Lab12. Security and Networking Concepts

INSTRUCTOR :MURAD NJOUM

Objectives

After completing this lab, the student should be able to:

- Understand through example the importance and usage of **set user id (suid)** and **set group id (permissions)** in Linux.
- Set and modify **suid and sgid values** on Linux files.
- Identify and learn some **Linux networking tool basics**.

Suid and Sgid

- Linux systems are **very secure** and have **multiple levels** of security that takes volumes to discuss .

- We have already talked about a part of one of those security levels **which is file security** where we explained the permissions (mode) and how they are used to control **who can access and use files and directories .**

- The permissions we talked about were the **read (r), write (w), and execute (x)**

- In this lab we will present a **less obvious**, but **very powerful permission** called the setuid (set user id) and setgid (set group id) permission usually referenced with **an (s) permission**

Introduction

Normally, on a Unix-like operating system, the ownership of files and directories is based on the default `uid` (user-id) and `gid` (group-id) of the user who created them.

The same thing happens when a process is launched:
it runs with the ***effective user-id and group-id*** of the user who started it, and with the corresponding privileges. This behavior can be modified by using special permissions.

The setuid bit

When the `setuid` bit is used, the behavior described above it's modified so that when an executable is launched, it does not run with the privileges of the user who launched it, but with that of the file owner instead.

So, for example, if an executable has the `setuid` bit set on it, and it's **owned by root**, when launched by a normal user, it will **run with root privileges**. It should be clear why this represents a potential security risk, if not used correctly.

An example of an executable with the setuid permission set is `passwd`, the utility we can use to change our login password. We can verify that by using the `ls` command:

Set User Id (suid) Permission

passwd
command which we use to change our passwords.

Run the command
which passwd

/bin/passwd

Run the command
ls -al /bin/passwd

What are the permissions ?

```
-rwsr-xr-x. 1 root root 27832 Jun 10  2014 /bin/passwd
```

ls -al /etc/shadow

What are the permissions ?

The (s) on the user part of the mode is the **suid**. This (s) is very important and without it a user **will not be able to** change his/her password.

- The process resulting from running the passwd command has an **effective uid as root** (owner of the file passwd) which is why this command is able to open and modify files (e.g. **/etc/passwd and /etc/shadow files**) which the **user running the command** is not allowed to. This gives great flexibility by giving regular users the ability to access files through running commands which they cannot access normally

- This gives **great flexibility** by giving regular users the ability to **access files through running commands** which they cannot access normally.

How to identify the `setuid` bit?

- As you surely have noticed looking at the output of the command above, the `setuid` bit is represented by an `s` in place of the `x` of the executable bit.

- The `s` implies that the executable bit is set, otherwise you would see a capital `S`. This happens when the `setuid` or `setgid` bits are set, but the executable bit is not, showing the user an inconsistency:

- the `setuid` and `setgid` bits have no effect if the executable bit is not set. The setuid bit has no effect on directories.

The setgid bit

Unlike the `setuid` bit, the `setgid` **bit has effect on both files and directories**. In the first case, the file which has the `setgid` bit set, when executed, instead of running with the privileges of the group of the user who started it, runs with those of the group which owns the file: **in other words, the group ID of the process will be the same of that of the file.**

When used on a directory, instead, the `setgid` bit alters the standard behavior so that the group of the files created inside sgid directory, will not be that of the user who created them, but that of the parent directory itself. This is often used to ease the sharing of files (**files will be modifiable by all the users that are part of sgid group**). Just like the `setuid`, the `setgid` bit can easily be spotted (in this case on a test directory):

This time the `s` is present in place of the executable bit on the group sector.

```
ls -ld test drwxrwsr-x. 2 egdoc egdoc 4096 Nov 1 17:25 test
```

