

Sri Lanka Institute of Information Technology

**Systems and Networking Programming**

**Individual Assignment**

“Buffer Overflow” Linux vulnerability

**R.M.S Sankalpa**

**IT19068374**

**Content**

Title page No

1 Abstract ……………………………………………………………. 03

2 Introduction ……………………………………………………….. 04

3 Exploitation process and mechanisms use to exploit ……………… 06

4 conclusion …………………………………………………………. 18

5 References …………………………………………………………. 19

**Abstract**

This report describes about what is the “Buffer Overflow” Linux vulnerability, how it happens, what is the damage it can cause, what are the exploitation techniques and how can we exploit it.

In here we show how anyone can exploit it with using a simple code. But in modern operating systems there are strong mechanisms to avoid this “Buffer Overflow” vulnerability. So, to exploit this vulnerability we need to disable them and run over code which occurs overflow in buffers. After a successful exploit, we can easily inject malicious codes to the system.

This vulnerability happens in a little mistake which is done by the programmer. Although it is very little mistake, it causes very huge damage by crashing our programs and giving opportunity to the attacker to inject malicious code into our programs.

**Introduction**

**Buffer overflow** is a Linux vulnerability which occurs when the buffer overflow. A memory buffer is an area in the computer’s memory (RAM) meant for temporarily storing data. This overflow happens when we try to fill this buffer with more data than it can handle. It’s like we are trying to fill 1 liter bottle with 2 liter water, so eventually an overflow happens.

When an attacker sends 11 integers (44 bytes) to a computer where there only can store 10 integers (40 bytes). Then the integers in the 10th locations will be overwritten by the overflow bytes. So the stack goes backward and the data in buffer are copied from lower memory address to highest memory address. When these overwritten things happen, programs will be crashed. This vulnerable thing happens when we using “strcpy” function in our code to store more values in less memory.

So, the solution is to use “strncpy” function. The difference is there is a “n” and it is safer than “strcpy”. With “strncpy” we can limit the data we are going to copy and avoid the buffer overflow. Then the program will copy only the limited data amount and others will be discarded. But “strcpy” function copy the data into memory addresses until there are no inputs, so the stored data in that addresses will be overwritten and happen overflow. This is a common software coding mistake that an attacker could exploit to gain access to your system.

**History**

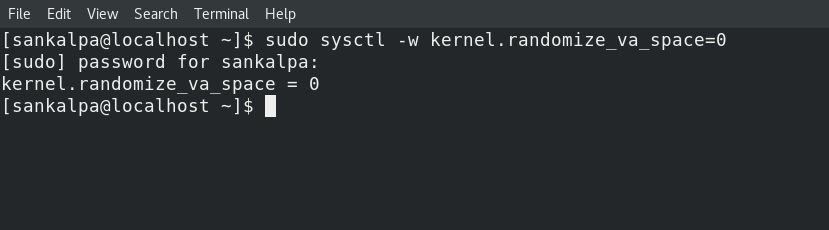
Buffer overflows were understood and partially publicly documented as early as 1972, when the Computer Security Technology Planning Study laid out the technique: "The code performing this function does not check the source and destination addresses properly, permitting portions of the monitor to be overlaid by the user. This can be used to inject code into the monitor that will permit the user to seize control of the machine." Today, the monitor would be referred to as the kernel.

The earliest documented hostile exploitation of a buffer overflow was in 1988. It was one of several exploits used by the [Morris worm](https://en.wikipedia.org/wiki/Morris_worm) to propagate itself over the Internet. The program exploited was a [service](https://en.wikipedia.org/wiki/Service_(systems_architecture)) on [Unix](https://en.wikipedia.org/wiki/Unix) called [finger](https://en.wikipedia.org/wiki/Finger_protocol). Later, in 1995, Thomas Lopatic independently rediscovered the buffer overflow and published his findings on the [Bugtraq](https://en.wikipedia.org/wiki/Bugtraq) security mailing list. A year later, in 1996, [Elias Levy](https://en.wikipedia.org/wiki/Elias_Levy) (also known as Aleph One) published in [*Phrack*](https://en.wikipedia.org/wiki/Phrack) magazine the paper "Smashing the Stack for Fun and Profit", a step-by-step introduction to exploiting stack-based buffer overflow vulnerabilities[1]

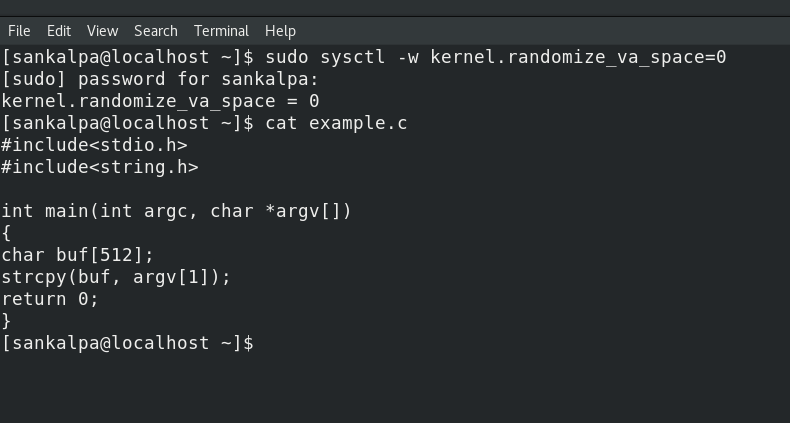
**Exploitation process and mechanisms use to exploit.**

Throughout the exploit procedure there are different techniques and steps to follow. These are the techniques and the steps.

