Ho Chi Minh City University Of Science

Faculty of Information Technology

Cybersecurity

Project 03

io.netgarage.org warfare

Nguyen Phuc Thuan - 20120380

Ho Chi Minh City - 2024

# I. Introduction

Dear sir,

My name is Nguyen Phuc Thuan, with the student id of 20120380. In this report, I will mainly state about my path of playing io.netgarage.org warfare. I write this report as a “hacking story”, and in English. With my not very good English, there are some mistakes about grammar stuff.

From the author of io.netgarage.org (in short, from now I will call it the “warfare”), The problems are presented to you as a series of programs. They will vary in size from a few lines to real software. The point is usually to exploit this bug in such a way that you can control the program's execution flow. With the aim of having it read out the password file for the next level.

The way this works is that the programs are "SUID binaries" (<http://en.wikipedia.org/wiki/Setuid>). Set-user-id programs run with the privileges of the owner of the program. Not the user starting the program. This is also how for example the "passwd" program on a standard unix works.

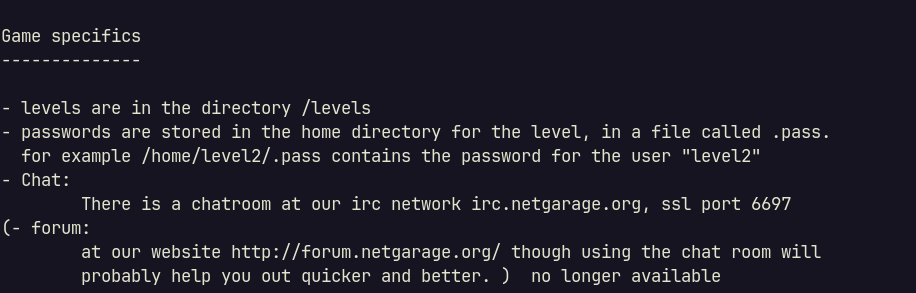
We will need to hijack these elevated privileges of the level programs and use them to read the file in /home/levelX+1/.pass. which contains the password for that level.

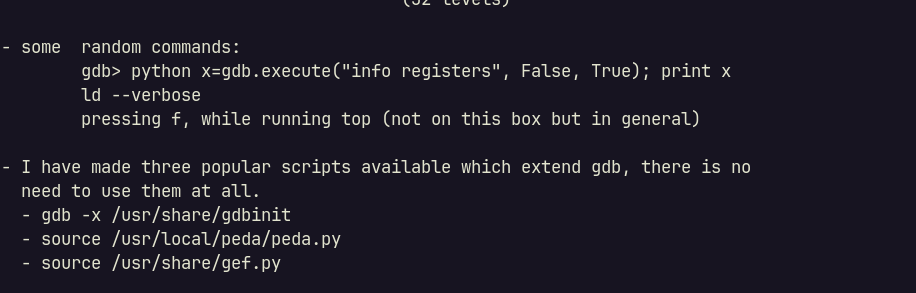
Easy to understand huh. Seem like this warfare is mainly about software exploit.

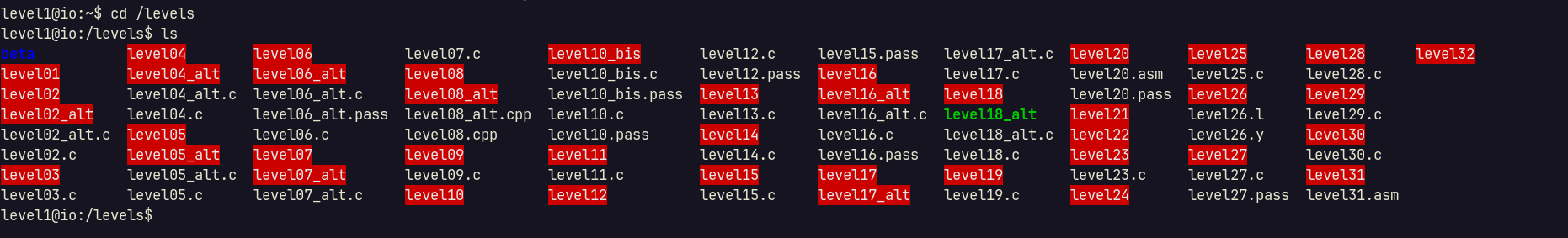
I hope you will enjoy this report, away from usually academic report at all. Let’s get started.

# II. Write Up

## Level 1

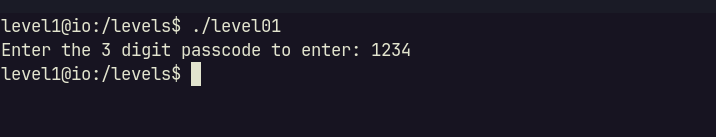
What got me first in level 1 seem like the ssh banner, which tells something about using gdb. Maybe we need to use gdb to get the key somewhere in the software. In the other hand, the banner show that we should start with level01 in the ./levels directory





32 levels at all. I not a very good at this field (reverse engineering), but I would give it a try.

