

# Security by File Permissions in Linux

**Dr. Vimal Baghel, Assistant Professor, SCSET, BU**

# Outline

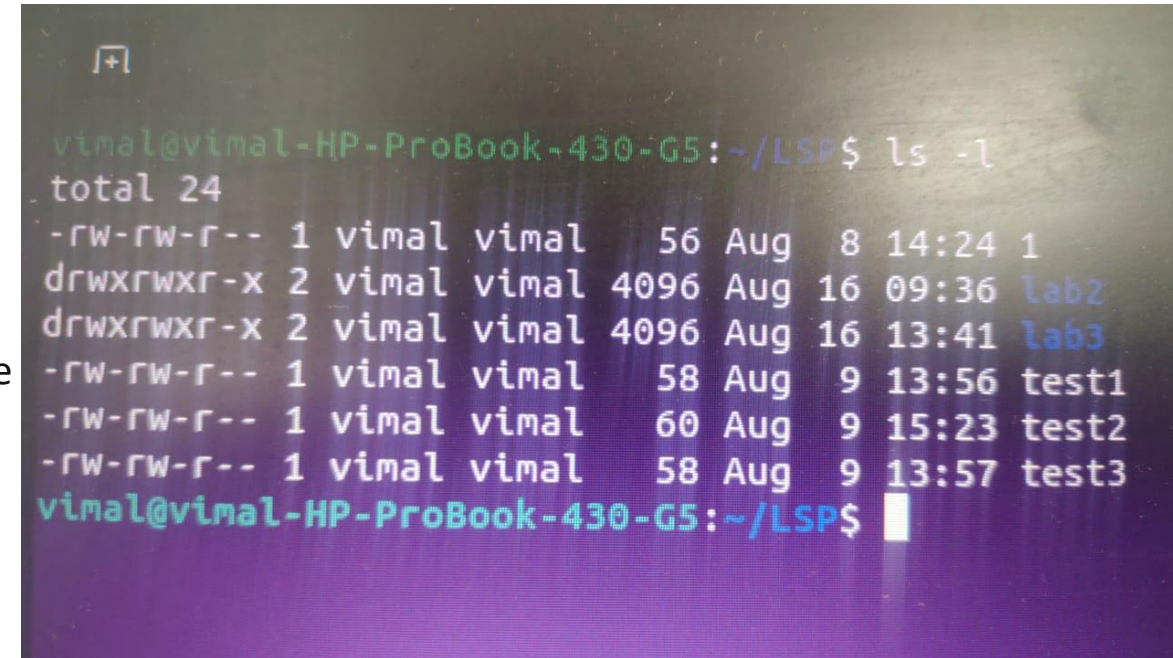
- Understanding Linux file permissions
- How do you view Linux file permissions?
- How do you read file permissions?
- What are octal values?
- What do Linux file permissions actually do?
- How do directory permissions work?
- How do you modify Linux file permissions?
- What are special file permissions?
- Q & A

# Understanding Linux file Permissions

- File permissions are core to the security model used by Linux systems.
  - determine who can access files and directories on a system and how.
- **Permission Groups**
  - Owner (u)
  - Group (g)
  - Others (o)
  - All Users (a)
- **Permission Types**
  - Read (r)
  - Write (w)
  - Execute (x)

# Viewing the Permissions

- \$ ls -l, output is in the format
  - ***\_rwxrwxrwx. 1 owner:group***
    - File type: -
    - Permission settings: *rw-r--r--*
    - Extended attributes: dot (.)
    - Numeric value (e.g.) 1: Number of hard links to the file
    - User owner: *vimal*
    - Group owner: *vimal*
- What are octal values?
  - $r = 4, w = 2, x = 1$



```
vimal@vimal-HP-ProBook-430-G5:~/LSP$ ls -l
total 24
-rw-rw-r-- 1 vimal vimal  56 Aug  8 14:24 1
drwxrwxr-x 2 vimal vimal 4096 Aug 16 09:36 lab2
drwxrwxr-x 2 vimal vimal 4096 Aug 16 13:41 lab3
-rw-rw-r-- 1 vimal vimal  58 Aug  9 13:56 test1
-rw-rw-r-- 1 vimal vimal  60 Aug  9 15:23 test2
-rw-rw-r-- 1 vimal vimal  58 Aug  9 13:57 test3
vimal@vimal-HP-ProBook-430-G5:~/LSP$
```



# What happens when we interact with a file?

- When the system is looking at a file's permissions to determine what information to provide you when you interact with a file, it runs through a series of checks:
  1. It first checks to see whether you are the user that owns the file.
  2. If you are not the user that owns the file, next your group membership is validated to see whether you belong to the group that matches the group owner of the file
  3. "Others" permissions are applied when the account interacting with the file is neither the user owner nor in the group that owns the files.