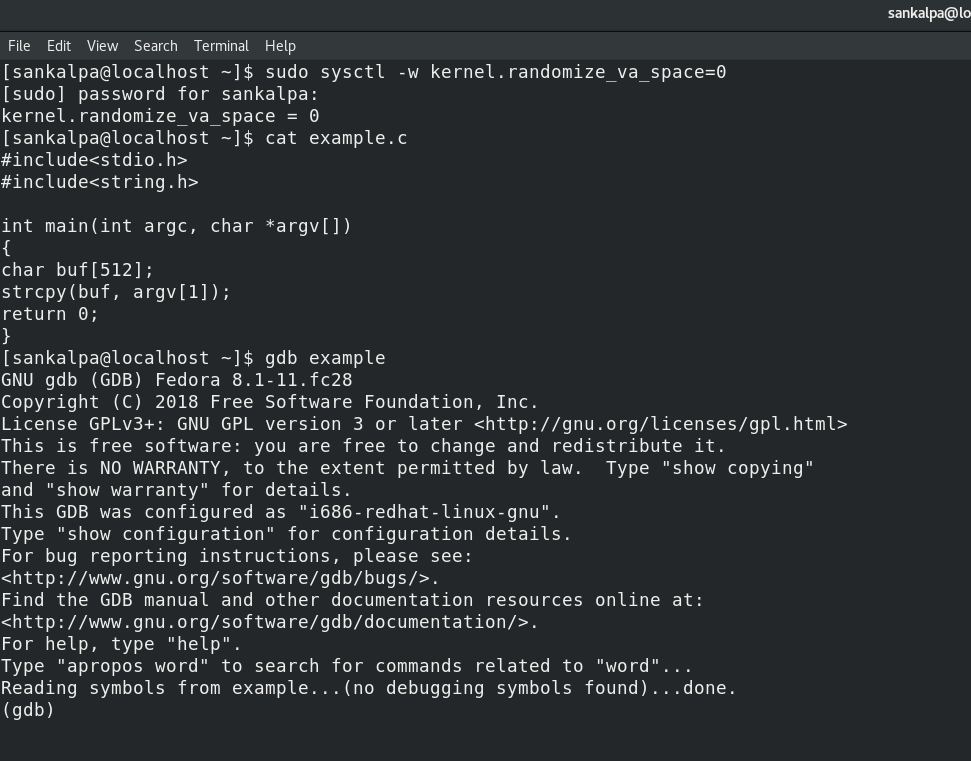
1 Disable the “kernel randomize \_va\_space” security method. [2], [3]



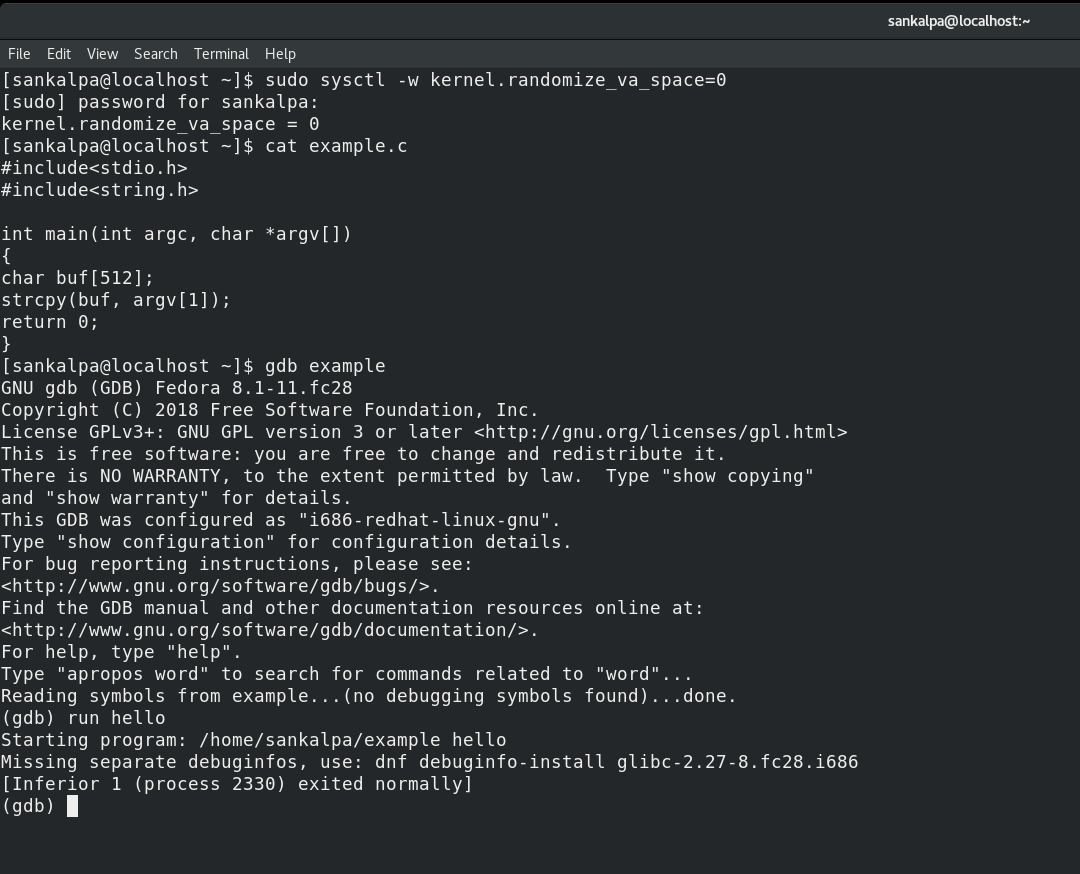
2 Then we look at our code where there is the “Buffer Overflow” vulnerability. [2], [3]



