**CS6008 Cryptography and Network Security.**

**Assignment No.3**

**Name:** G Vignesh **Reg No:**2018103078

**Date:** 29.4.2022

**Module-3:**  EXPLOIT TECHNIQUES

**External Learning-** Implementing Return Oriented Programming.

**Topic:** Spawning a shell using Return oriented programming

**Aim:**

To check and find the ROP gadgets in the executable files of our c program. String or combine those gadgets together in order make those gadgets perform spawning a shell. Finally pass those gadgets and place them in the return address of the function to successfully launch a shell.

**Tools-used:**

1.Linux Terminal

2.GCC compiler.

3.ROP-Gadgets python script.

4.GNU Debugger (GDB).

**Description:**

Return oriented programming is a security exploit technique used by hackers to execute their code in our system by finding suitable ROP gadgets in our program. ROP gadgets are sequence of instructions ending with **return**  statements. In the exercise we implement a c program which takes file name as input and copy its content to a buff. We find ROP gadgets identify the suitable ones to launch a shell, string them together pad with space in order place them in return address of the program save it in a file using python and pass that file as argument to the program and spawn the shell.

**Input:** Program containing ROP vulnerability , ROP gadgets stringed together passed saved in file passed as argument.

**Output:** Spawning a shell(i.e able to launch a shell and get access to the system)

**The C program:**

**ExecuteROP.c**

#include <string.h>

#include <stdio.h>

void overflow (void\* inbuf, int inbuflen)

{

  char buf[4];

  memcpy(buf, inbuf, inbuflen);

}

int main (int argc, char\*\* argv)

{

  char filebuf[750];

  FILE\* file = fopen(argv[1], "rb");

  int bytes = fread(filebuf, sizeof(char), 750, file);

  fclose(file);

  overflow(filebuf, bytes);

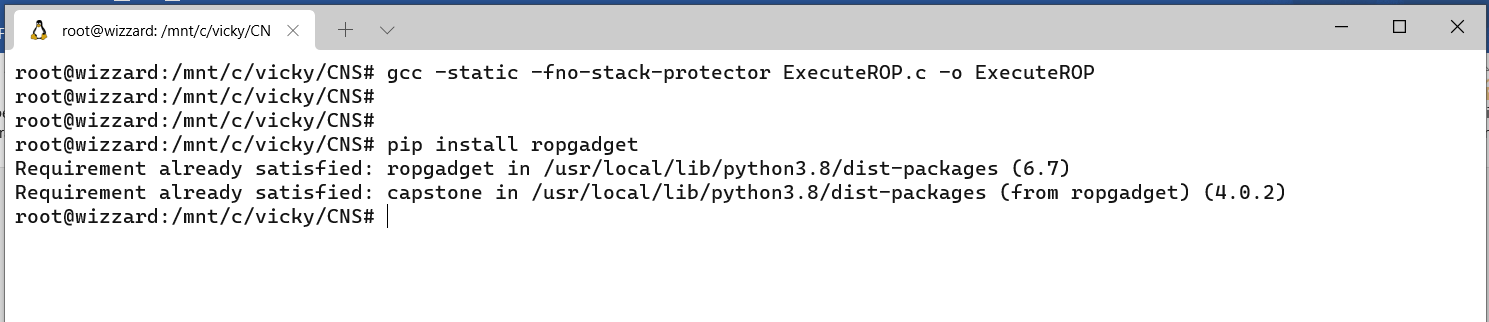
  return 0;

}

This c program takes file name as input read its contents ,count no.of bytes and sends to a function overflow where the contents of the file are copied to the buffer using memcpy function.