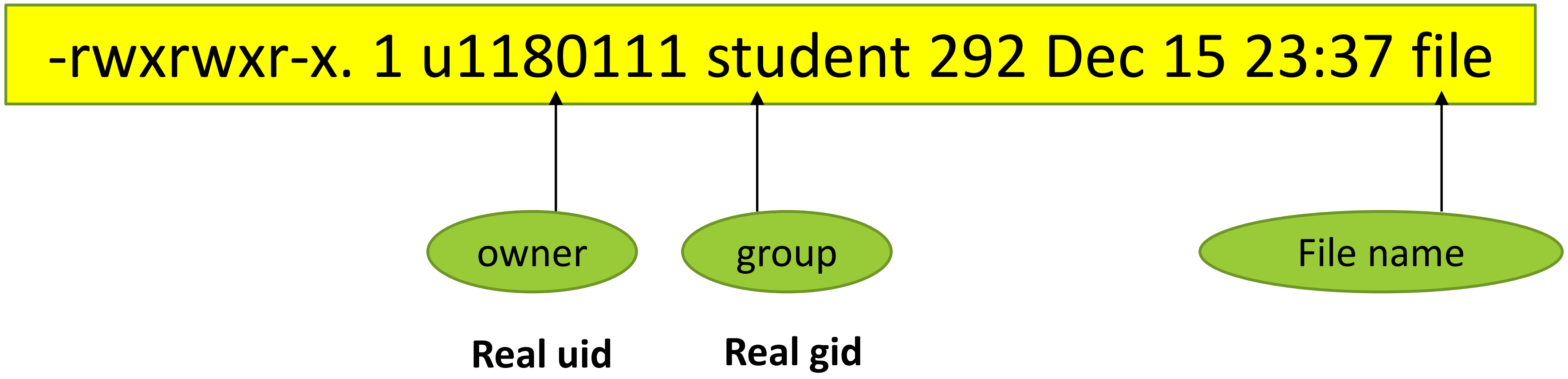
When a command such as passwd is executed, a process is created as explained in lab 7.That process has many properties.

Four of those are:

1.real uid (real user id)
2.real gid (real group id)
3.effective uid (effective user id)
4.effective gid (effective group id)

Cont..

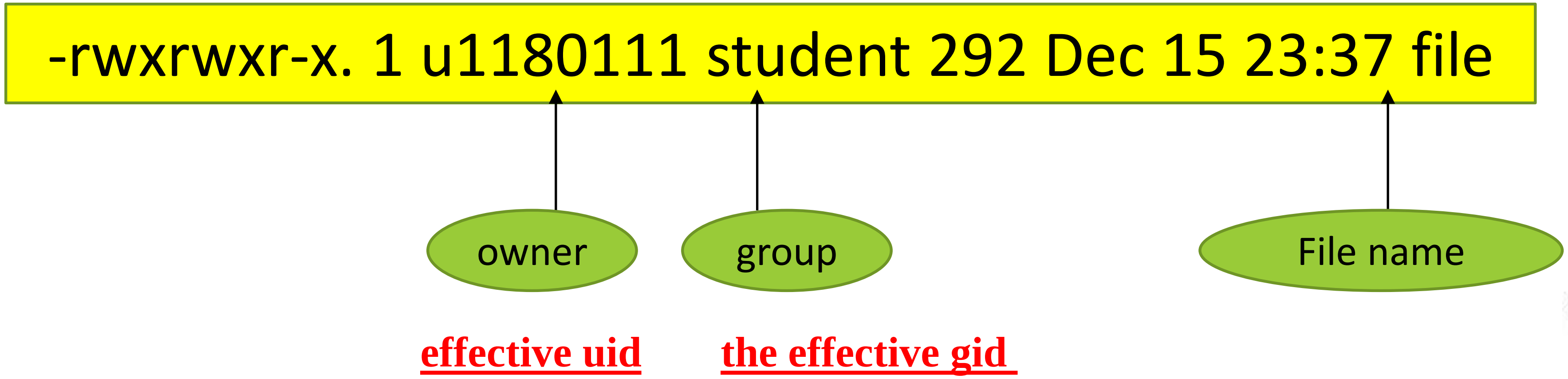
- The **real (uid and gid)** are the same as the **username and group** of the user executing that command



```
-rwxrwxr-x. 1 u1180111 student 292 Dec 15 23:37 file
```

Cont..

