Introduzione alla CyberSecurity

Pasquale Caporaso
Università di Tor Vergata

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Indice

- Informazioni sulla CyberChallenge
- Hacking Etico
- Cos’ è una CTF?
- Linux 101

La Squadra

15 Studenti
4 Sapienza - 11 Tor Vergata
5 Scuole Superiori
5 Istruttori

Pasquale

Francesco

Milena

Pierciro

Sara

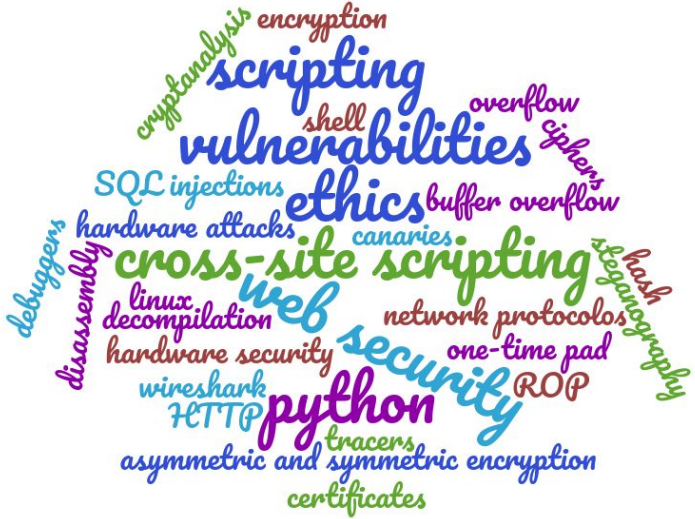
Chi sono io

- Pasquale Caporaso, phd student, security researcher for CNIT
- Ex-Malware Analyst for Leonardo spa
- Research in cyber security, malware and operating systems
- Addicted to CTFs
- Finalist in CC Sapienza 2020

Cosa faremo

Per i prossimi 4 mesi:

- 6 ore di “lezione” a settimana
  - 2 di teoria
  - 4 di pratica



Cosa faremo

Lunedì:

- 16:00 - 18:00

Mercoledì:

- 16:00 - 18:00

Venerdì:

- 16:00 - 18:00

Data	Argomento lezione	Tipo	Speaker
17/02/2025	Introduzione Generale	Teoria	Pasquale
19/02/2025	Introduzione Python - libreria pwntools	Teoria/Pratica	Pierciro
21/02/2025	Pratica - Python	Pratica	Pasquale/Pierciro
24/02/2025	Network Sec	Teoria	Sara
26/02/2025	Network Sec	Pratica	Sara
28/02/2025	Network Sec	Pratica	Sara
03/03/2025	Web 1	Teoria	Francesco
05/03/2025	Web 1	Pratica	Milena
07/03/2025	Web 1	Pratica	Milena
10/03/2025	Rev 1	Teoria	Pasquale
12/03/2025	Rev 1	Pratica	Pasquale
14/03/2025	Rev 1	Pratica	Pasquale
17/03/2025	Web 2	Teoria	Francesco
19/03/2025	Web 2	Pratica	Milena
21/03/2025	Web 2	Pratica	Milena
24/03/2025	Crypto 1	Teoria	Pierciro

Cosa faremo

Gara finale:

- 28 Maggio
- Gara CTF Jeopardy: 8 ore
- 6 qualificati per la squadra finale

SIAMO NOI A DECIDERE LA SQUADRA FINALE!

Dove trovare il materiale

Siti utili:

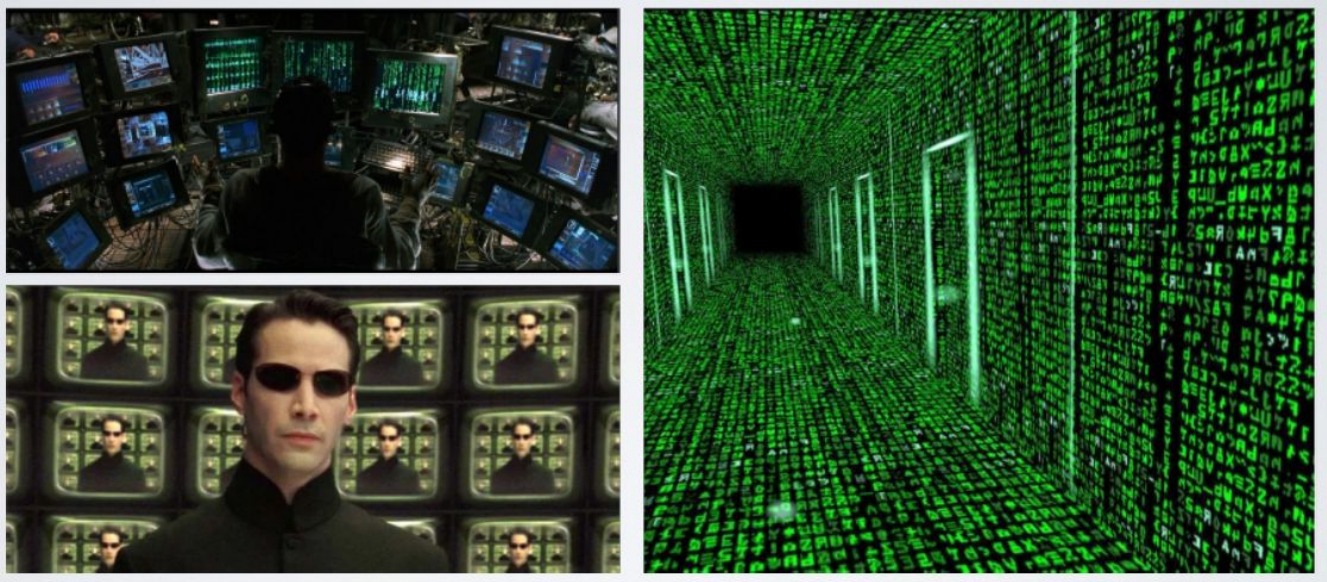
- Cartella Drive
- Piattaforma di Addestramento
- Canale Teams

Per le domande:

- Telegram/Discord

La Dura Realtà

Quello che vi aspettate:



