

# Linux DAC Lab

## By Michael Ambeguia

**Purpose:** The purpose of this lab is to go over the default Linux system access control method, discretionary access control. I will be using my Ubuntu Server VM to demonstrate the discretionary access controls. I will also go over some extra permissions and attributes that can be applied on Linux files.

### Skills applied:

1. Linux DAC
2. Linux DAC ACLs
3. Linux special permissions ( Setuid, Setgid, sticky bit)
4. Extra attributes ( immutable, append only)

### Sections:

1. Understanding DAC
2. Demonstrating DAC features on an Ubuntu Server

## Section #1 Understanding DAC

What is DAC?

DAC (discretionary access control) is an access control mechanism that allows users to set permissions on folders, files, and programs at their discretion. Users are able to choose the necessary permissions they need and want for their data.

Why would you use DAC?

There are four main reasons why you would want to use DAC:

1. **Simplicity:** DAC is straightforward to implement and understand, making it accessible for end users without extensive technical knowledge. With DAC, users can easily manage access permissions for their own resources, reducing the reliance on IT administrators for routine access control tasks. This simplicity streamlines access management processes and empowers users to control their data effectively.
2. **Flexibility:** DAC offers a high degree of flexibility by allowing resource owners to dynamically adjust permissions as needed. Owners can modify access rights on a whim, quickly adapting to changing requirements or business needs. This agility enables organizations to respond promptly to access requests, security incidents, or changes in user roles without cumbersome administrative overhead.
3. **Decentralization:** DAC decentralizes access control management by assigning ownership and control of resources to their respective data owners. Unlike centralized access control models, such as Role-Based Access Control (RBAC), DAC distributes responsibility for

access decisions across the organization. This decentralized approach fosters accountability and empowers data owners to enforce access policies that align with their specific data governance requirements.

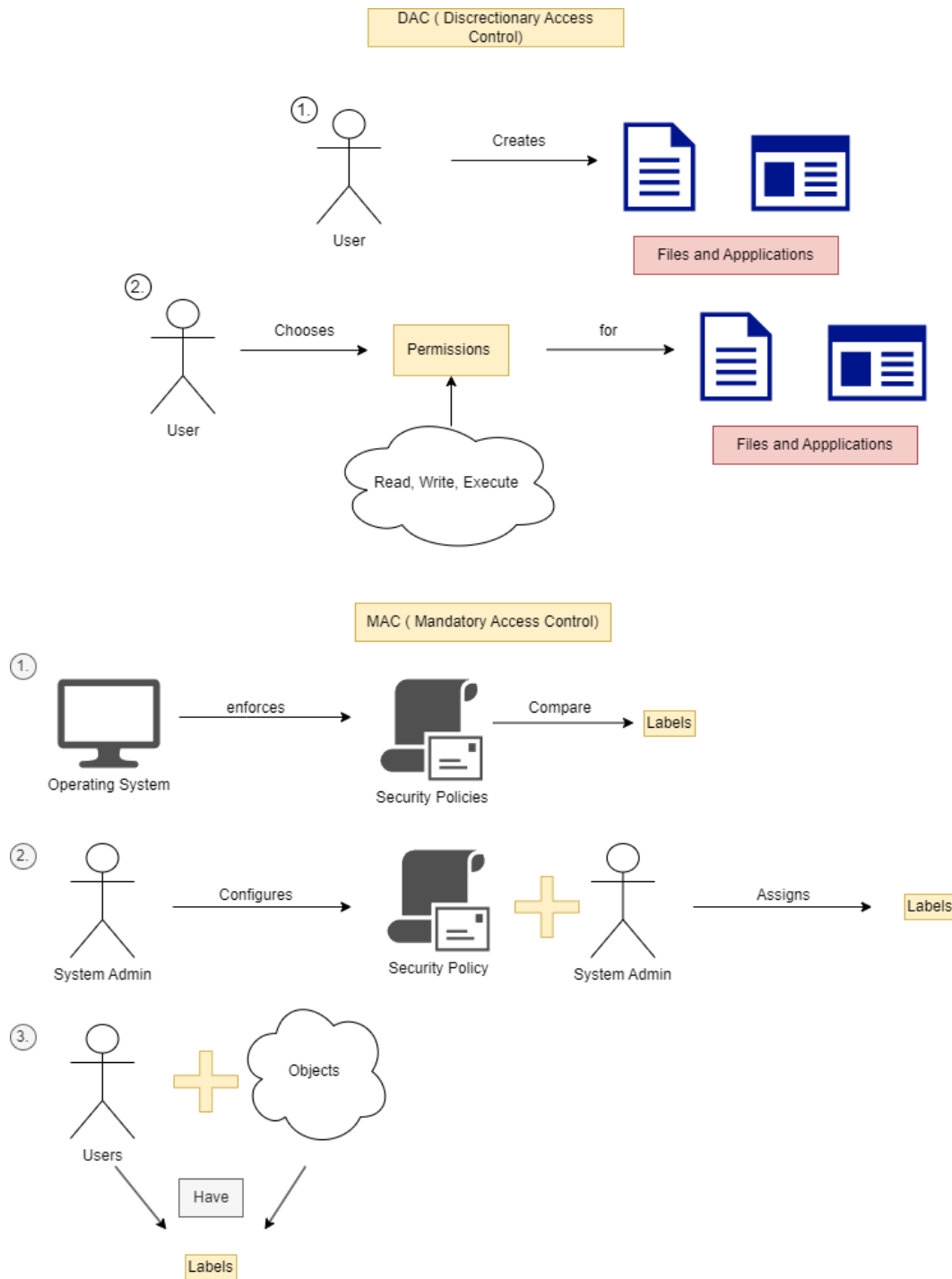
4. Customization: DAC facilitates granular control over access permissions, allowing users to define precise access rights for individual resources or groups of users. This granularity enables fine-tuning of access privileges to match the sensitivity and importance of data assets. Additionally, DAC supports group-based permissions, simplifying access management for large sets of users with similar access requirements while still allowing for customization at the individual level.

Are there security concerns associated with DAC?

Yes there are quite a few security concerns associated with DAC? DAC grants data owners the ability to set permissions at their own discretion, and this can cause some data security concerns.