Start with level 1, the program prompt me to enter a three-digits number:

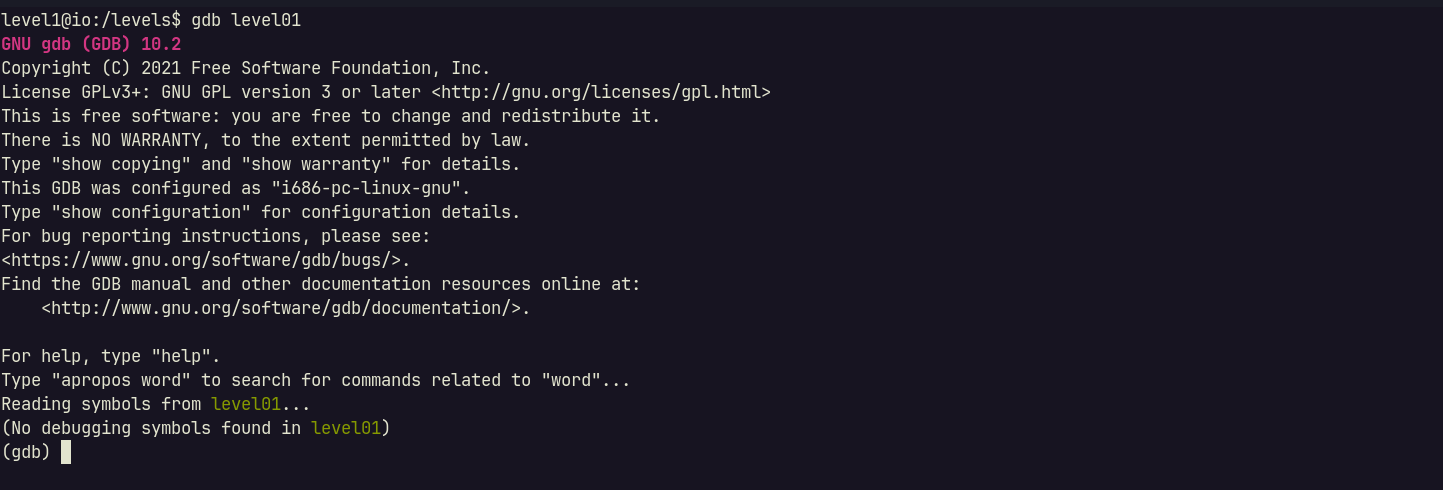


In my sense, I guess that there will be a comparison inside this program, to compare the input with some constant, such as:

|  |
| --- |
| C++  if (input == some\_number) {  show\_some\_output();  } else {  show\_nothing();  } |

Using my guessing with above hint about using gdb, this should be the solution.

Using gdb, first I set a breakpoint on main (program start), then run it:



Try to disassable the main function show me a comparison as expected:

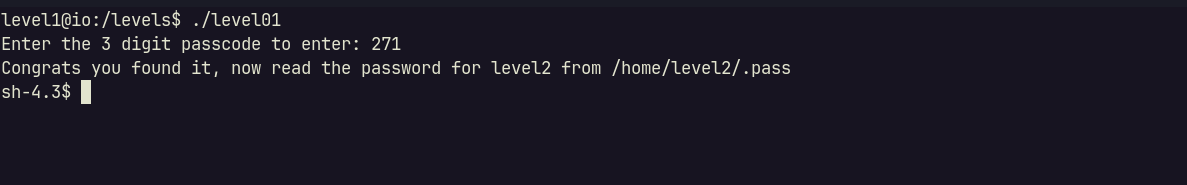


The memory at 0x10f should store some value. Simply display it show us the code:

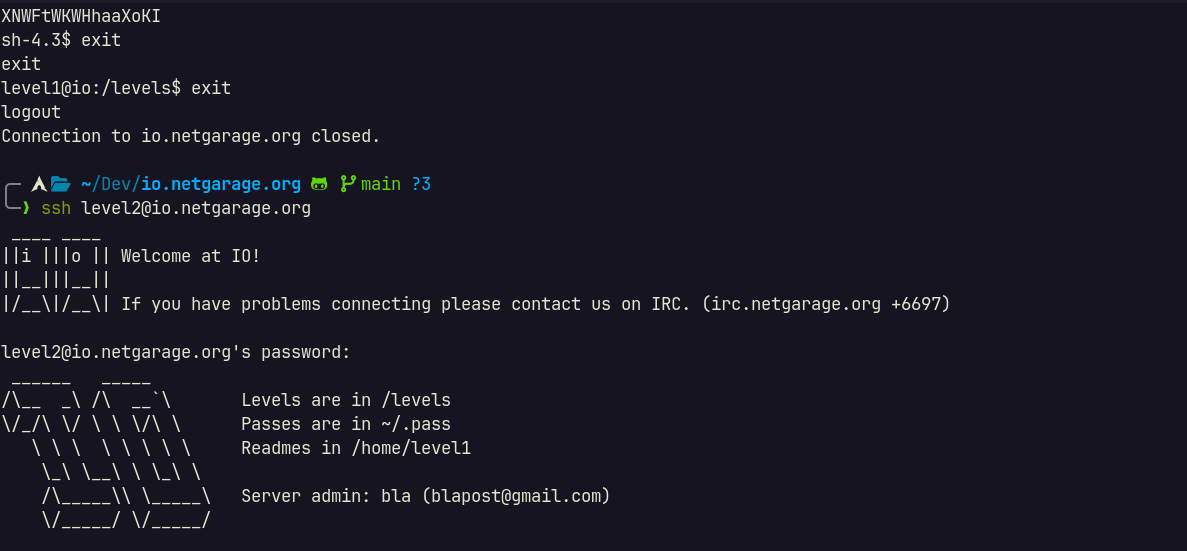


271 is seem to be the key to level 2.