**Execution:**

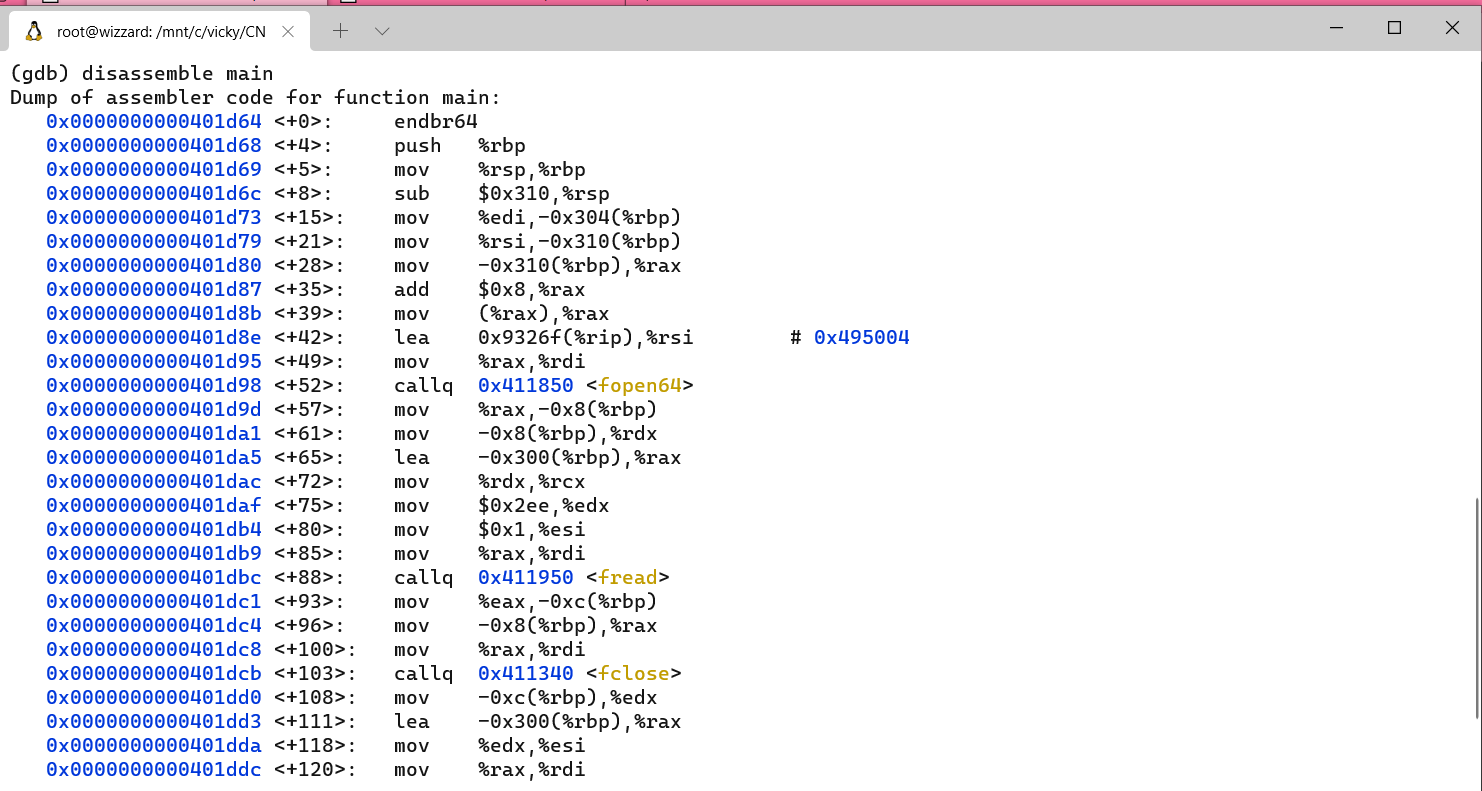
**Step-1:**



The c program above shown is compiled using options like **-fno-stack-protector**

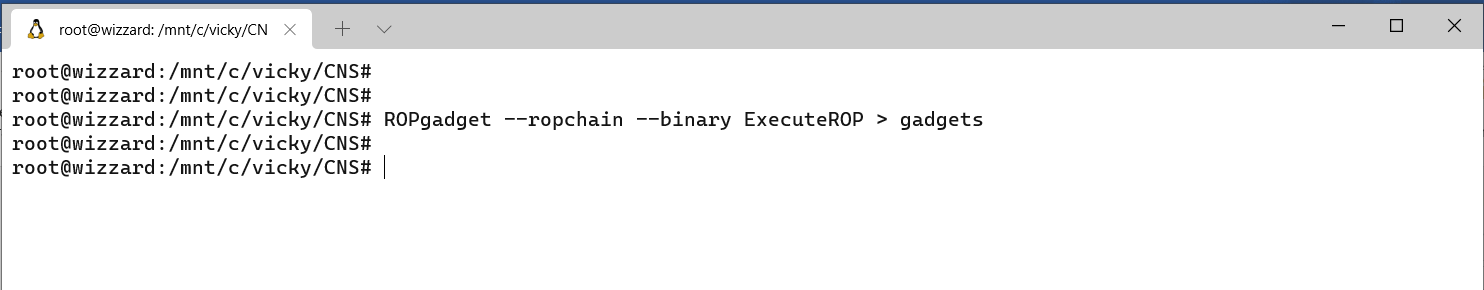
So that stack smashing done by us is not shown as warning and **-static** option makes the library function required for this program to be compiled along with the code so that we can get more Gadgets.

