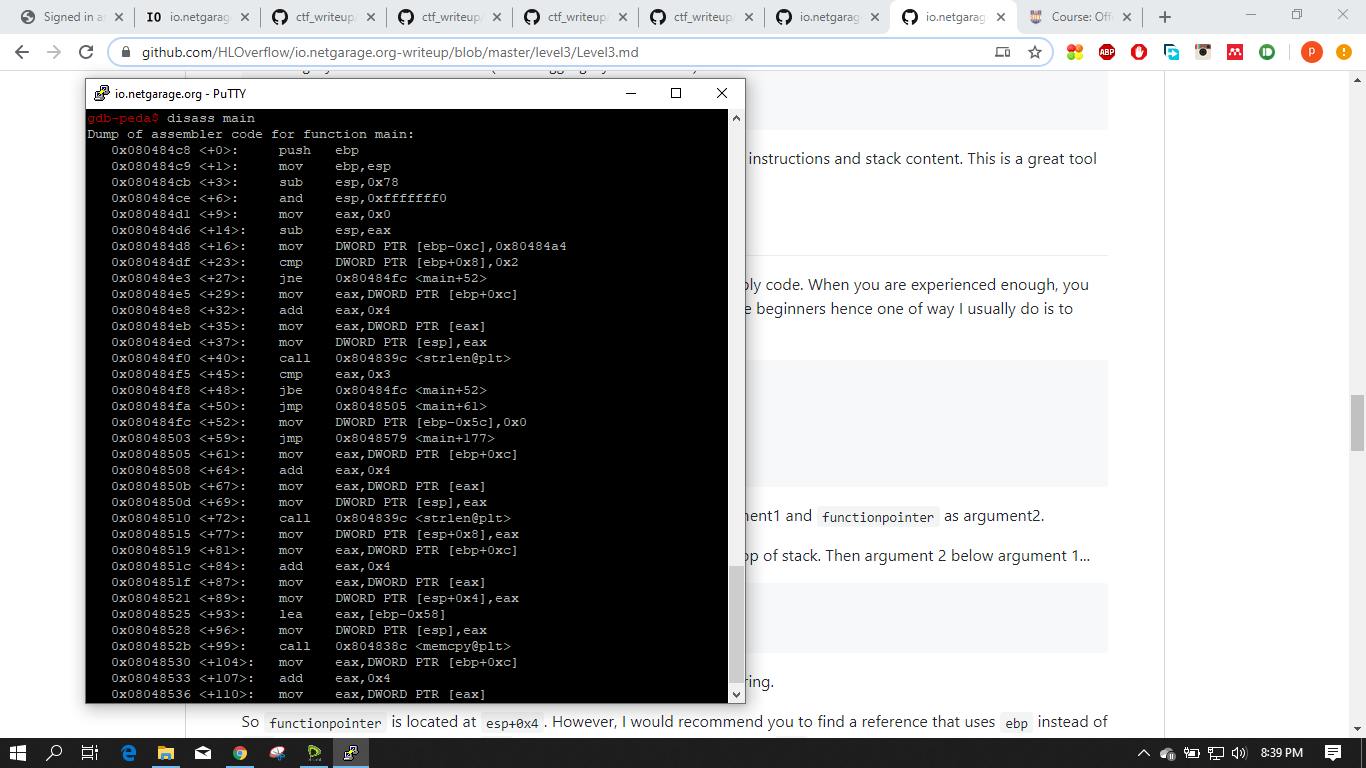
This program introduce the concept of buffer overflow to overwrite nearby local variable (more specifically, variables below the buffer) .



gdb peda is a GDB plugin that helps greatly in visualization of registers, instructions and stack content. This is a great tool for dynamic analysis.



When you disass main, it can be very scary to see a long list of assembly code. When you are experienced enough, you can figure out which offset refers to which variable. I will assume you are beginners hence one of way I usually do is to look out for functions that reference those local variables.



