

Storage Covert Channel Lab

2025-03-27

Table of contents

1	Setting up an FTP server	1
1.1	Installation	1
1.2	Login	2
1.3	Tweaking the FTP server	3
1.4	Restart the FTP server	3
2	Storage Covert Channel	4
2.1	7 bits	4
2.2	10 bits	7

In the last lecture, you covered different ways that someone might set up and take advantage of a Covert Channel. In this lab, we shall set up one that takes advantage of the file permissions in an FTP server.

1 Setting up an FTP server

Tip

Depending on how far you went with a previous lab, you might not need to do the **Setting up an FTP server** section at all.

1.1 Installation

We shall be using vsftpd (Very Secure FTP Daemon). To do that run the following commands in the terminal

```
sudo apt update      # to update your software listings
sudo apt install vsftpd # to actually install vsftpd
```

Once the installation is done, you should be able to confirm that it is installed and running correctly with the following command.

```
sudo netstat -alpn | grep ' LISTEN '
```

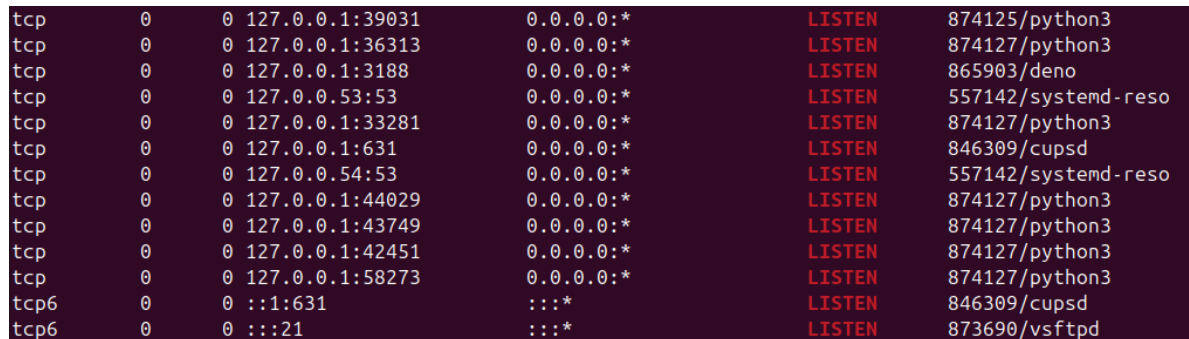
Tip

if the `netstat` command above doesn't work, you might need to install it using the command

```
sudo apt install net-tools
```

Once installed, re-run the `netstat / grep` command from earlier.

The command above should produce a list of services that are running on your Linux machine. One of the lines in that list should have vsftpd in it. Below is a screenshot showing what the line above produces on my system.



tcp	0	0	127.0.0.1:39031	0.0.0.0:*	LISTEN	874125/python3
tcp	0	0	127.0.0.1:36313	0.0.0.0:*	LISTEN	874127/python3
tcp	0	0	127.0.0.1:3188	0.0.0.0:*	LISTEN	865903/deno
tcp	0	0	127.0.0.53:53	0.0.0.0:*	LISTEN	557142/systemd-reso
tcp	0	0	127.0.0.1:33281	0.0.0.0:*	LISTEN	874127/python3
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN	846309/cupsd
tcp	0	0	127.0.0.54:53	0.0.0.0:*	LISTEN	557142/systemd-reso
tcp	0	0	127.0.0.1:44029	0.0.0.0:*	LISTEN	874127/python3
tcp	0	0	127.0.0.1:43749	0.0.0.0:*	LISTEN	874127/python3
tcp	0	0	127.0.0.1:42451	0.0.0.0:*	LISTEN	874127/python3
tcp	0	0	127.0.0.1:58273	0.0.0.0:*	LISTEN	874127/python3
tcp6	0	0	:::1:631	:::*	LISTEN	846309/cupsd
tcp6	0	0	:::21	:::*	LISTEN	873690/vsftpd

Figure 1: The last line in this Screenshot shows vsftpd is installed and working on my system

1.2 Login

Now try logging in to your ftp server from the same computer/system.

```
ftp localhost
```

Running the command above should allow you to log in to your own system using the username and password of your system. Feel free to navigate and see what you have access to.