- The **effective uid** is the same as the **real uid ,** and **the effective gid is** the same as **the real gid** except when there is an (s) permission



```
-rwxrwxr-x. 1 u1180111 student 292 Dec 15 23:37 file
```

owner

group

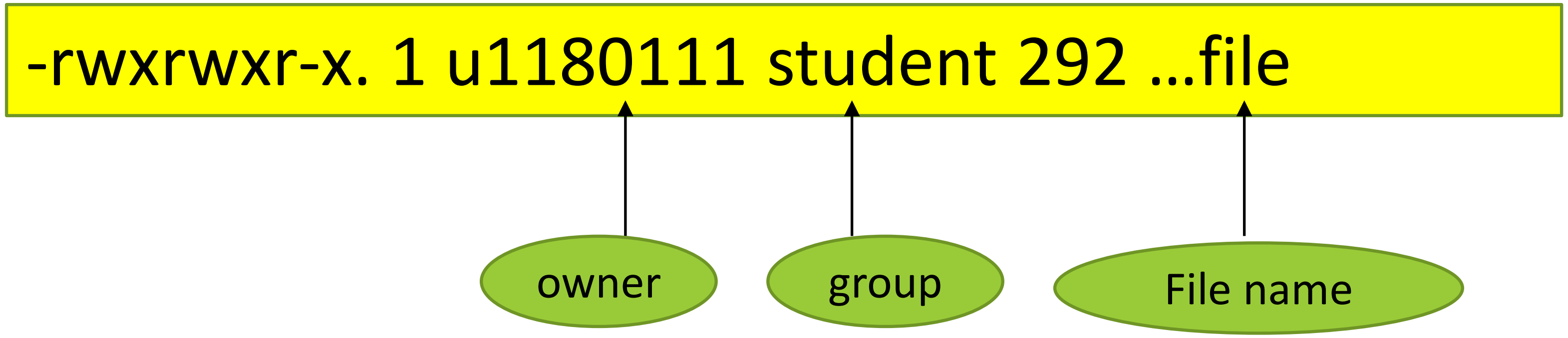
File name

effective uid

the effective gid

Cont..

- In this case (**S permission**) the **effective uid** will be **same** as the **owner of the file** and the **effective gid** will be **same as the group name on the file**.



-rwxrwxr-x. 1 u1180111 student 292 …file

owner

group

File name

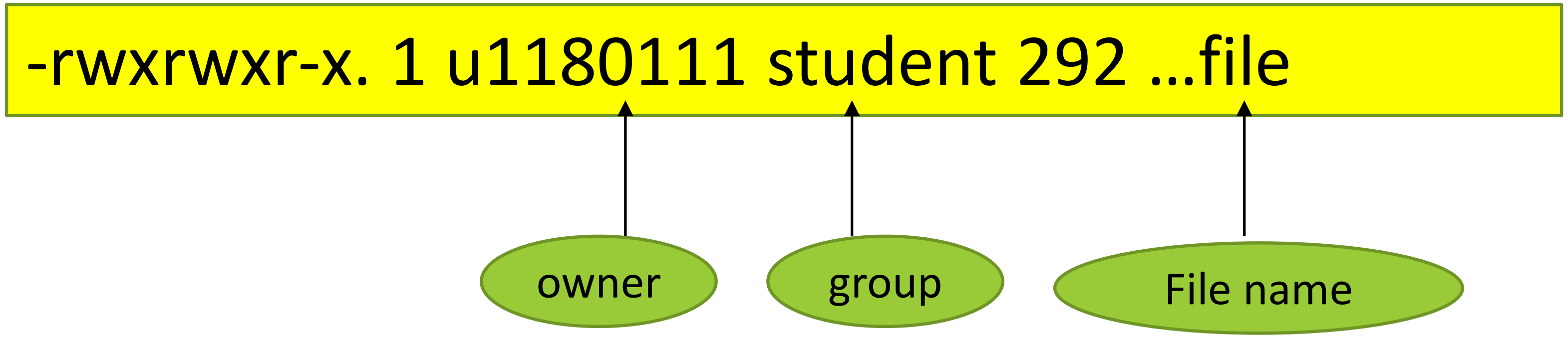
- Now, you are try to run command *runscript* as following properties

-rwsrw-r-x. 1 ahmad root 50 ……. runningscript

real uid	**u1180111**
real gid	**student**
effective uid	**ahmad**
effective gid	**student**

Cont..

- In this case (**S permission**) the **effective uid** will be **same** as the **owner of the file** and the **effective gid** will be **same as the group name on the file**.



