Sergio Diaz 1155107144 IERG4130 - Lab #1

# Buffer Overflow Vulnerability Lab

## Task 1: Exploiting the Vulnerability

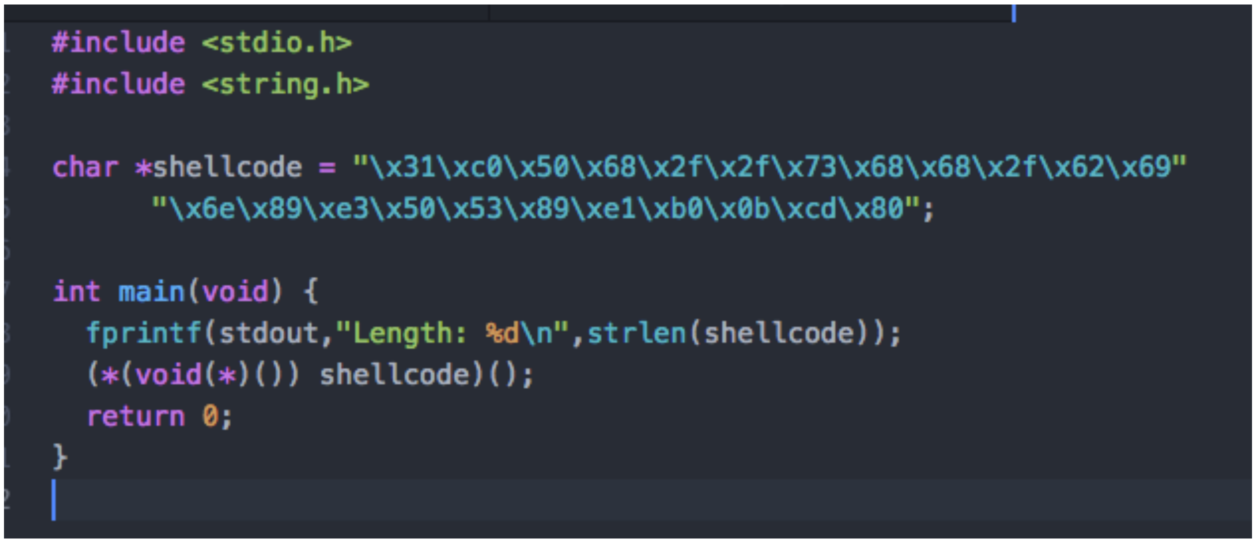
First of all we must turn off the address randomizer so we can actually do a guess of where the return address will be. Then we must compile our shell code program, which actually contains assembly instructions so the root shell can appear when we do our buffer overflow attack. Then we must compile the stack.c program as a root user (also we need to turn off the stack protector, otherwise we will not be able to execute our program since the OS will know that there is a buffer overflow threat) so when we do the buffer overflow attack we can be able to get a super user shell and not a normal user one. Finally we need to add some code to our exploit file so we can manage to change the return address and to get our malicious code to be executed. What we need to add are the addresses of our shell code into the buffer of size 517 so we can change the return address. Besides, we need to make sure that our code is not indexed at the beginning of our buffer, otherwise it would not be executed due to the structure of the linux stack (also because we turned off the address randomization).

## 2.5 Task 2: Protection in /bin/bash

Now, we let /bin/sh point back to /bin/bash, and run the same attack developed in the previous task.

**Can you get a shell?**

No, since we are no longer using zsh. We could get a shell if we use apposite shell_code for /bin/bash such as the following:



```c
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
      "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void) {
  fprintf(stdout,"Length: %d\n",strlen(shellcode));
  (*(void(*)()) shellcode)();
  return 0;
}
```

**Disclaimer: This code is not mine. It is from http://shell-storm.org/shellcode/files/shellcode-827.php

**Is the shell the root shell?**

No, it is not because we are no longer using zsh.

**What has happened?**

We are no longer using zsh

### 2.6 Task 3: Address Randomization

**Now, we turn on the Ubuntu's address randomization. We run the same attack developed in Task 1. Can you get a shell? If not, what is the problem? How does the address randomization make your attacks difficult? You should describe your observation and explanation in your lab report.**

We cannot get a shell, since now the memory randomization it's on. In other words, since we don't know the return address it's more difficult to complete the attack, however this doesn't mean we cannot do it. The buffer is only 517, eventually, if we run our code in a loop we will be manage to "guess" the correct address.

### 2.7 Task 4: Stack Guard

**So far, we disabled the "Stack Guard" protection mechanism in GCC when compiling the programs. In this task, you may consider repeating task 1 in the presence of Stack Guard. To do that, you should compile the program without the *-fno-stack-protector'* option. For this task, you will recompile the vulnerable program, stack.c, to use GCC's Stack Guard, execute task 1 again, and report your observations. You may report any error messages you observe.**

**In the GCC 4.3.3 and newer versions, Stack Guard is enabled by default. Therefore, you have to disable Stack Guard using the switch mentioned**

