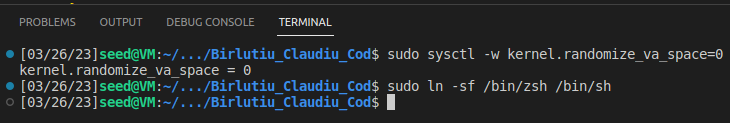
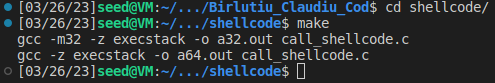
# Raport pentru lucrarea 3: Atacul cu depășire de zonă de memorie (versiunea Set-UID)

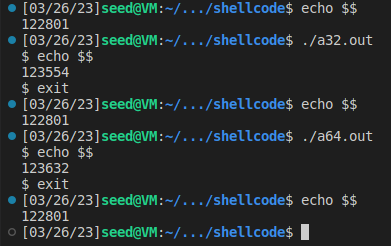
Autor: Birlutiu Claudiu-Andrei

## Sarcina 1: Familiarizarea cu ShellCode

* Pregatirea mediului prin executarea comenzilor din laborator
* am rulat comanda **make** în directorul shellcode și aceasta a creat cele 2 fisiere executabilele ale shellcode-ului (pe 32 de biti și 64 de biti) și de asemenea la crearea lor s-a folosit optiunea de execuție de pe stiva

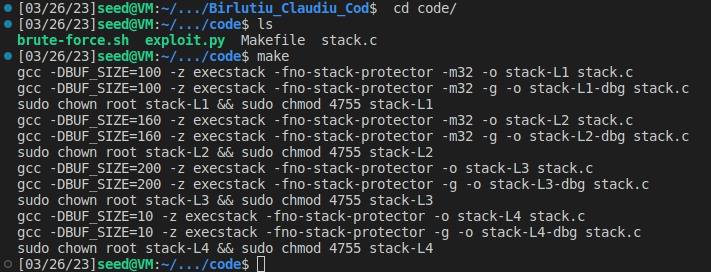


* am lansat în execuție cele 2 executabile și am observat ca în ambele cazuri se deschide un shell nou pe un proces nou (id-uri diferite). Se poate observa în ss-ul de mai jos cum sa deschid cele 2 shell-uri cu id diferit

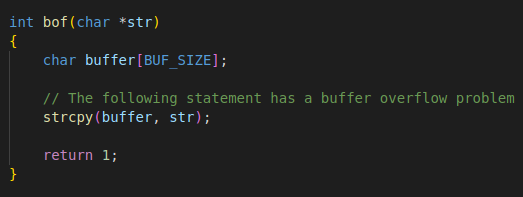


## Sarcina 2: Înțelegerea programului vulnerabil

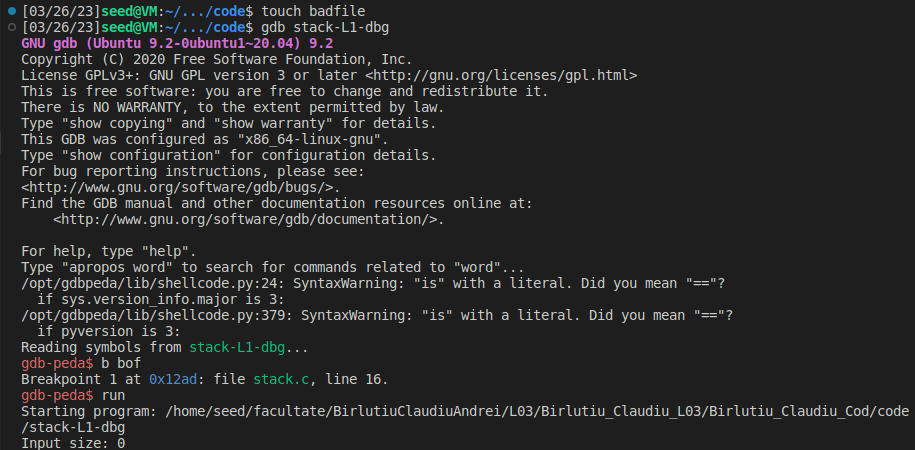
* Am rulat comanda **make** astfel ca s-au construit mai multe fisiere executabile ale preogramului stack (pe 32 și 64 de biti) cu valori ale dimenisunii buffer-ului diferite și de asemenea crearea programelor pe care se poate face debugging