-rwxrwxr-x. 1 u1180111 student 292 …file

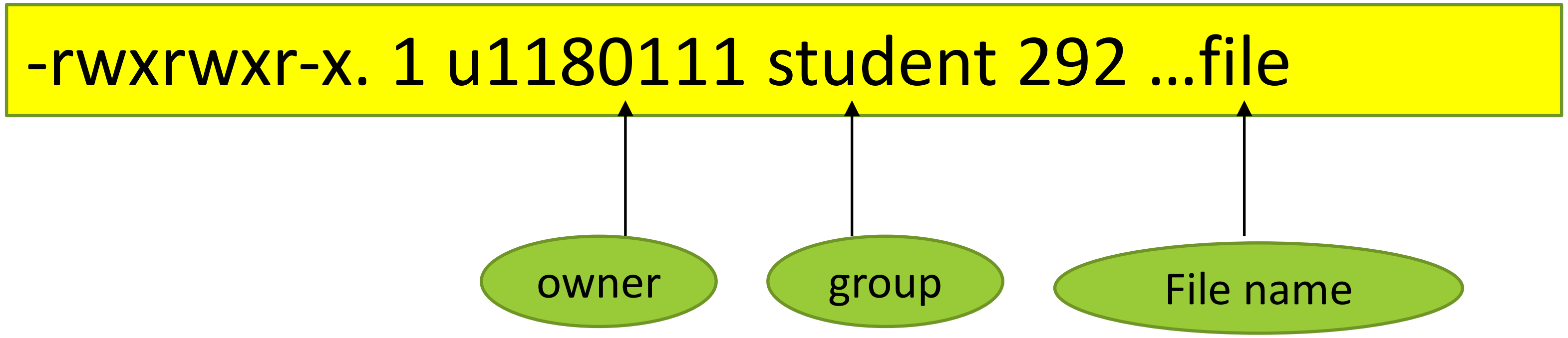
- Now, you are try to run command *runscript* as following properties

-rwxrwsr-x. 1 ahmad root 50 ……. runningscript

real uid	**u1180111**
real gid	**student**
effective uid	**u1180111**
effective gid	root

Cont..

- In this case (**S permission**) the **effective uid** will be **same** as the **owner of the file** and the **effective gid** will be **same as the group name on the file**.



```
-rwxrwxr-x. 1 u1180111 student 292 ...file
```

- Now, you are try to run command *runscript* as following properties

```
-rwsrwsr-x. 1 ahmad root  50 ....... runningscript
```

real uid	**u1180111**
real gid	**student**
effective uid	**ahmad**
effective gid	root

```
-rwsr-x--x. 1 root root 27832 Jun 10  2014 /bin/passwd
```

4751

A user named **‘Ahmad**' attempts to execute the file. The executable permission for all users is set (the '1') so ‘Ahmad' can execute the file. The file **owner** is 'root' and the SUID permission is set (the '4') - so the file is executed as 'root'.

The reason an executable would be run as 'root' is so that it can modify specific files that the user would not normally be allowed to, **without giving the user full root access.**

Set Group id (sgid) Permission Let us now see how the same idea is applied to the sgid.

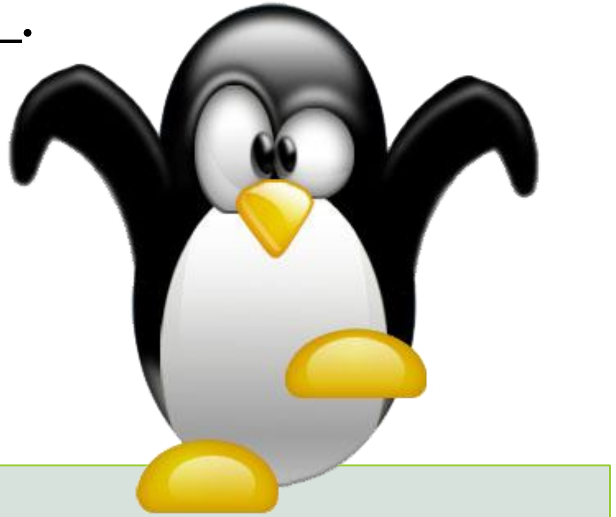
run the command:

which write

what are the permissions on the write command? ______________________________.

```
-rwxr-sr-x. 1 root tty 19624 Oct 31 00:48 /bin/write
```

What is the name of the group name on the write command ? ____________________.