Some commands that might be of help to you include:

```
ls
ls -lh
cd foldername
cd ..
Ctrl + D
```

1.3 Tweaking the FTP server

We can make some few modifications to make it a better experience. These modifications include adjusting the welcome message, and adjusting the folder that an ftp user is automatically sent to.

```
sudo vim /etc/vsftpd.conf    # feel free to use another editor other than vim
```

When inside the config file, look for and adjust the following lines accordingly.

```
anonymous_enable=YES
local_enable=NO
ftpd_banner=My first and Best FTP server # feel free to adjust this
anon_root=***    # where *** is the path to an appropriate folder with
                  # appropriate permissions. A suggestion is /home/ftp which you
                  # will need to create if it doesn't already exist.
```

1.4 Restart the FTP server

Restart the server and log in again. This time, you don't need to use your system's credentials to log in. You should be able to log in **anonymously**. That means your username is *anonymous* and your password is blank i.e. just hit **Enter**

```
sudo service vsftpd restart
ftp localhost
```

You can also try to ftp into your ftp server from another computer. All you'll need is the ip address of the computer on which the ftp server is located.

If you set up this server on a virtual machine and set up the network correctly, then your virtual machine will have a different IP address from your host computer and you can do this part from your host computer.

```
ftp 138.47.something.something # recall you are running this command on your
# host computer but using the ip address of your virtual machine (or
# whatever computer the server is on)
```

2 Storage Covert Channel

We are now going to use the ftp server as a Storage covert channel. More specifically, we shall use the permissions of the files on that server as a way of hiding a message.

Note that the permissions are typically made up of 10 characters that could either be a letter [drwxl] or a dash [-]. We could take the presence of any letter to be a 1 and its absence to be a 0. This allows us 10 bits of information for each file.

There are two ways we could use these 10 bits

2.1 7 bits

We could ignore the first three bits and only embed ASCII letters with the latter 7 bits of each file's permissions.

```
---xrwxrwx
```

The file permission above would be translated to 1111111.

Unfortunately, a list of files where the first 3 bits of file permissions were consistently not being used would rouse too much suspicion. So in order to make it look more convincing/surreptitious, we could add random files as noise into the file list. These “noise” files would have to have at least one of their first 3 bits set so that they could be differentiated from the actual hidden message.

Here is an example of what we're trying to do.

Suppose we have the message **Demigod** that we want to embed.

First we break it down into its ASCII representation.

```
D = 68 = 1000100
e = 101 = 1100101
m = 109 = 1101101
i = 105 = 1101001
g = 103 = 1100111
o = 111 = 1101111
d = 100 = 1100100
```

File permissions are made up of three categories of flags that determine whether three user categories (user, group, other) can read, write, or execute the associated file respectively.

If we are going to embed our message into these 9 bits, we will have to change our message bits from 7 to 9 bits by prepending 2 bits to each character's representation.

```
D = 001 000 100
e = 001 100 101
m = 001 101 101
i = 001 101 001
g = 001 100 111
o = 001 101 111
d = 001 100 100
```

Conveniently, it is possible to quickly set the permissions of a file to a 9 bit pattern. All one needs is the decimal equivalent of each 3 bit grouping. This will end up with each 3 bit pattern being replaced with a number in the range [0,7] i.e. the smallest 3 bit number - 000 to the largest 3 bit number - 111.

```
D = 104
e = 145
m = 155
i = 151
g = 147
o = 157
d = 144
```

We shall now create a bunch of random files (with the `touch` command), and adjust their permissions (with the `chmod` command) using the numbers identified above.

```
touch file1
chmod 104 file1
touch file2
chmod 145 file2
touch file3
chmod 155 file3
touch file4
chmod 151 file4
touch file5
chmod 147 file5
touch file6
chmod 157 file6
touch file7
chmod 144 file7
```

This should produce something like this in your folder.

```
---x---r-- 1 prof prof 0 Mar 27 16:15 file1*
---xr--r-x 1 prof prof 0 Mar 27 16:15 file2*
---xr-xr-x 1 prof prof 0 Mar 27 16:15 file3*
---xr-x--x 1 prof prof 0 Mar 27 16:15 file4*
---xr--rwx 1 prof prof 0 Mar 27 16:15 file5*
---xr-xrwx 1 prof prof 0 Mar 27 16:15 file6*
---xr--r-- 1 prof prof 0 Mar 27 16:15 file7*
```

Like we mentioned before, note that all the files that actually have something in their permissions do not have any of the first 3 flags set.

