

Homework-3

Access Control - 20 Points

1. Briefly describe the differences between Capabilities and Access Control lists. Under what conditions is one approach better than the other one?

Answer:

Difference 1: Predicates

If we compare the predicates of two systems then we can find significant difference between the two systems:

ACL: $op \text{ in } acl(object, principal(p))$

Capability: $op \text{ in } ops(caps(p,i))$

a) Difference is that the capability model makes no reference to any notion of "principal"

b) The capability model has parameter "i", only this model can solve the confused deputy problem.

Difference 2: Access Rights and Persistence

Condition: When the computer shuts down and all of the processes disappear.

ACL: a) NO problem: Reason: login sessions disappear too.

The user identity for a process is derived from who starts it, which is derived from login session.

No need to record permission on pre-process basis.

Capabilities: a) Definite problem. Reason: Solutions vary.

Some systems associate an initial capability list with each login.

Difference 3: Least Privilege

Condition: Trust on Programs

a) ACL is Better: Access control list systems are coarser; every process executed by "fred" has the same rights. If there is always trust on programs then ACL is better.

Condition: Computer viruses

b) Capability is better: If computer viruses are front page news.

Difference 4: Revocation

Condition:

a) ACL: You can remove a user from the list, and that user can no longer gain access to the object.

b) Capability: There is no equivalent operation. Some mechanisms are there but they are cumbersome.

ACL is object/resource centric, works perfectly where access control is set up by a central authority, but not good when users want to create more complex policies like delegating their rights, or where the user population is large and changes frequently. ACL is not good for revocation or for checking system-wide the access rights of all users as it requires to go through millions of files.

Capability is user centric. It is easier to create more complex policies with Capability.

2. Describe the differences between Discretionary Access Control (DAC) and Mandatory Access Control (MAC). Are the file permissions in Linux an example of DAC or of MAC?

Ans: Discretionary Access Control (DAC)

In discretionary access control (DAC),

- a) DAC provides access based on identity.
- b) DAC is more labor intensive than MAC
- c) DAC access can be provided by other users
- d) The owner of the object specifies which subjects can access the object. This model is called discretionary because the control of access is based on the discretion of the owner.
- e) Here, one needs to know each person who needs the resource so that they can be given access.
- f) DAC provides users who have access to the resource to also provide access to other users by including them in the list. This can be problematic in case of a malicious access addition.

Mandatory Access Control (MAC)

In mandatory access control (MAC),

- a) MAC provides access based on levels.
- b) MAC is less flexible than DAC.
- c) MAC access can only be changed by admins.

d) The system admins (and not any user) specifies which subjects can access which specific objects. This model is called mandatory because it is mandatory to obey the policy set by the admin.