The three fields are mutually exclusive: You can not be covered under more than one of the fields of permission settings on a file.

# Principle of Least Privilege

Only grant users, applications, and processes as much access as they need, and no more!

- **Proper example:** UserA needs to be able to read file1, so they are granted the read permission but not the write permission.
- **Improper example:** UserA needs to be able to read file2, but they are granted read, write, and execute permissions and could potentially change the file.
- **Proper example:** UserA needs to perform standard system activities, such as creating files associated with their own job, and does not need to perform system administrative tasks, so the user logs on with a standard user account.
- **Improper example:** UserA needs to perform standard system activities, such as creating files associated with their own job, and does not need to perform system administrative tasks, so the user logs on with the root user account.

# Recognizing Access Levels

## Files

- **Read:** The ability to access and view the contents of a file.
- **Write:** The ability to save changes to a file.
- **Execute:** The ability to run a script, program, or other software file.

## Directories

- **Read:** The ability to list the contents of a directory.
- **Write:** The ability to create, rename, and delete files in a directory. Requires the execute attribute to also be set.
- **Execute:** The ability to access a directory, execute a file from that directory, or perform a task on that directory.

# Recognizing Access Identities

## User (u)

- Single account designated as the owner.
- By default, the resource creator.

## Group (g)

- Single group associated with the file or directory.

## Others (o)

- All users and groups who are not the designated user (owner) or associated group.



# Display Permissions

```
student@ubuntu20:/projects$ ls -l
total 4
drwxr-xr-x 2 root      root      4096 Dec  9 11:11 old-projects
-rwxr--r-- 1 root      root         0 Dec  9 11:11 project1.txt
-rw-rw---- 1 student5  2022projects 0 Dec  9 11:11 project2.txt
-r----- 1 student9  2022projects 0 Dec  9 11:11 project3.txt
student@ubuntu20:/projects$
```

# When Permissions Are Important?

- **home directories**– drwx\_\_\_\_\_ (700)
  - \$ chmod 700 /home/user1.
- **bootloader configuration files**– If you decide to implement password to boot specific operating systems then you will want to remove read and write permissions from the configuration file from all users but root. To do you can change the permissions of the file to 700.
- **system and daemon configuration files**– It is very important to restrict rights to system and daemon configuration files to restrict users from editing the contents, it may not be advisable to restrict read permissions, but restricting write permissions is a must. (644).
- **firewall scripts** – It may not always be necessary to block all users from reading the firewall file, but it is advisable to restrict the users from writing to the file. (700)

# Key Demonstration: Display Permissions



1. Sign in to at least one system (either RH or Debian-based) and then demonstrate how to display permissions. Explain the output.
2. Be in the logged-in user's home directory.
3. If necessary, create two-three text files by using the touch command: `touch fileA fileB`
4. Run `ls -l` to display permissions
5. Explain each permission field for the user, group, and others
6. Identify the owner and group field

# Advanced Permissions

- **\_** – no special permissions
- **d** – directory
- **l** – The file or directory is a symbolic link
- **s** – This indicated the setuid/setgid permissions.
- **t** – This indicates the sticky bit permissions.

# Interpret Permissions Strings (slide 1/4)

## First field

- File ( - ) or Directory (d)

```
student@ubuntu20:/projects$ ls -l
total 4
drwxr-xr-x 2 root      root      4096 Dec  9 11:11 old-projects
-rwxrw-r-- 1 root      root           0 Dec  9 11:11 project1.txt
-rw-rw---- 1 student5  2022projects  0 Dec  9 11:11 project2.txt
-r-----  1 student9  2022projects  0 Dec  9 11:11 project3.txt
student@ubuntu20:/projects$
```