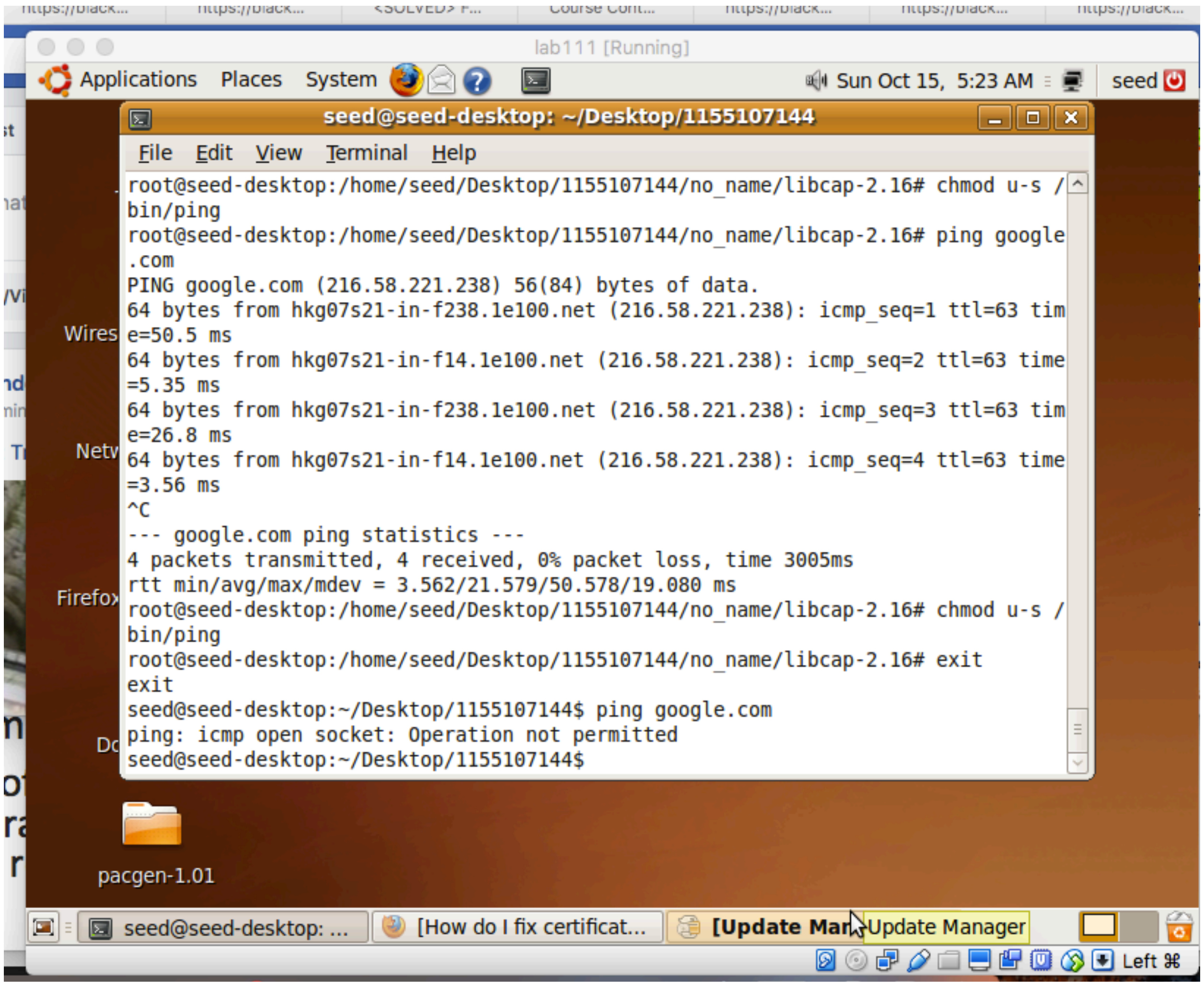
**before. In earlier versions, it was disabled by default. If you use a older GCC version, you may not have to disable Stack Guard.**

When we try to execute stack.c without using the -fno-stack-protector, the system does not allow us to run ./stack since it knows that there might be a possible threat if the latter program is executed. This is done because there's a stack protection that will make sure to abort the program if it detects a possible overflow.

# Linux Capability Exploration Lab

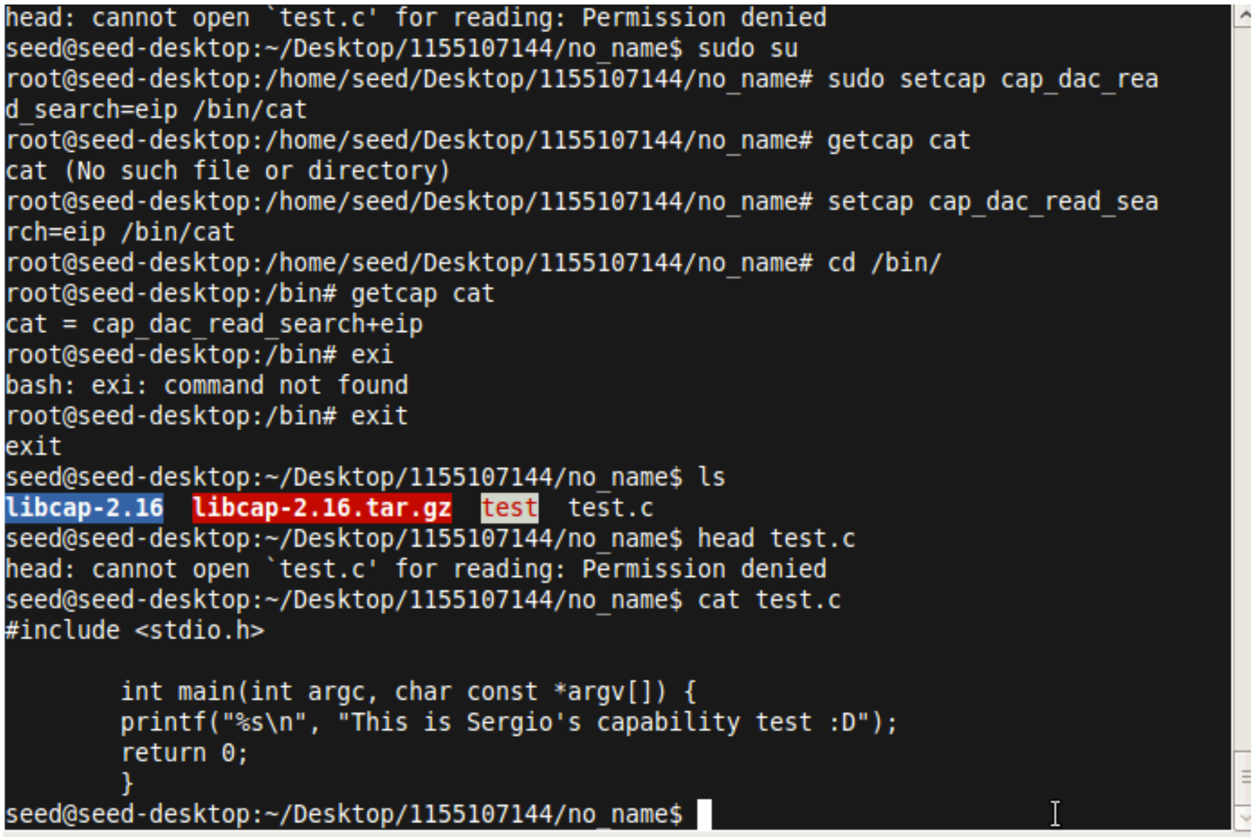
### 3.1 Task 1: Experiencing Capabilities

Ping to google.com works:

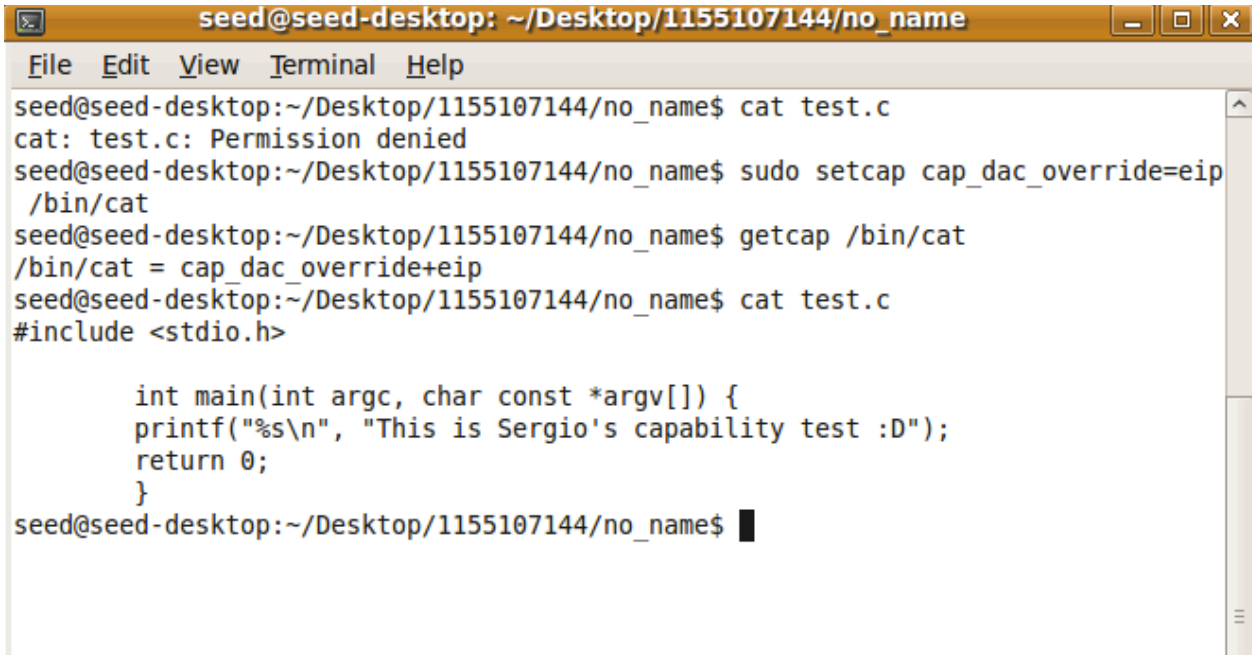


```
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# chmod u-s /
bin/ping
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# ping google
.com
PING google.com (216.58.221.238) 56(84) bytes of data.
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221.238): icmp_seq=1 ttl=63 tim
e=50.5 ms
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=2 ttl=63 time
=5.35 ms
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221.238): icmp_seq=3 ttl=63 tim
e=26.8 ms
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=4 ttl=63 time
=3.56 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 3.562/21.579/50.578/19.080 ms
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# chmod u-s /
bin/ping
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# exit
exit
seed@seed-desktop:~/Desktop/1155107144$ ping google.com
ping: icmp open socket: Operation not permitted
seed@seed-desktop:~/Desktop/1155107144$
```

Ping to google.com doesn't work because we removed the privilege to be able to open a raw socket (only root can do this).

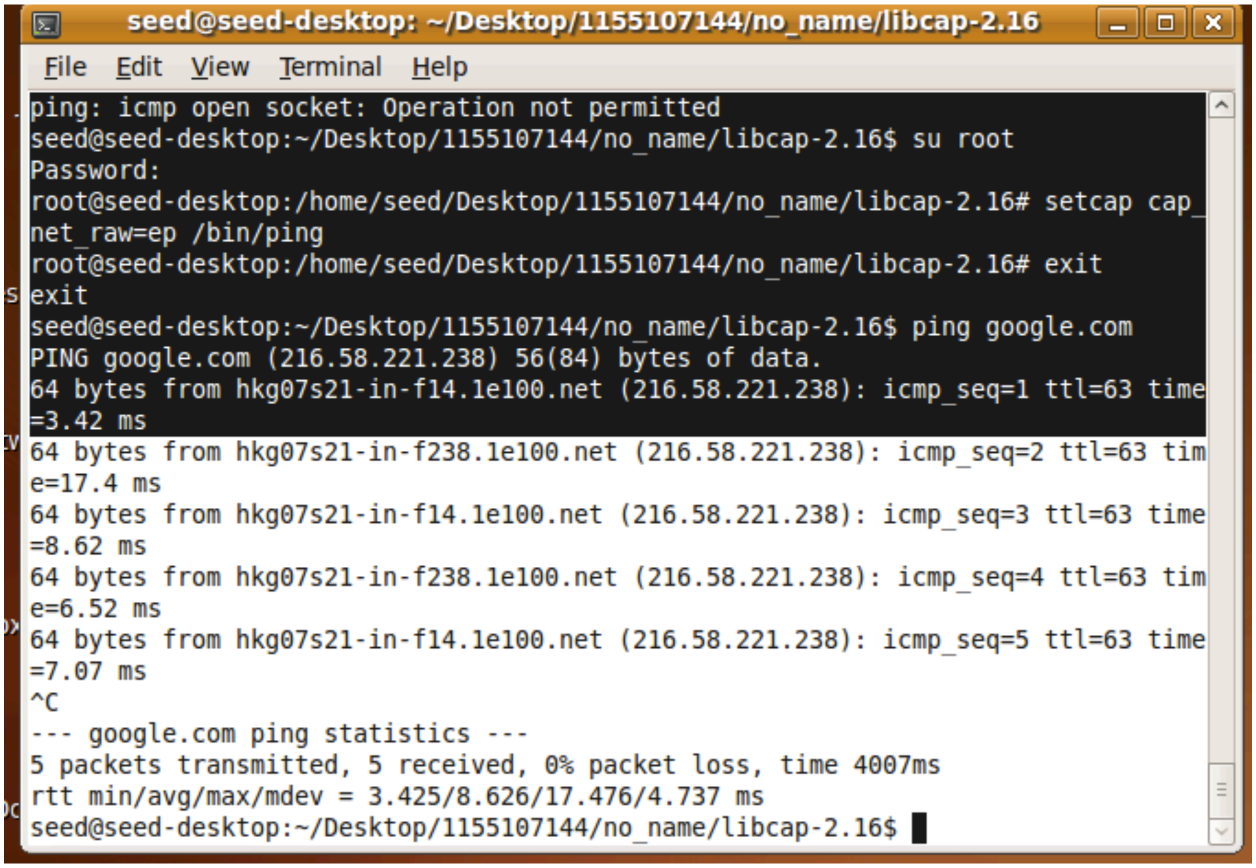


```
head: cannot open `test.c' for reading: Permission denied
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo su
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# sudo setcap cap_dac_rea
d_search=eip /bin/cat
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# getcap cat
cat (No such file or directory)
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# setcap cap_dac_read_sea
rch=eip /bin/cat
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# cd /bin/
root@seed-desktop:/bin# getcap cat
cat = cap_dac_read_search+eip
root@seed-desktop:/bin# exi
bash: exi: command not found
root@seed-desktop:/bin# exit
exit
seed@seed-desktop:~/Desktop/1155107144/no_name$ ls
libcap-2.16  libcap-2.16.tar.gz  test  test.c
seed@seed-desktop:~/Desktop/1155107144/no_name$ head test.c
head: cannot open `test.c' for reading: Permission denied
seed@seed-desktop:~/Desktop/1155107144/no_name$ cat test.c
#include <stdio.h>

        int main(int argc, char const *argv[]) {
        printf("%s\n", "This is Sergio's capability test :D");
        return 0;
        }
seed@seed-desktop:~/Desktop/1155107144/no_name$
```