DAC Concerns:

1. Lack of accountability: DAC relies on trust. The organization is trusting that users will apply the appropriate permissions on data. Unfortunately, there is no easy way to audit DAC implementations to govern this trust.
2. Data Leaks: Improperly set permissions can lead to sensitive data being available to non-authorized users.
3. Dependency on user awareness: DAC relies on end users having the proper training to know how to protect sensitive data. Organizations must ensure that users are trained on proper DAC permission usage, but end users can always make mistakes. Also users would have to know what data the organization considers to be sensitive.



DAC:

1. Users create files and download apps on their device.
2. Users choose the permissions they want to set on their files and downloaded applications.

## Section #2 Demonstrating DAC features on an Ubuntu Server

To demonstrate Linux discretionary access control features I will be using my Ubuntu Server. Before demonstrating the DAC features I will first go over how Linux system DAC permissions work.

## Linux DAC permissions:

Linux systems have a pretty simplified permissions scheme that focuses on three main permissions: read, write, and execute.

What does each permission allow a user to do on a Linux system?

1. Read: Read allows a user on a linux system to read the contents of a file. The user can read the contents by simply using a text editor like nano or vim, or by using commands like less, more and cat.
2. Write: Write allows a user to add, modify and delete content from a file. You would typically need both read and write permissions though since without read a user won't be able to view the contents of the file to know what is already there.
3. Execute: Execute allows a user to cd into a directory and travel through the directories subdirectories. Execute allows a user to run a program or script as well.

How are DAC permissions represented on a Linux system? How are the permissions applied and who do they apply to?

1. Linux permissions can be represented in two ways, by using letters, and by using octal values.

Letter representation: Read = r Write = w Execute= x

Octal representation: Read = 4 Write =2 Execute = 1

2. Linux permissions in the context of DAC are applied to the owner, the group and others. The owner is the user who created the file. The group by default is the primary group of the owner, but it can be set to a secondary group after the fact. Others refers to anyone else on the system who is not the owner or a member of the associated group.
3. Permissions on a linux system are applied to owners, groups, and others through the use of either the aforementioned letter representation or the octal representation:

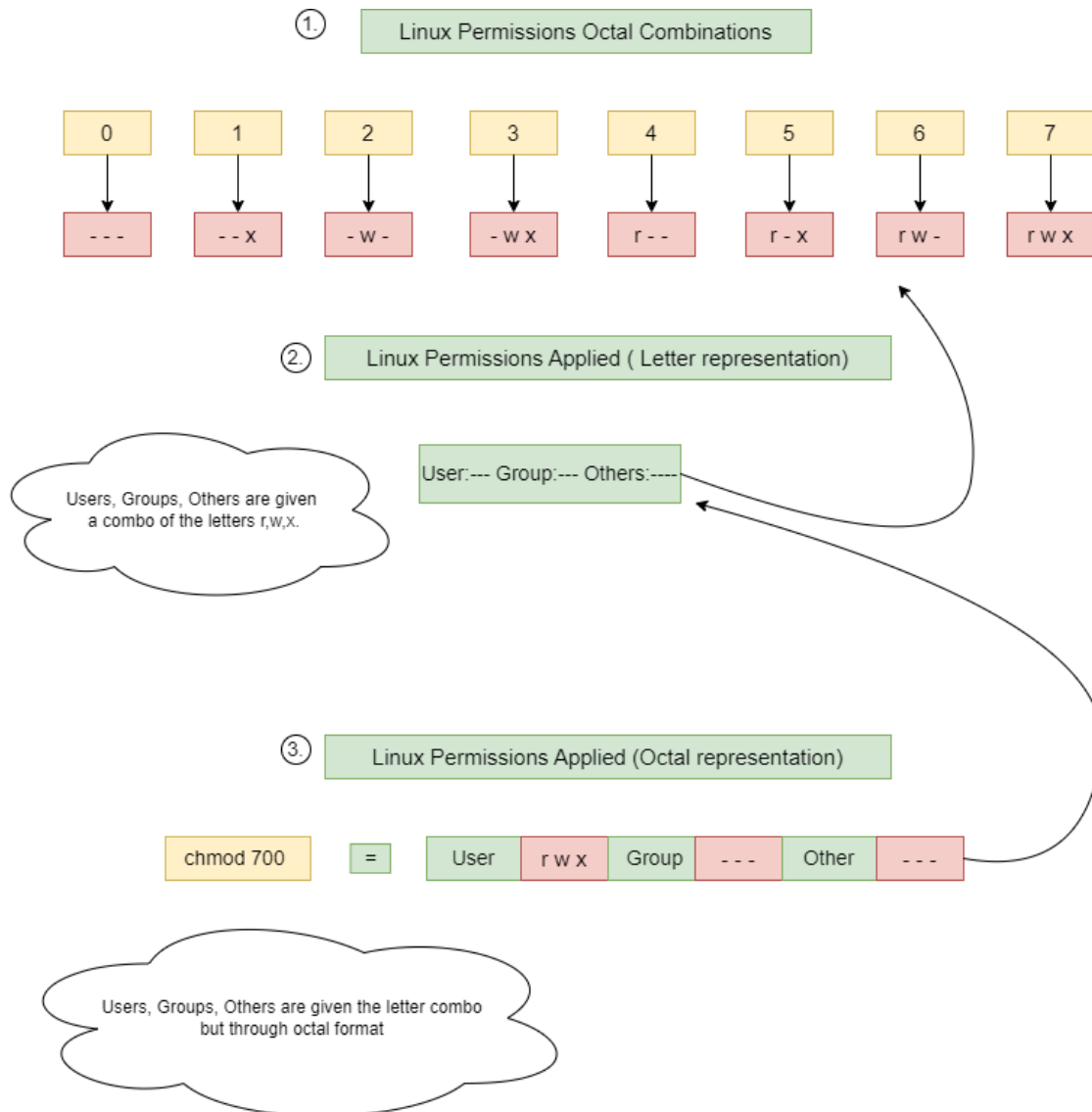


Diagram explanation:

1. Octal permissions can be used to represent a combination of the three letters representing read, write, and execute. Octal permissions are simply a combination of 4,2, and 1 added up. If a permission is not granted, then 0 is added. For example, octal value 3 represents read and execute since read (2) + execute (1) = 3 (rx).
2. User owners, groups, and others can be assigned three permissions, so a file can have a total of 9 permission slots that use the letter representation of the permissions.
3. Octal permissions are typically three digits long. The hundredth place represents the user, the tens place represents the group, and the one place represents others.