La Dura Realtà

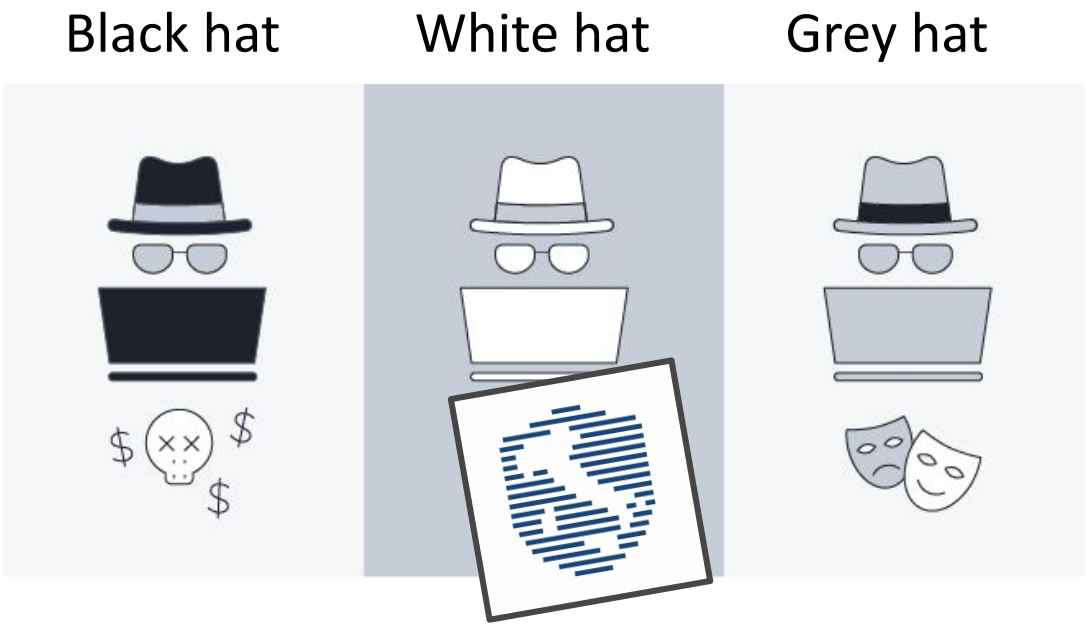
Quello che sarà:

Ethical Hacking

Hacker:

- chi, servendosi delle proprie conoscenze nella tecnica di programmazione degli elaboratori elettronici, penetri abusivamente in una rete di calcolatori per utilizzare dati e informazioni in essa contenuti, per lo più allo scopo di aumentare i gradi di libertà di un sistema chiuso e insegnare ad altri come mantenerlo libero ed efficiente.

Ethical Hacking



La nostra filosofia

Imparare facendo

- vi diamo gli strumenti per lavorare e vi lasciamo andare avanti da soli

Capture The Flag

Un **Capture the Flag** (abbreviato in **CTF**) è un gioco di **hacking** dove team o singoli cercano **vulnerabilità** in sistemi e software messi a disposizione dagli organizzatori della competizione al fine di sfruttarle e di collezionare le varie **flag** nascoste sul sistema bersaglio.

Diversi Tipi:

- Jeopardy
- Attacco / Difesa

https://ctftime.org/

Jeopardy CTF

Tante challenge, un solo obiettivo: Trovare la flag

- flag{Th1s_is_4_flag_3x@mpl3}

Tanti livelli di difficoltà:

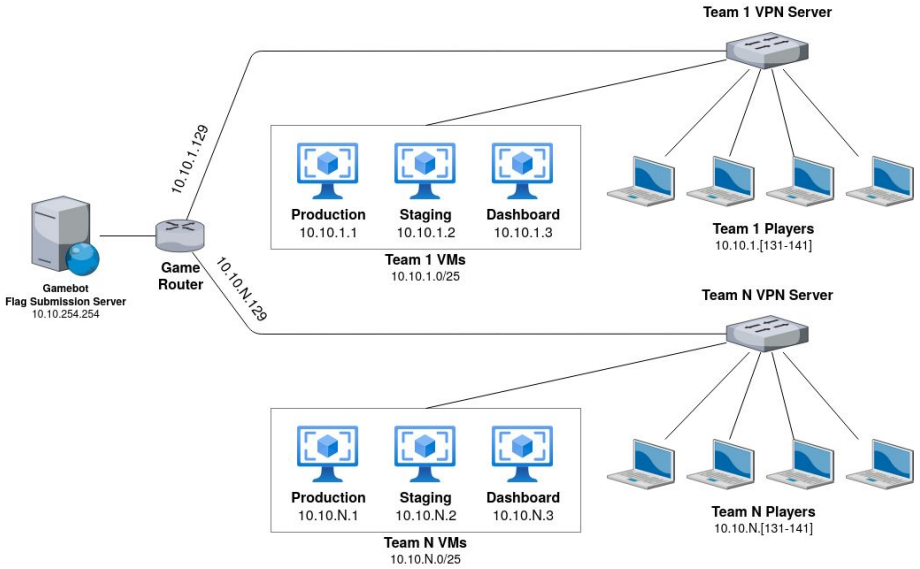
- cliccare su link nascosto
- reversare una cpu implementata con i domino

Web	Crypto	Forensics	Reverse	Misc	Pwn
1	165	100	50	50	50
150	150	150	100	100	150
204	150	150	150	165	200
203	200	200	200	150	250
206	257	200	300	200	323
318	334	250	300	300	440
325	400	347	400		
	430	350			

CTF Attacco/Difesa

Ogni squadra ha dei servizi, dovete:

- mantenerli funzionanti
- attaccare gli altri
- patchare i vostri



Introduzione a Linux

Pasquale Caporaso
Università di Tor Vergata

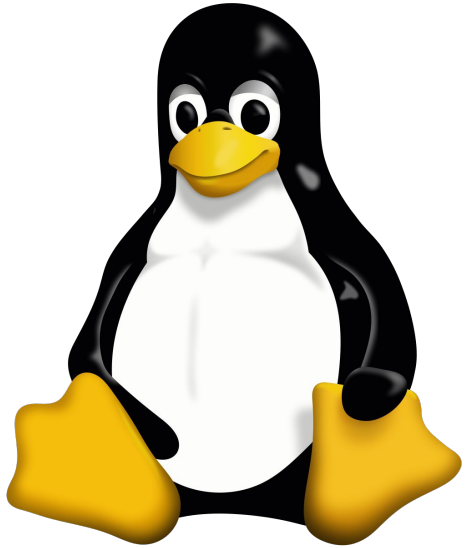
CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