We should now add some noise to the files, making sure that every file that is termed as *noise* has at least one of the first 3 flags set. This will make our file server contents look more believable.

```
---x---r-- 1 prof prof 0 Mar 27 16:15 file1*
d--xrw-r-- 1 prof prof 0 Mar 27 16:15 file1.5*
---xr--r-x 1 prof prof 0 Mar 27 16:15 file2*
---xr-xr-x 1 prof prof 0 Mar 27 16:15 file3*
-r-xrwxrwx 1 prof prof 0 Mar 27 16:15 file3.5*
---xr-x--x 1 prof prof 0 Mar 27 16:15 file4*
---xr--rwx 1 prof prof 0 Mar 27 16:15 file5*
-rwx--xr-x 1 prof prof 0 Mar 27 16:15 file5.5*
---xr-xrwx 1 prof prof 0 Mar 27 16:15 file6*
-rw-r---wx 1 prof prof 0 Mar 27 16:15 file6.5*
```

```
---xr--r-- 1 prof prof 0 Mar 27 16:15 file7*  
drwxr-xrw- 1 prof prof 0 Mar 27 16:15 file7.5*
```

SUCCESS. Covert channel has been set up and hidden message has been embedded.

How would we go about retrieving the message?

```
---x---r-- 1 prof prof 0 Mar 27 16:15 file1*  
0001000100 = 68 = D  
d--x---r-- 1 prof prof 0 Mar 27 16:15 file1.5*  
1001000100 = ignored  
---xr--r-x 1 prof prof 0 Mar 27 16:15 file2*  
0001100101 = 101 = e  
...and so on...
```

2.2 10 bits

One criticism of the covert channel above is that we are wasting 30% of the bits available to us. With just a small tweak to our algorithm, it is possible to use all the 10 bits in the file permissions instead of just 7.

There are a few implications we need to know though, chief of which is that we can no longer afford to include noise files. This is because there is no way to differentiate the noise files from the files that have legitimate portions of the message embedded in their permissions.

To create the message, we'll need to

1. Order the files in the server alphabetically,
2. Create a complete message by appending all the bits of the hidden message together, and then splitting them into groups of 10. Each 10 bit grouping will be applied to a single file.
3. If the message we are trying to embed isn't made up of a perfect multiple of 10bits, then we pad the message with extra 0's.
4. During the decoding process, we should collect all the bits into a single stream, split them into groups of 7s (since we are using basic ASCII), and then convert each group of 7 into its associated character, and then reassemble the message.

try to decode the following

```

d---r---rwx 2 prof prof 4K Mar 27 20:57 0fd1b45f22e18b3
-r-xrw--w- 1 prof prof 0 Mar 27 20:57 17c455d90e49
-rw--w-r-x 1 prof prof 0 Mar 27 20:57 302289542768697c
-rw---x--- 1 prof prof 0 Mar 27 20:57 4bdf419390d83b860cec
--wxr-xrwx 1 prof prof 0 Mar 27 20:57 51451ddb647ff3566601f232
d-w---xr-- 2 prof prof 4K Mar 27 20:57 6e8dd5f0924ce30b35aeaed9
d-wxr--w- 2 prof prof 4K Mar 27 20:57 70a8cbb30
dr--r-x-w- 2 prof prof 4K Mar 27 20:57 79bf30d265cbd436079e
-rwxrwx--x 1 prof prof 0 Mar 27 20:57 81052541de641ff1ed7ca40
d-w-----wx 2 prof prof 4K Mar 27 20:57 a8b18ffb171e161c753ab8d
-rw-rwxrw- 1 prof prof 0 Mar 27 20:57 c52eda933ff95be8f914eaf62
-r-x-----x 1 prof prof 0 Mar 27 20:57 daf9509999adb4f6e6b49c7e91
d---rwxr-- 2 prof prof 4K Mar 27 20:57 f35c8e8ed0fb8a609
--wxrw--w- 1 prof prof 0 Mar 27 20:57 f4ed4ab4e61c850de968
-rwxrwx-w- 1 prof prof 0 Mar 27 20:57 f59a77545fe6d10
---x----- 1 prof prof 0 Mar 27 20:57 fce47615d2

```

Answer

The first file

```
d---r---rwx 2 prof prof 4K Mar 27 20:57 0fd1b45f22e18b3
```

decodes to 1000100111

The second file

```
-r-xrw--w- 1 prof prof 0 Mar 27 20:57 17c455d90e49
```

decodes to 0101110010

We keep on going till we get all the bits

```

100010011101011100100110010101011000100000111011111010001100
101111001011001010100111111001101000001101101111100101000001
1000111100001111001001111110100001000000

```

We then group them into 7s and convert each group into its own ASCII character.

```

1000100 1110101 1100100 1100101 0101100 0100000 1110111 1101000
D      u      d      e      ,      [space] w      h
1100101 1110010 1100101 0100111 1110011 0100000 1101101 1111001
e      r      e      '      s      [space] m      y
0100000 1100011 1100001 1110010 0111111 0100001 0000000
[space] c      a      r      ?      !      [ignored]

```

which produces the message *Dude, where's my car?!*