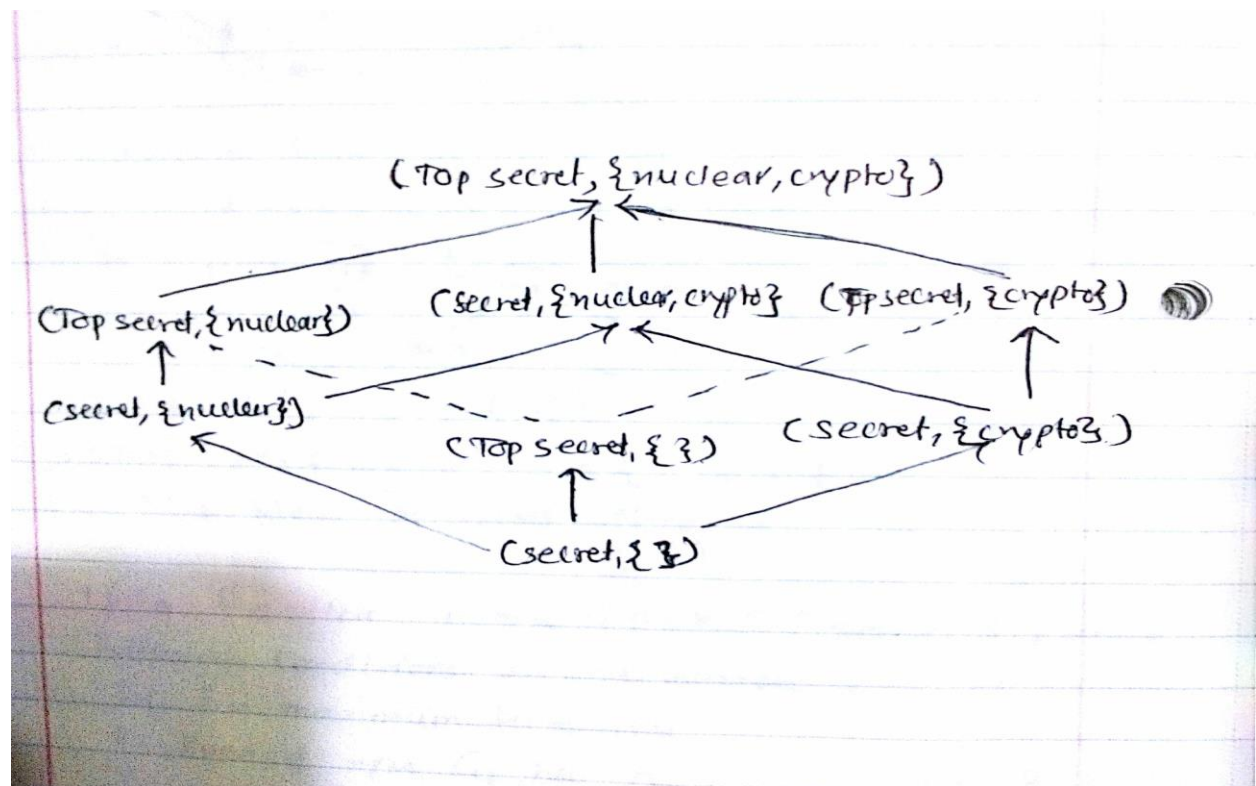
e) MAC is easier to implement as establishing and maintaining access control we just need to establish a single level for each resource and one level for each user. It is easier for admins to keep track of who has access to what because it is only them who can change the permission levels with MAC.

File permissions in Linux is an example of DAC.

3. Is the Chinese Wall security policy an example of DAC or MAC?

Ans: It is multi-Lateral Security Policy. Chinese Wall security policy is an example of both DAC and MAC.

4.



5. Read Ben Laurie's description of Capsicum at <http://www.links.org/?p=1242>. Summarize in one paragraph the main intuition behind Capsicum (i.e., what is it useful for, and how is it implemented) .

Ans: I have read Ben Laurie's description of Capsicum at given link. I found that Capsicum is a lightweight OS capability and sandbox framework which implements a hybrid capability system model. Capsicum can be used for application and library compartmentalization, the decomposition of larger bodies of software into isolated (sandboxed) components in order to

implement **security policies and limit the impact of software vulnerabilities**. He also used error driven development to achieve goal.

The paragraph also tells about the writers encounter with capsicum and him changing from bzip2 to Capsicum. Bzip2 was prone to buffer overflows or integer over/underflows – and these often lead to remote code execution. But Capsicum is solution to mitigate this problem, to defend a decompressor it is necessary to run decompression engine in a separate engine. Capsicum runs the decompression engine in a separate process with no privilege beyond that needed to get its job done, once the appropriate files are open, fork the process and enter capability mode in the child. Removes all permissions except the ability to read the input and write the output and then go ahead and decompress. Then you have to make one observation that If a bug is discovered the attacker has pretty much same what he had before, the ability to read the input file and the ability to write arbitrary content to the output file, burn CPU and consume memory. But that's it – no access to your files, the network, any other running process, or anything else interesting. Also he is asking for feedback to present development in experimental way.

6. Apache web servers provide access controls with their .htaccess file. How can you configure this file to allow access only to an specific IP address?

Ans: We can use .htaccess to ban or allow access to your web site by IP address.