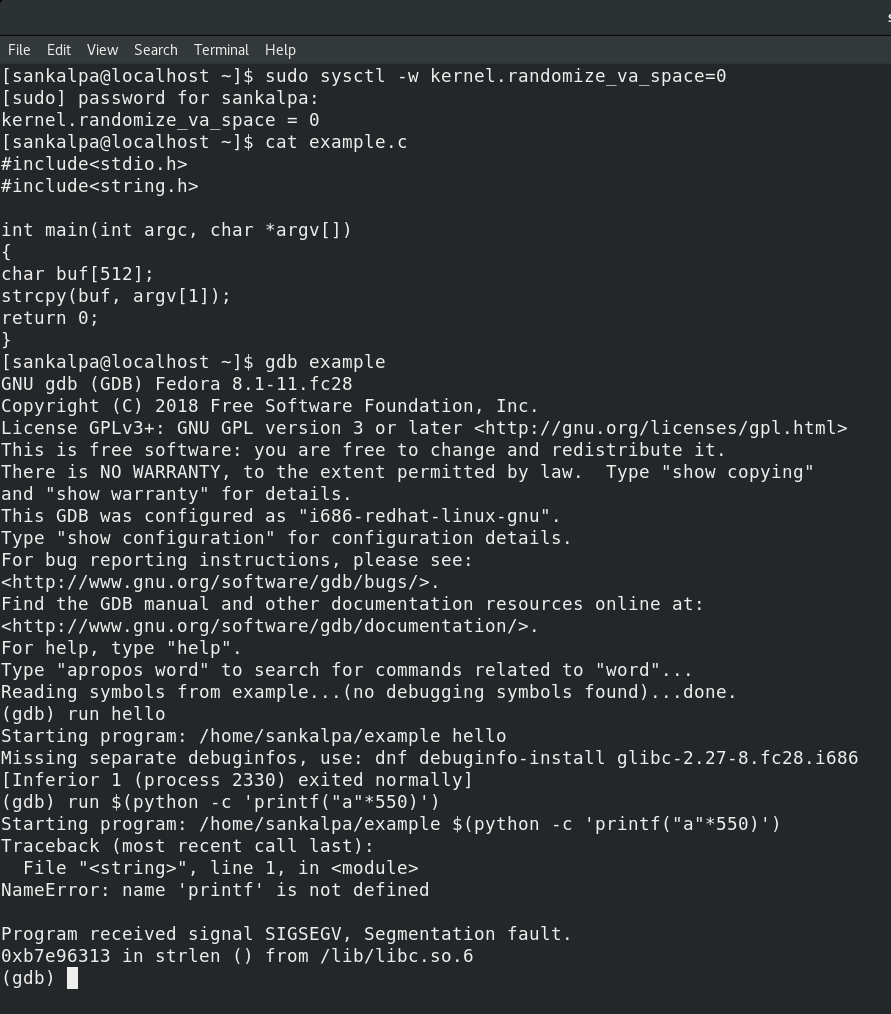
3 Then we open our executable file in “gdb”. [2], [3]

****

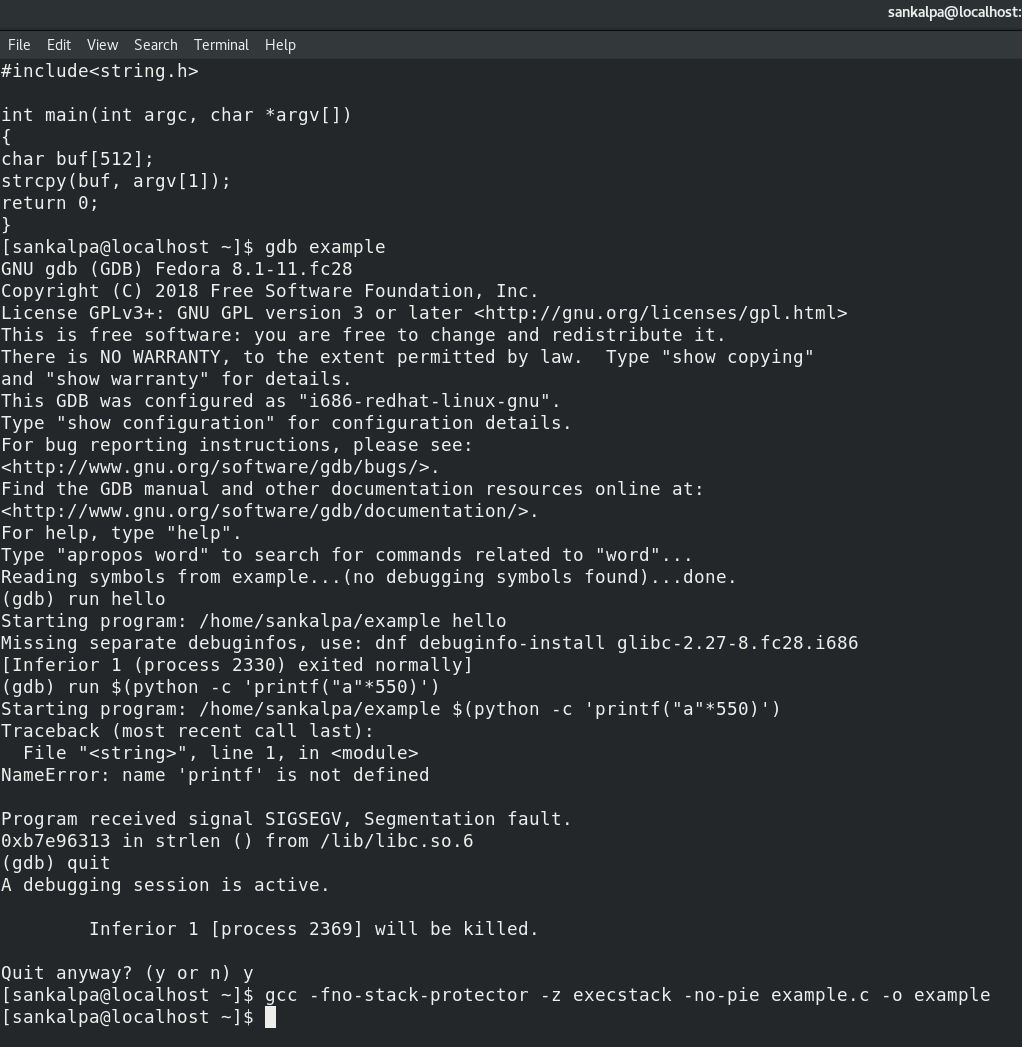
4 Then we enter some random input. [2], [3]