Using the ***who*** command find the pts file for your neighbor.

ls -al /dev/pts/5

what is the group on that file and what permission does the group have?

```
crw--w----. 1 u1160661 tty 136, 2 Dec  4 22:50 /dev/pts/2
```

echo hello > /dev/pts/5

What happened? Why?

```
-rwxr-xr-x. 1 root root 33128 Oct 30 21:16 /bin/echo
```

```
[u1172082@localhost ~]$ who
comp311ta :0          2018-12-04 19:06 (:0)
comp311ta pts/0       2018-12-04 19:07 (:0)
comp311ta pts/1       2018-12-04 21:31 (Dell)
u1160661 pts/2        2018-12-04 22:50 (Dell)
u1172082 pts/3        2018-12-04 22:54 (Dell)
[u1172082@localhost ~]$ echo hello > /dev/pts/2
-bash: /dev/pts/2: Permission denied
```

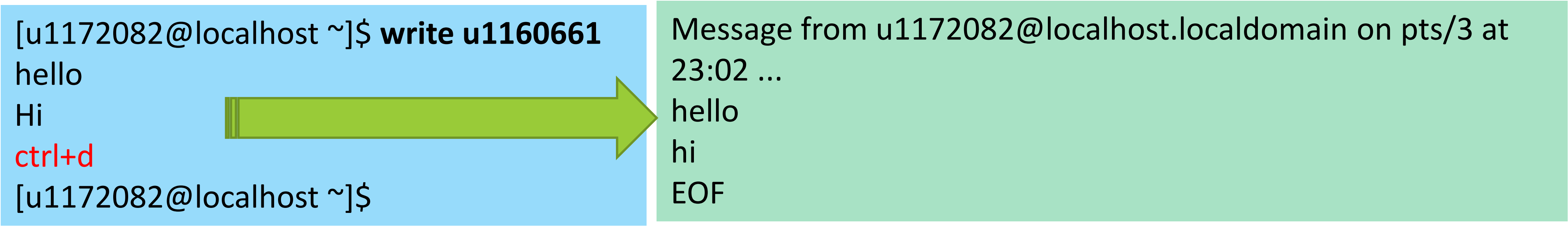
Now using the ***write*** command write the same message to your neighbor’s terminal as follows:

write u1112233

hello

ctrl-d

What happened?____



```
[u1172082@localhost ~]$ write u1160661
hello
Hi
ctrl+d
[u1172082@localhost ~]$
```

```
Message from u1172082@localhost.localdomain on pts/3 at
23:02 ...
hello
hi
EOF
```

In both cases, it was you (same user with same permissions) that was trying to write a message to the other user’s terminal. **Why did it not work when you tried to do it directly while it worked using the write command? (hint the (s) permission on the write command*)***

Adding (s) Permission

```
[u1172082@localhost ~]$ touch newfile
[u1172082@localhost ~]$ chmod 2777 newfile
[u1172082@localhost ~]$ ls -al newfile
-rwxrwsrwx. 1 u1172082 students 0 Dec  4 23:07 newfile
```

To add the s permission to your files, use the **chmod** command with four digits instead of three as before, for example:

create a file called newfile (touch newfile).

-rwxrwxr-x

chmod 2777 newfile

What permissions are now on file newfile? -rwxrwsrwx

chmod 4777 newfile

What permissions are now on file newfile? -rwsrwxrwx

chmod 6777 newfile

What permissions are now on file newfile? -rwsrwsrwx

As you can see adding an even digit (2 or 4 or 6) will put (s) on group, user, or both respectively

What command would you use to set the permissions on newfile to:

1. r-s -wx rwx

chmod 4537 myfile

2. r-x rws r--

chmod 2574 myfile

3. rwS rws r--

chmod 6674 myfile

How do you get a capital s (S) and a small s (s)?

The s is lowercase when the executable bit is set, and uppercase if not.

Networking:

As users we are not allowed to modify network setups, but we can view some information on how networks are configured on Linux.

Run the command:

ifconfig

What is the ip address (inet) of your machine?

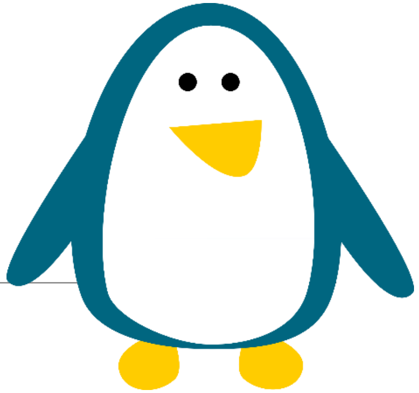
____

What is the MAC (ether) address of your machine?