DAC demonstrated:

DAC is evident on Linux systems since file permissions are set and modified by the owner of a resource. If a user is an owner of a file, or a member of a group that owns a file and the group has write permissions, they can modify the permissions for a given file. Also the root user and a user using sudo privileges can change the permissions of any file despite the current permissions. DAC on linux is discretionary since the owner can change the permissions on a whim to meet new demands.

## Basic DAC implementation:

### 1. Modifying file permissions:

As my spy3 user I will create a file named grocery\_list.txt.

```
spy3@spyserver1:~/spy3_data$ ls -l grocery_list.txt
-rw-rw-r-- 1 spy3 spy3 544 May 23 21:31 grocery_list.txt
spy3@spyserver1:~/spy3_data$
```

The default permissions set on the grocery\_list.txt file in octal format is 664. So Spy3 can read and write to the file. The Spy3 group ( Spy3's primary group) can read and write to the file. Others on the system can only read my file.

I want to change these permissions. I don't want others to see my grocery list! I have two ways to change the permissions for my grocery\_list.txt file. The first method is to use octal permission, the second is to use the letter representation.

#### 1. Using octal notation.

I will use chmod 660 to prevent others from even reading my grocery\_list.txt file.

```
spy3@spyserver1:~/spy3_data$ chmod 660 grocery_list.txt
spy3@spyserver1:~/spy3_data$ ls -l grocery_list.txt
-rw-rw---- 1 spy3 spy3 544 May 23 21:31 grocery_list.txt
spy3@spyserver1:~/spy3_data$
```

Now others lost their read permission on the file!

#### 2. Using letter representation

I will set the read permission back for others first so I can demonstrate using chmod with letters.

```
spy3@spyserver1:~/spy3_data$ chmod 664 grocery_list.txt
spy3@spyserver1:~/spy3_data$ ls -l grocery_list.txt
-rw-rw-r-- 1 spy3 spy3 544 May 23 21:31 grocery_list.txt
spy3@spyserver1:~/spy3_data$ _
```

Now I will use chmod 0-r grocery\_list.txt to get rid of the read permission for others.

```
spy3@spyserver1:~/spy3_data$ chmod o-r grocery_list.txt
spy3@spyserver1:~/spy3_data$ ls -l grocery_list.txt
-rw-rw---- 1 spy3 spy3 544 May 23 21:31 grocery_list.txt
spy3@spyserver1:~/spy3_data$ _
```

Now others have lost their read permission once more!

Notice that I was changing the permission as Spy3! Spy3 had no special privileges, but because they were the user owner I was able to use the account to change the permissions.

## 2. Changing the group ownership of a file:

Another vital aspect of Linux DAC is the ability to change the group ownership of a file to expand who can work with it.

1. First I will create a new group named cia. I will add Spy3 and spyuser1 to this group.

The new cia group was created using the groupadd command.

```
spy3@spyserver1:~/spy3_data$ sudo groupadd cia
[sudo] password for spy3:
spy3@spyserver1:~/spy3_data$ _
```

Now I will add Spy3 and spyuser1 to this new group.

```
[sudo] password for spy3:
spy3@spyserver1:~/spy3_data$ sudo usermod -aG cia spy3
spy3@spyserver1:~/spy3_data$ sudo usermod -aG cia spyuser1
spy3@spyserver1:~/spy3_data$
```

Verify group membership using the groups command.

```
spy3@spyserver1:~/spy3_data$ groups spyuser1
spyuser1 : spyuser1 cia
spy3@spyserver1:~/spy3_data$ groups spy3
spy3 : spy3 sudo storageusers cia
spy3@spyserver1:~/spy3_data$ _
```

2. Now that the new group is created and users are assigned to it I will change the group that owns grocery\_list.txt from Spy3 to cia.

```
spy3@spyserver1:~/spy3_data$ sudo chgrp cia grocery_list.txt
spy3@spyserver1:~/spy3_data$ ls -l grocery_list.txt
-rw-rw---- 1 spy3 cia 544 May 23 21:31 grocery_list.txt
spy3@spyserver1:~/spy3_data$ _
```

3. Test that spyuser1 has access to the file.

```
spy3@spyserver1:~/spy3_data$ su spyuser1
Password:
spyuser1@spyserver1:/home/spy3/spy3_data$ cat grocery_list.txt
Grocery List
1. Watermelon
2. Milk
3. Cereal
4. Sphagetti Noodles
5. Pasta Sauce
6. Garlic Bread
7. Ground Beef
8. Onions
9. Salad
10. Sparkling Water
spyuser1@spyserver1:/home/spy3/spy3_data$ _
```

```
GNU nano 6.2 grocery_list.txt
Grocery List
1. Watermelon
2. Milk
3. Cereal
4. Sphagetti Noodles
5. Pasta Sauce
6. Garlic Bread
7. Ground Beef
8. Onions
9. Salad
10. Sparkling Water
11. Vanilla Icecream
```

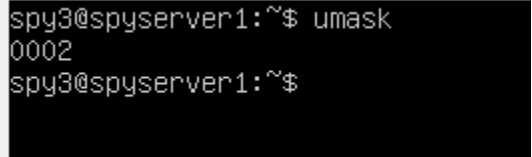
Spyuser1 can view the contents of grocery\_list.txt and can even add items to it! This is possible since they are a member of the group that owns the file.

4. Default permissions (Umask)



On Linux systems you are able to define the default directory and file permissions . These default permissions can be set on a specific user or system wide. The umask “masks” out permissions from the default permission set, basically subtracting the octet value to get rid of certain permissions.

1. Viewing current umask settings:

A terminal window with a black background and white text. The prompt is 'spy3@spyserver1:~\$'. The command 'umask' has been entered, and the output '0002' is displayed on the next line. The prompt 'spy3@spyserver1:~\$' is shown again on the third line.

```
spy3@spyserver1:~$ umask
0002
spy3@spyserver1:~$
```

Using the umask command you are able to view the currently applied umask. The current umask is 0002 so the default file permissions will now be 664 and for directories it will be 775.