seed@seed-desktop: ~/Desktop/1155107144/no_name
File Edit View Terminal Help

```
seed@seed-desktop:~/Desktop/1155107144/no_name$ cat test.c
cat: test.c: Permission denied
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo setcap cap_dac_override=eip
 /bin/cat
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap /bin/cat
/bin/cat = cap_dac_override+eip
seed@seed-desktop:~/Desktop/1155107144/no_name$ cat test.c
#include <stdio.h>

        int main(int argc, char const *argv[]) {
        printf("%s\n", "This is Sergio's capability test :D");
        return 0;
        }
seed@seed-desktop:~/Desktop/1155107144/no_name$
```

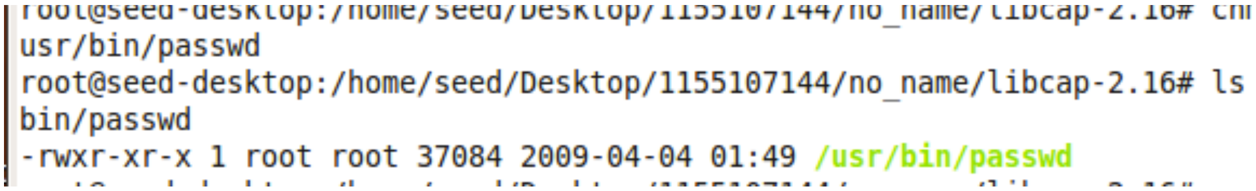
Ping works again because we assigned a cap_net_raw capability:



```
ping: icmp open socket: Operation not permitted
seed@seed-desktop:~/Desktop/1155107144/no_name/libcap-2.16$ su root
Password:
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# setcap cap_
net_raw=ep /bin/ping
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# exit
exit
seed@seed-desktop:~/Desktop/1155107144/no_name/libcap-2.16$ ping google.com
PING google.com (216.58.221.238) 56(84) bytes of data.
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=1 ttl=63 time
=3.42 ms
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221.238): icmp_seq=2 ttl=63 tim
e=17.4 ms
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=3 ttl=63 time
=8.62 ms
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221.238): icmp_seq=4 ttl=63 tim
e=6.52 ms
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=5 ttl=63 time
=7.07 ms
^C
--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 3.425/8.626/17.476/4.737 ms
seed@seed-desktop:~/Desktop/1155107144/no_name/libcap-2.16$
```

**Question 1: Please turn the following Set-UID programs into non-Set-UID programs, without affect- ing the behaviors of these programs.**

After performing question #1, we can see the passwd program has the s-bit off, which we can clearly see using the ls -l command:

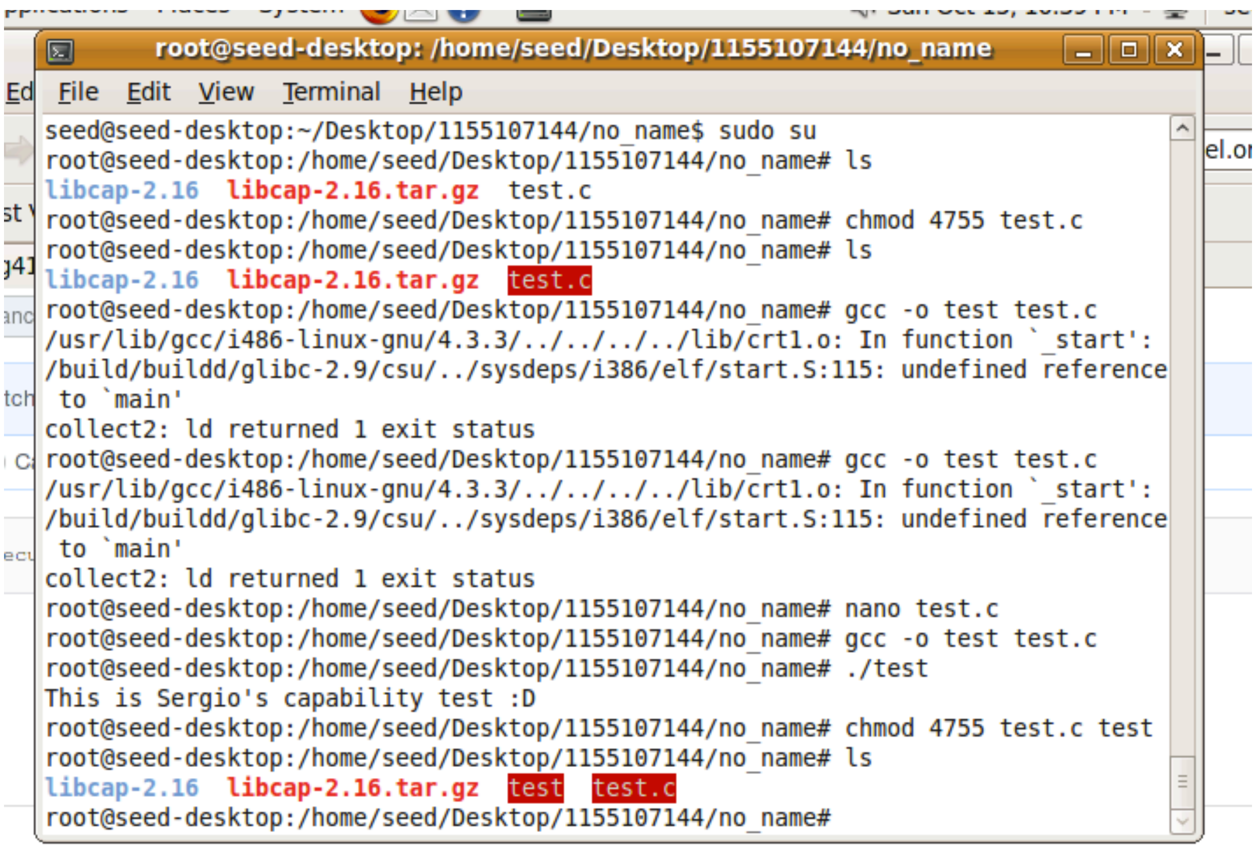


```
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# chm
usr/bin/passwd
root@seed-desktop:/home/seed/Desktop/1155107144/no_name/libcap-2.16# ls
bin/passwd
-rwxr-xr-x 1 root root 37084 2009-04-04 01:49 /usr/bin/passwd
```