# Interpret Permissions Strings (slide 2/4)

## Second, Third, and Fourth Fields

- Permissions for the User identity
- Read (r), Write (w), Execute (x), or none ( - )

```
student@ubuntu20:/projects$ ls -l
total 4
drwxr-xr-x 2 root      root      4096 Dec  9 11:11 old-projects
-rwxrw-r-- 1 root      root         0 Dec  9 11:11 project1.txt
-rw-rw---- 1 student5  2022projects 0 Dec  9 11:11 project2.txt
-r----- 1 student9  2022projects 0 Dec  9 11:11 project3.txt
student@ubuntu20:/projects$
```

# Interpret Permissions Strings (slide 3/4)

## Fifth, Sixth, and Seventh Fields

- Permissions for the Group identity
- Read (r), Write (w), Execute (x), or none ( - )

```
student@ubuntu20:/projects$ ls -l
total 4
drwxr-xr-x 2 root      root      4096 Dec  9 11:11 old-projects
-rwxrw-r-- 1 root      root           0 Dec  9 11:11 project1.txt
-rw-rw---- 1 student5  2022projects  0 Dec  9 11:11 project2.txt
-r----- 1 student9  2022projects  0 Dec  9 11:11 project3.txt
student@ubuntu20:/projects$
```

# Interpret Permissions Strings (slide 4/4)

## Eighth, Ninth, and Tenth Fields

- Permissions for the Others identity
- Read (r), Write (w), Execute (x), or none ( - )

```
student@ubuntu20:/projects$ ls -l
total 4
drwxr-xr-x 2 root      root      4096 Dec  9 11:11 old-projects
-rwxrw-r-- 1 root      root           0 Dec  9 11:11 project1.txt
-rw-rw---- 1 student5  2022projects 0 Dec  9 11:11 project2.txt
-r----- 1 student9  2022projects 0 Dec  9 11:11 project3.txt
student@ubuntu20:/projects$
```

# Set Permissions with Absolute and Symbolic Modes

## Absolute mode

- Uses octal values
- Ex: `chmod 764 file1`

## Symbolic mode

- Uses characters
- Ex: `chmod u+r file1`

set permissions



Note that you must be able to interpret both modes!

# Absolute Mode

## Octal Values

- Read = 4
- Write = 2
- Execute = 1
- Displayed with three fields, with permissions values added.

## Example

- user has rwx, or  $4+2+1 = 7$
- group has r-x, or  $4+0+1 = 5$
- other has ---, or  $0+0+0 = 0$
- Permissions: 750 for ugo
- Syntax to set this access level:
  - `chmod 750 file1`



# Symbolic Mode

## Characters:

- Read = r, write = w, execute = x
- User = u, group = g, others = o

## Operators:

- + grants a permission,
- - removes a permission
- = sets a permission

## • Examples:

- Give r to o for file1:  
`chmod o+r file1`
- Remove r for o for file1:  
`chmod o-r file`
- Give rw to go for file1:  
`chmod go+rw file1`

# Key Demonstration: Absolute and Symbolic Modes (slide 1)



Use `chmod` to demonstrate absolute and symbolic modes. Display the permissions changes using `ls -l` and interpret the new permissions.

1. Be in the logged-in user's home directory
2. Create a directory in the logged-in user's home folder named Permissions: `mkdir Permissions`
3. Change into the Permissions directory: `cd Permissions`
4. Create dirA and file1: `mkdir dirA` then run `touch file1`
5. Use `ls -l` to display current permissions

*(continued on next slide)*



*(continued from previous slide)*

6. Use absolute mode to see rwx for ugo on file1: `chmod 777 file1`

7. Display the new permissions: `ls -l`

8. Use absolute mode to set rwx for the user and group, and no access for others on dirA: `chmod 770 dirA`

9. Display the new permissions: `ls -l`

# Use the chown Command

## **Example 1:** Change the owner but not the group:

- `chown newowner filename`

## **Example 2:** Change the owner and the group:

- `chown newowner:newgroup filename`

## **Example 3:** Change the group but not the owner:

- `chown :newgroup filename`

# Modifying the Permissions

- Let file1 has the permissions as `_rw_rw_rw`. We want to remove the read and write permissions from the all-users: \$ ***chmod a-rw file1***
- To add the permissions again: \$ ***chmod a+rw file1***
- Using Binary References to Set permissions: \$ ***chmod 640 file1***
- *Other Ways:*
  - \$ ***chmod ug+rw example.txt***
  - \$ ***chmod o+r example2.txt***



# Owners and Groups

- Syntax: \$ ***chown owner:group filename***,
  - ***Example: \$ chown user1:family file1.***
- the **chgrp** command can be used to change the group ownership of a file.