Identifying ROP gadgets is a difficult task which can be done by running the GDB mode and view the machine identifying the statements that end with ret.



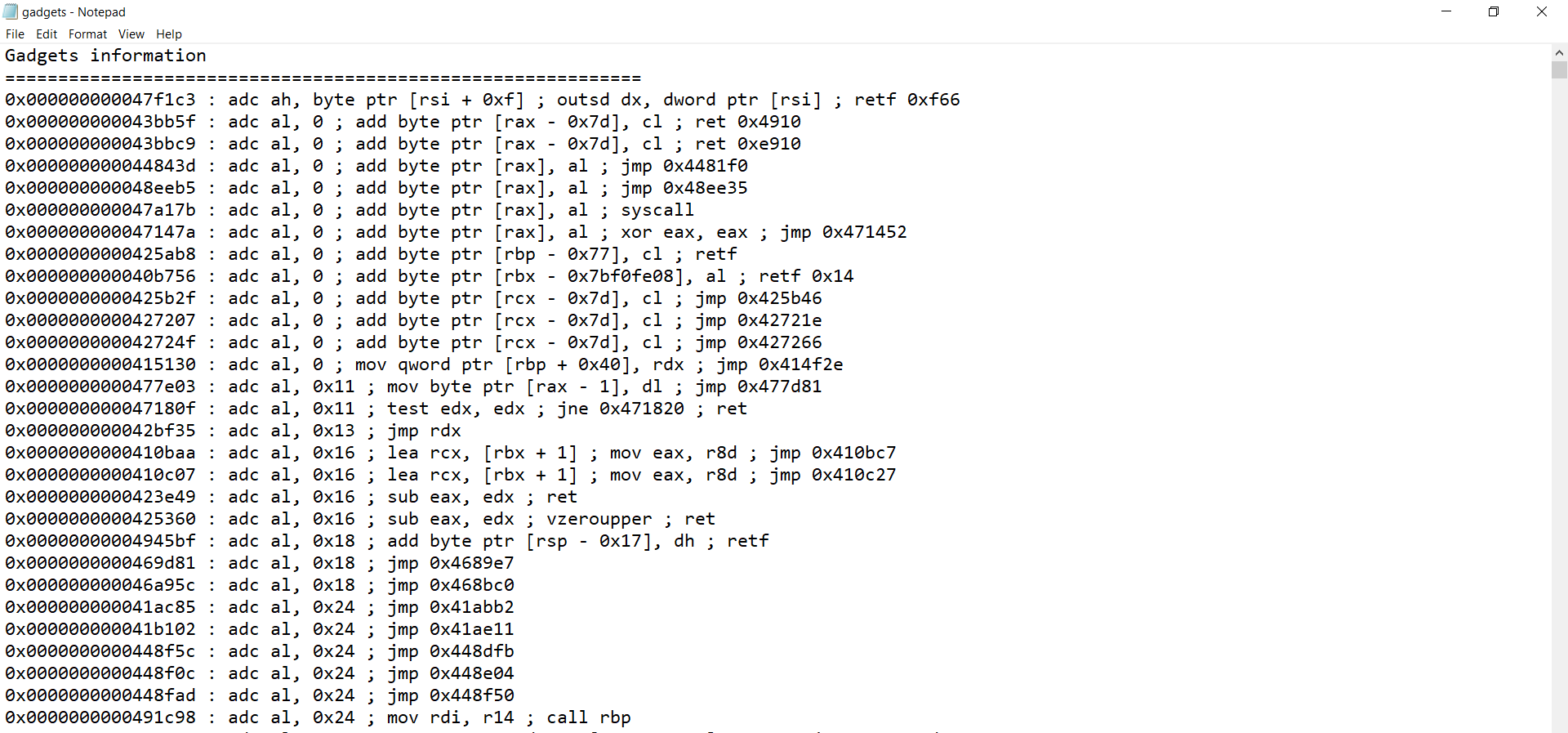
But it is very difficult to open each system call and find ROP gadgets so that we can find suitable no.of gadgets for our task. So, we make use python module **ropgadget.py**  which is installed using **pip install ropgadget**. This tool helps us to identify ROP gadgets present in our program.