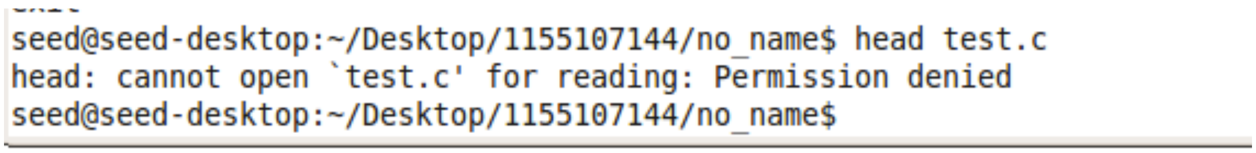
**Question 2: You have seen what we can do with the cap net raw capability. We would like you to get familiar with several other capabilities. For each of the following capabilities, do the following: (1) explain the purpose of this capability; (2) find a program to demonstrate the effect of these capabilities (you can run the application with and without the capability, and explain the difference in the results). You can also write your own applications if you prefer, as long as they can demonstrate the effect of the capability. Here is the list of capabilities that you need to work on (read include/linux/capability.h to learn about the capabilities).**

**cap dac read search:** Do not check file read permissions. In other words you can run a program with R permission.

Here we can see that I made a test.c program. Then I disabled all types of permissions. Hence when I wanted to use the cat command to read test.c contents I couldn't, however, after setting the cap_dac_read_search I was able to use the command cat to read the file's contents.



```
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo su
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# ls
libcap-2.16  libcap-2.16.tar.gz  test.c
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# chmod 4755 test.c
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# ls
libcap-2.16  libcap-2.16.tar.gz  test.c
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# gcc -o test test.c
/usr/lib/gcc/i486-linux-gnu/4.3.3/../../../../lib/crt1.o: In function `_start':
/build/buildd/glibc-2.9/csu/../sysdeps/i386/elf/start.S:115: undefined reference
 to `main'
collect2: ld returned 1 exit status
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# gcc -o test test.c
/usr/lib/gcc/i486-linux-gnu/4.3.3/../../../../lib/crt1.o: In function `_start':
/build/buildd/glibc-2.9/csu/../sysdeps/i386/elf/start.S:115: undefined reference
 to `main'
collect2: ld returned 1 exit status
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# nano test.c
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# gcc -o test test.c
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# ./test
This is Sergio's capability test :D
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# chmod 4755 test.c test
root@seed-desktop:/home/seed/Desktop/1155107144/no_name# ls
libcap-2.16  libcap-2.16.tar.gz  test  test.c
root@seed-desktop:/home/seed/Desktop/1155107144/no_name#
```



```
seed@seed-desktop:~/Desktop/1155107144/no_name$ head test.c
head: cannot open `test.c' for reading: Permission denied
seed@seed-desktop:~/Desktop/1155107144/no_name$
```

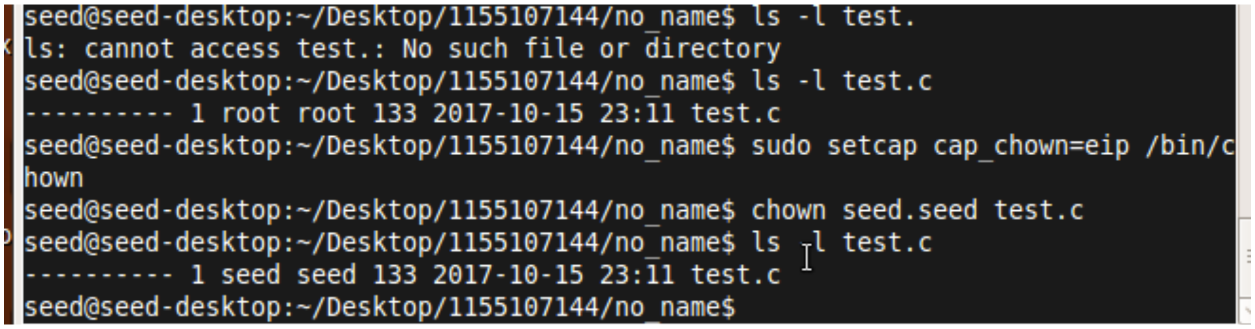
**cap dac override:** Do not check for read, write and execute flags.

Once again I tried to read the test.c file using cat and the permission was denied, however I set the cap_dac_override command and I was able to read the file again.

**cap fowner:** Do not check for permissions on commands that usually needs the UID of the process to match the one of the desired file.

**cap chown:** It is able to alternate or modify files' UIDs and GIDs.

With this command we managed to change the ownership of the program (root to seed)



```
seed@seed-desktop:~/Desktop/1155107144/no_name$ ls -l test.
ls: cannot access test.: No such file or directory
seed@seed-desktop:~/Desktop/1155107144/no_name$ ls -l test.c
---------- 1 root root 133 2017-10-15 23:11 test.c
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo setcap cap_chown=eip /bin/c
hown
seed@seed-desktop:~/Desktop/1155107144/no_name$ chown seed.seed test.c
seed@seed-desktop:~/Desktop/1155107144/no_name$ ls -l test.c
---------- 1 seed seed 133 2017-10-15 23:11 test.c
seed@seed-desktop:~/Desktop/1155107144/no_name$
```

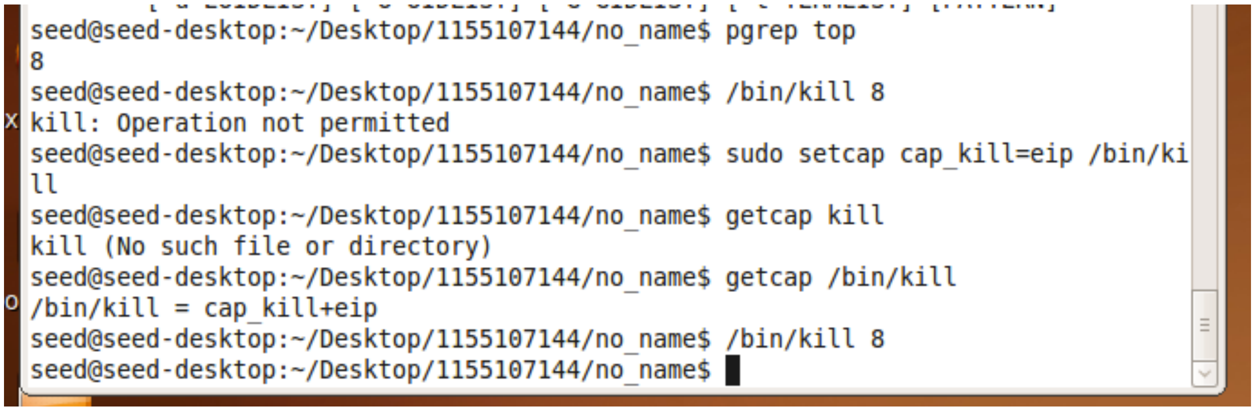
In the above image we can see that when we used the command "ls -l test.c" for the first time it belonged to root. And after we used cap_chown we were able to change the owner to seed.