From the source code, we know that printf will have a string as argument1 and functionpointer as argument2.

The calling convention of x86 for function is to have argument on the top of stack. Then argument 2 below argument 1...

gdb-peda$ x/s 0x80486c0

0x80486c0: "This is exciting we're going to %p\n"

With gdb examine, we can confirm that argument1 really contain the string.

So functionpointer is located at esp+0x4. However, I would recommend you to find a reference that uses ebp instead of esp. This is because the value of ebp is constant in a stack frame whereas esp can vary.

0x0804856d <+165>: mov eax,DWORD PTR [ebp-0xc]

0x08048570 <+168>: call eax

This is another place that reference your function pointer. Notice call eax from this, you know function pointer is at ebp-0xc.

You can do the same to find out where is buffer.

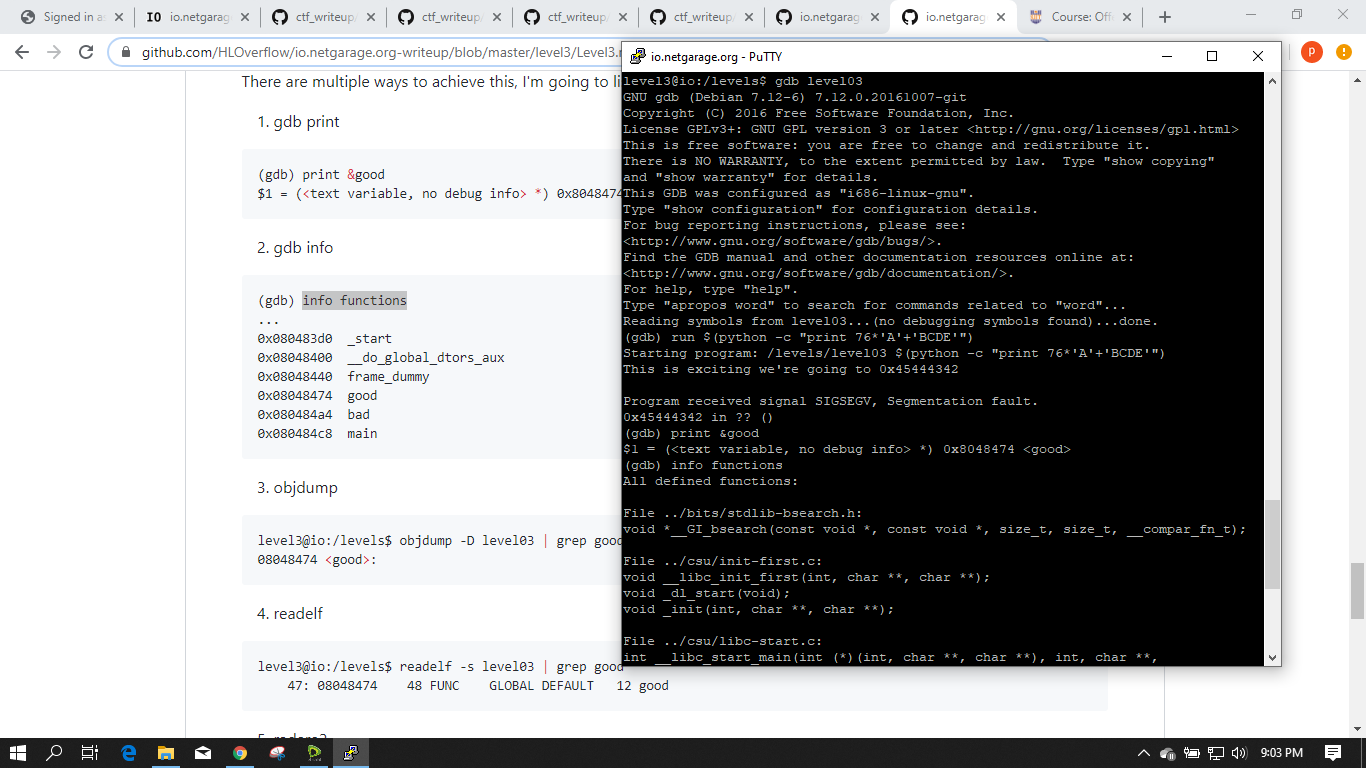
Now, you just need to find the distance between buffer and functionpointer.