Suppose IP address: 1.2.3.4

Place this code in .htaccess file

```
# ALLOW USER BY IP
```

```
<Limit GET POST>
```

```
order deny,allow
```

```
deny from all
```

```
allow from 1.2.3.4
```

```
</Limit>
```

.htaccess files (or "distributed configuration files") provide a way to make configuration changes on a per-directory basis.

In apache server access can be controlled based on the client hostname, IP address, or other characteristics of the client request, as captured in environment.

The Allow and Deny directives let you allow and deny access based on the host name, or host address, IP address range.

Buffer Overflow Vulnerability Lab - 80 Points

Lab Tasks:

2.1. Initial SetUp: Installed SeedUbuntu on the Virtual Box.

Task 1: Exploiting the vulnerability:

Disable Address Space Randomization as SeedUbuntu uses address space randomization to randomize the starting address of heap and stack which makes guessing the exact addresses difficult.

```
$ su root
```

Password: *****

```
#sysctl -w kernel.randomize_va_space=0
```

The StackGuard Protection Scheme and Non-Executable Stack will be disabled while running the code.

Next, I compiled stack.c vulnerable program and make it set-root-uid. Achieved this by compiling it in the root account, and chmod the executable to 4755.

execstack and -fno-stack-protector options to turn off the non-executable stack and StackGuard protections.

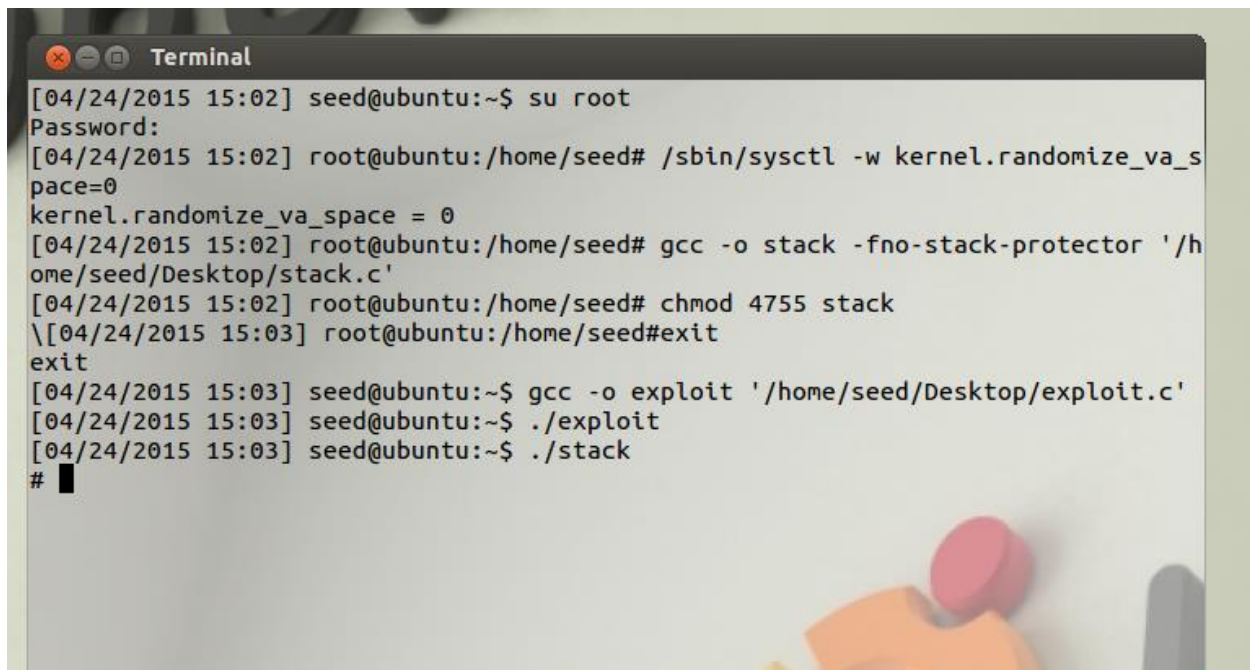
The goal of exploit.c code is to construct contents for “badfile”. In this code, the shellcode is given to us we need to develop the rest.