**cap fsetid:** Do not modify set-UID and set-GUID flags when a file is modified.

**cap sys module:** Load and unload kernel modules, which are pieces of code that can be mounted and unmounted from the kernel on demand.

**cap kill:** When you want to send a signal, the permission flags aren't checked.

In this picture we want to kill process number 8, however we are not permitted to this. But after we set the capability to do it we successfully managed to do it.

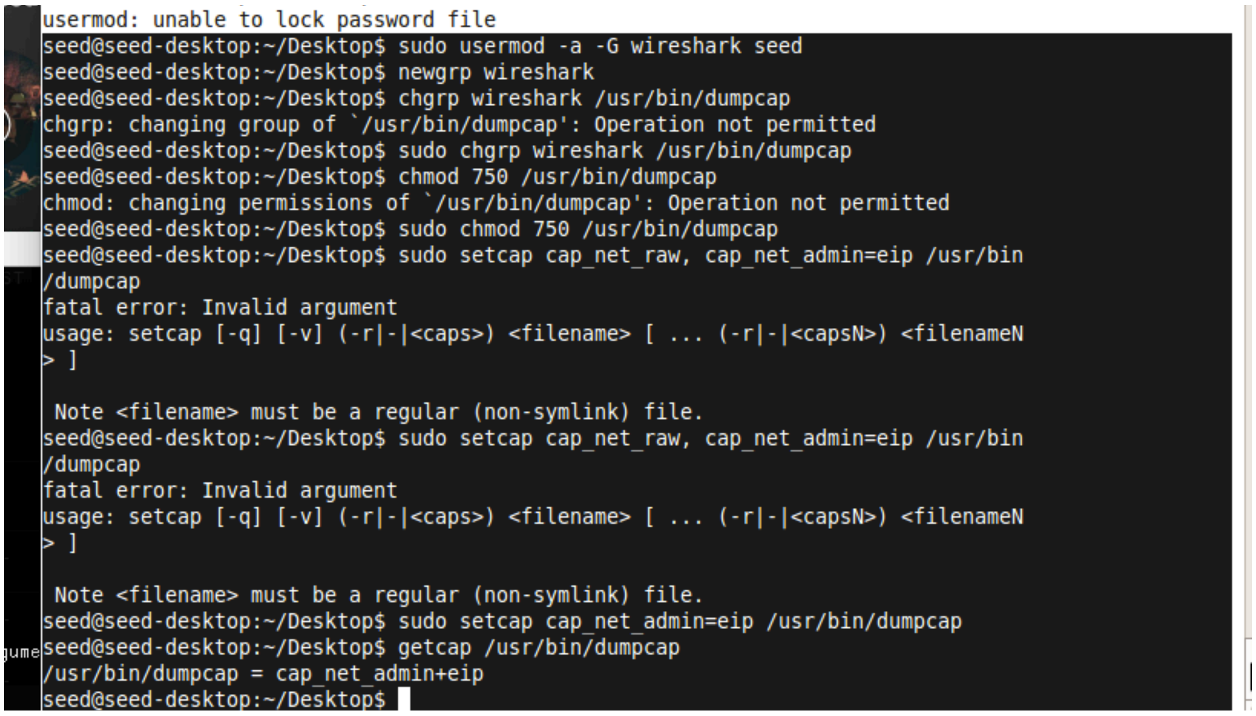


```
seed@seed-desktop:~/Desktop/1155107144/no_name$ pgrep top
8
seed@seed-desktop:~/Desktop/1155107144/no_name$ /bin/kill 8
kill: Operation not permitted
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo setcap cap_kill=eip /bin/ki
ll
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap kill
kill (No such file or directory)
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap /bin/kill
/bin/kill = cap_kill+eip
seed@seed-desktop:~/Desktop/1155107144/no_name$ /bin/kill 8
seed@seed-desktop:~/Desktop/1155107144/no_name$
```

**cap net admin:** It can perform several operations such as:

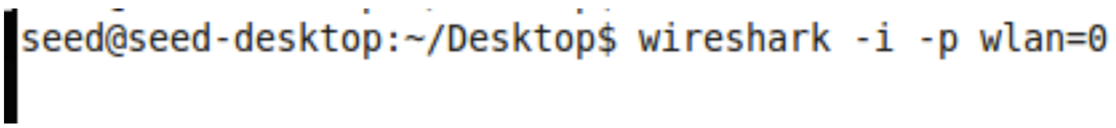
- Being able to enable promiscuous mode
- Make the configurations of an interface
- Administrate the firewall
- Add/edit/delete/modify routing table
- Set the types of service

Since cap_net_admin can be used for many purposes I decided to focus on being able to turn on the promiscuous mode in Wireshark, so the user can sniff all traffic.



```
usermod: unable to lock password file
seed@seed-desktop:~/Desktop$ sudo usermod -a -G wireshark seed
seed@seed-desktop:~/Desktop$ newgrp wireshark
seed@seed-desktop:~/Desktop$ chgrp wireshark /usr/bin/dumpcap
chgrp: changing group of `/usr/bin/dumpcap': Operation not permitted
seed@seed-desktop:~/Desktop$ sudo chgrp wireshark /usr/bin/dumpcap
seed@seed-desktop:~/Desktop$ chmod 750 /usr/bin/dumpcap
chmod: changing permissions of `/usr/bin/dumpcap': Operation not permitted
seed@seed-desktop:~/Desktop$ sudo chmod 750 /usr/bin/dumpcap
seed@seed-desktop:~/Desktop$ sudo setcap cap_net_raw, cap_net_admin=eip /usr/bin
/dumpcap
fatal error: Invalid argument
usage: setcap [-q] [-v] (-r|-|<caps>) <filename> [ ... (-r|-|<capsN>) <filenameN
> ]

 Note <filename> must be a regular (non-symlink) file.
seed@seed-desktop:~/Desktop$ sudo setcap cap_net_raw, cap_net_admin=eip /usr/bin
/dumpcap
fatal error: Invalid argument
usage: setcap [-q] [-v] (-r|-|<caps>) <filename> [ ... (-r|-|<capsN>) <filenameN
> ]

 Note <filename> must be a regular (non-symlink) file.
seed@seed-desktop:~/Desktop$ sudo setcap cap_net_admin=eip /usr/bin/dumpcap
seed@seed-desktop:~/Desktop$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_admin+eip
seed@seed-desktop:~/Desktop$
```