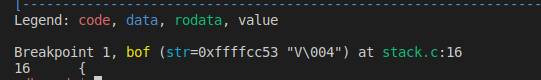


* codul stack.c manifesta o vulnerailitate în cadrul functiei bof din cauza faptului ca dimensiunea string-ului primit ca parametru și care e copiat cu ajutorul functiei strcpy (care nu verifica dimensiunile maxime destinație – sursa) => lungimea sirului str poate fi mai mare decât BUF\_SIZE = > se vor copia bytes într-o zona nedorita

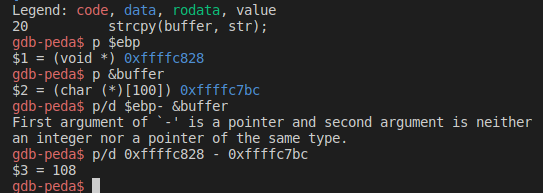


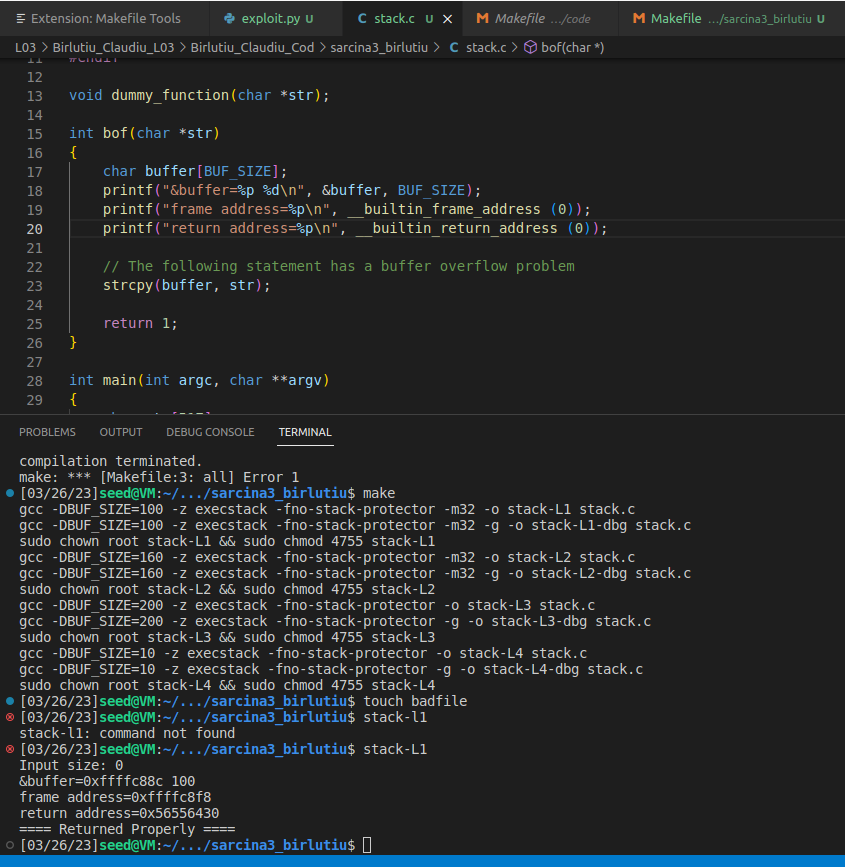
## Sarcina 3: Lansarea atacului asupra programului pe 32 biți