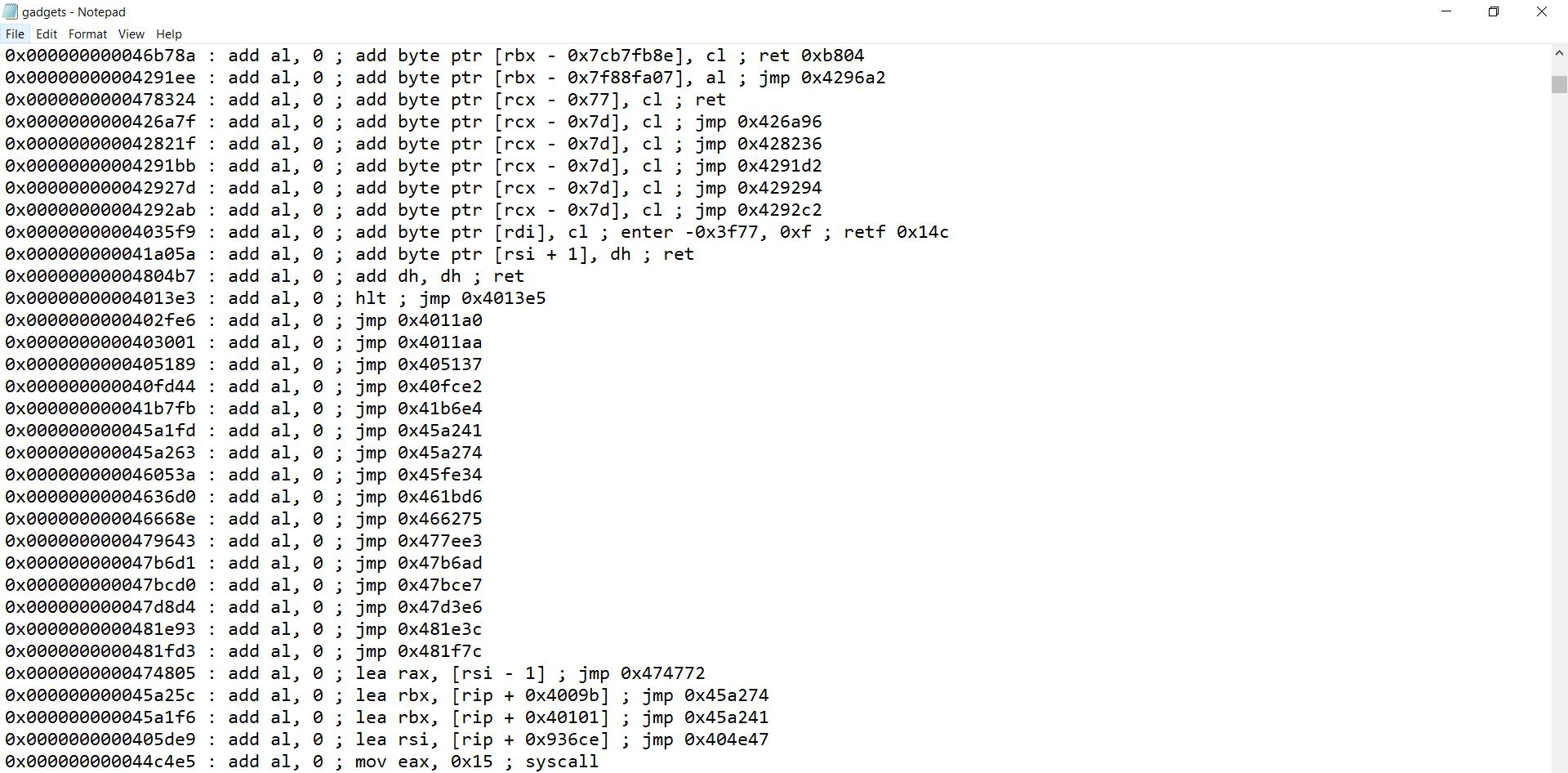
**Step-2:**



The ropgadget tool takes our executable code as input finds ROP gadgets available in the program that can be used to chain together and stores those gadgets into a file **gadgets.**