2. Setting umask on a per user basis:

Setting the umask on a per user basis will mean that you are determining what the default file and directory permissions will be for a specific user. It will only apply to files and directories created by that user and not any other users on the system. To apply the umask on a per user basis you will need to set the umask value on the `/.bashrc` file or the `/.bash_profile` file.

I will set the umask value to 0006 for Spy3 by configuring the Spy3 `/.bashrc` file. This will result in 660 and 771 for files and directories respectively.

```
GNU nano 6.2 .bashrc *
# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]} && echo terminal || echo error)" "${history}'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

umask 0006

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo
^X Exit      ^R Read File ^N Replace   ^U Paste     ^J Justify   ^_ Go To Line  M-E Redo
```

Enforce this new setting and check to see if the new umask applied:

```
spy3@spyserver1:~$ source ~/.bashrc
spy3@spyserver1:~$ umask
0006
spy3@spyserver1:~$ _
```

Test new umask on directories:

```
drwxrwxr-x 2 spy3 spy3 4096 May  9 22:37 Scripts
drwxrwxr-x 2 spy3 spy3 4096 May 23 22:08 spy3_data
drwx----- 4 spy3 spy3 4096 Apr 26 23:50 spy3keys
drwxrwx--x 2 spy3 spy3 4096 May 24 22:03 test
spy3@spyserver1:~$ _
```

The new test directory now has 771 permissions. My old directories had 775 permissions since the old umask was 0002.

Test new umask on files:

```

spy3@spyserver1:~$ cd test
spy3@spyserver1:~/test$ touch umask_test.txt
spy3@spyserver1:~/test$ ls -l
total 0
-rw-rw---- 1 spy3 spy3 0 May 24 22:05 umask_test.txt
spy3@spyserver1:~/test$ _

```

Now Spy3 files will have 660 permissions, effectively blocking others from accessing them and interacting with them.

### 3. Setting umask on a systemwide basis:

I will set the system wide umask to 0006 as well to block others from snooping into directories and looking at files that don't belong to them. I will accomplish this by configuring the new system wide umask in the /etc/profile file.

```

GNU nano 6.2 /etc/bash.bashrc *
#fi

# sudo hint
if [ ! -e "$HOME/.sudo_as_admin_successful" ] && [ ! -e "$HOME/.hushlogin" ] ; then
  case " $(groups) " in *\ admin\ *|\ sudo\ *)
    if [ -x /usr/bin/sudo ]; then
      cat <<-EOF
        To run a command as administrator (user "root"), use "sudo <command>".
        See "man sudo_root" for details.
      EOF
    fi
  esac
fi

# if the command-not-found package is installed, use it
if [ -x /usr/lib/command-not-found -o -x /usr/share/command-not-found/command-not-found ]; then
  function command_not_found_handle {
    # check because c-n-f could've been removed in the meantime
    if [ -x /usr/lib/command-not-found ]; then
      /usr/lib/command-not-found -- "$1"
      return $?
    elif [ -x /usr/share/command-not-found/command-not-found ]; then
      /usr/share/command-not-found/command-not-found -- "$1"
      return $?
    else
      printf "%s: command not found\n" "$1" >&2
      return 127
    fi
  }
fi

umask 0006_

```

Test the new umask by logging in as a new user:

```
spyuser1@spyserver1:~$ umask
0006
spyuser1@spyserver1:~$
```

I logged in as spyuser1 and checked the umask settings, and they are now 0006.

## 5. Recursive permissions

Recursive permissions can be sent upon a directory so that all files and subdirectories below it have the same permissions applied for the owner, group, and others. This is possible by using the `chmod` command with the `-R` option. The only thing is the umask will override the permissions of new files created, so `chmod -R` will only work on previously created files and subdirectories.

1. I will use `chmod -R 700 test` to set the permissions on the test directory recursively and only give the owner permissions.

```
spy3@spyserver1:~$ chmod -R 700 test
spy3@spyserver1:~$ ls -l
total 16
drwxrwxr-x 2 spy3 spy3 4096 May  9 22:37 Scripts
drwxrwxr-x 2 spy3 spy3 4096 May 23 22:08 spy3_data
drwx----- 4 spy3 spy3 4096 Apr 26 23:50 spy3keys
drwx----- 2 spy3 spy3 4096 May 24 22:05 test
spy3@spyserver1:~$ _
```

2. Create files in the test directory and see the permissions.

```
spy3@spyserver1:~/test$ ls -l
total 4
-rwx----- 1 spy3 spy3  0 May 24 22:28 r-test.txt
-rw-rw---- 1 spy3 spy3  0 May 24 22:32 t1
drwx----- 2 spy3 spy3 4096 May 24 22:31 test1
-rwx----- 1 spy3 spy3  0 May 24 22:05 umask_test.txt
spy3@spyserver1:~/test$ _
```

## 6. Setuid

The setuid bit can allow a user to run an executable with the permissions of the owner rather than their own permissions. This permission is typically used for scripts and programs that are technically owned by root, but that need to be executed by regular non privileged users. There are many examples of this permission being used on Linux systems such as the following:

```
spy3@spyserver1:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 59976 Feb  6 2024 /usr/bin/passwd
spy3@spyserver1:~$
```

```

-rwsr-xr-x 1 root root      55680 Apr  9 2024 su
-rwsr-xr-x 1 root root  232416 Apr  3 2023 sudo

```

## 7. Setgid

The setgid is a special permission that can be set on linux files and directories. Setgid has many special qualities. The first quality is that it allows a user to run an executable with the permissions of the group that owns the file. The second quality is that when it is applied to a directory, new files will inherit the group that already owns the directory. Typically a file will inherit group ownership from the user that created it. But with the setgid special permissions set on a directory, new files will now inherit the group ownership associated with the directory. For example, if a file called secret1 is created by spy3 in a directory called secrets that is owned by the secrets group, the secret1 file will be owned initially by spy3 and whatever spy3's primary group is, say spies. But with the setgid bit, the secret1 file will now be owned by the spies group.

Let's look at an example of this:

I will first show what happens when a file is created in a directory without a setgid applied to it, then I will show what happens if a setgid is applied.