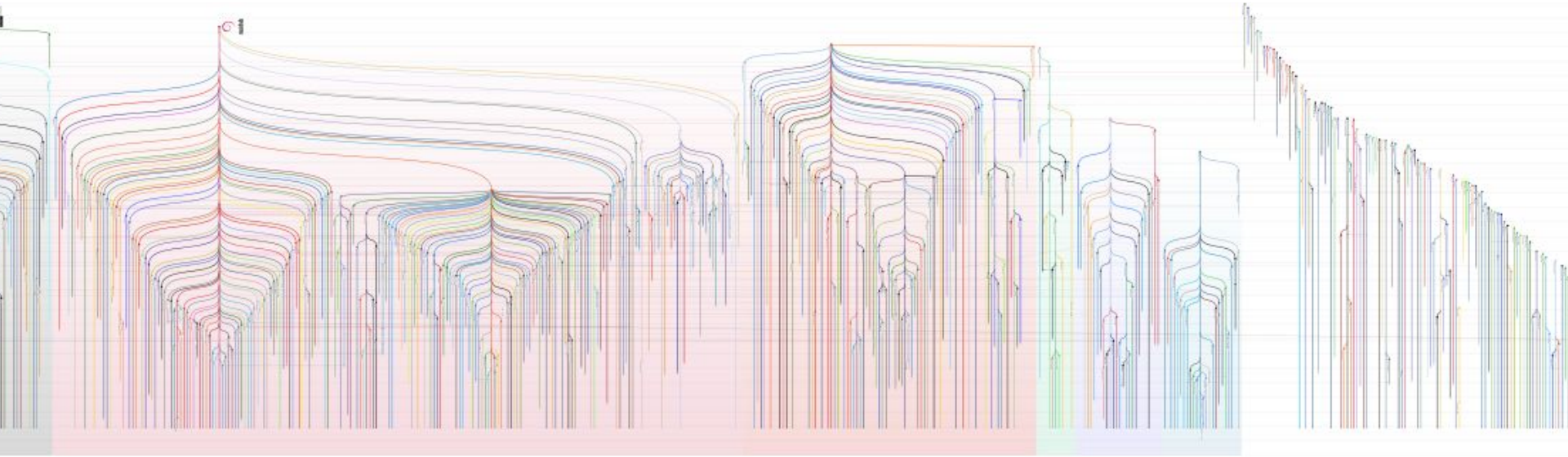
https://cybersecnatlab.it

Linux

- Sistema operativo Open-Source
- Una storia molto complicata…
- Un solo kernel, tante distribuzione

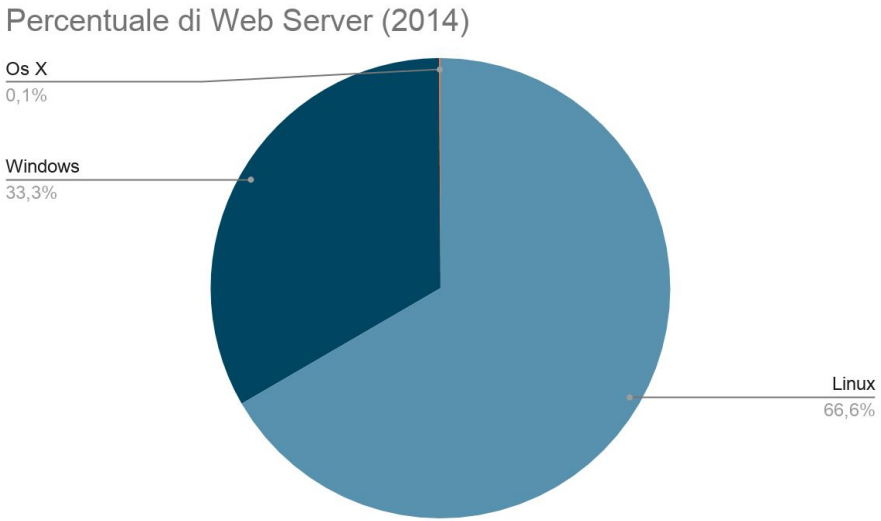


Linux



Ma perchè?

- Molto comodo
- Linux domina Internet (e i super computer)



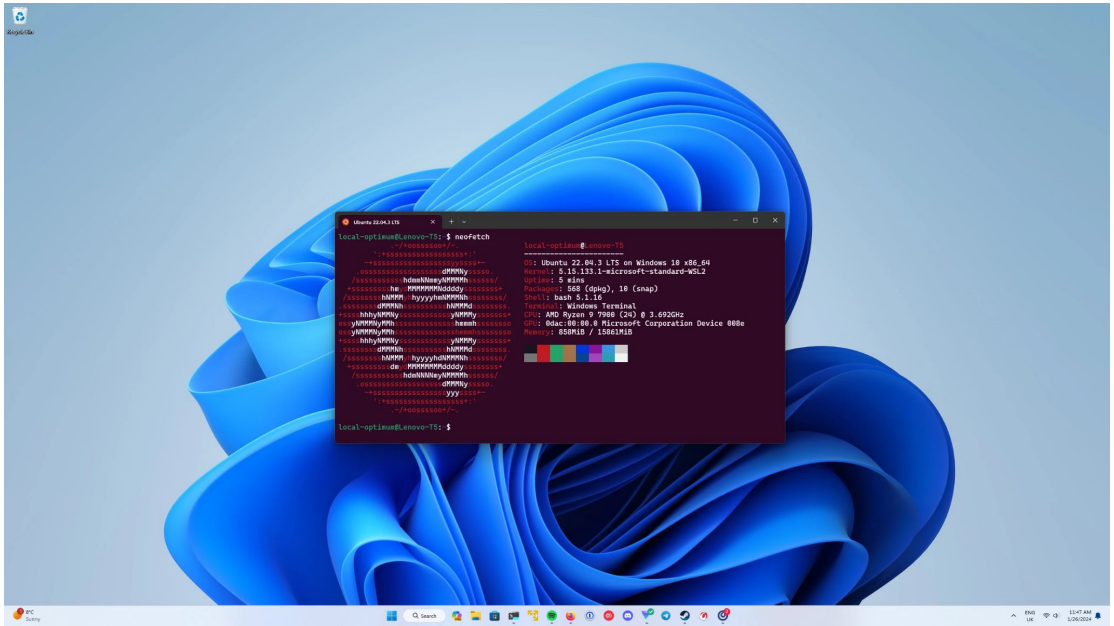
Setup your Linux machine

- Step 0:

https://ctf.cyberchallenge.it/training/environment

Setup your Linux machine

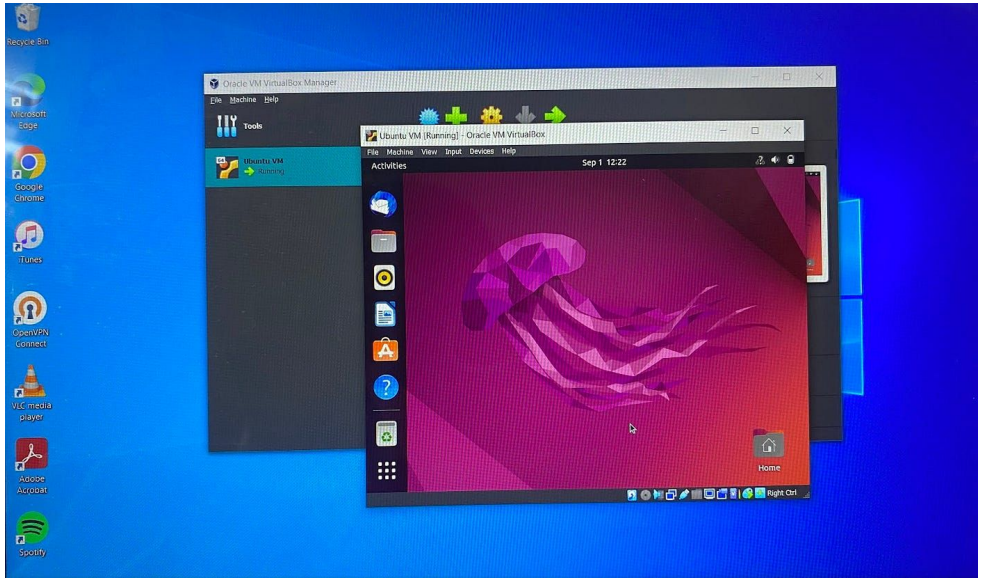
- Choice 1: WSL



Setup your Linux machine

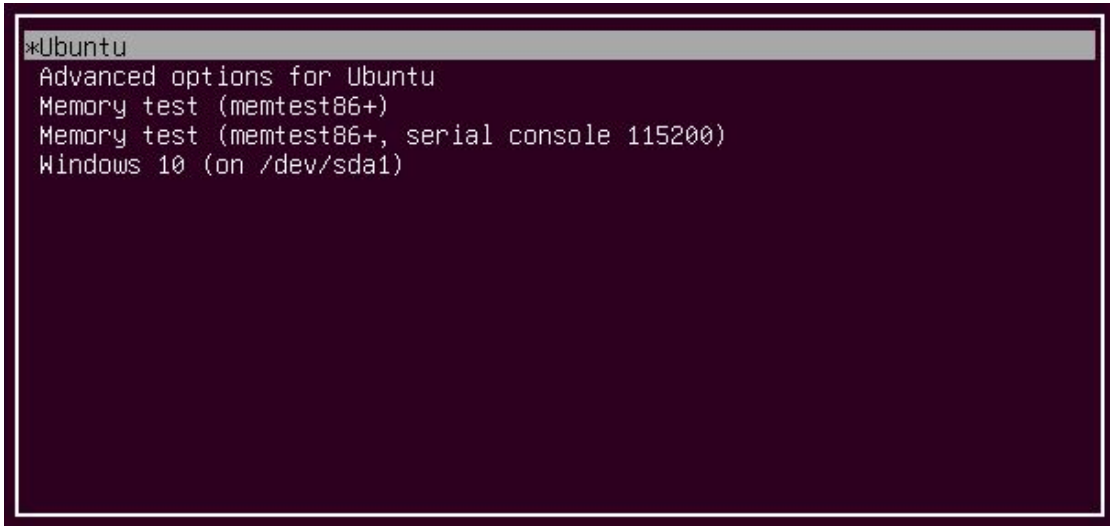
- Choice 2: Virtual Machine

https://www.makeuseof.com/install-ubuntu-on-vmware-workstation/



Setup your Linux machine

- Choice 3: Dual boot



Which distro

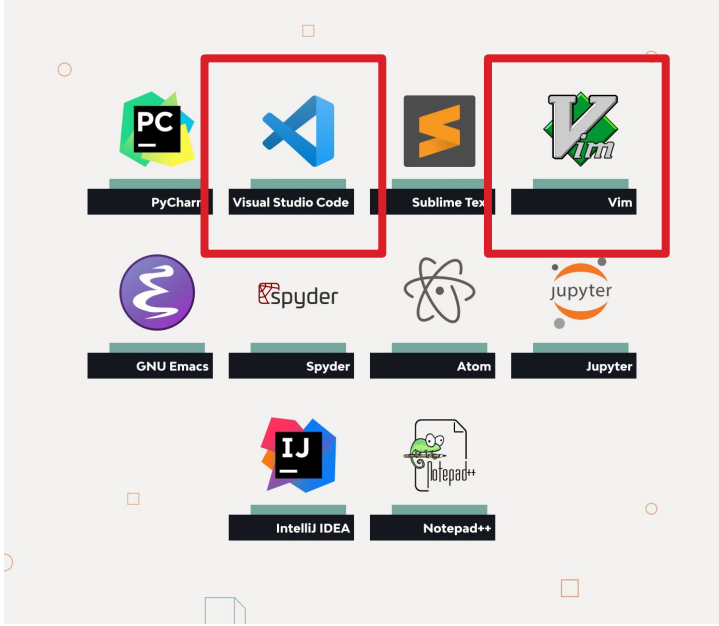
Linux users discussing which distro is the best

Which distro

- My choice:

Where to write code

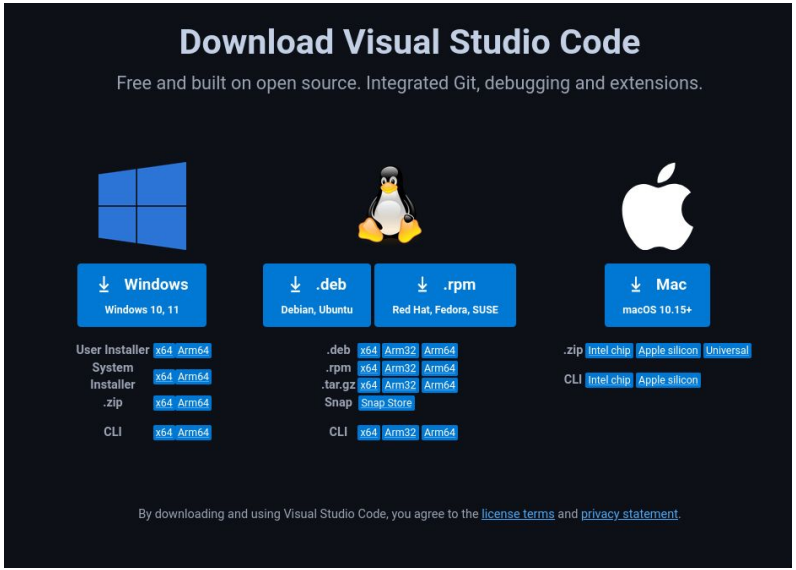
- Wherever you want



Where to write code

- Vscode

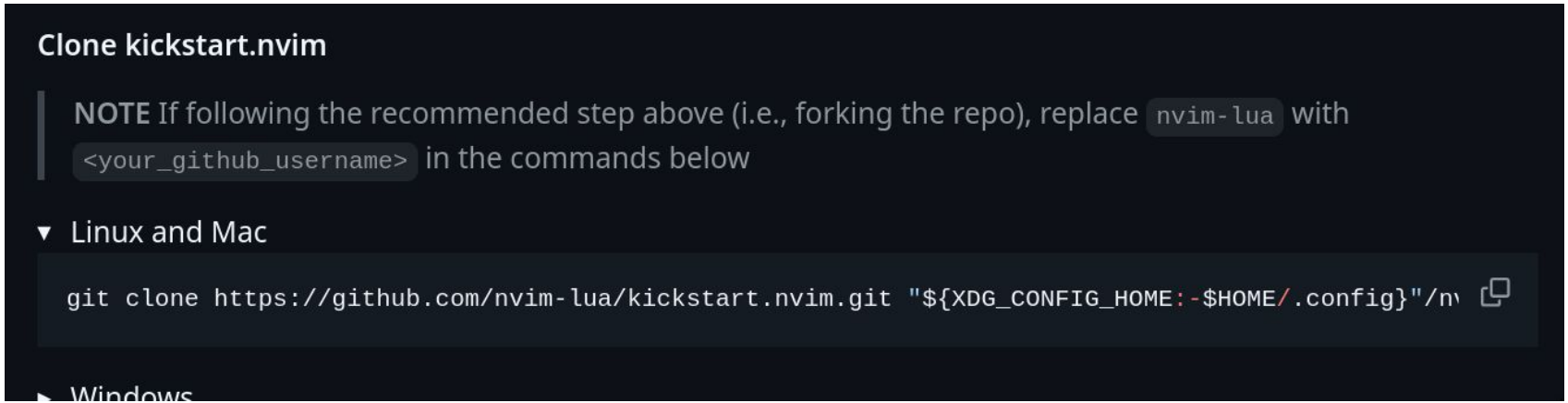
https://code.visualstudio.com/download



Where to write code

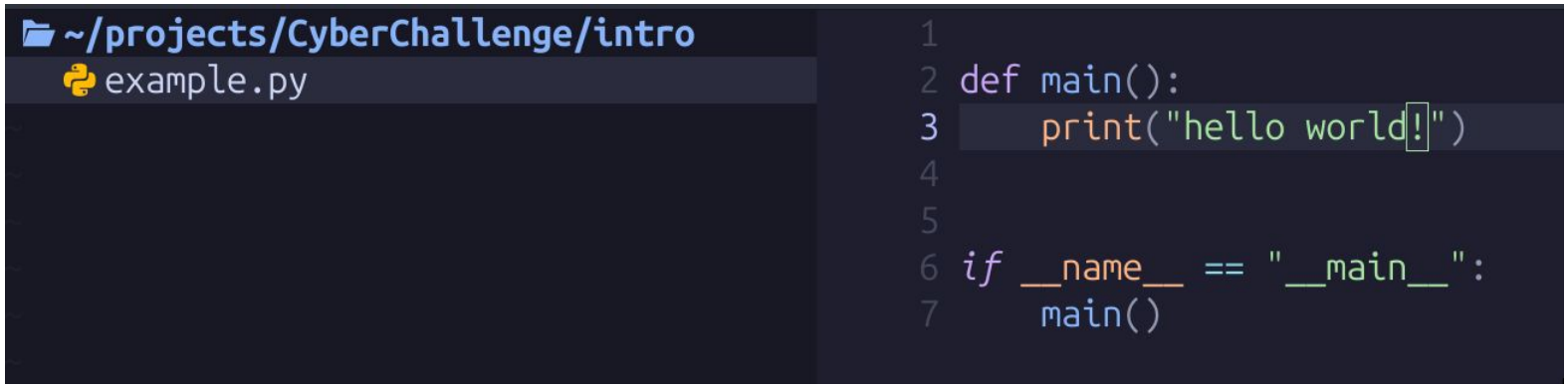
- Nvim

https://github.com/nvim-lua/kickstart.nvim



How to run code

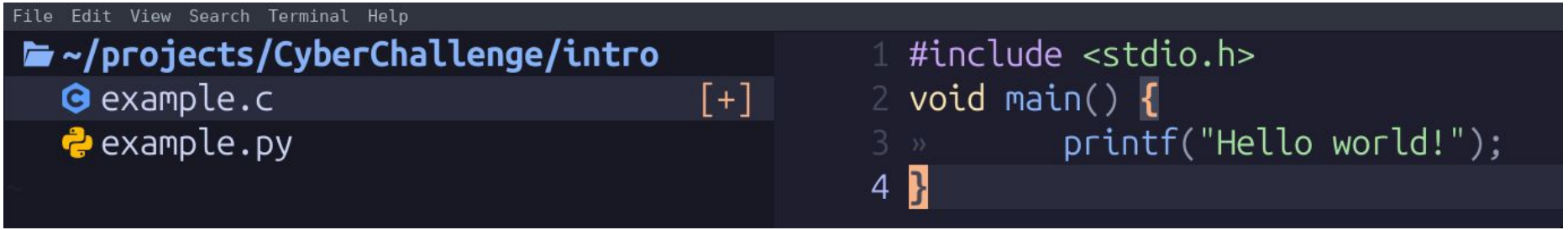
- python




```
~/projects/CyberChallenge/intro  sudo apt install python3
~/projects/CyberChallenge/intro  python3 example.py
hello world!
~/projects/CyberChallenge/intro
```

How to run code

- C



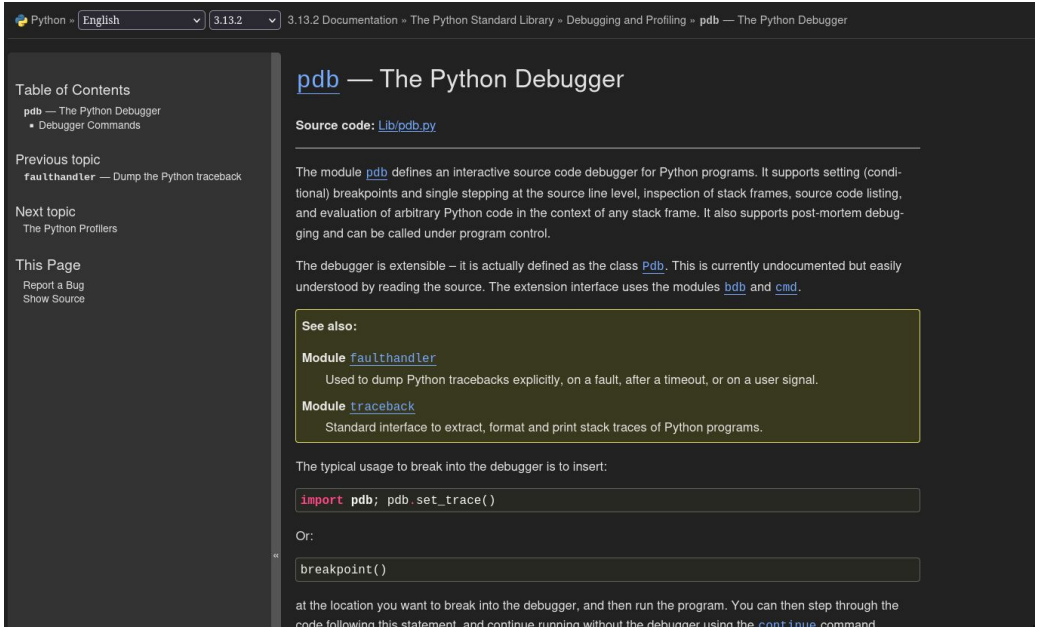

```
~/projects/CyberChallenge/intro  sudo apt install gcc
~/projects/CyberChallenge/intro  gcc example.c -o example
~/projects/CyberChallenge/intro  ./example
Hello world!%
~/projects/CyberChallenge/intro
```