**Gadgets file:**



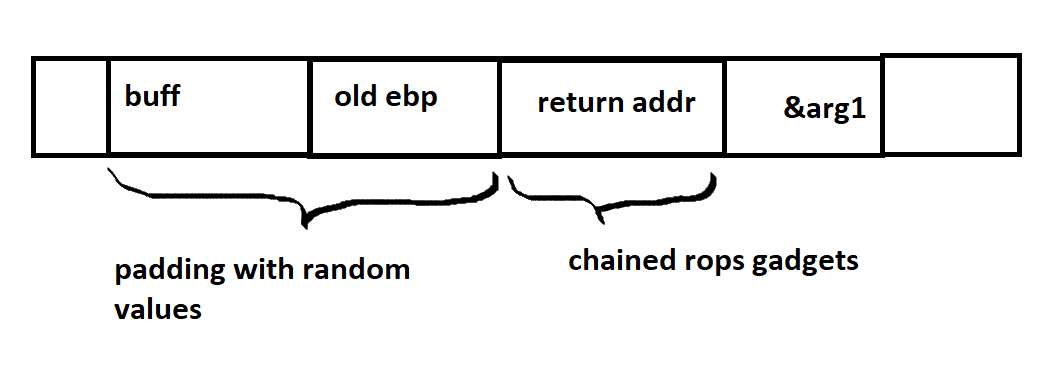


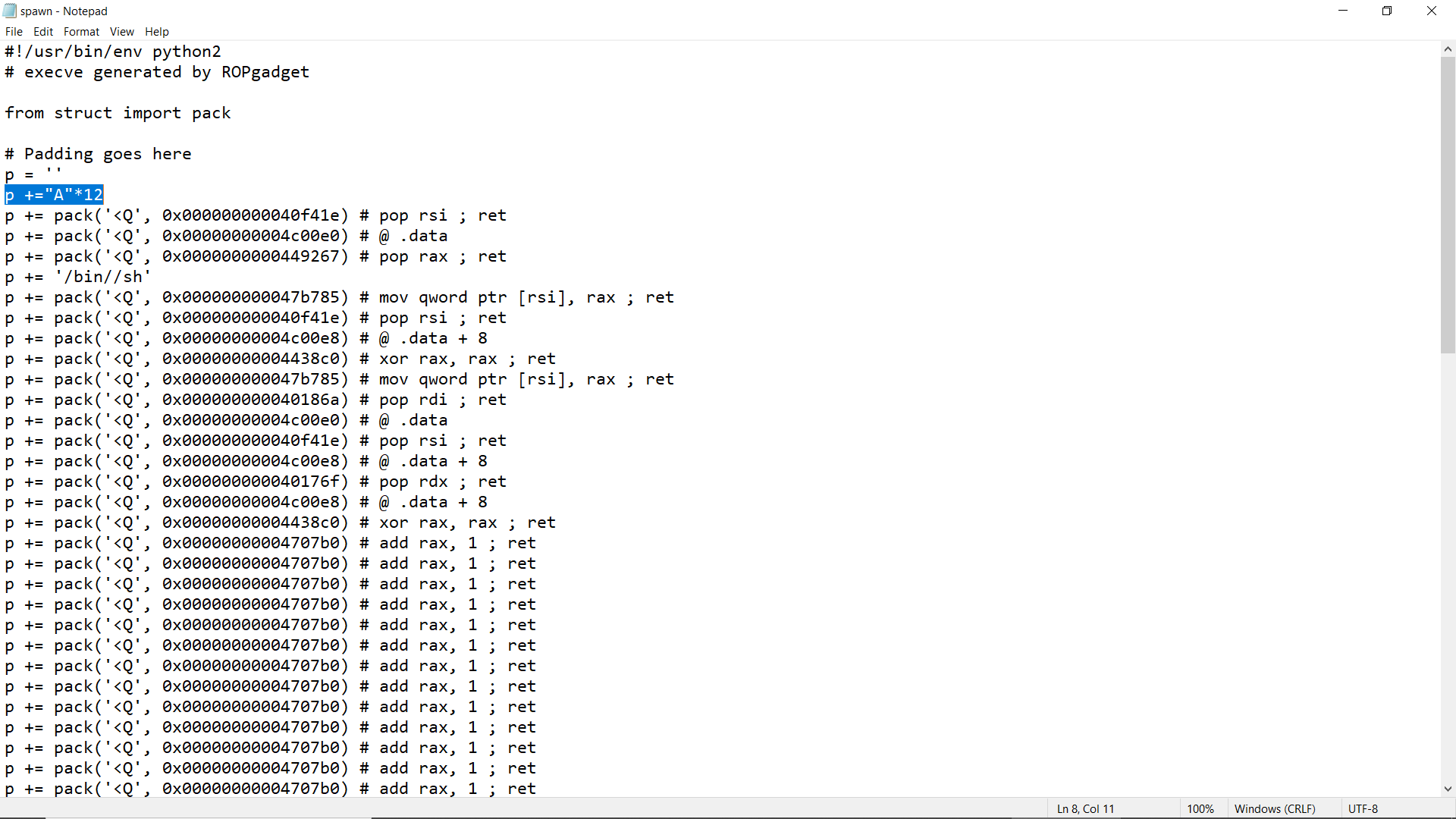
There are more than 1000 gadgets available in our program which is stores into this file.