### 1. Directory without setgid:

```

spy3@spyserver1:~$ ls -l
total 16
drwxrwxr-x 2 spy3 spy3 4096 May  9 22:37 Scripts
drwxrwx--x 2 spy3 spy3 4096 May 24 23:00 secrets
drwxrwxr-x 2 spy3 spy3 4096 May 23 22:08 spy3_data
drwx----- 4 spy3 spy3 4096 Apr 26 23:50 spy3keys
spy3@spyserver1:~$ _

```

The directory secrets is owned by spy3 and the spy3 group.

```

spy3@spyserver1:~$ mkdir secrets
spy3@spyserver1:~$ ls
Scripts secrets spy3_data spy3keys
spy3@spyserver1:~$ cd secrets
spy3@spyserver1:~/secrets$ touch secret1
spy3@spyserver1:~/secrets$ ls -l
total 0
-rw-rw---- 1 spy3 spy3 0 May 24 23:00 secret1
spy3@spyserver1:~/secrets$ _

```

The new file secret1 is owned by spy3 and the spy3 group.

Now I will change the group ownership of the directory to the cia group.

```

spy3@spyserver1:~$ chown :cia secrets/
spy3@spyserver1:~$ ls -l
total 16
drwxrwxr-x 2 spy3 spy3 4096 May  9 22:37 Scripts
drwxrwx--x 2 spy3 cia  4096 May 24 23:00 secrets
drwxrwxr-x 2 spy3 spy3 4096 May 23 22:08 spy3_data
drwx----- 4 spy3 spy3 4096 Apr 26 23:50 spy3keys
spy3@spyserver1:~$ _

```

Again, I will create a new file to demonstrate that the cia group will not own it.

```

spy3@spyserver1:~$ cd secrets/
spy3@spyserver1:~/secrets$ touch secret2
spy3@spyserver1:~/secrets$ ls -l
total 0
-rw-rw---- 1 spy3 spy3 0 May 24 23:00 secret1
-rw-rw---- 1 spy3 spy3 0 May 24 23:04 secret2
spy3@spyserver1:~/secrets$ _

```

The new secret2 file is still owned by spy3 and the spy3 group, and not the cia group.

## 2. Directory with setgid:

I will apply the setgid on the secrets directory now.

```

spy3@spyserver1:~$ chmod g+s secrets
spy3@spyserver1:~$ ls -l
total 16
drwxrwxr-x 2 spy3 spy3 4096 May  9 22:37 Scripts
drwxrws--x 2 spy3 cia  4096 May 24 23:04 secrets
drwxrwxr-x 2 spy3 spy3 4096 May 23 22:08 spy3_data
drwx----- 4 spy3 spy3 4096 Apr 26 23:50 spy3keys
spy3@spyserver1:~$

```

Now I will create a new secret3 file, and observe the group ownership after.

```

spy3@spyserver1:~/secrets$ touch secret3
spy3@spyserver1:~/secrets$ ls -l
total 0
-rw-rw---- 1 spy3 spy3 0 May 24 23:00 secret1
-rw-rw---- 1 spy3 spy3 0 May 24 23:04 secret2
-rw-rw---- 1 spy3 cia  0 May 24 23:10 secret3
spy3@spyserver1:~/secrets$

```

Now the new secret3 file inherits the cia group ownership from the secrets directory!

## 8. Sticky bit

The sticky bit for a Linux system is a special permission that prevents the deletion and renaming of files in a directory. When a sticky bit is applied to the directory only the file owner can delete it. The group owner and its members and others cannot delete the file. The only person that can delete the file is the directory owner and the file owner. The sticky bit is useful when you have a directory that is used by multiple users and you want to prevent users from deleting each other's work either accidentally or on purpose..

1. Create a directory owned by the cia group.

```
spy3@spyserver1:~$ mkdir sticky_test
spy3@spyserver1:~$ chown :cia sticky_test/
spy3@spyserver1:~$ ls -l
total 20
drwxrwxr-x 2 spy3 spy3 4096 May  9 22:37 Scripts
drwxrws--x 2 spy3 cia  4096 May 24 23:10 secrets
drwxrwxr-x 2 spy3 spy3 4096 May 23 22:08 spy3_data
drwx----- 4 spy3 spy3 4096 Apr 26 23:50 spy3keys
drwxrwx--x 2 spy3 cia  4096 May 30 20:23 sticky_test
spy3@spyserver1:~$
```

2. I will now apply the sticky bit to the directory. I will be using the letter representation to set the sticky bit on it.

```
spy3@spyserver1:/tmp$ mkdir sticky_test
spy3@spyserver1:/tmp$ chown :cia sticky_test/
spy3@spyserver1:/tmp$ chmod +t sticky_test/
spy3@spyserver1:/tmp$ ls -l
total 24
drwx----- 3 root root 4096 May 30 20:22 snap-private-tmp
drwxrwx--t 2 spy3 cia  4096 May 30 20:43 sticky_test
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-ModemManag
er.service-ek0gIy
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-systemd-lo
gind.service-kX2j18
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-systemd-re
solved.service-Coonbe
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-systemd-ti
mesyncd.service-exIjuE
spy3@spyserver1:/tmp$ _
```

Notice that the permissions for the directory are now drwxrwx-t. The t represents the sticky bit permission. To set the sticky bit using numeric representation you would use a 1 and the other permissions. For example, I can do 1770.

3. Now that the sticky bit is set, I will test it by creating two files. One will be created by spyuser1, and the other by spy3.

Spy3:

```

spy3@spyserver1:/tmp$ cd sticky_test/
spy3@spyserver1:/tmp/sticky_test$ touch spy3_test.txt
spy3@spyserver1:/tmp/sticky_test$ chown :cia spy3_test.txt
spy3@spyserver1:/tmp/sticky_test$ ls -l
total 0
-rw-rw---- 1 spy3 cia 0 May 30 20:44 spy3_test.txt
spy3@spyserver1:/tmp/sticky_test$ _

```

Spyuser1:

```

spyuser1@spyserver1:/tmp$ cd sticky_test/
spyuser1@spyserver1:/tmp/sticky_test$ touch spyuser1_test.txt
spyuser1@spyserver1:/tmp/sticky_test$ chown :cia spyuser1_test.txt
spyuser1@spyserver1:/tmp/sticky_test$ ls -l
total 0
-rw-rw---- 1 spy3      cia 0 May 30 20:44 spy3_test.txt
-rw-rw---- 1 spyuser1 cia 0 May 30 20:46 spyuser1_test.txt
spyuser1@spyserver1:/tmp/sticky_test$

```

Now spy3 will try to delete and rename spyuser1's file, and vice-versa.