How to debug code

- python

How to debug code

- python



How to debug code

- c

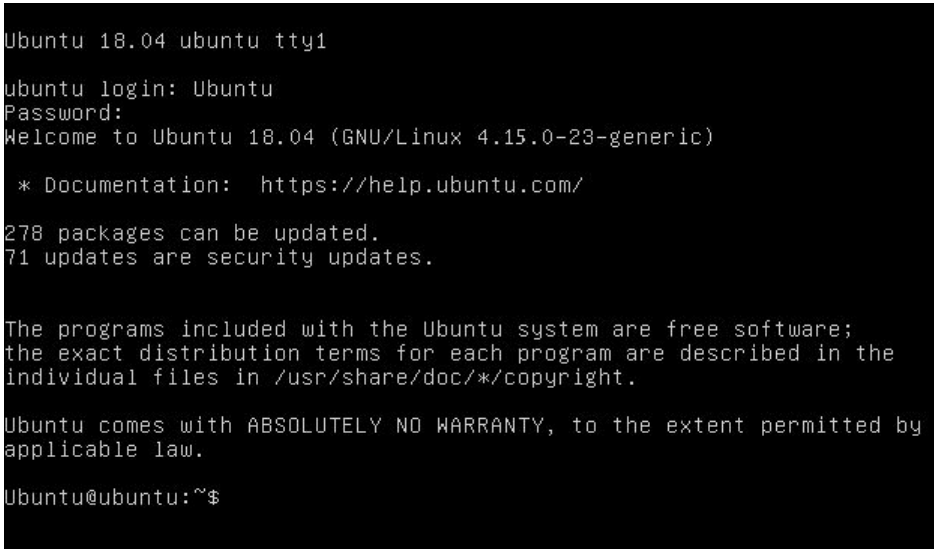
GDB + GEF

Linux OS – Filesystem

- Tree structure
  - /
    - /boot/ → system boot files
    - /dev/ → hardware devices (each device is a file! more or less…)
    - /etc/ → system configuration
    - /home/ → user home directories
    - /lib/ → system libraries
    - /mnt/ → removable devices (e.g. usb)
    - /proc/ → kernel interaction (file abstraction) [now deprecated but still in use]
    - /sys/ → kernel interaction
    - /tmp/ → temporary files
    - /usr/ → universal system resources → mostly user installed programs
    - /var/ → variable files

Linux OS – tty

- tty = virtual terminal
  - It's the main interface to the OS
    - Try to install Linux without any graphical interface, you'll be welcomed by a tty ☺️
  - It's called virtual because it's a virtual version of old *teletypes*
  - *tty = text input/output interface provided by the kernel*
    - *input → processing → output*
  - tty files are located in */dev*
- Console = terminal + physical tools (e.g. keyboard, screen)



Linux OS – shell and file descriptors

- shell = command line interpreter
  - *input → shell → output*
- There are 3 default files (with associated **file descriptors**) the shell works with:
  - 0 – stdin
  - 1 – stdout
  - 2 – stderr
- On a virtual terminal, by default
  - fd 0 is connected to the keyboard
  - fd 1 and 2 are connected to the screen

Linux OS – GUI

- GUI = Graphical User Interface
  - Windows, Icons, Mouse
  - User-friendly interface to the OS
  - Runs on top of a tty
  - Allows ***pseudo-tty*** thanks to programs called ***terminal emulators***

Linux OS – shell /2

- The shell can be used in 2 ways:
  - Interactively
  - By executing a file, written in the “shell language”
- There are many programs which implement a shell:
  - sh, bash, zsh, …
- A shell welcomes the user with a **shell prompt**
  - It means that the shell is ready to accept commands to be executed
  - Whatever is written at the shell prompt is
    - A program to be executed
    - A shell command
- The shell makes use of **environmental variables**
  - Global, OS-level variables which configure the system environment (e.g. *$PATH)*
  - Global, run-time variables defining the local environment (e.g. *$USER*, $TERM)

Linux OS – shell /3

- The program **echo**
  - prints back whatever it finds as *command line argument*
    - Command line arguments are strings following the program name, separated by spaces
  - can also be used to print env vars
    - echo $PATH
- Why does “echo $PATH” works? Where is the executable file “echo”?
- The program **pwd**
  - prints the current working directory
  - How to change directory? use **cd**
  - How to list files/directories inside a directory? Use **ls**
    - play with ls arguments
    - Notice that “.” and “..” folders inside every folder?

Linux OS – shell /4

- The program **cat** *(concatenate)*
  - cat file.txt
  - cat file1.txt file2.txt
  - cat –
- CLI editors
  - nano
  - vim
  - …

Linux OS – redirection

- Input and Output of a program can be redirected an can be used to feed other programs.
  - operator “>” redirects **output**
    - By default it redirects **stdout** (i.e. fd 1)
  - operator “<“ redirects **input**
    - By default it redirects **stdin** (i.e. fd 0)

- The general syntax for redirection
  - fd1 [operator] &fd2 → redirects *fd1* to *fd2* (watch the “&”!)
  - fd1 [operator] filename → redirects *fd1* to *filename*
    - cat file.txt > output.txt is equivalent to cat file.txt 1> output.txt

Linux OS – redirection/2

- Redirect multiple fds to same destination
  - *cat file.txt > output.txt 2>&1 [cat file.txt 1>output.txt 2>&1]*
    - redirects **stdout** to **output.txt** and **stderr** to **stdout** (so to output.txt). Result: stdout and stderr redirected to txt file

- The special file */dev/null*
  - Bytes written to this file are simply trashed
  - Useful when you want to ignore some output
  - *find / -type f –name sudo 2>/dev/null*
    - Hey, ignore stderr and show just stdout!