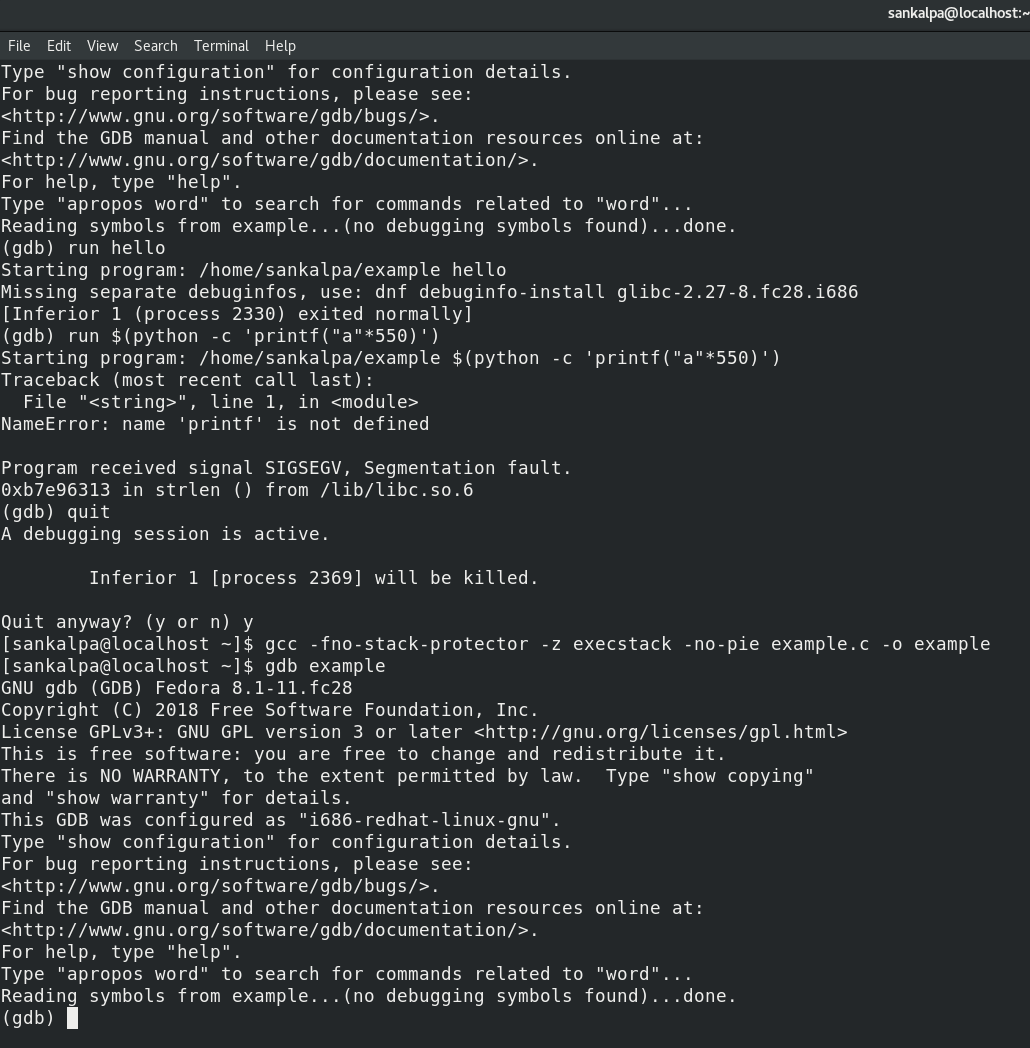
5 Then we give 550 “a” s, which is over than buffer. [2], [3]



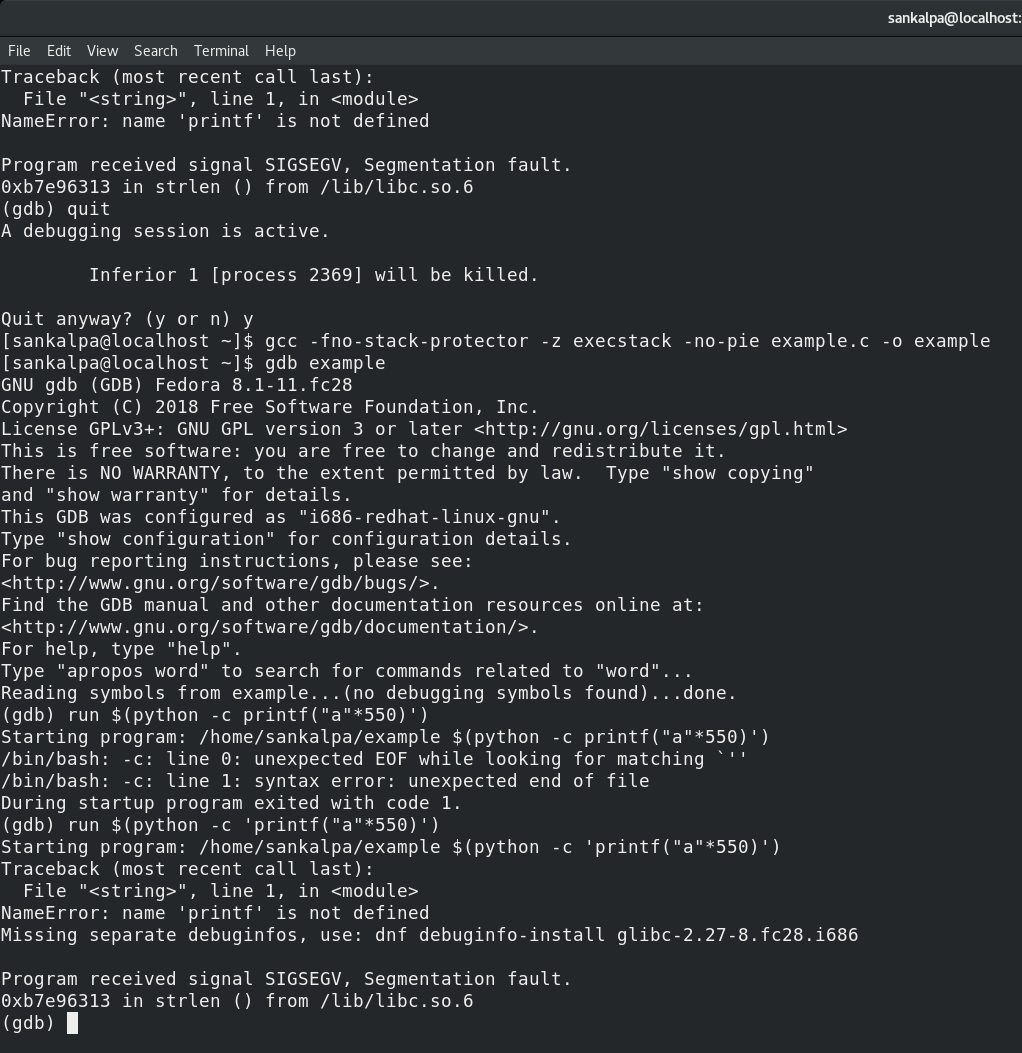
6 Then we exit from gdb and run a code with flags to disable other security methods which protect operating system from “Buffer Overflow”. [2], [3]