**Step-3:**

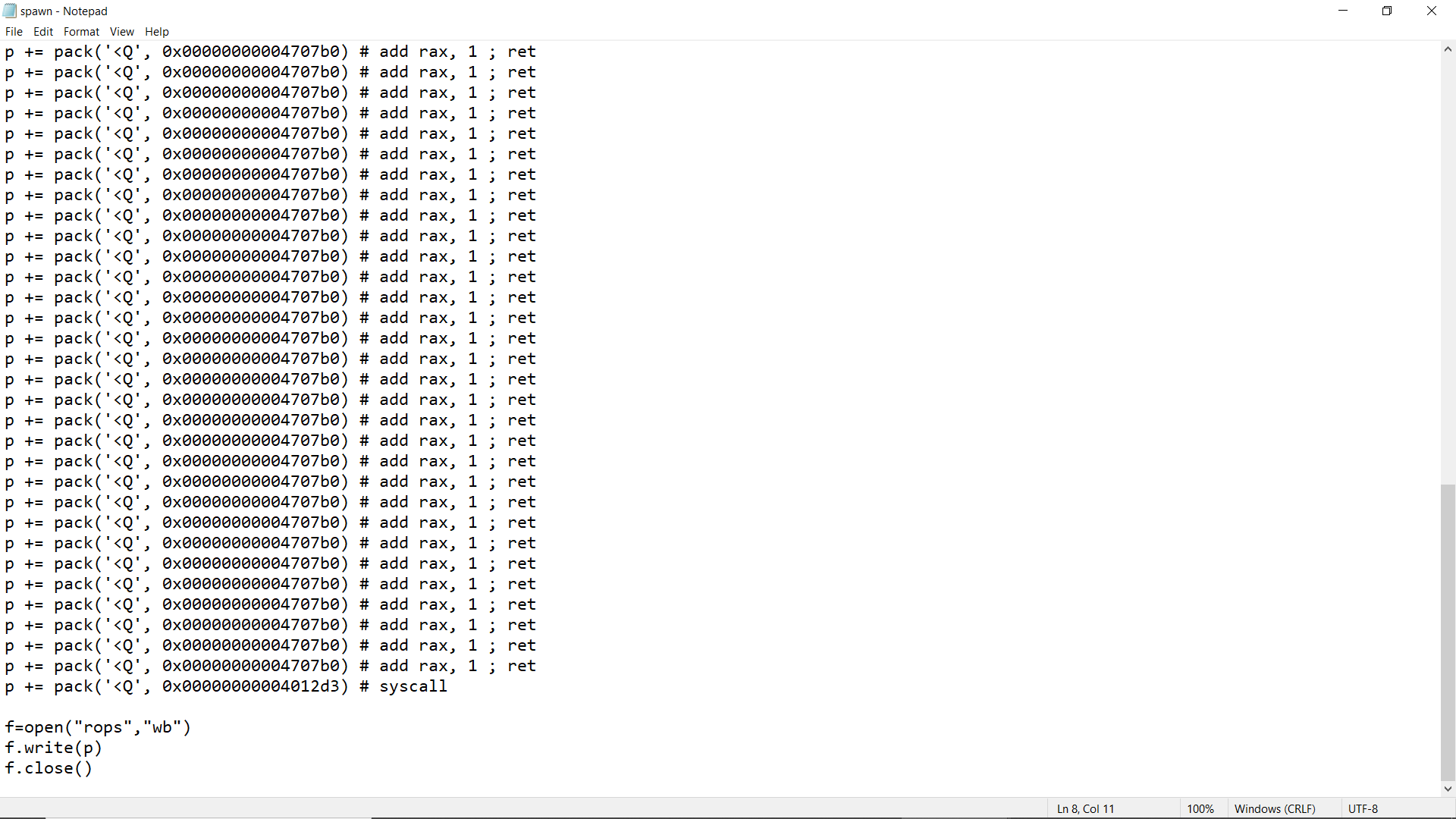
we need to find the string the Gadgets so that when we place them in the return address of the program instead of returning from the function , execute our ROP gadgets and the task desired by us. Here our task is to spawn a shell. The string of gadgets combined for spawning a shell is provided by ropgadget python module itself we place them in a file **spawn.**