What is happening in the code is copying our shellcode into the buffer. strcpy() will then copy this onto buffer with strcpy() not checking the boundaries, and will overflow the return address, overwriting it with the address where our code is now located.

Once we reach the end of main and it tried to return it jumps to our code and executes our shellcode.

Then we run the vulnerable code and if our attack is successful we must get the shell prompt now.

Hence, we were successful in gaining the root access using Buffer Overflow Vulnerability.

A terminal window titled "Terminal" with a dark header bar. The window shows a series of commands and their outputs in a monospaced font. The background of the terminal is light gray, and the text is black. The commands include switching to root, using sysctl to disable address randomization, compiling a stack.c file, setting permissions, and then compiling and running exploit.c and stack.c files. The terminal is overlaid on a blurred background of colorful geometric shapes.

```
[04/24/2015 15:02] seed@ubuntu:~$ su root
Password:
[04/24/2015 15:02] root@ubuntu:/home/seed# /sbin/sysctl -w kernel.randomize_va_s
pace=0
kernel.randomize_va_space = 0
[04/24/2015 15:02] root@ubuntu:/home/seed# gcc -o stack -fno-stack-protector '/h
ome/seed/Desktop/stack.c'
[04/24/2015 15:02] root@ubuntu:/home/seed# chmod 4755 stack
\[04/24/2015 15:03] root@ubuntu:/home/seed#exit
exit
[04/24/2015 15:03] seed@ubuntu:~$ gcc -o exploit '/home/seed/Desktop/exploit.c'
[04/24/2015 15:03] seed@ubuntu:~$ ./exploit
[04/24/2015 15:03] seed@ubuntu:~$ ./stack
#
```

Task 2: Address Randomization:

I ran task 1 as it is with address randomization switched on.

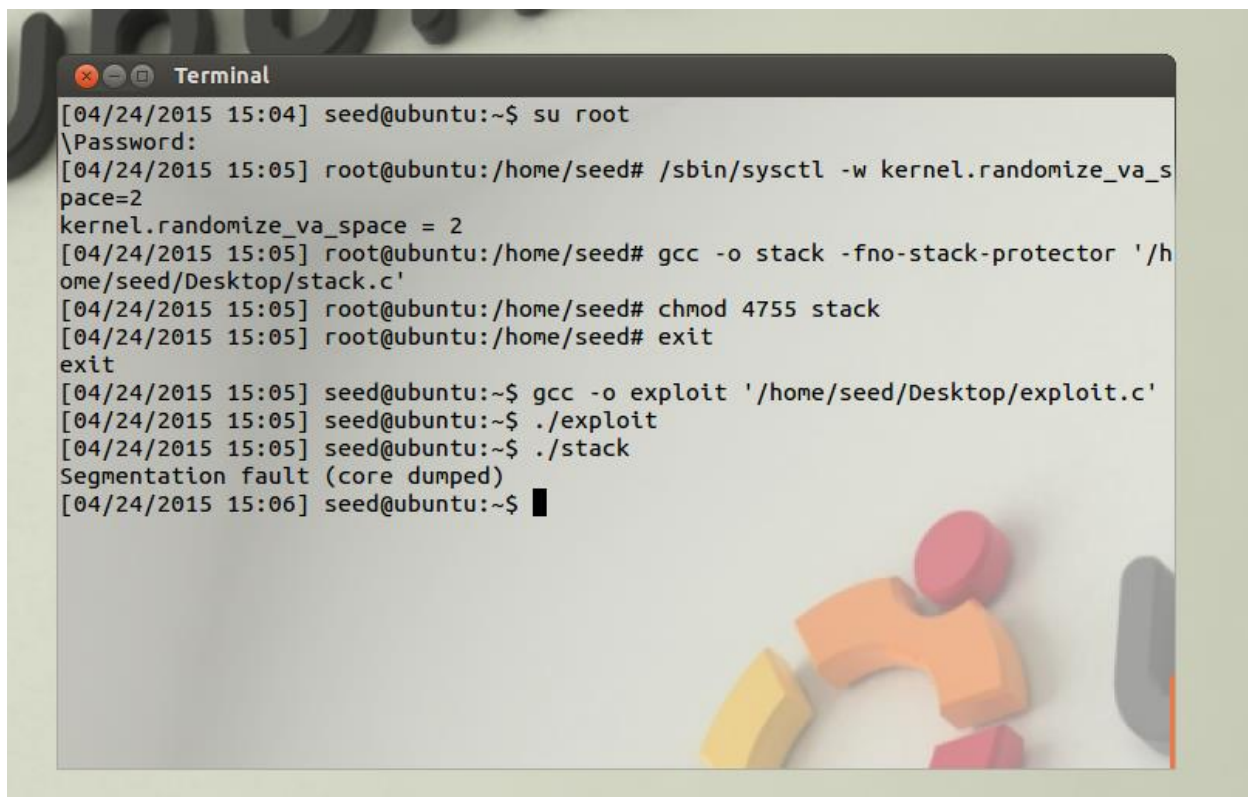
I got a segmentation fault as Address randomization was on. (refer Screen shot)