1. Spyuser1 will try to delete spy3's file.

```

spyuser1@spyserver1:/tmp/sticky_test$ ls -l
total 0
-rw-rw---- 1 spy3      cia 0 May 30 20:44 spy3_test.txt
-rw-rw---- 1 spyuser1 cia 0 May 30 20:46 spyuser1_test.txt
spyuser1@spyserver1:/tmp/sticky_test$ rm spy3_test.txt
rm: cannot remove 'spy3_test.txt': Operation not permitted
spyuser1@spyserver1:/tmp/sticky_test$

```

Spyuser1 could not delete spy3's file even though they are both in the same group! The sticky bit works.

Spyuser1 will now try to rename spy3's file.

```

spyuser1@spyserver1:/tmp/sticky_test$ mv spy3_test.txt spy3.txt
mv: cannot move 'spy3_test.txt' to 'spy3.txt': Operation not permitted
spyuser1@spyserver1:/tmp/sticky_test$ _

```

Renaming was not permitted as well.

2. Spy3 will try to delete spyuser1's file.



```

spy3@spyserver1:/tmp/sticky_test$ ls -l
total 0
-rw-rw---- 1 spy3      cia 0 May 30 20:44 spy3_test.txt
-rw-rw---- 1 spyuser1 cia 0 May 30 20:46 spyuser1_test.txt
spy3@spyserver1:/tmp/sticky_test$ rm spyuser1_test.txt
spy3@spyserver1:/tmp/sticky_test$ ls -l
total 0
-rw-rw---- 1 spy3 cia 0 May 30 20:44 spy3_test.txt
spy3@spyserver1:/tmp/sticky_test$

```

It worked for spy3! This is because spy3 is the owner of the directory. Sticky bits still permit the directory owner to delete files in the directory.

```

spy3@spyserver1:/tmp$ ls -l
total 24
drwx----- 3 root root 4096 May 30 20:22 snap-private-tmp
drwxrwx--t 2 spy3 cia 4096 May 30 20:57 sticky_test
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-ModemManager.service-ek0gIy
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-systemd-logind.service-kX2j18
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-systemd-resolved.service-Coonbe
drwx----- 3 root root 4096 May 30 20:22 systemd-private-e6f6a96f08c647828b304667f65000eb-systemd-timesyncd.service-exIjuE
spy3@spyserver1:/tmp$ _

```

Spy3 is the directory owner for sticky\_test!

## 9. ACLs

Acls are a useful permission mechanism that allows granular permissions to be granted on files and directories. They can help you deal with one off cases such as when you need to allow a non group member access and write to a file. Acls provide you with flexibility that goes above the regular user owner, group, and others permission scheme. Now it is time to see how acls are implemented on a Linux system.

1. Check if acls are supported:

I tried simply running the two main acl commands, setfacl and getfacl. Apparently on my Ubuntu server these commands are not installed so I will install them.

```

spy3@spyserver1:~$ setfacl
Command 'setfacl' not found, but can be installed with:
sudo apt install acl
spy3@spyserver1:~$ getfacl
Command 'getfacl' not found, but can be installed with:
sudo apt install acl
spy3@spyserver1:~$

```

2. Install acl onto my Ubuntu server.

```

no VM guests are running outdated hypervisor (qemu) binaries on this host.
spy3@spyserver1:~$ sudo apt install acl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
acl is already the newest version (2.3.1-1).
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
spy3@spyserver1:~$ _

```

Now the acl commands are installed on the server.2

3. Verify that acls work now.

```

spy3@spyserver1:~$ ls
Scripts secrets spy3_data spy3keys
spy3@spyserver1:~$ cd secrets/
spy3@spyserver1:~/secrets$ ls
secret1 secret2 secret3
spy3@spyserver1:~/secrets$ getfacl secret1
# file: secret1
# owner: spy3
# group: spy3
user::rw-
group::rw-
other::---
spy3@spyserver1:~/secrets$

```

I used getfacl on the secret1 file, and the command worked so everything is working well.

## ACL commands:

1. Getfacl: Getfacl is used to view the acls applied on a file. Here are some options used with the command:

Option	Purpose
-t	Displays output in a tabular format.
-R	Lists acls of files and directories recursively.
-c	Don't display header.
-s	Skips files with only the base entries ( owner, group, and

	others).
--	----------

2. Setfacl: Used to set the acl permissions on a file or directory. Here are some options

Option	Purpose
-m	Modify the acl of a file or directory.
-x	Removes a specific acl entry.
-b	Removes all acl entries.
-k	Removes default acl.
-d	Sets default acl entry.
-M	Sets mask explicitly.
-n	Do not recalculate effective rights mask.
-N	Grants no rights or permissions. Disables effective rights mask calculation.

### How do ACLs work?

1. First of all there are three main sets of permissions when dealing with acls: regular unix permissions, acl permissions, and default mask permissions. The regular unix permissions are the regular owner, group, and others permissions. Acl permissions are the granular permissions granted onto a user, group, or others outside of the traditional unix permissions. Lastly the default mask is the default acl permissions applied to new files in a directory.
2. In order to apply acls you need to use setfacl. When using setfacl you use the following syntax: setfacl -m user-type:name:permissions filename. To set default acl permissions you would use setfacl -d user-type:name:permissions directory path/name. You can apply acls recursively by using the -dR option.
3. The acl permissions granted to the user and group in question will be the effective permissions. The effective permissions take into account the default mask for the acl.

Effective permissions examples:

## ACL Permissions

### Example #1

Default unix permissions: `rw-rw-r-x`

ACL permissions: `user:joe:rw-`

Mask: `r-x`

What permissions will Joe Have?

Joe will only have `r--`

### Example #1

Default unix permissions: `rw-rwxr-x`

ACL permissions: `group:managers:rwx`

Mask: `r-x`

What permissions will managers have?

Managers will only have `r-x`!

## ACLs applied:

1. I will apply an acl to a file in a directory under /tmp that is owned by spy3 and the cia group. The acl will allow spymaster19 to read and execute the file.