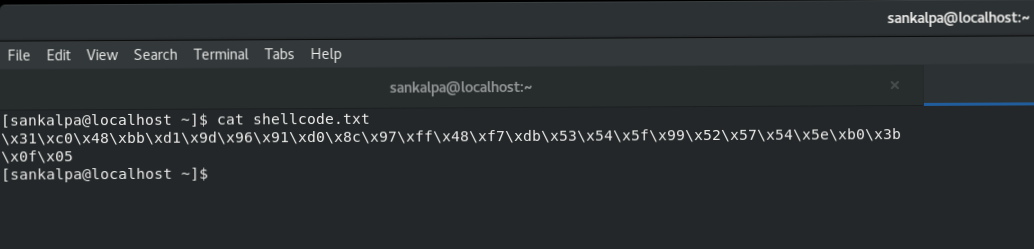
7 Then we come back to the gdb mode. [2], [3]



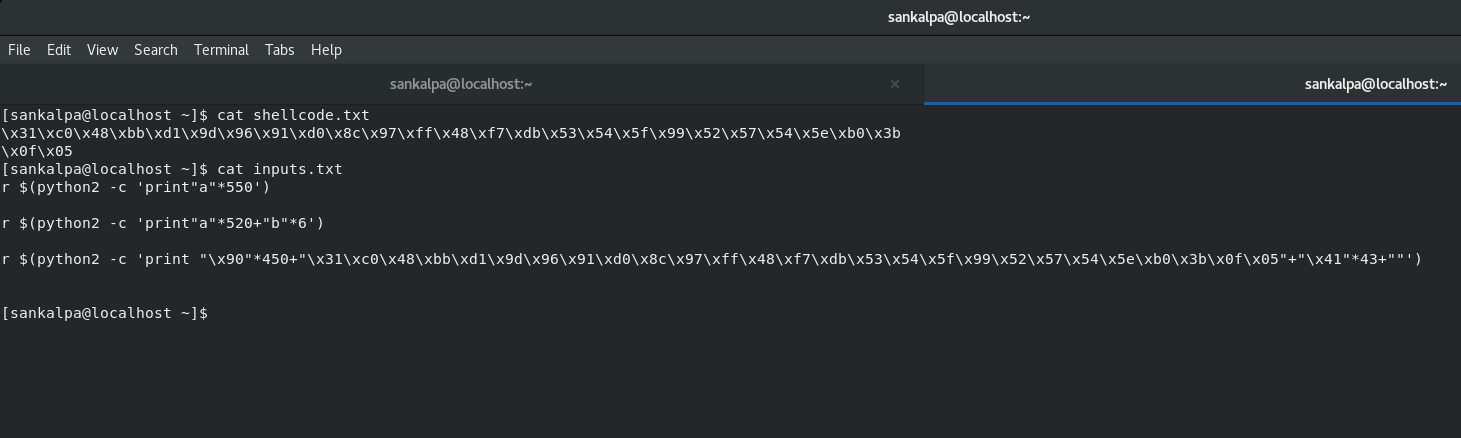
8 Then we run the previous code again. [2], [3]