Try it again with level binary:



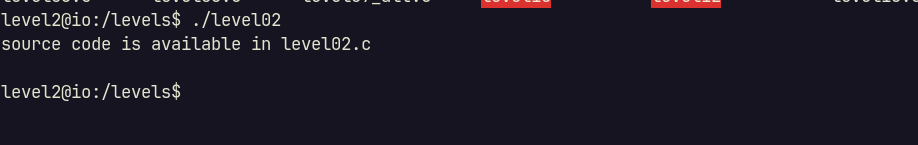
Got out ticket to level 02.



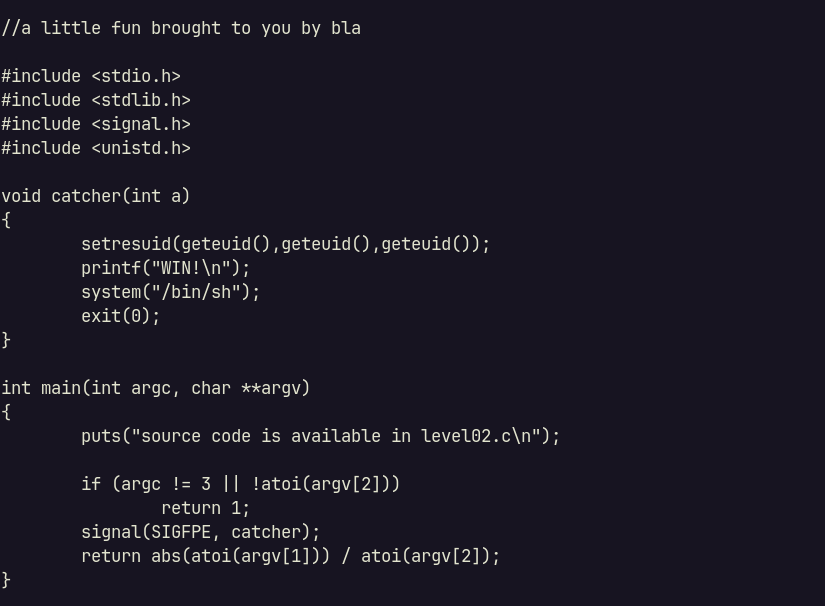
Credential found: level2:XNWFtWKWHhaaXoKI

## Level 2

Level 02 need us to investigate its code:



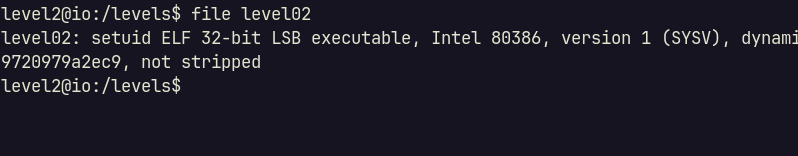
And here is its code:



What do we need? Three arguments, and the second one must be a number different from zero. Then it will setup a SIGFPE, which will catch the SIGFPE (a signal about arimethic stuff, such as division by zero???), by calling catcher.

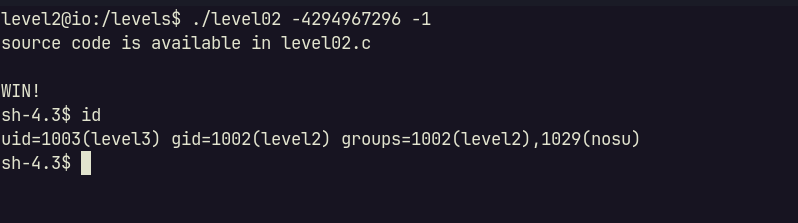
To exploit this binary, we cannot leverage division by zero. So the last way is exploit integer overflow.

First check for binary arch:



A 32-bit program. So atoi should convert the input to a 32-bit integer, which will vary from -2^32 to (2^32 – 1), in case it is signed integer.

So what if we divide (-2^32) to (-1)? In math, we will get 2^32. But it is out of range.



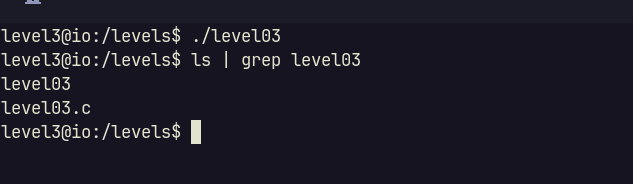
Gotcha.