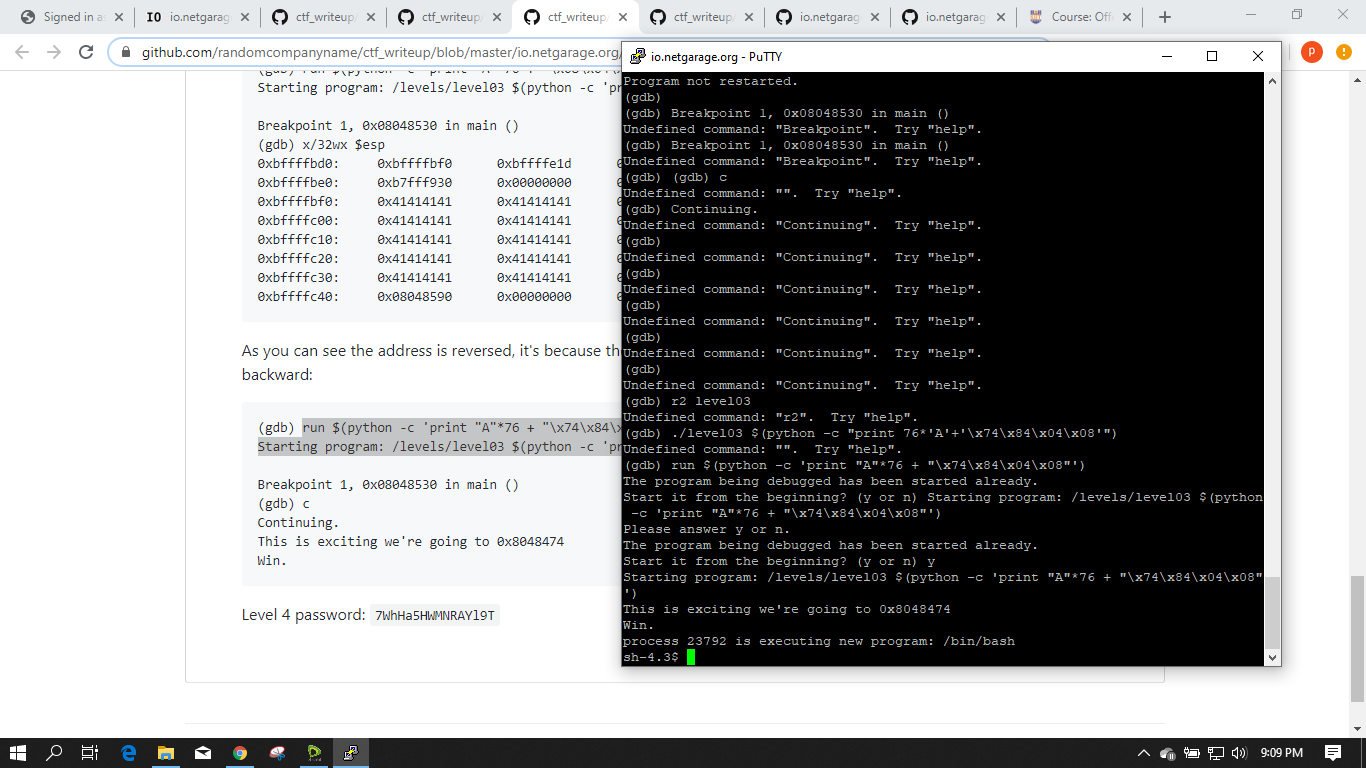
Run python inside /levels and then get this value.