Initially we need to add padding so that the buffer is filled with the padding provided by us and the chained ROP gadgets are placed in the return address field of the stack. At last, we are writing this chained string p inro a file called **rops** since our c program takes only files as argument.





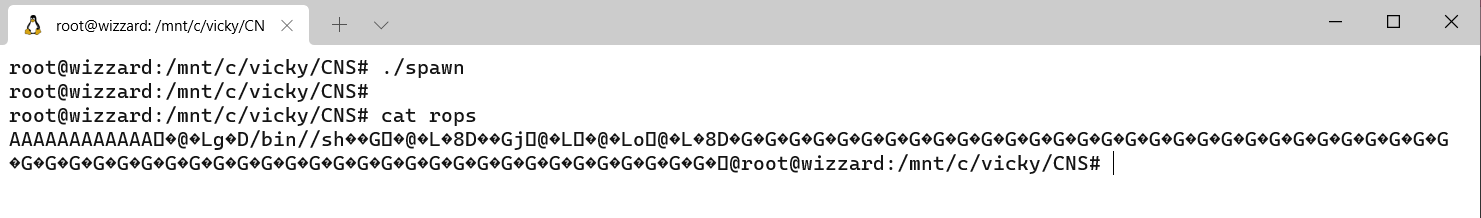
**Padding** is done by p=”A”\*12



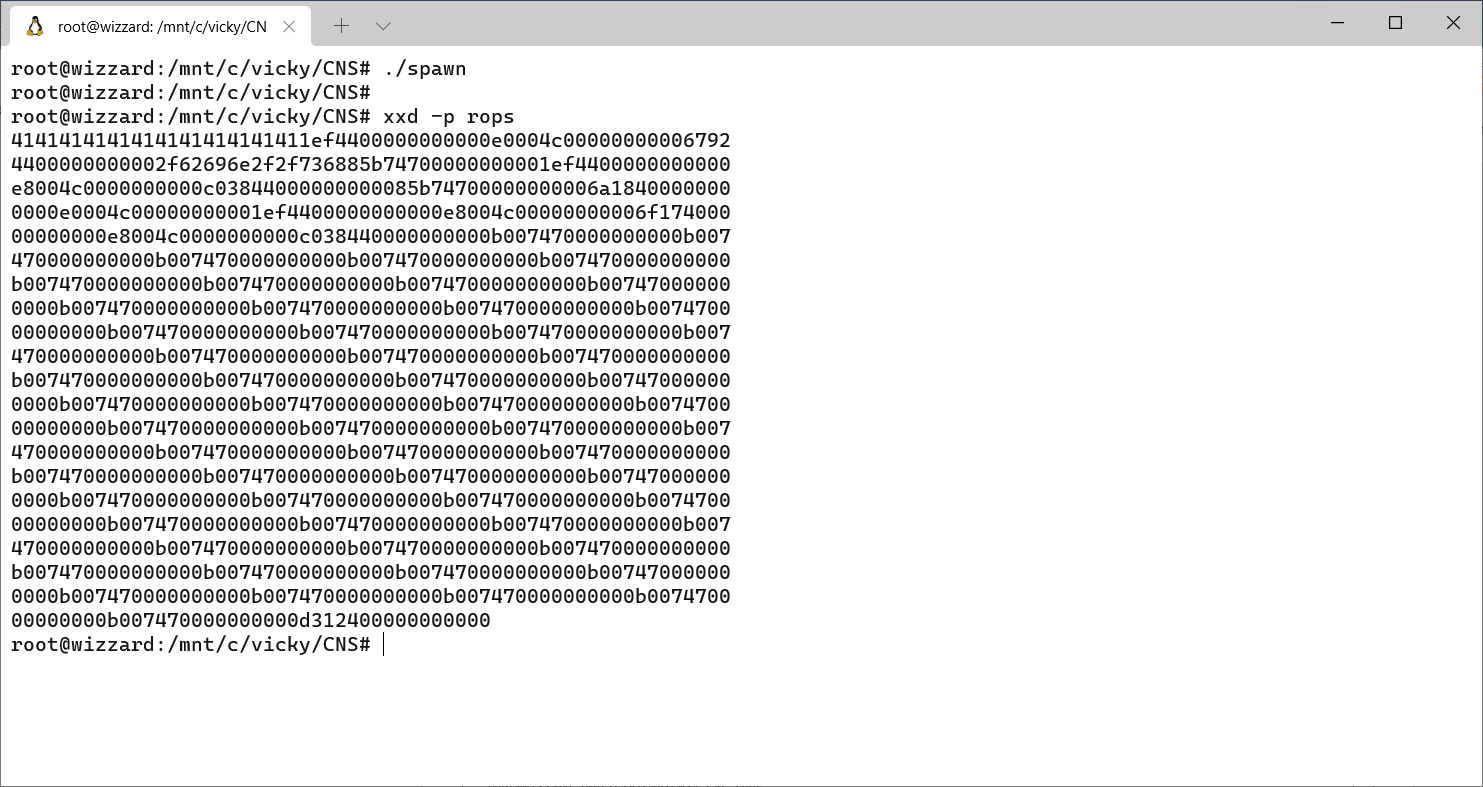
When this spawn file is executed via python it stores this chained rop gadgets **p**  into the file rops. Which is done in last 3 lines.