____.

What is the netmask used by your machine?

____.



Run the command:

netstat -n | grep 23 (ie. Ip address is 192.168.1.12:23)

This will give information about the **telnet (port number is 23)** connections made to/from the system.List the quad (Socket Connection) for your telnet connection:

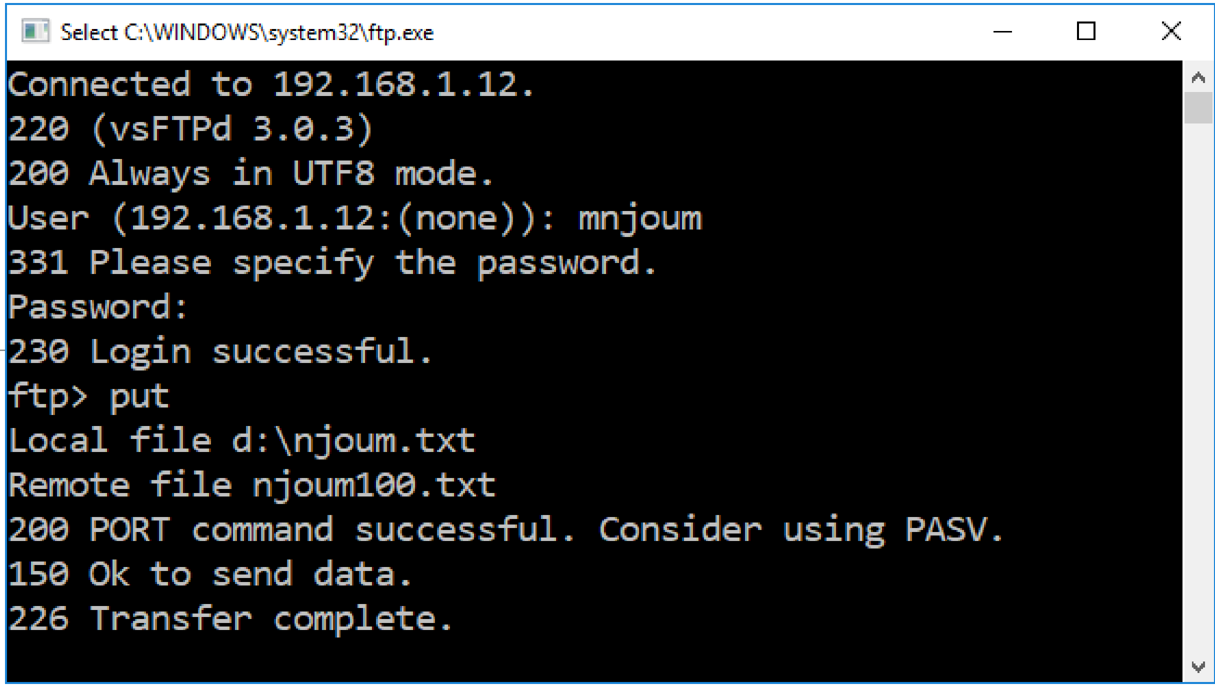
____.

```
 netstat -n|grep 23
tcp6       0       0 192.168.1.41:23         192.168.1.11:56818      ESTABLISHED
tcp6       0       0 192.168.1.41:23         192.168.1.11:56647      ESTABLISHED
tcp6       0       2 192.168.1.41:23         192.168.1.11:53410      ESTABLISHED
```

TCP/UDP (Some Is support by, but others not)

FTP:20/21, SMTP:25, POP:110, HTTPS (SSL/TLS):443

Put (upload: from windows to linux)
File1
File2
Go to your home user directory, file is their



```
Connected to 192.168.1.12.
220 (vsFTPd 3.0.3)
200 Always in UTF8 mode.
User (192.168.1.12:(none)): mnjoum
331 Please specify the password.
Password:
230 Login successful.
ftp> put
Local file d:\njoum.txt
Remote file njoum100.txt
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
```

Or (Linux ot Linux OS)

ftp –p (passive mode) or pftp

Use passive mode for data transfers. Allows use of ftp in environments where a firewall prevents connections from the outside world back to the client machine. Requires that the ftp server support the PASV command. This is the default if invoked as pftp.

Well Done!