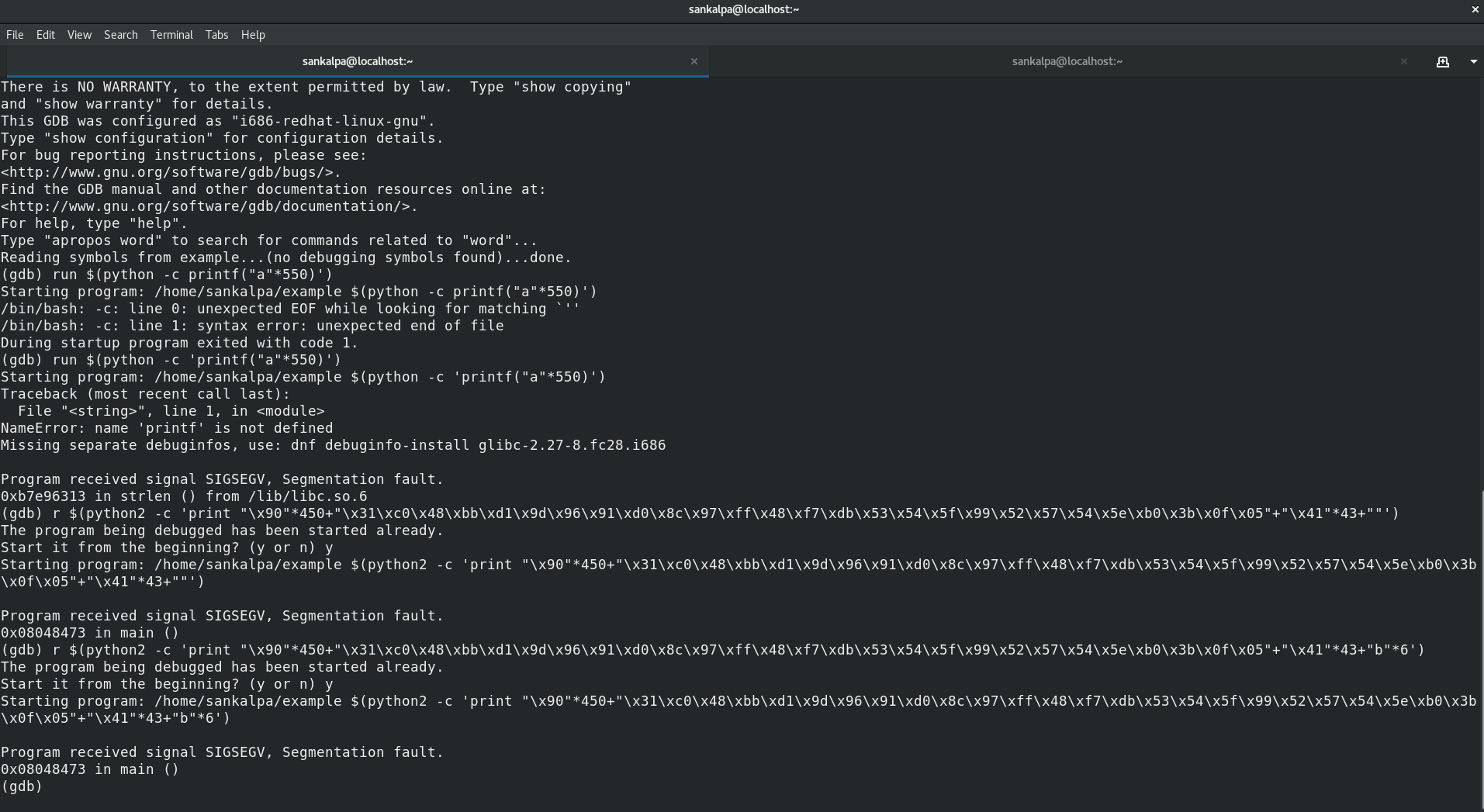
9 Then open a shellcode in another tab. [2], [3]



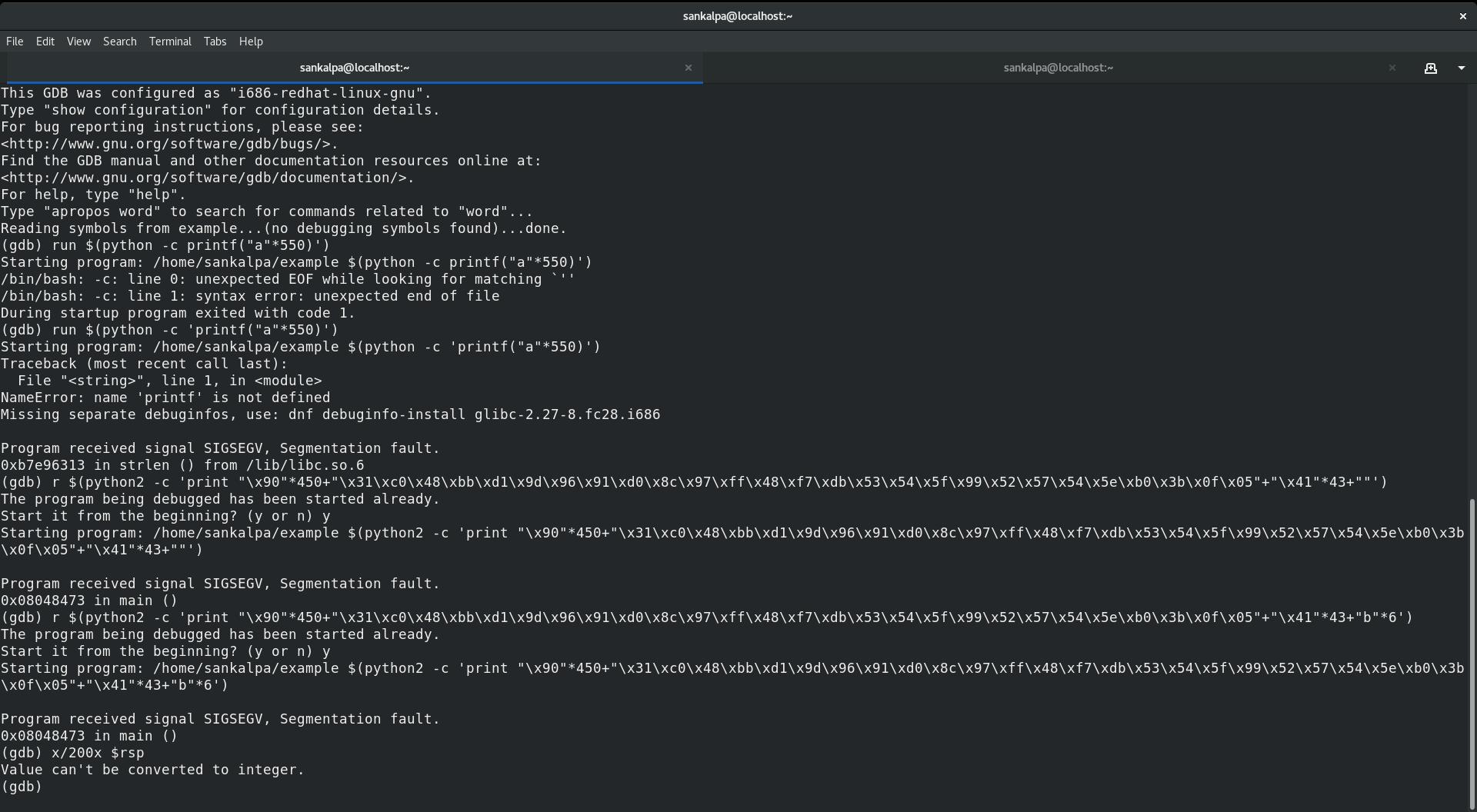
10 Open a file with NoOps. [2], [3]