1a.

```
spy3@spyserver1:/tmp$ mkdir acl_test
spy3@spyserver1:/tmp$ ls
acl_test
snap-private-tmp
systemd-private-270daddb509746ffa358ff3aeebf51f8-ModemManager.service-4bN9EJ
systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-logind.service-nP1Q3Q
systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-resolved.service-hI2jKA
systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-timesyncd.service-ExI9ag
spy3@spyserver1:/tmp$ ls -l
total 24
drwxrwx--x 2 spy3 spy3 4096 Jun  2 20:07 acl_test
drwx----- 3 root root 4096 Jun  2 19:58 snap-private-tmp
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-ModemManager.service-4bN9EJ
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-logind.service-nP1Q3Q
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-resolved.service-hI2jKA
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-timesyncd.service-ExI9ag
spy3@spyserver1:/tmp$ chown :cia acl_test/
spy3@spyserver1:/tmp$ ls -l
total 24
drwxrwx--x 2 spy3 cia 4096 Jun  2 20:07 acl_test
drwx----- 3 root root 4096 Jun  2 19:58 snap-private-tmp
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-ModemManager.service-4bN9EJ
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-logind.service-nP1Q3Q
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-resolved.service-hI2jKA
drwx----- 3 root root 4096 Jun  2 19:58 systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-timesyncd.service-ExI9ag
spy3@spyserver1:/tmp$ _
```

Created the new directory and made sure that it is owned by spy3 and the cia group.

1b.

```
spy3@spyserver1:/tmp/acl_test$ touch acl_test1.txt
spy3@spyserver1:/tmp/acl_test$ ls -l
total 0
-rw-rw---- 1 spy3 spy3 0 Jun  2 20:09 acl_test1.txt
spy3@spyserver1:/tmp/acl_test$ _
```

Created a test file in the /tmp/acl\_test directory.

1c.

```

spy3@spyserver1:/tmp/acl_test$ setfacl -m u:spymaster19:rx acl_test1.txt
spy3@spyserver1:/tmp/acl_test$ getfacl acl_test1.txt
# file: acl_test1.txt
# owner: spy3
# group: spy3
user::rw-
user:spymaster19:r-x
group::rw-
mask::rwx
other:---
spy3@spyserver1:/tmp/acl_test$

```

I set the acl on acl\_test1.txt so that spymaster19 has read and executed permissions on the file. I then used getfacl so that I can review the acl for the file.

1d. Test the acl as spymaster19. I will first write some stuff to the file as spy3, and test the read permissions for spymaster19.

```

spy3@spyserver1:/tmp/acl_test$ echo "Testing the acl for this file!" >> acl_test1.txt
spy3@spyserver1:/tmp/acl_test$ cat acl_test1.txt
Testing the acl for this file!
spy3@spyserver1:/tmp/acl_test$ _

```

```

spy3@spyserver1:/tmp/acl_test$ su spymaster19
Password:
spymaster19@spyserver1:/tmp/acl_test$ cat acl_test1.txt
Testing the acl for this file!
spymaster19@spyserver1:/tmp/acl_test$ _

```

Spymaster19 can read the acl\_test1.txt file! The acl works!.

2. Testing acls with groups. I will create a new group with a spymaster19 and a new user, ciachief as the members. It will be called the boss group. This group should have rwx on the acl\_test directory (meaning recursively).

2a. Create the new user and group. Add users to the group.

```

spy3@spyserver1:/tmp/acl_test$ sudo groupadd boss
groupadd: group 'boss' already exists
spy3@spyserver1:/tmp/acl_test$ usermod -aG boss spymaster19
usermod: Permission denied.
usermod: cannot lock /etc/passwd; try again later.
spy3@spyserver1:/tmp/acl_test$ !!
usermod -aG boss spymaster19
usermod: Permission denied.
usermod: cannot lock /etc/passwd; try again later.
spy3@spyserver1:/tmp/acl_test$ sudo usermod -aG boss spymaster19
spy3@spyserver1:/tmp/acl_test$ groups spymaster19
spymaster19 : spymaster19 adm cdrom sudo dip plugdev lxd boss
spy3@spyserver1:/tmp/acl_test$

```

```

spy3@spyserver1:/tmp/acl_test$ adduser ciachief
adduser: Only root may add a user or group to the system.
spy3@spyserver1:/tmp/acl_test$ sudo adduser ciachief
Adding user `ciachief' ...
Adding new group `ciachief' (1006) ...
Adding new user `ciachief' (1003) with group `ciachief' ...
Creating home directory `/home/ciachief' ...
Copying files from `/etc/skel' ...
New password:
BAD PASSWORD: The password is shorter than 10 characters
New password:
BAD PASSWORD: The password contains less than 4 character classes
New password:
BAD PASSWORD: The password is shorter than 10 characters
passwd: Have exhausted maximum number of retries for service
passwd: password unchanged
Try again? [y/N] y
New password:
Retype new password:
Sorry, passwords do not match.
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for ciachief
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] Y
spy3@spyserver1:/tmp/acl_test$

```

```

spy3@spyserver1:/tmp/acl_test$ sudo usermod -aG boss ciachief
spy3@spyserver1:/tmp/acl_test$ groups ciachief
ciachief : ciachief boss
spy3@spyserver1:/tmp/acl_test$ _

```

I created the new ciachief user with password HelloW0rld! and then added them to the boss group.

2b. Set the acl on the acl\_test directory. I will grant the boss group rwx on the whole acl\_test directory recursively.

```

spy3@spyserver1:/tmp/acl_test$ cd
spy3@spyserver1:~$ cd /tmp
spy3@spyserver1:/tmp$ ls
acl_test
snap-private-tmp
systemd-private-270daddb509746ffa358ff3aeebf51f8-ModemManager.service-4bN9EJ
systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-logind.service-nP1Q3Q
systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-resolved.service-hI2jKA
systemd-private-270daddb509746ffa358ff3aeebf51f8-systemd-timesyncd.service-ExI9ag
spy3@spyserver1:/tmp$ setfacl -R -m g:boss:rwx acl_test/
spy3@spyserver1:/tmp$ getfacl acl_test/
# file: acl_test/
# owner: spy3
# group: cia
user::rwx
group::rwx
group:boss:rwx
mask::rwx
other::--x

spy3@spyserver1:/tmp$ _

```

Now the acl is set and the boss group can now have rwx permissions on the acl\_test directory files like acl\_test1.txt!