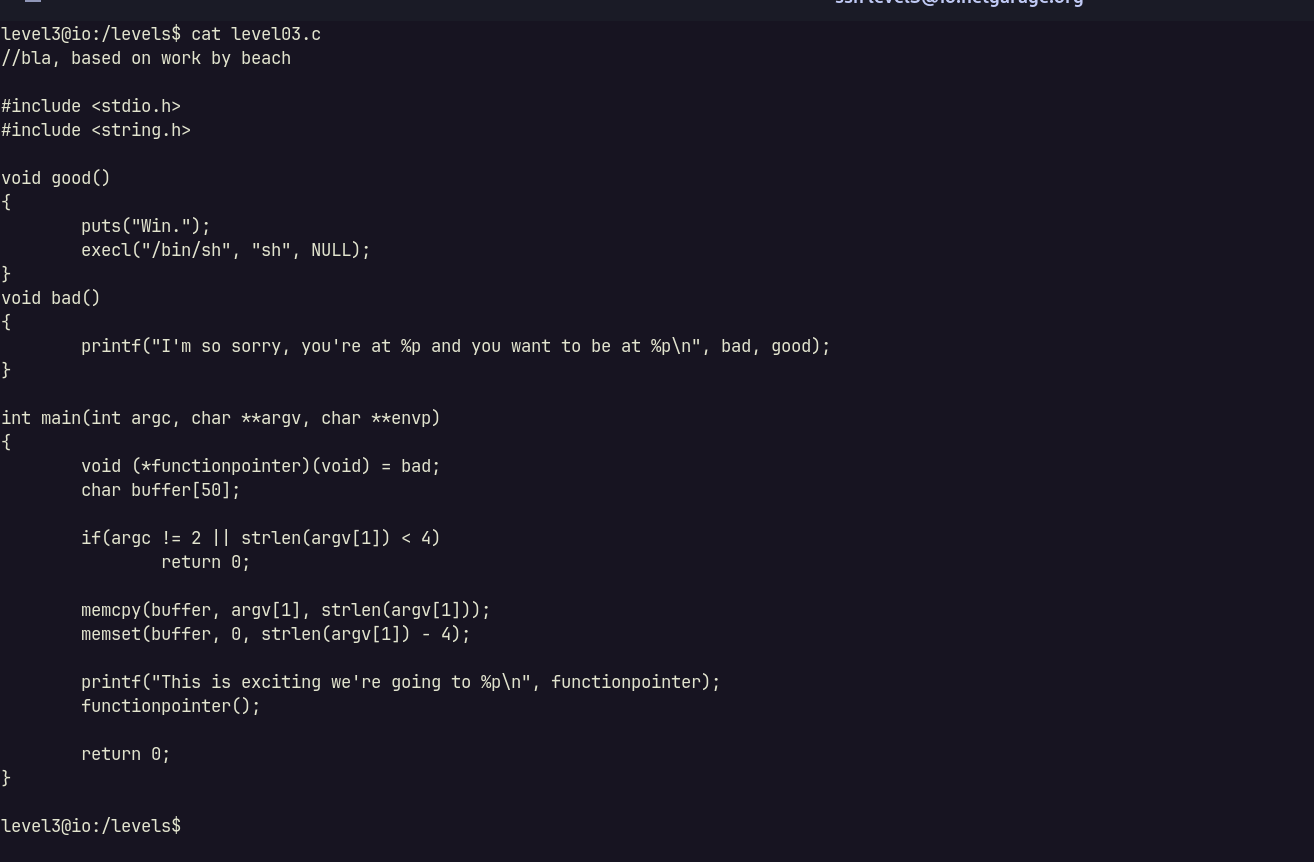
Credential found: level3:OlhCmdZKbuzqngfz

## Level 3

First try to run the binary level03, but it does not show me anything. But we got a level03.c:

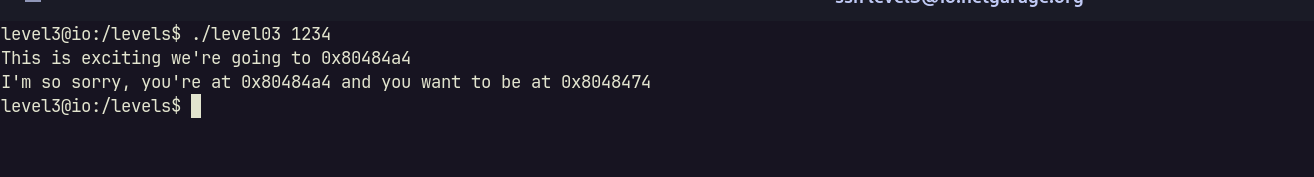


Here is it source code:

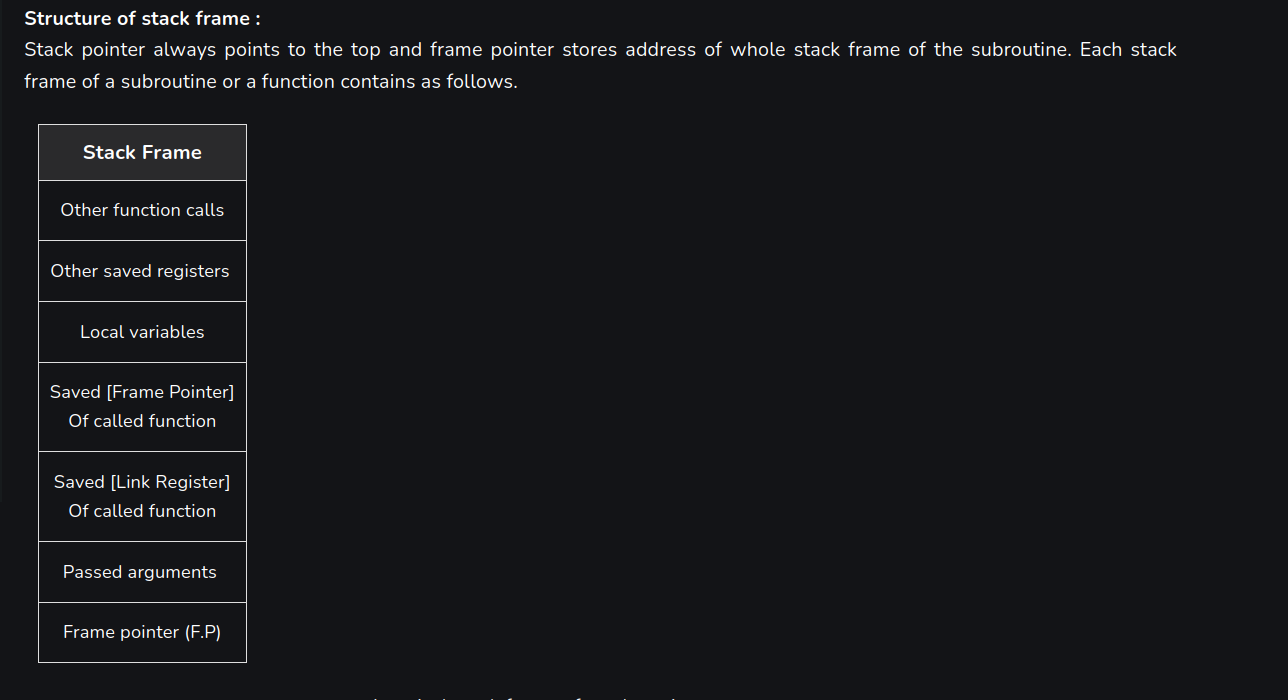


Seem like we need to provide agrv[1] that have length bigger than 4.

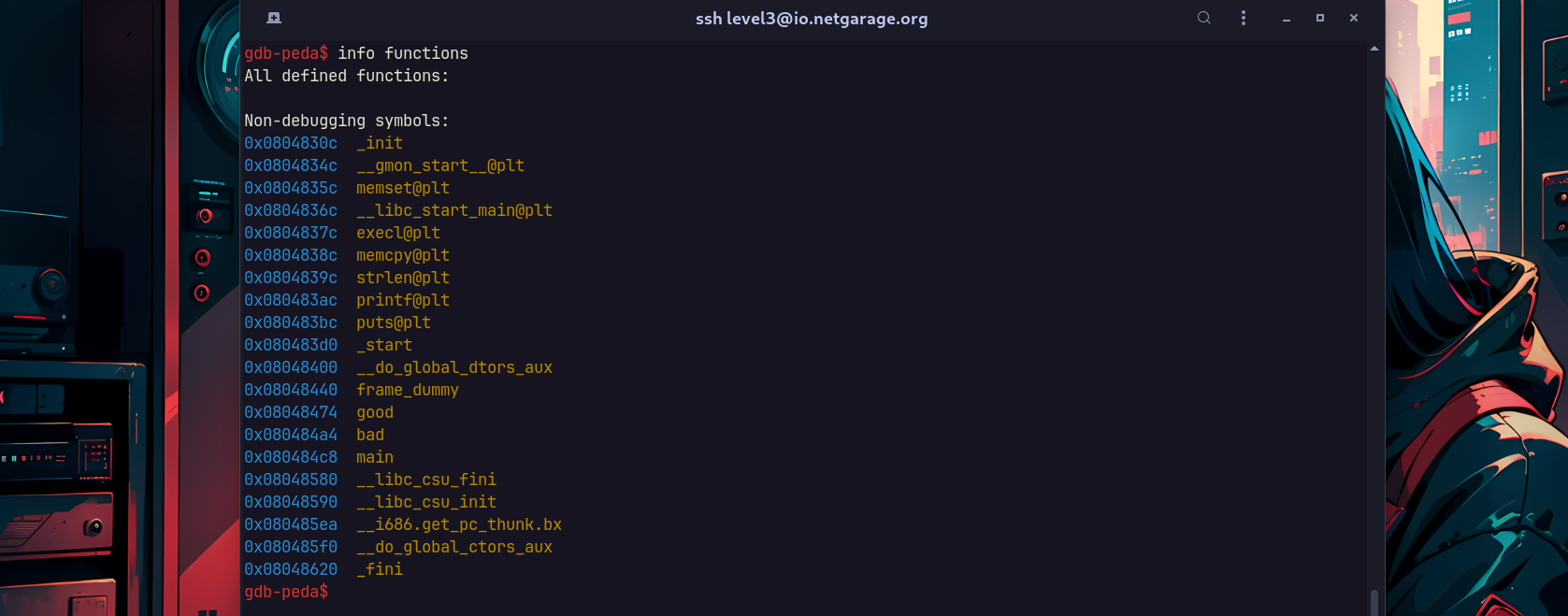
What is the hint here? It is the buffer[50], which will be store in stack. This level is about stack buffer overflow, to make the functionpointer points to good().