# Key Demonstration: Using `chown` and `chgrp`



Sign in to at least one system (either RH or Debian-based) and then use `chown` to demonstrate changing owners, groups, and both on a file. Demonstrate the `chgrp` command, too. Use `ls -l` to display the changes.

1. Be in the logged-in user's home directory.
2. If necessary, create two or three text files by using the `touch` command: `touch fileA fileB`
3. If necessary, create a user account and a group for demonstration purposes: `useradd {USERA}` and then run `groupadd {GROUP1}`
4. Demonstrate the use of `chown` by changing the associated file owner and group: `chown {USERA}:{GROUP1} fileA`
5. Demonstrate the use of `chgrp` by changing the associated group: `chgrp {GROUP1} fileB`

# File Attributes

- Apart from the file mode bits that control user and group read, write and execute permissions, several file systems support file attributes that enable further customization of allowable file operations.
- The e2fsprogs package contains the programs lsattr(1) and chattr(1) that list and change a file's attributes, respectively.
- These are a few useful attributes. Not all filesystems support every attribute.
  - **a - append only:** File can only be opened for appending.
  - **c - compressed:** Enable filesystem-level compression for the file.
  - **e – extent:** the file is using extents for mapping the blocks on disk.
  - **i - immutable:** Cannot be modified, deleted, renamed, linked to. Can only be set by root.
  - **j - data journaling:** Use the journal for file data writes as well as metadata.
  - **m - no compression:** Disable filesystem-level compression for the file.
  - **A - no atime update:** The file's atime will not be modified.
  - **C - no copy on write:** Disable copy-on-write, for filesystems that support it.

# Use Attribute Management Commands (slide 1)

- To display attributes for file1:

```
lsattr file1
```

```
student@ubuntu20:~/Documents$ lsattr file2  
----i-----e----- file2  
student@ubuntu20:~/Documents$
```

- To set the immutable flag attribute for file1:

```
chattr +i file1
```

```
student@ubuntu20:~/Documents$ sudo chattr +i file3
student@ubuntu20:~/Documents$ lsattr file3
----i-----e----- file3
student@ubuntu20:~/Documents$
```



# Troubleshooting Permissions

## Is the user a member of the sales group?

- Confirm the user's membership in sales with the group or id commands
- Confirm the permissions applied to sales for the file by using ls -l

## Are the permissions set correctly?

- Display permissions with ls -l
- Check permissions of file
- Are permissions set recursively from parent directory?
- Use su to test access
- Reapply permissions, being careful of your absolute or symbolic mode syntax



## Review Activity: Standard Linux Permissions

1. How does the principle of least privilege help mitigate threats and mistakes?
2. What octal value is used in absolute mode to set permissions at all access for all identities?
3. Write the command by using symbolic mode that removes the read permission from others for fileA without impacting other permissions.
4. Interpret the results of the following command: `chown -R USERA:sales dirA`

# Understand SUID and SGID

## SUID

- Allows the specified user to have similar permissions to the owner without being the owner.
- Files or commands can be executed as the owner.

## SGID

- Allows the group associated with a directory to be inherited by files created in the directory.

# What are special file permissions?

- The **setuid/setgid** permissions are used **to tell the system to run an executable as the owner with the owner's permissions.**
  - **Be careful** using setuid/setgid bits in permissions. If you incorrectly assign permissions to a file owned by root with the setuid/setgid bit set, then you can open your system to intrusion.
- You can only assign the setuid/setgid bit by explicitly defining permissions. The character for the setuid/setgid bit is **s**.
- So do set the setuid/setgid bit on file2.sh you would issue the command \$ ***chmod g+s file2.sh***.

# Key Demonstration: Configure with SGID



Sign in to at least one system (either RH or Debian-based) and then configure a directory with SGID, allowing files created in this directory by different users retain the directory's group association.