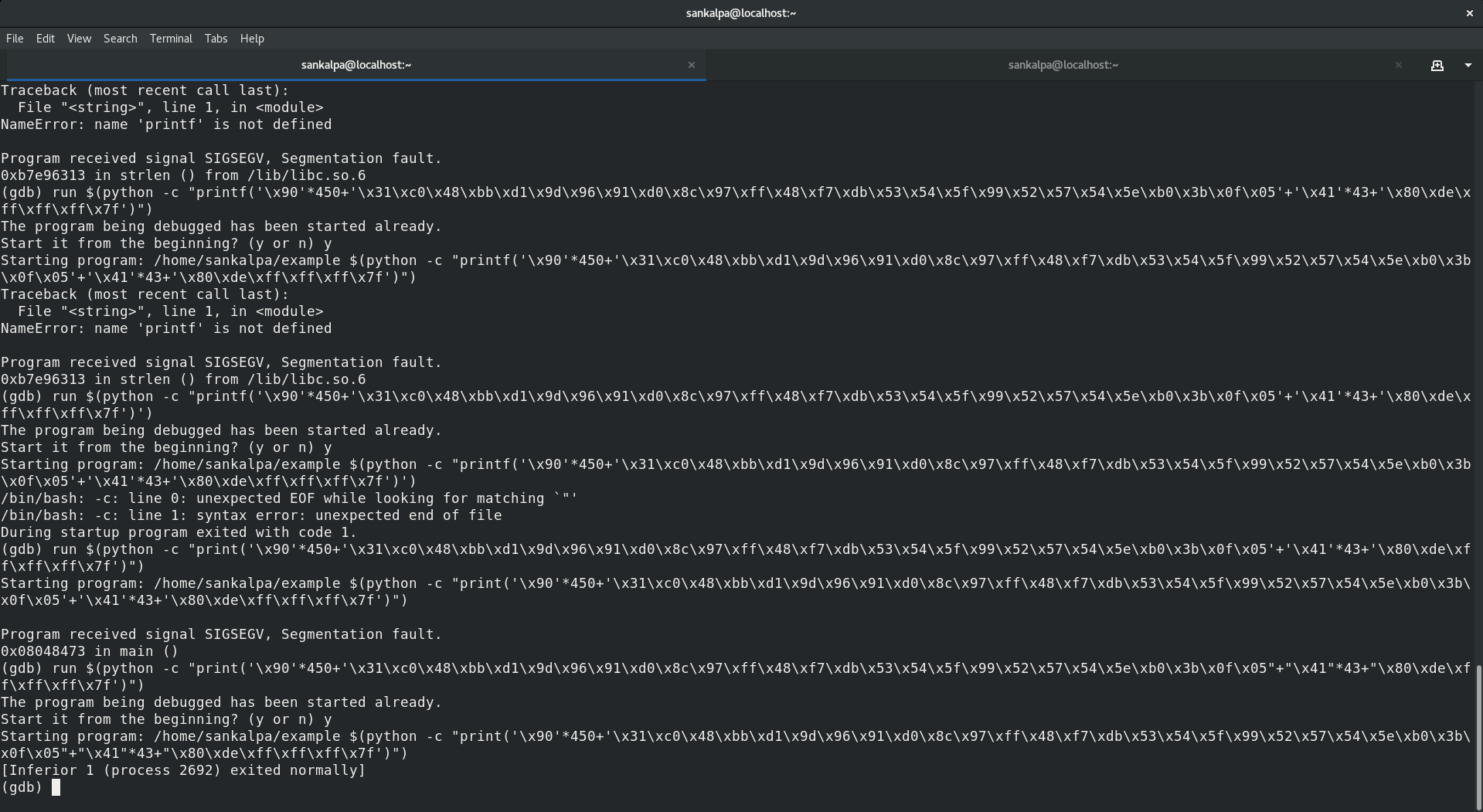
11 Run the NoOps in gdb. [2], [3]



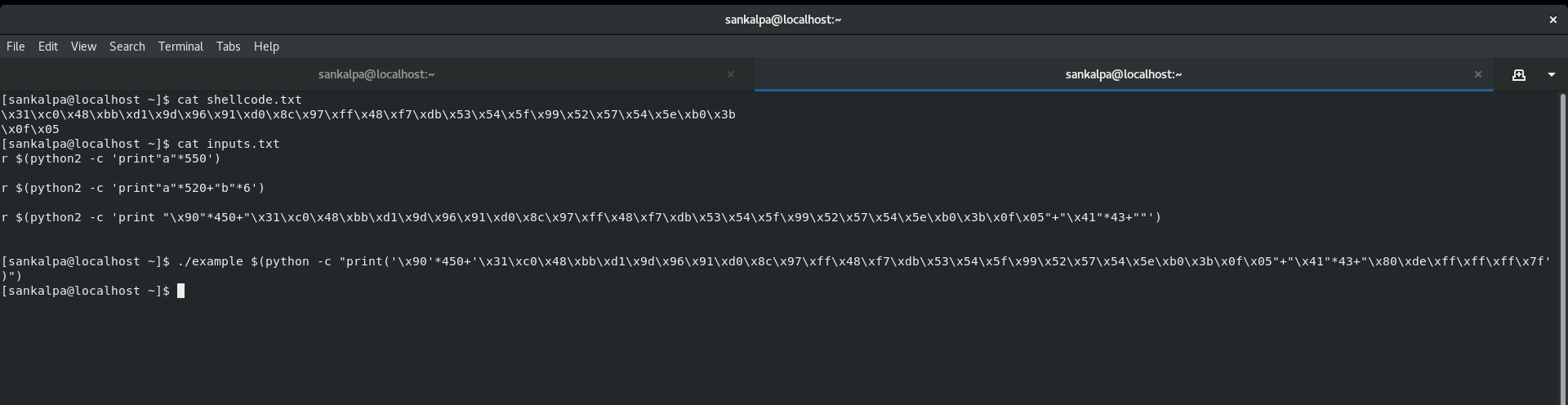
12 Then we run a code by using rsp. [2], [3]