The input is seem like an address. Doing a quick research in GeeksForGeeks, I found below figure, state about stack frame:



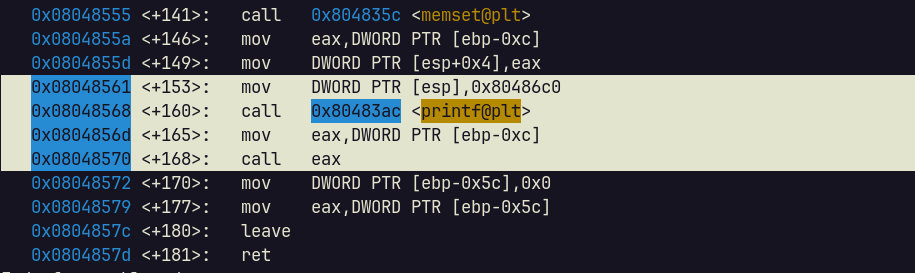
First, using gdb, we can get a list of function address:



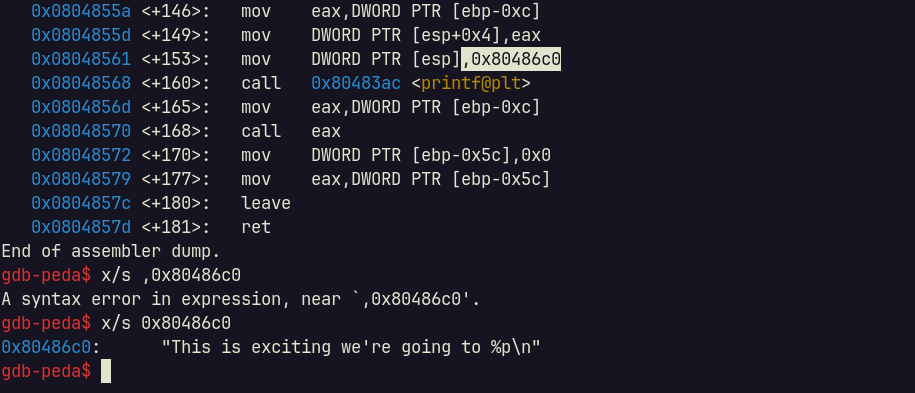
|  |
| --- |
| good: 0x08048474 |
| bad: 0x080484a4 |

In the other hand, we must find out how far is buffer and functionpointer.

First, there is a line that using printf and reference to functionpointer. If we found the string, we can also find functionpointer address:



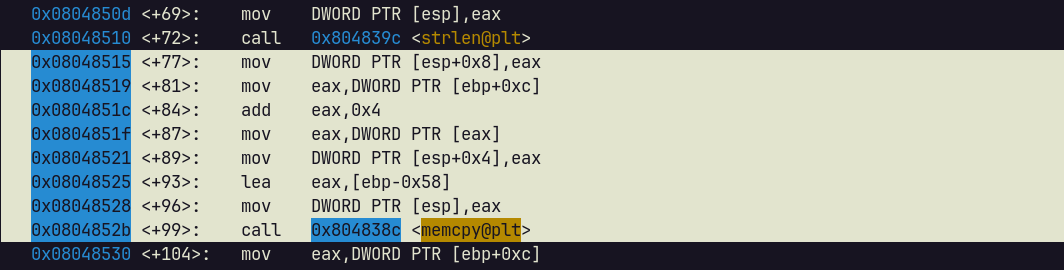
Here is it.



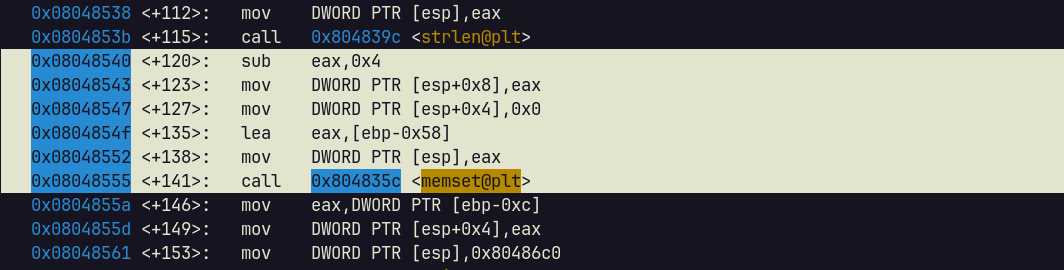
We can assume that functionpointer using the address ebp-0xc.