1. Be in the logged-in user's home directory.
2. If necessary, create a directory at the root of the filesystem named `/projects`, create one standard user named `USERA`, and one group named `MANAGERS`
3. Display the new directory's permissions: `ls -l /`
4. Set the SGID permission on the new `/projects` directory for the group: `chmod g+s /projects`
5. Use `su - USERA`, create a file named `test` in the `/projects` directory, and then confirm the group association was inherited: `ls -l /projects`

# Understand the Sticky Bit

- Protects files within a directory.
  - Only the file owner or root can delete the file.
  - Without the Sticky Bit, any user with write or execute could potentially delete the file.

# Sticky Bit Special Permissions

- The "sticky bit" is a directory-level special permission that restricts file deletion, meaning only the file owner can remove a file within the directory.
  - The sticky bit can be very useful in shared environment because when it has been assigned to the permissions on a directory it sets it so only file owner can rename or delete the said file.
- You can only assign the sticky bit by explicitly defining permissions. The character for the sticky bit is **t**.
  - To set the sticky bit on a directory named dir1 you would issue the command \$ **chmod +t dir1**.

# Troubleshoot Special Permissions Access

- Troubleshoot with `ls -l` and `chmod`.
- Confirm the SUID permission is set correctly for executable files.
- Confirm the SGID permission is set correctly for directories to permit files created in the directory to inherit the group association.
- Confirm the sticky bit permission is set correctly.





# Review Activity: Special Linux Permissions

1. How would SGID benefit users when set on the /projects directory where multiple users are members of the associated group and need access to each other's files in the directory?
2. Why might a sysadmin set the sticky bit on a configuration file?

# Understand Access Control List (ACL) Concepts

- Standard permissions are limited to one user, one group, and all others.
  - Cannot grant different access levels to two different users.
- ACLs permit multiple users to be given multiple levels of access.
- ACLs permit multiple groups to be given multiple levels of access.

# Display ACL Entries

- Use the getfacl command to display ACL entries.

```
student@ubuntu20:/projects$ getfacl project1.txt
# file: project1.txt
# owner: student5
# group: 2022projects
user::rw-
user:student9:r--
group::r--
mask::r--
other::r--

student@ubuntu20:/projects$
```

# Configure ACL Entries

- Use the **setfacl** command to configure ACL entries.
- To set an ACL entry for userA with rwx access:  
  
    `setfacl -m u:userA:rwx fileA`
- To set an ACL entry for groupA with rwx access:  
  
    `setfacl -m g:groupA:rwx fileA`

- To set an ACL entry userA with rwx access and group sales with rw access:

`setfacl -m u:userA:rwx,g:sales:rw fileA`

- To remove an ACL entry for userA for fileA:

`setfacl -x u:userA fileA`

```
student@ubuntu20:/projects$ sudo setfacl -m u:student9:r project2.txt
student@ubuntu20:/projects$ getfacl project2.txt
# file: project2.txt
# owner: root
# group: root
user::rw-
user:student9:r--
group::r--
mask::r--
other::r--

student@ubuntu20:/projects$
```

# Key Demonstration: Configure ACLs



Sign in to at least one system (either RH or Debian-based) and then configure ACLs on a file, demonstrating multiple users with multiple levels of access. Begin at the logged-in user's home directory.

1. If necessary, create an additional user named USERZ: `useradd USERZ`
2. Create a directory named acl-demo: `mkdir acl-demo`
3. Change to the acl-demo directory: `cd acl-demo`
4. Create fileA: `touch fileA`
5. Display default ACL settings: `getfacl fileA`
6. Configure USERZ with read-only access: `setfacl SYNTAX`

# Troubleshoot ACLs

- Display ACL entries by using getfacl.
- Configure ACL entries by using setfacl.
- Confirm user identity.
- Confirm group membership.



# Review Activity: ACL Configuration

1. Explain the benefit offered by ACLs compared to standard Linux permissions.
2. What commands are used to set ACL entries for USERA with rwx and USERB with r-- for fileA?
3. Does the ACL structure replace standard permissions?

# Summary

- Understand the principle of least privilege, which enforces the idea that users should be given as little access to resources as necessary for them to do their jobs, with no additional unneeded access.
- Recognize access levels and identities.
- Absolute mode and symbolic mode provide the same information in different ways. Absolute mode displays in octal numerals, while symbolic mode displays information using operators.
- The immutable flag is an attribute of a file or directory that prevents it from being modified, even by the root user.





Thanks

Q & A