Can you get a shell?

No, we couldn't. refer screen shot.

If not, what is the problem? How does the address randomization make your attacks difficult?

Address Randomization randomizes where items are in memory to make the task of injecting malicious code more difficult. Our Buffer overflow attack fails as it depends on knowing where items are located in memory to be able to inject code that can make valid memory references.



```
Terminal
[04/24/2015 15:04] seed@ubuntu:~$ su root
Password:
[04/24/2015 15:05] root@ubuntu:/home/seed# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[04/24/2015 15:05] root@ubuntu:/home/seed# gcc -o stack -fno-stack-protector '/home/seed/Desktop/stack.c'
[04/24/2015 15:05] root@ubuntu:/home/seed# chmod 4755 stack
[04/24/2015 15:05] root@ubuntu:/home/seed# exit
exit
[04/24/2015 15:05] seed@ubuntu:~$ gcc -o exploit '/home/seed/Desktop/exploit.c'
[04/24/2015 15:05] seed@ubuntu:~$ ./exploit
[04/24/2015 15:05] seed@ubuntu:~$ ./stack
Segmentation fault (core dumped)
[04/24/2015 15:06] seed@ubuntu:~$
```

Now we run the attack in a loop as Address Randomization is on we will try to achieve our goal by doing this. The command we run after task 1 is as below:

```
$ sh -c "while [ 1 ]; do ./stack; done;"
```



```
Terminal
[04/24/2015 15:04] seed@ubuntu:~$ su root
Password:
[04/24/2015 15:05] root@ubuntu:/home/seed# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[04/24/2015 15:05] root@ubuntu:/home/seed# gcc -o stack -fno-stack-protector '/home/seed/Desktop/stack.c'
[04/24/2015 15:05] root@ubuntu:/home/seed# chmod 4755 stack
[04/24/2015 15:05] root@ubuntu:/home/seed# exit
exit
[04/24/2015 15:05] seed@ubuntu:~$ gcc -o exploit '/home/seed/Desktop/exploit.c'
[04/24/2015 15:05] seed@ubuntu:~$ ./exploit
[04/24/2015 15:05] seed@ubuntu:~$ ./stack
Segmentation fault (core dumped)
[04/24/2015 15:06] seed@ubuntu:~$ sh -c "while [ 1 ]; do ./stack; done;"
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
```

3. Task 3: Stack Guard

Turn off the address randomization first, otherwise we will not know which protection helps achieve the protection. And then run task1 again to obtain the result shown below.