>>> 0x58-0xc

76

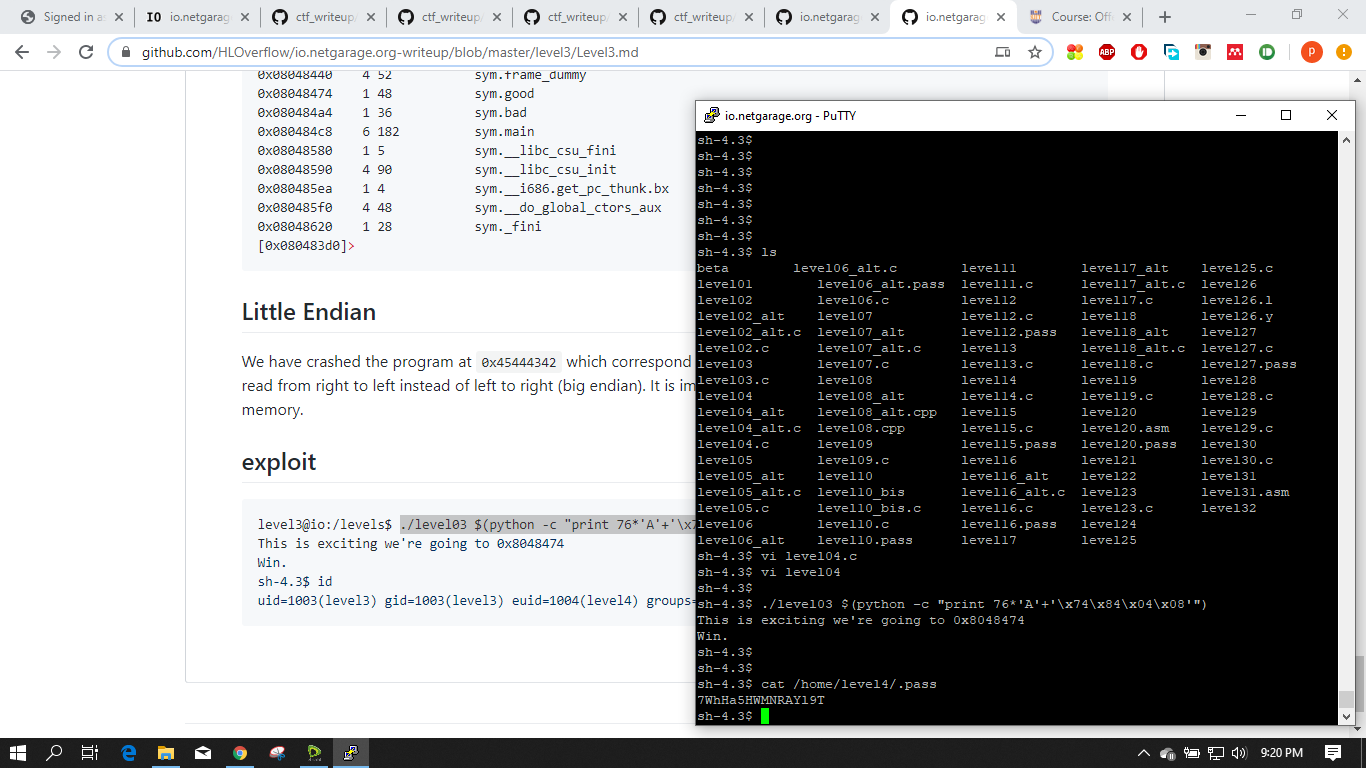
**testing out**





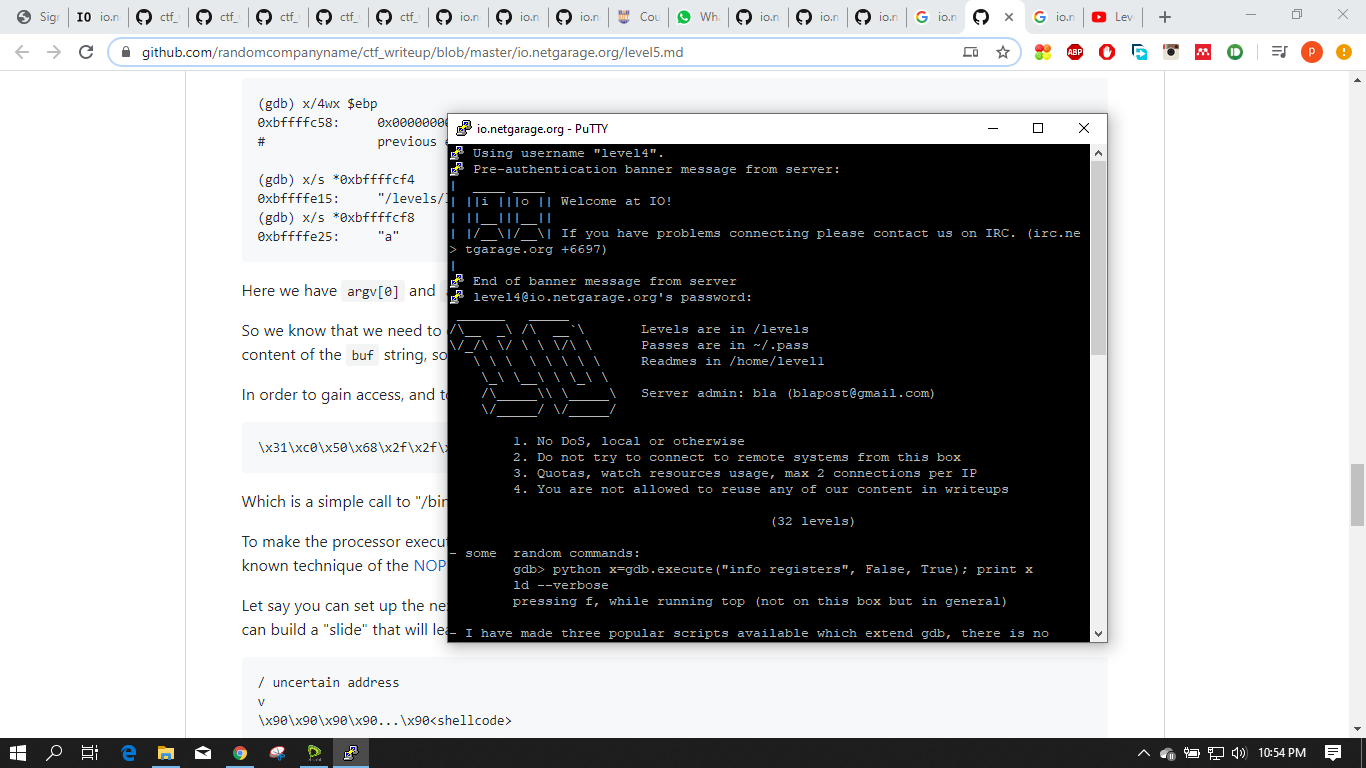
**Exploit**

Level 4 password: **7WhHa5HWMNRAYl9T**



**Level 04**

|  |  |
| --- | --- |
| **title** | **link** |
| Level 4 | ssh level4@io.netgarage.org |



The code for this one is straightforward and only call a system command with the popen function, reads its input and display it.

Under a shell, the binaries are found by searching the PATH variable until one executable file with the desired name is found.

In the code, it is susceptible to an attack that changes the $PATH environment variable because it is not using an absolute path to call whoami.