Then, using the same method, we should find the buffer address. The function memcpy and also memset using buffer as first argument. And the call of buffer should right behind the call of some strlen.

Then we found it here:

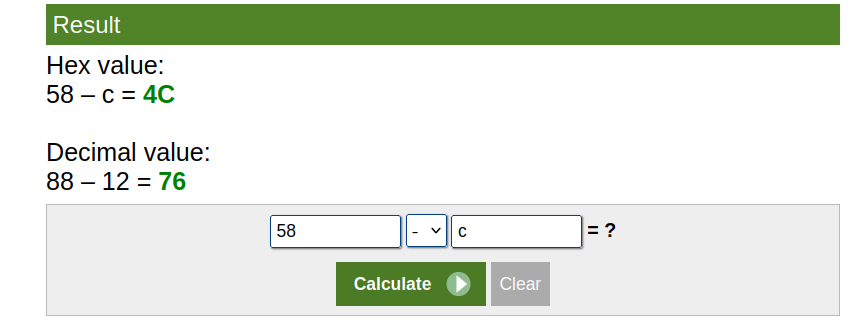


Also in the call of memset:



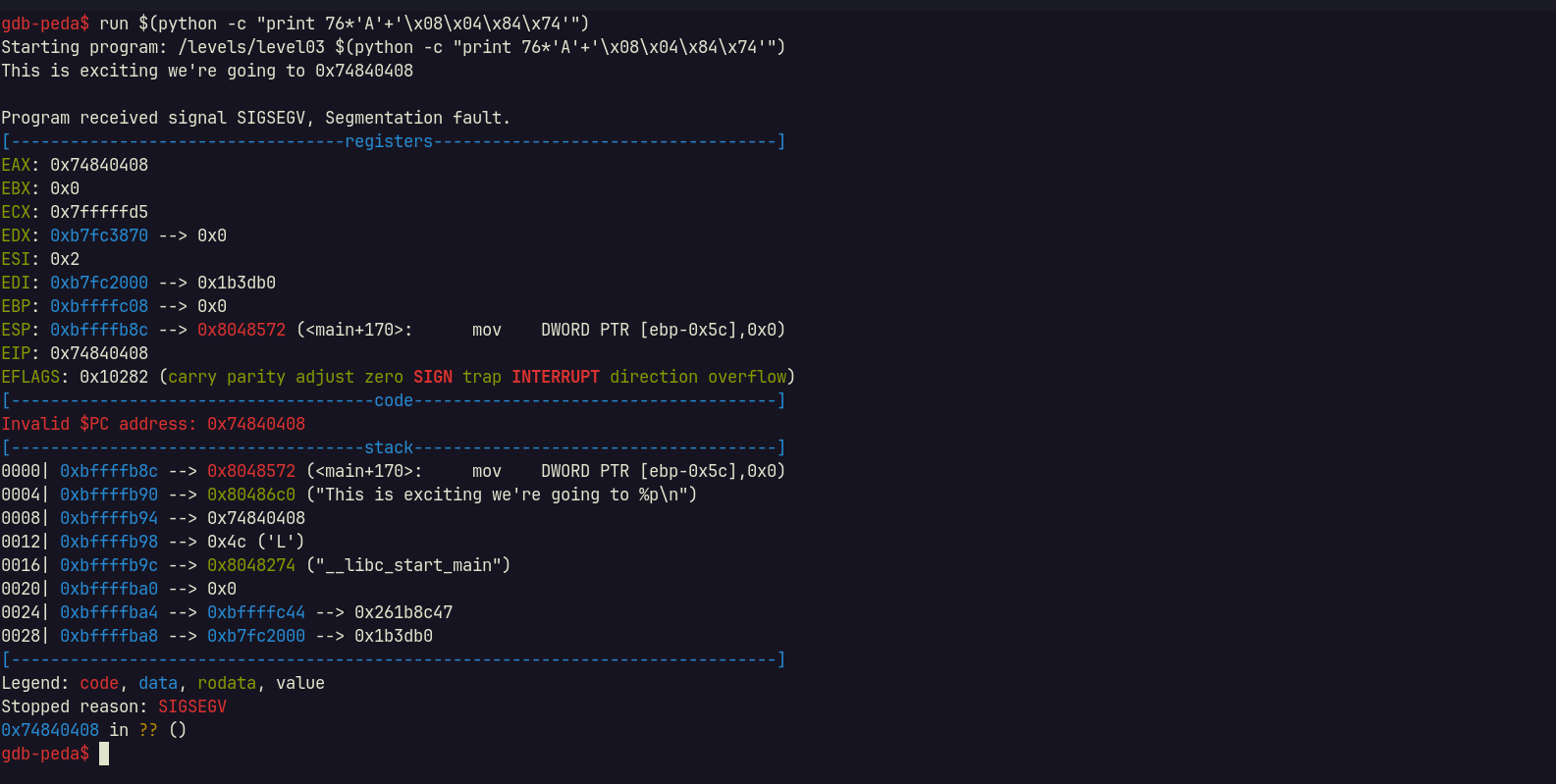
We can assume that buffer is start with 0x58.

So we can calculate distance from functionpointer and buffer in order to point to good.