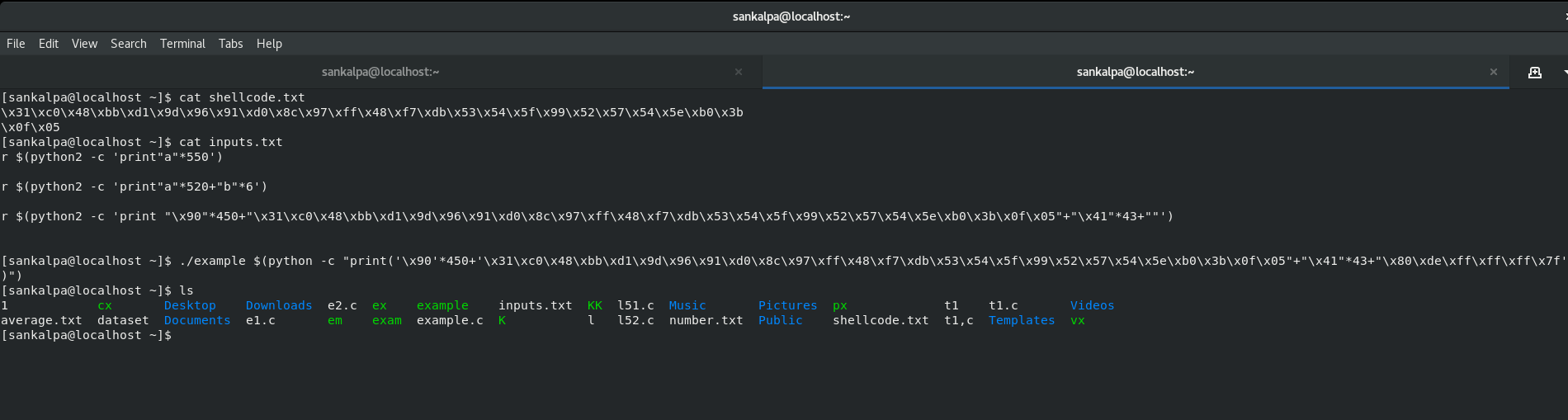
13 Again we run NoOps with some modifications. [2], [3]



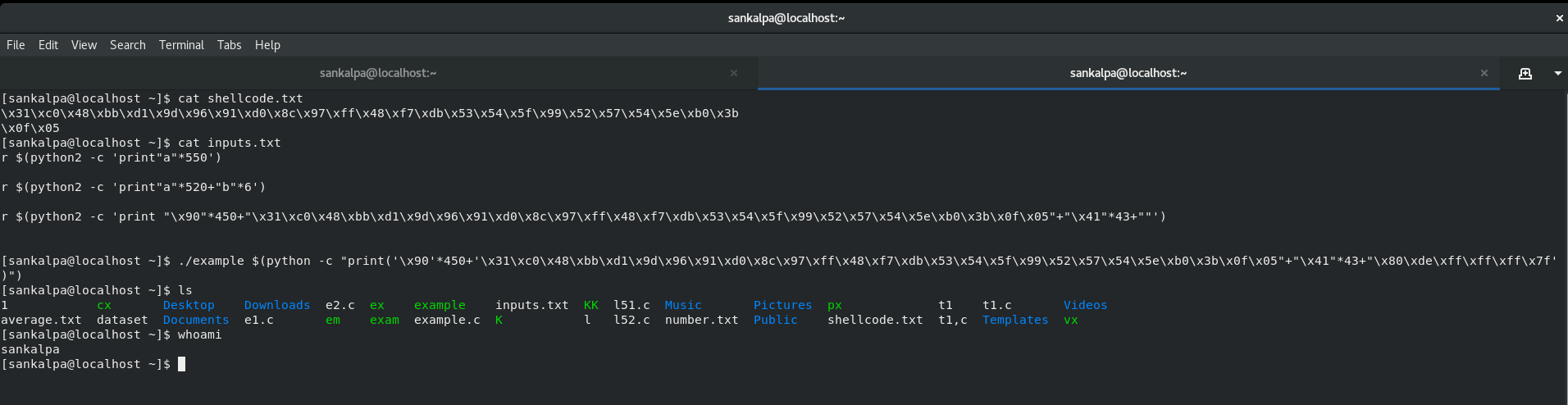
14 After that we run this NoOps with our executable file name.[2], [3]



15 Then we run “ls” command. [2], [3]



16 At last we can see we successfully exploit buffer overflow vulnerability by running “whoami” code. [2], [3]



**Conclusion**

By overflowing the memory area in RAM which we used to store temporary data, the user programs can be crashed and there is a chance of injecting malicious code. But anyone can exploit it by using a simple c code. Throughout this exploit, we should use different type of mechanisms. All these thing can be done in an operating system in Linux family. It takes only few minutes to exploit this vulnerability successfully. Modern operating systems have mechanisms to avoid buffer overflow vulnerability and in future there will be more secured operating system against buffer overflow vulnerability. So, in future the attacker have to pay more effort and time than now.

**References**

[1] “Buffer overflow - Wikipedia.” https://en.wikipedia.org/wiki/Buffer\_overflow (accessed May 12, 2020).

[2] “How to exploit a buffer overflow vulnerability - Practical.” .

[3] “Buffer Overflow attack on a 64-bit ubuntu using gdb-peda.” .

**Thank You!**