Linux OS – pipes

- Pipes are (guess what?) *files* used by processes to intercommunicate (IPC)
  - A *named pipe* or *fifo* exists on the filesystem
  - An *anonymous pipe* is managed directly by the kernel and doesn't exist on the filesystem

- Suppose that you want to use the output of a program as input to another program
  - You can use redirection
    - program1 > output
    - program2 < output
  - Or you can use an anonymous pipe!
    - program1 | program2
    - program1 | program2 | program3 | …

Linux OS – pipes /2

- Named pipes or fifos need to be created before they can be used
  - *mkfifo /tmp/myfifo*
  - *echo “let’s try” > /tmp/mkfifo*
    - notice that the program is blocked!
  - *[on another terminal] cat /tmp/mkfifo*
    - Look at the output. And notice that the echo process is now unlocked
- You get the same result if you first spawn the reader process and then the writer process.
- Pipes are closed when there is no writer associated
  - cat /tmp/myfifo
  - echo “test” > /tmp/myfifo
    - The cat process is now terminated, because there is no writer associated.

Linux OS – shell /5

- The program **grep**
  - filters the input based on rules
    - Very complex command, you can learn everything by reading the linux manual: **man grep**
- Example: See if any of the txt files inside the current directory contains the string “user”
  - cat *.txt | grep “user”
- Other text filtering programs
  - **awk** extract tokens
  - **sed** replace strings
  - **cut** split strings and grab only some parts
  - …

Linux OS – shell /6

- Examples
  - ls –l | grep ‘user’ | awk ‘{print $1}’
  - ls –l | grep –iE ‘ic$’
  - ls –l | awk ‘{print $9}’ | grep –iE ‘^P’
  - …

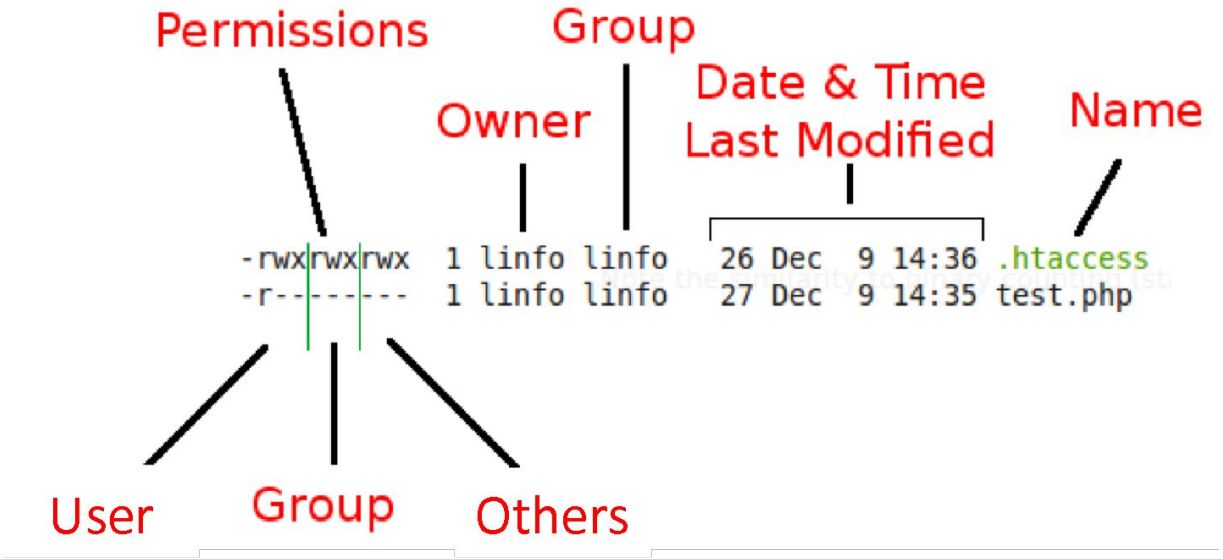
Linux OS – strings

- Strings are sequences of characters enclosed in quotes (single or double).
  - You can omit quotes when the string doesn't contain spaces or *other characters*
  - You must use quotes otherwise!
- Examples
  1. ls –l "file.txt"
  2. ls –l file.txt
  3. cat file with spaces.txt
  4. cat "file with spaces.txt"
- You can use ' inside " " and " inside ' '
- What if there is a file named: hey"joh'n.txt?
  - Cat "hey\"joh'n.txt"

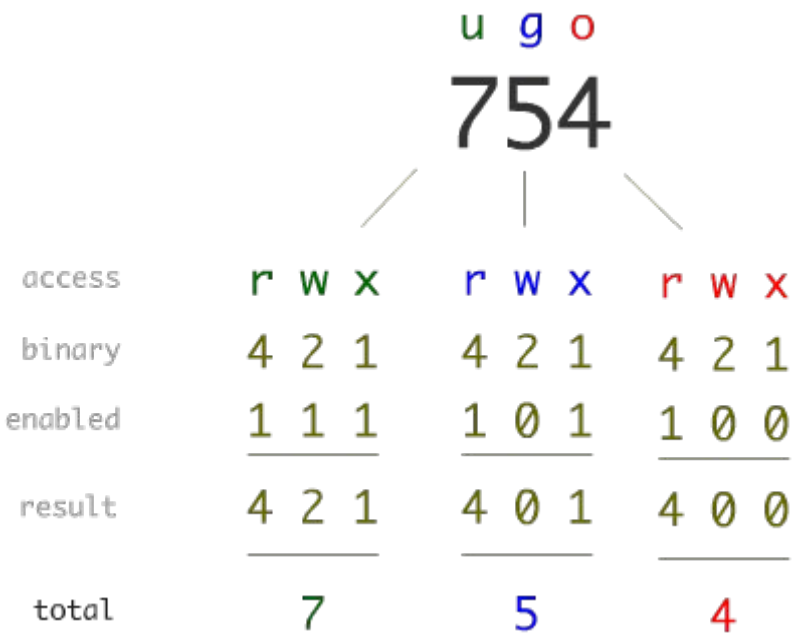
Linux OS – escape sequence

- \" is an **escape sequence**
  - A string is *escaped* when all dangerous characters are replaced with the corresponding escape sequence
- Dangerous characters?
  - cat file with spaces.txt cat file\ with\ spaces.txt
    - Here the space is dangerous because it is used as separator for command line arguments. So it can't be used to specify file names containing spaces.
  - " inside " " and ' inside ' '
  - unprintable characters (\n, \r, \t, …)
- Hey, I want to print "\n" (and not a newline)
  - echo –e "\n"
  - echo –E "\n"

Linux permissions



Linux permissions



root@kali# chown user:root test □ ?

root@kali# chown user:user test

root@kali# chmod 600 test

root@kali# chmod o+x test

root@kali# chmod g+rwx test

Sudoers

- root@kali# adduser user *(unprivileged user)*
- root@kali# *cat /etc/passwd | grep "bash\|sh”* *(get users of the system)*
- root@kali# id user (information about the user user)
- *root@kali# su user (log as user user)*
- *user@kali$ cat /etc/shadow PERMISSION DENIED*

Sudoers

- root@kali# visudo (edit /etc/sudoers file)
  - user ALL=(root) NOPASSWD: /bin/cat *
- user@kali# sudo –l (what can I sudo?!)
- user@kali# sudo cat /etc/shadow (we can cat everything?! As root user!)
- How can we bypass when sudoers has user ALL=(root) NOPASSWD: /bin/less /var/log/*
- What about this? user ALL=(root) NOPASSWD: /tmp/myprogram

Setuid/Setgid

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

- When the **setuid/setgid** bit are used, the executable does not run with the privileges of the user who launched it, but with that of the file owner/group instead.