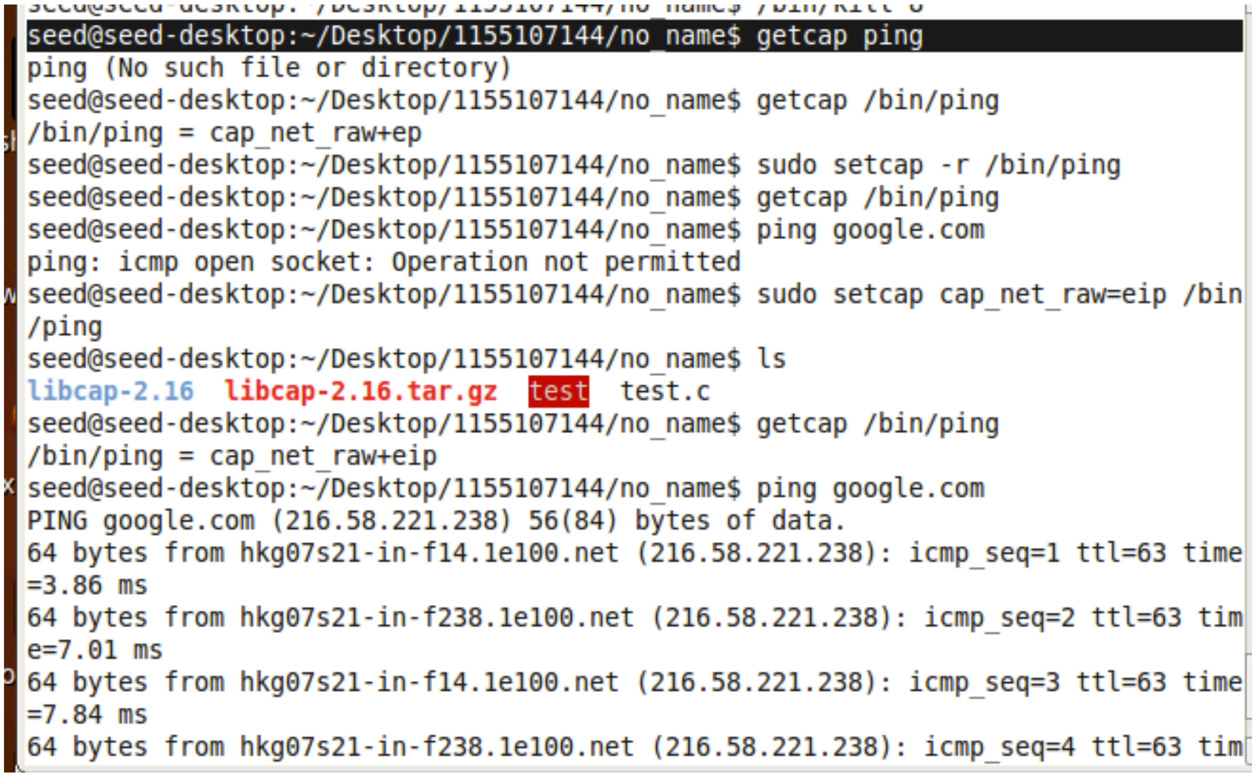
In the aforementioned picture we managed to do our goal, so now the current user is able to capture packages in promiscuous mode :D.



```
seed@seed-desktop:~/Desktop$ wireshark -i -p wlan=0
```

**cap net raw:**

- Gives the capability of socket binding to any address only and only if is intended for transparent proxying



```
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap ping
ping (No such file or directory)
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo setcap -r /bin/ping
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap /bin/ping
seed@seed-desktop:~/Desktop/1155107144/no_name$ ping google.com
ping: icmp open socket: Operation not permitted
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo setcap cap_net_raw=eip /bin
/ping
seed@seed-desktop:~/Desktop/1155107144/no_name$ ls
libcap-2.16  libcap-2.16.tar.gz  test  test.c
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap /bin/ping
/bin/ping = cap_net_raw+eip
seed@seed-desktop:~/Desktop/1155107144/no_name$ ping google.com
PING google.com (216.58.221.238) 56(84) bytes of data.
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=1 ttl=63 time
=3.86 ms
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221.238): icmp_seq=2 ttl=63 tim
e=7.01 ms
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=3 ttl=63 time
=7.84 ms
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221.238): icmp_seq=4 ttl=63 tim
```

Here we try to ping the famous site called google.com, however we receive an error about the opening of a socket. However, after we successfully put the capability into the ping command we successfully manage to ping google.com

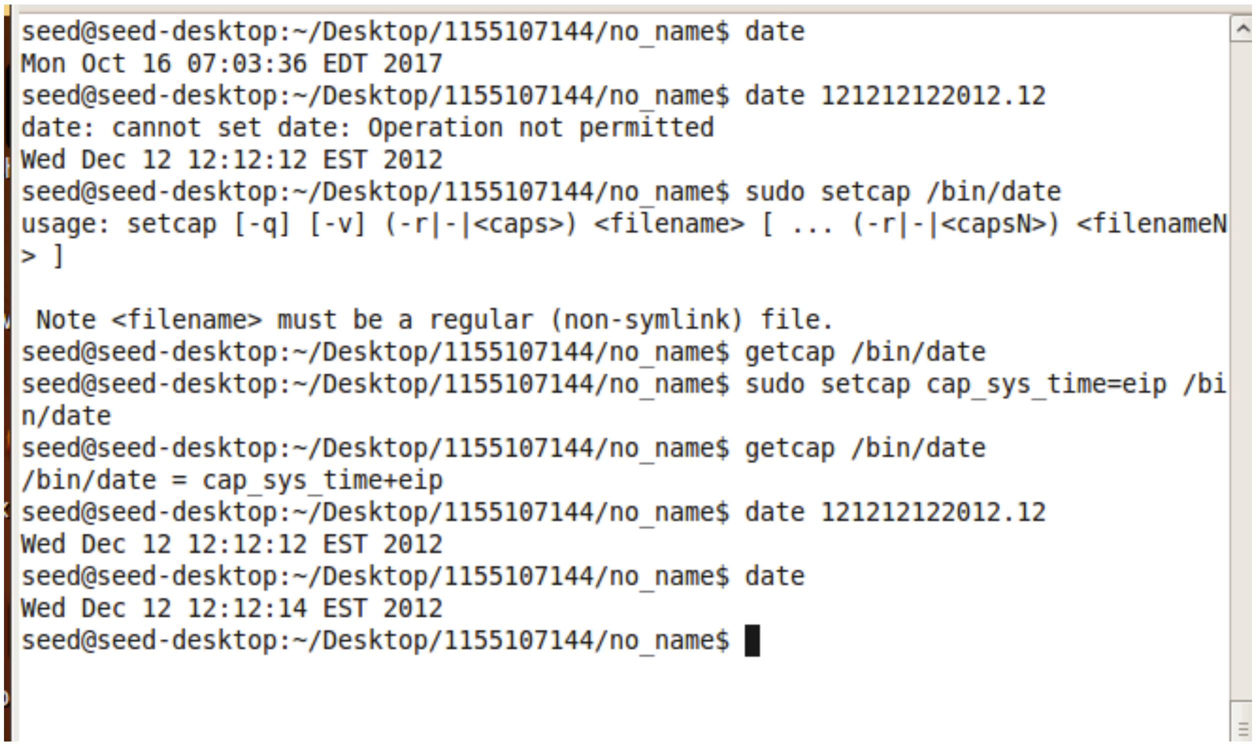
**cap sys nice:**

- This command enables to change the nice value
- Set Input and output scheduling for any desired process