Next the address randomization is disabled and stack.c is recompiled without -fno-stack-protector option. I could not get the root privilege because stack protector checks the boundary of the buffer and it restrict the malicious assembly language code to execute.

```

Terminal
[04/24/2015 15:32] seed@ubuntu:~$ clear

[04/24/2015 15:33] seed@ubuntu:~$ su root
Password:
[04/24/2015 15:33] root@ubuntu:/home/seed# /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[04/24/2015 15:33] root@ubuntu:/home/seed# gcc -o stack -z execstack '/home/seed/Desktop/stack.c'
[04/24/2015 15:35] root@ubuntu:/home/seed# chmod 4755 stack
[04/24/2015 15:35] root@ubuntu:/home/seed# exit
exit
[04/24/2015 15:35] seed@ubuntu:~$ gcc -o exploit '/home/seed/Desktop/exploit.c'
[04/24/2015 15:35] seed@ubuntu:~$ ./exploit
[04/24/2015 15:35] seed@ubuntu:~$ ./stack
*** stack smashing detected ***: ./stack terminated
===== Backtrace: =====
/lib/i386-linux-gnu/libc.so.6(__fortify_fail+0x45)[0xb7f240e5]
/lib/i386-linux-gnu/libc.so.6(+0x10409a)[0xb7f2409a]
./stack[0x8048513]
[0xbffff2b6]
===== Memory map: =====
08048000-08049000 r-xp 00000000 08:01 1575430 /home/seed/stack
08049000-0804a000 r-xp 00000000 08:01 1575430 /home/seed/stack

```

```

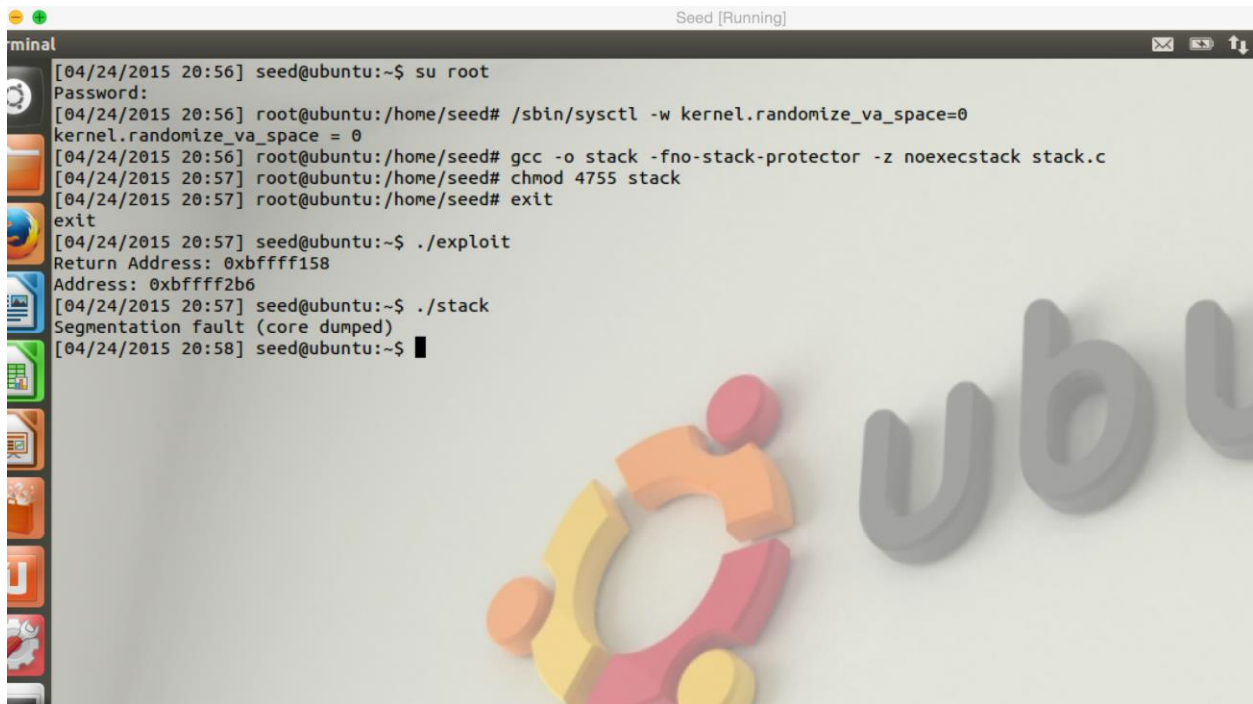
Terminal
0804a000-0804b000 rwxp 00001000 08:01 1575430 /home/seed/stack
0804b000-0806c000 rwxp 00000000 00:00 0 [heap]
b7def000-b7e0b000 r-xp 00000000 08:01 2360149 /lib/i386-linux-gnu/libgcc_s.so.1
b7e0b000-b7e0c000 r-xp 0001b000 08:01 2360149 /lib/i386-linux-gnu/libgcc_s.so.1
b7e0c000-b7e0d000 rwxp 0001c000 08:01 2360149 /lib/i386-linux-gnu/libgcc_s.so.1
b7e1f000-b7e20000 rwxp 00000000 00:00 0
b7e20000-b7fc3000 r-xp 00000000 08:01 2360304 /lib/i386-linux-gnu/libc-2.15.so
b7fc3000-b7fc5000 r-xp 001a3000 08:01 2360304 /lib/i386-linux-gnu/libc-2.15.so
b7fc5000-b7fc6000 rwxp 001a5000 08:01 2360304 /lib/i386-linux-gnu/libc-2.15.so
b7fc6000-b7fc9000 rwxp 00000000 00:00 0
b7fd9000-b7fdd000 rwxp 00000000 00:00 0
b7fdd000-b7fde000 r-xp 00000000 00:00 0 [vdso]
b7fde000-b7ffe000 r-xp 00000000 08:01 2364405 /lib/i386-linux-gnu/ld-2.15.so
b7ffe000-b7fff000 r-xp 0001f000 08:01 2364405 /lib/i386-linux-gnu/ld-2.15.so
b7fff000-b8000000 rwxp 00020000 08:01 2364405 /lib/i386-linux-gnu/ld-2.15.so
bffd000-c0000000 rwxp 00000000 00:00 0 [stack]
Aborted (core dumped)
[04/24/2015 15:35] seed@ubuntu:~$