- root@kali# ls -la /usr/bin/passwd (-rwsr-xr-x)

Setuid/Setgid

- root@kali# chmod +s sh
- root@kali# chown root:vdsi sh
- root@kali# chmod u+s sh
- root@kali# chmod -s sh
- root@kali# chmod g+s sh

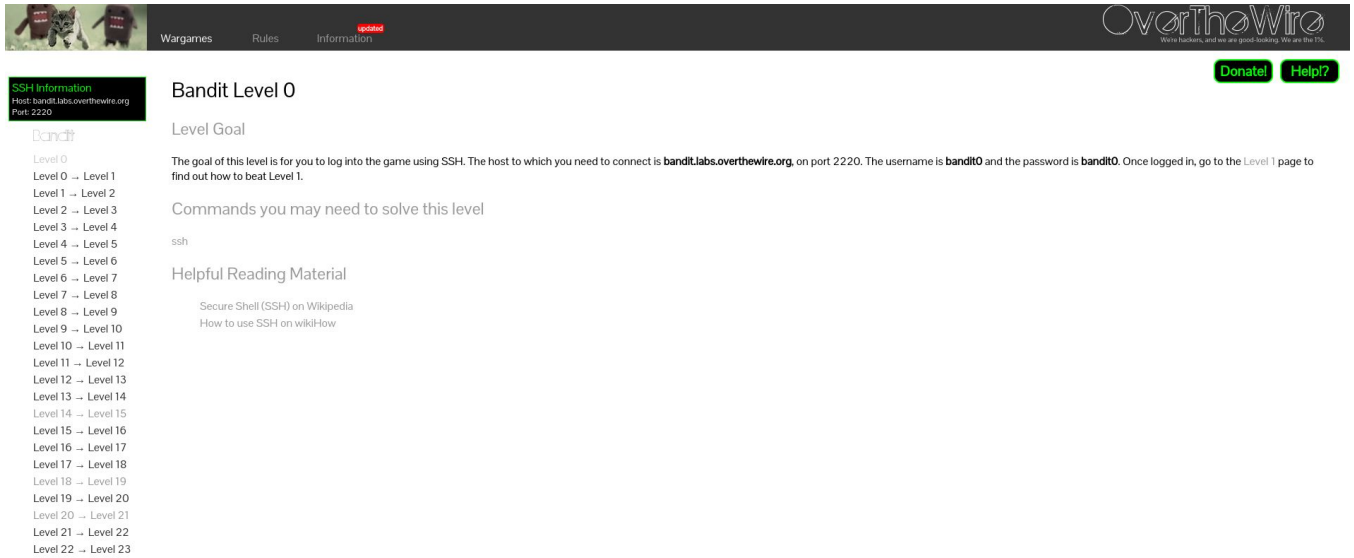
Setuid/Setgid

- Try it with /bin/bash binary….it does not work?! Just use –p (preserve privileges)
- What about creating our own binary that preserves suid/gid?

- root@kali# cat > suid.c <<EOF
- #include <stdio.h>
- #include <sys/types.h>
- #include <unistd.h>
- int main(void) { setuid(0); setgid(0); system("/bin/bash"); }
- EOF

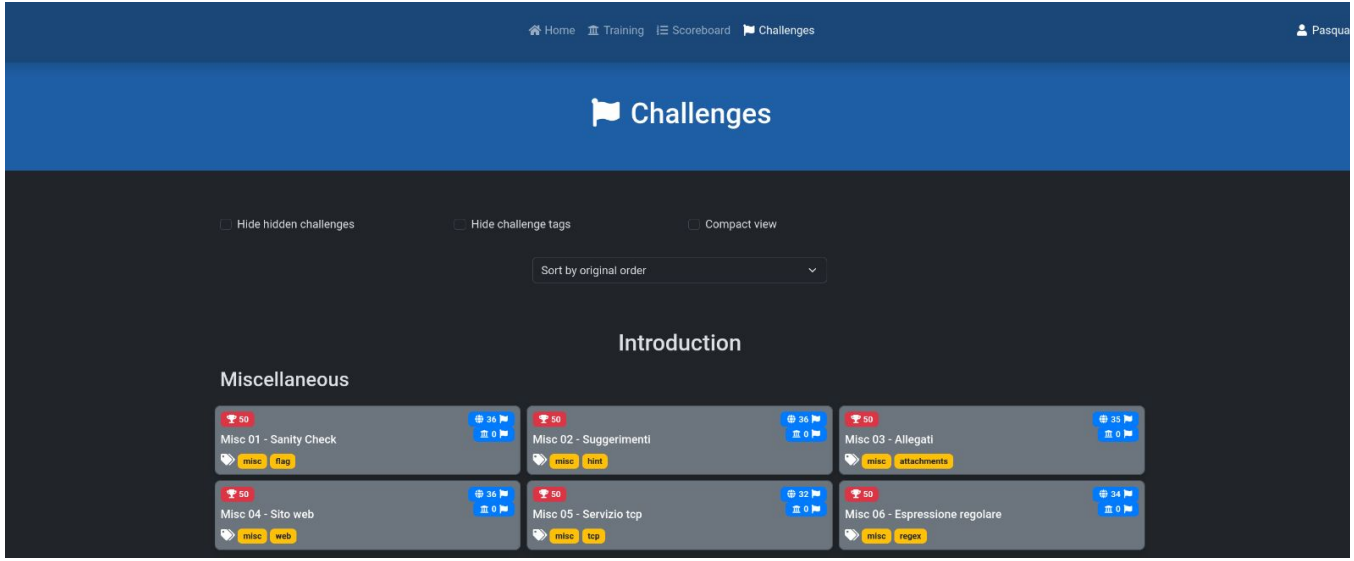
Linux Training

https://overthewire.org/wargames/bandit



Piattaforma di allenamento

https://ctf.cyberchallenge.it



Domande?