2c. Test the acl permissions with the ciachief user, who is in the boss group. Ciachief should be able to write to acl\_test1.txt, cd into the acl\_test directory, and should also be able to read it as well.

```

ciachief@spyserver1:/tmp$ cd acl_test/
ciachief@spyserver1:/tmp/acl_test$ ls -l
total 4
-rw-rwx---+ 1 spy3 spy3 64 Jun  2 21:09 acl_test1.txt
ciachief@spyserver1:/tmp/acl_test$ _

```

```

GNU nano 6.2                                acl_test1.txt *
Testing the acl for this file!
Ciachief can write to this file!_

```



## 10. Extended file permissions ( i and a)

Linux extended file permissions are used as extra permissions that can be granted on files. The two extended file permissions are i which stands for immutable and a for append only. Immutable means that the file cannot be changed. A file with this attribute cannot be deleted, renamed, written to, have changes for permissions and ownership, or have hard links created for it. The append only, a, means that you can only add content to a file, and cannot change any data written to the file beforehand. The immutable attribute is useful for protecting the integrity and availability of important files such as log files. Since no data can be removed or added to a file with the immutable attribute the file data will remain unchanged, preserving its integrity. Also, since the file cannot be deleted or renamed, the availability of the file will be kept since no one can try to hide or remove the file from the system.

1. Applying immutable permission onto the auth.log file.

```
root@spyserver1:/var/log# chattr +i auth.log
root@spyserver1:/var/log# lsattr auth.log
----i-----e----- auth.log
root@spyserver1:/var/log#
```

To set the immutable attribute you use the chattr command and use +i and the file name. To view the extended file permissions you can use lsattr on a file. As you can see auth.log now has the i permission.

2. Test the limits of the i attribute.
  - 2a. Try changing permissions on the file

```
root@spyserver1:/var/log# chmod 777 auth.log
chmod: changing permissions of 'auth.log': Operation not permitted
root@spyserver1:/var/log#
```

- 2b. Try adding data

```
GNU nano 6.2                                auth.log
Jun  9 20:21:31 spyserver1 login[719]: pam_unix(login:session): session opened for user spy3(uid=1000) by spy3.
Jun  9 20:21:31 spyserver1 systemd-logind[697]: New session 1 of user spy3.
Jun  9 20:21:31 spyserver1 systemd: pam_unix(systemd-user:session): session opened for user spy3(uid=1000) by spy3.
Jun  9 20:22:10 spyserver1 sudo:      spy3 : TTY=tty1 ; PWD=/root ; USER=root ; COMMAND=/bin/bash
Jun  9 20:22:10 spyserver1 sudo: pam_unix(sudo-i:session): session opened for user root(uid=0) by spy3.

[ File 'auth.log' is unwritable ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^N Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo
```

## 2c. Try removing data

```
GNU nano 6.2                                auth.log
Jun  9 20:21:31 spyserver1 login[719]: pam_unix(login:session): session opened for user spy3(uid=1000) by spy3.
Jun  9 20:21:31 spyserver1 systemd-logind[697]: New session 1 of user spy3.
Jun  9 20:21:31 spyserver1 systemd: pam_unix(systemd-user:session): session opened for user spy3(uid=1000) by spy3.
Jun  9 20:22:10 spyserver1 sudo:      spy3 : TTY=tty1 ; PWD=/root ; USER=root ; COMMAND=/bin/bash
Jun  9 20:22:10 spyserver1 sudo: pam_unix(sudo-i:session): session opened for user root(uid=0) by spy3.

[ File 'auth.log' is unwritable ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^N Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo
```

## 2d. Try deleting the file

```
root@spyserver1:/var/log# rm auth.log
rm: cannot remove 'auth.log': Operation not permitted
root@spyserver1:/var/log#
```

2e. Try changing the owner

```
root@spyserver1:/var/log# chown spy3 auth.log
chown: changing ownership of 'auth.log': Operation not permitted
root@spyserver1:/var/log#
```

2f. Try renaming the file

```
root@spyserver1:/var/log# mv auth.log login.log
mv: cannot move 'auth.log' to 'login.log': Operation not permitted
root@spyserver1:/var/log# _
```

3. Removing the immutable attribute.

```
root@spyserver1:/var/log# chattr -i auth.log
root@spyserver1:/var/log# lsattr auth.log
-----e----- auth.log
root@spyserver1:/var/log#
```

4. Adding the append only attribute to a file.

4a. Create a new file, secrets\_list.txt

```
spy3@spyserver1:~/secrets$ touch secrets_list.txt
spy3@spyserver1:~/secrets$
```

4b. Set the a attribute on the file

```
spy3@spyserver1:~/secrets$ sudo chattr +a secrets_list.txt
[sudo] password for spy3:
spy3@spyserver1:~/secrets$ lsattr secrets_list.txt
-----a-----e----- secrets_list.txt
spy3@spyserver1:~/secrets$ _
```

4d. Test the append only attribute.

```
spy3@spyserver1:~/secrets$ echo "Testing append only attribute" > secrets_list.txt
-bash: secrets_list.txt: Operation not permitted
spy3@spyserver1:~/secrets$ echo "Testing append only attribute" >> secrets_list.txt
spy3@spyserver1:~/secrets$ _
```

```
GNU nano 6.2 secrets_list.txt *
Testing append only attribut_

[ Error writing secrets_list.txt: Operation not permitted ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^_ Replace    ^U Paste      ^J Justify    ^_ Go To Line M-E Redo
```

I can't delete past data in the file!, I can only add new data using the append >> symbols!

```
GNU nano 6.2                                secrets_list.txt *
Testing append only attribute
Adding stuff using nano!_

[ Error writing secrets_list.txt: Operation not permitted ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo
```

```
spy3@spyserver1:~/secrets$ echo "Appending data can only be done using >>" >> secrets_list.txt
spy3@spyserver1:~/secrets$ cat secrets_list.txt
Testing append only attribute
Appending data can only be done using >>
spy3@spyserver1:~/secrets$ _
```