**cap sys time:**

- Is used to set the machine's system clock.

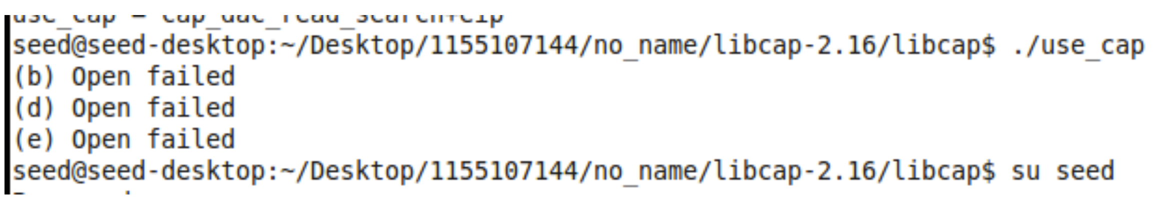
Here we try to change the system clock however we receive a warning saying that we aren't allowed to do that. However, after we successfully put the capability into the date command we successfully manage to complete change the date to Wed Dec 12 12:12:14 EST 2012



```
seed@seed-desktop:~/Desktop/1155107144/no_name$ date
Mon Oct 16 07:03:36 EDT 2017
seed@seed-desktop:~/Desktop/1155107144/no_name$ date 121212122012.12
date: cannot set date: Operation not permitted
Wed Dec 12 12:12:12 EST 2012
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo setcap /bin/date
usage: setcap [-q] [-v] (-r|-|<caps>) <filename> [ ... (-r|-|<capsN>) <filenameN
> ]

 Note <filename> must be a regular (non-symlink) file.
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap /bin/date
seed@seed-desktop:~/Desktop/1155107144/no_name$ sudo setcap cap_sys_time=eip /bi
n/date
seed@seed-desktop:~/Desktop/1155107144/no_name$ getcap /bin/date
/bin/date = cap_sys_time+eip
seed@seed-desktop:~/Desktop/1155107144/no_name$ date 121212122012.12
Wed Dec 12 12:12:12 EST 2012
seed@seed-desktop:~/Desktop/1155107144/no_name$ date
Wed Dec 12 12:12:14 EST 2012
seed@seed-desktop:~/Desktop/1155107144/no_name$ 
```

**3.2 Task 2: Adjusting Privileges**

**Question 3: Compile the following program, and assign the cap dac read search capability to the executable. Login as a normal user and run the program. Describe and explain your observations:**



```
use_cap = cap_dac_read_search+eip
seed@seed-desktop:~/Desktop/1155107144/no_name/libcap-2.16/libcap$ ./use_cap
(b) Open failed
(d) Open failed
(e) Open failed
seed@seed-desktop:~/Desktop/1155107144/no_name/libcap-2.16/libcap$ su seed
```

After compiling the latter file we can clearly see the following:

- A: Success.
- B: Open failed.
- C: Since C is enabled, it successfully opens.

- D: Open failed.
- E:  Open failed.

**Question 4:**

**If we want to dynamically adjust the amount of privileges in ACL-based access control, what should we do?**

We need to log in as a super user to be able to change a specific permission for a file. What we need to do is use the command chmod following the numbers (base 10 representation of on and off bits) to indicate whether a group or user have a specific permission (read, write, execute)

**Compared to capabilities, which access control is more convenient to do so?**

It's always more convenient to not log as a super user and also to care about whether a user/group have an specific permission (read, write, execute). So using capabilities is more convenient, however it can be more dangerous since users can interact with a file even though they don't posses specific permissions.

**Question 5: After a program (running as normal user) disables a capability A, it is compromised by a buffer-overflow attack. The attacker successfully injectes his malicious code into this program's stack space and starts to run it. Can this attacker use the capability A? What if the process deleted the capability, can the attacker uses the capability?**

It depends, If the capability was deleted the attacker cannot take advantage of the latter. However, if the capability is only disabled the attacker can enable it, hence he can use his malicious code.

**Question 6: The same as the previous question, except replacing the buffer-overflow attack with the race condition attack. Namely, if the attacker exploites the race condition in this program, can he use the capability A if the capability is disabled? What if the capability is deleted?**

If an attacker manages to do a race condition attack he would be able to use the aforementioned capability, no matter if it is enabled or disabled, since can manage to run the malicious code first.

**Question : Buffer Overflow**

**(a) Suppose the following C function is running on web server of insecure.com. where username[] is the content provided by user from the following HTML fragment:**

**The developers of insecure.com have implemented their server-side program in C very carefully so that whenever insecure() is called by the code they implemented, the argument length is equal to the number of elements in username[] (i.e., the precondition of insecure() always holds).**

**Explain how an attacker can bypass the length restriction to launch buffer overflow attack.**

First of all the max length of the user input is 20 and the size of the buffer is 18. So Actually there is a possibility of having an attack.

**(b) To fix the problem in (a), the developers decide to add the code below for server side validation. Does any form of buffer overflow attack still exist? Justify your answer.**

No, since a buffer overflow attack can only occur when the size of the input is differ than the buffer, so as long as the input validation in the code, there are no possibilities of a buffer overflow attack.

**(c) Name and briefly describe two ways to mitigate buffer overflow attack.**

Although the buffer overflow attack can be very lethal there are several solutions to at least prevent/mitigate the possibilities of being a victim of this attack:

First of all the software engineer must be aware of this threat and at least need to have conscience about the possibility of exploitation. For example: You can try to use a language that cannot interact with the memory in the way C and C++ do, such as Java or Python. However this is not always possible, since you might be in the situation in which using a low-level language (C, C++) is not an option, hence we must use another solutions such as validating user input (quality assurance and testing) and using safe methods provided by the language that actually can check for a possible buffer overflow.