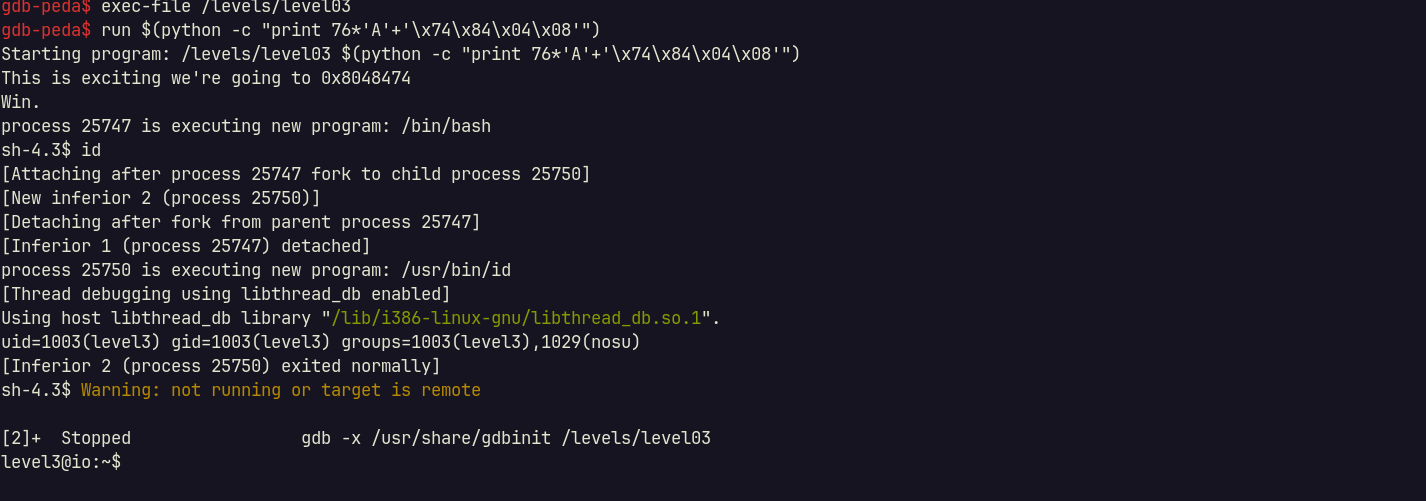


We must overflow buffer by 76, then write the address of good to buffer.

Run with 76 character to overflow, then point to good:

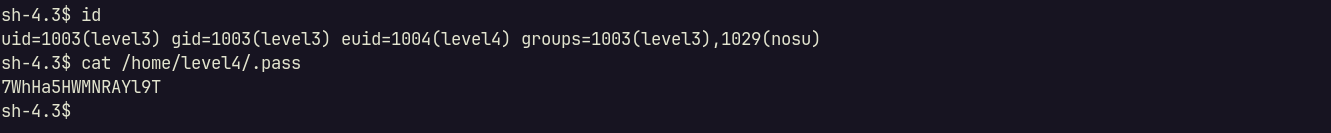


Hmm, seem like we point to wrong place. The address seem written in reverse order. This could because the edianess (Big Edian or Little Edian). Try to send reverse order of edianess:



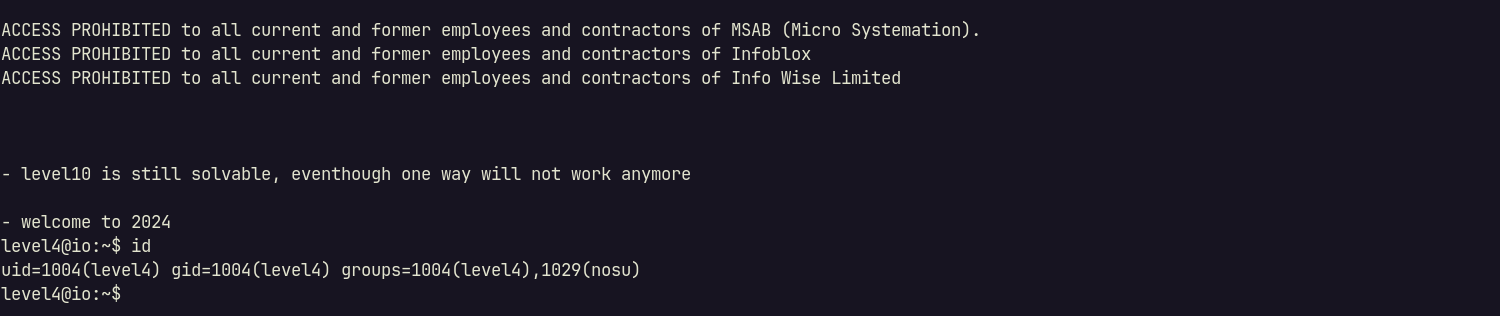
Gotcha.

Gdb may restrict our operation, so we go back to normal binary run and get the elevated session:



Credential found: level4:7WhHa5HWMNRAYl9T

## Level 4



## Level 5

## Level 6

## Level 7

## Level 8

## Level 9

## Level 10