```

Task 4: Non executable Stack

Turn off the address randomization first, otherwise we will not know which protection helps achieve the protection. And then run task1 again to obtain the result shown below.

Next, the address randomization is disabled and stack.c is recompiled without execstack option. I could not get the root privilege because execstack option makes it executable and checks the boundary of the buffer and restrict the malicious assembly language code to execute. Hence we disable it.(without execstack option)

A terminal window titled "Seed [Running]" showing a series of commands and their outputs. The user 'seed' switches to root using 'su root'. The root user disables address randomization with 'sysctl -w kernel.randomize_va_space=0'. Then, 'stack.c' is compiled with 'gcc -o stack -fno-stack-protector -z noexecstack stack.c' and permissions are set to 4755 with 'chmod 4755 stack'. The user exits root and runs './exploit', which shows a return address of 0xbffff158. Then, the user runs './stack', which results in a segmentation fault (core dumped). The terminal background features the Ubuntu logo.

```
[04/24/2015 20:56] seed@ubuntu:~$ su root
Password:
[04/24/2015 20:56] root@ubuntu:/home/seed# /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[04/24/2015 20:56] root@ubuntu:/home/seed# gcc -o stack -fno-stack-protector -z noexecstack stack.c
[04/24/2015 20:57] root@ubuntu:/home/seed# chmod 4755 stack
[04/24/2015 20:57] root@ubuntu:/home/seed# exit
exit
[04/24/2015 20:57] seed@ubuntu:~$ ./exploit
Return Address: 0xbffff158
Address: 0xbffff2b6
[04/24/2015 20:57] seed@ubuntu:~$ ./stack
Segmentation fault (core dumped)
[04/24/2015 20:58] seed@ubuntu:~$
```