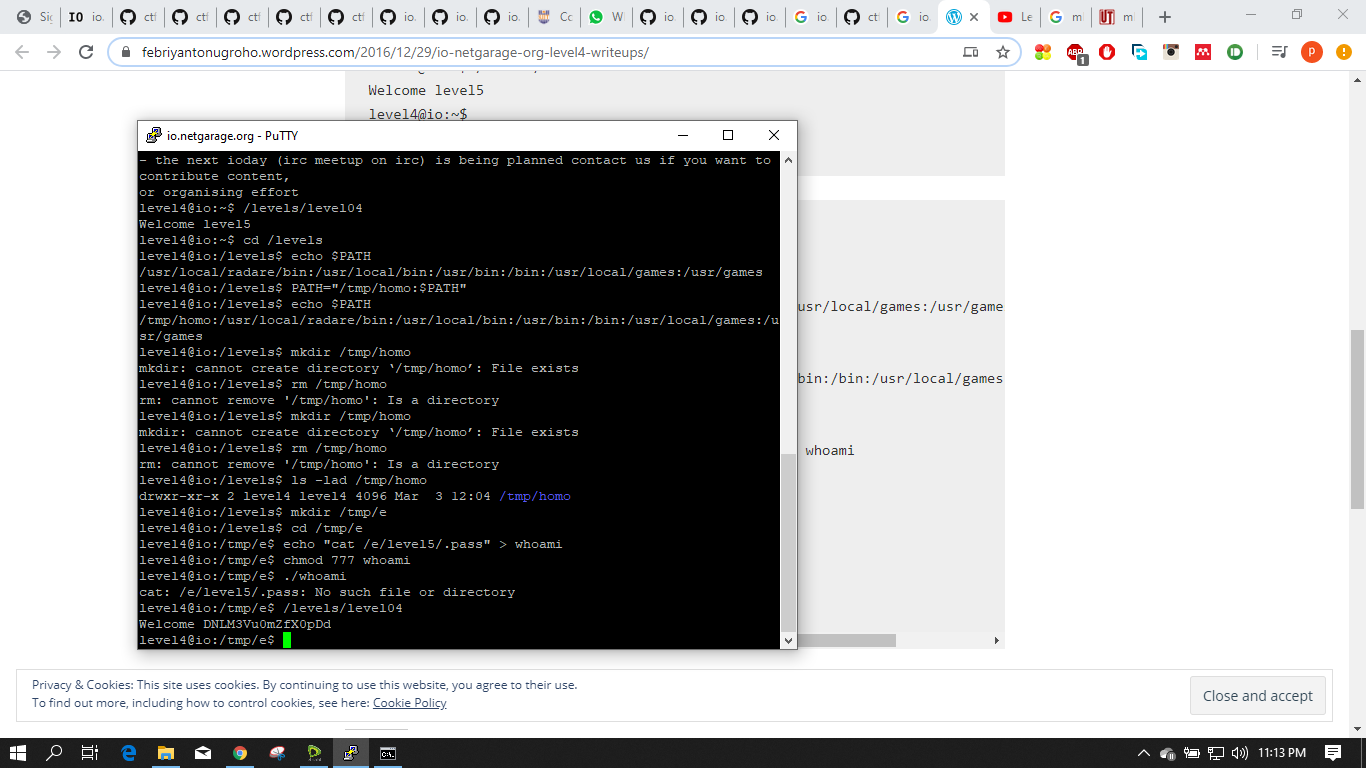
The OS will look at $PATH for list of directories to search the binary for in a left to right order. When it finds a binary that has the requested name, it will execute that binary.

You can trick the OS into running our own whoami program that spawn a shell. As long as our directory comes before the actual location of whoami, your binary will be run instead.

You can make a directory for yourselves to put the above C source code.

Next, compile the program with the same name as whoami.

You have prepended a "." in the $PATH. The OS will start searching from current directory first for a whoami. You can see that your euid is level5.



**Level 05**

|  |  |
| --- | --- |
| **title** | **link** |
| Level 5 | ssh level5@io.netgarage.org |