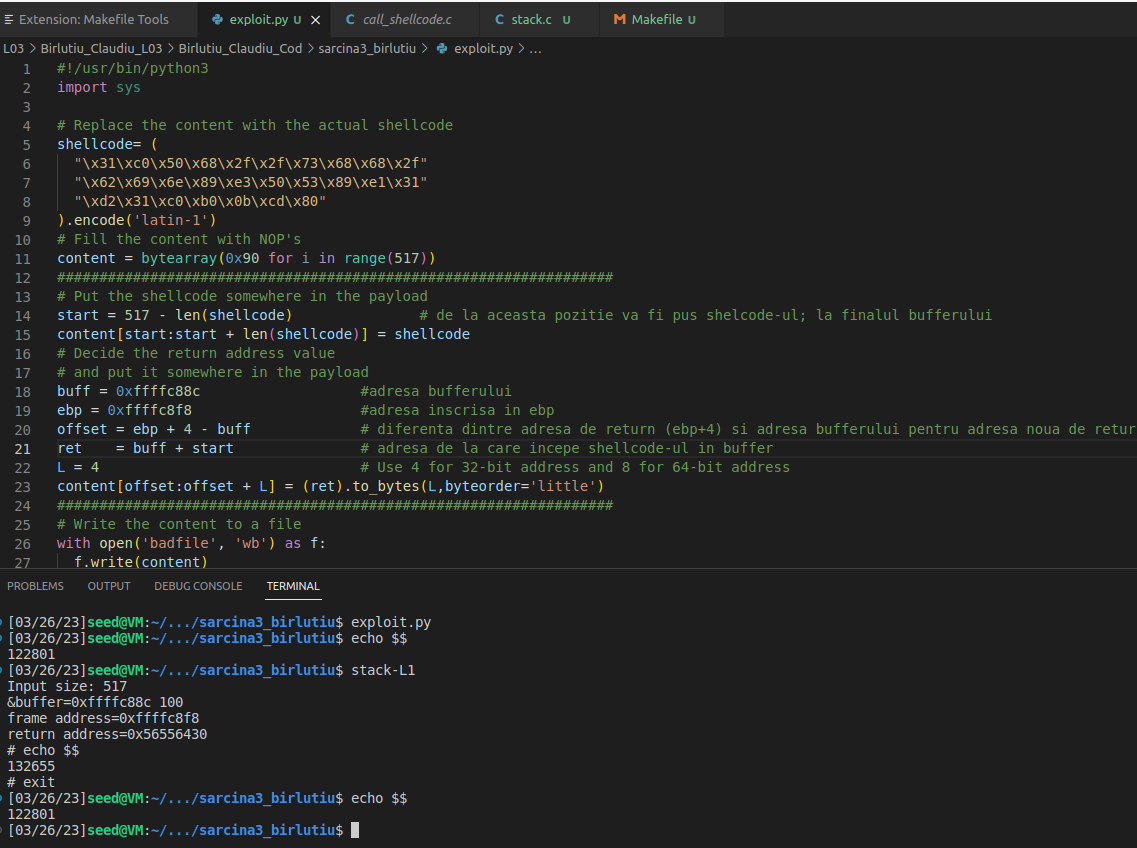
* În prima faza se va afla distanta dintre pozittia de inceput a tamponului si locul unde este stocata adresa de retur.
* Am lansat cu ajutorul tool-ului gdb debugging-ul pe programul stack-L1-gdb și am pus un breakpoint la nivelul functiei bof
* se observa ca executia se oprește la nivelul breakpoint-ului pus



* se da un **next** și se printeaza apoi valoarea registrului ebp (în acest caz ebp va indica cadrul de stiva al functiei **bof**) și a adresa buffer-ului; Trebuie remarcat faptul ca valoarea pointerului de cadru obtinu de la gdb este diferita de cea din timpul executiei propriu-zise (fara a folosi gdb).



* S-a observat ca diferența intre adresa buffer-ului și adresa functiei se afla la o diferența de 0x108 = 264
* deaorece aceste valori nu sunt corecte din cauza introducerii unor valori diferite în stiva de modul debug am recurs la metoda de a indtroduce în codul **stack.c** a unor printf-uri pentru adresele necesare
* onbservam ca adresa buffer-ului este 0xffffc88c, iar adresa frame-ului este 0xffffc8f8 => diferența este 6C, adică 108
* adresa de retur va fi ebp + 4 => de unde rezulta ca diferenta este de 108+4 = 112 : un calcul mai simplu se face in felul urmator
* start = 517 - len(shellcode) # de la aceasta pozitie va fi pus shelcode-ul; la finalul bufferului
* content[start:start + len(shellcode)] = shellcode
* buff = 0xffffc88c #adresa bufferului
* ebp = 0xffffc8f8 #adresa inscrisa in ebp
* offset = ebp + 4 - buff # diferenta dintre adresa de return (ebp+4) si adresa bufferului pentru adresa noua de return; va reprezenta pozitia din buffer unde va fi pusa adresa de return noua
* ret = buff + start # adresa de la care incepe shellcode-ul in buffer; aceasta va reprezenta adresa de return a functiei => se va executa astfel shelcode-ul



## Sarcina 4: Lansarea atacului asupra programului pe 64 biți

## Sarcina 5: Înfrângerea contra-măsurii din dash

## Sarcina 6: Înfrângerea randomizării adreselor

## Sarcina 7a: Protecția StackGuard activată

## Sarcina 7b: Protecția prin stivă ne-executabilă activată