**Step-4:**

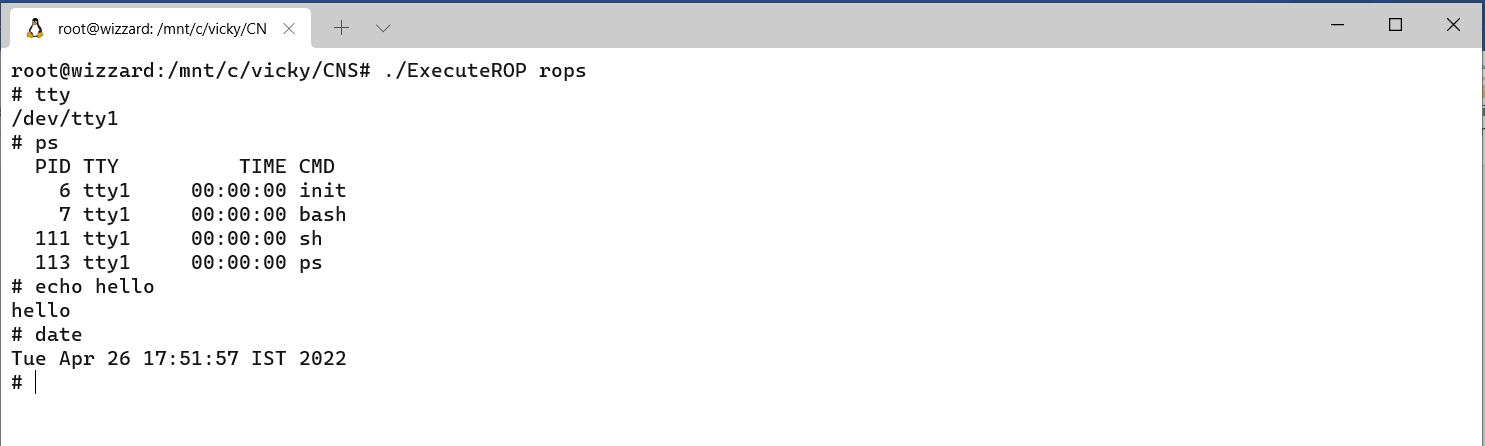
when we run the spawn file where the gadgets are padded, chained and placed into a file **rops.**



By cat command we can view those gadgets since they are binary format it is not visible.



Using xxd command we can see them.



These rop gadgets are passed to the c program which replaces the return address of the function stack and launch’s the shell. Now shell is launch with this the attacker can perform shell scripting tasks which can cause damage to our system.

**Conclusion:**

Rather than using libc function attackers are able to find return instructions called Gadgets and string them together, so that they can place it in the stack and execute. This can be prevented by using control flow integrity which uses control flow graphs and checks whether of our program flow deviates from what we have assigned to do. This finds if there is any change in flow by return oriented gadgets it terminates the program.