

LINUX

Local Security Principles

AUGUST 2025



User Accounts

- The Linux kernel allows properly authenticated users to access files and applications. While each user is identified by a unique integer (the user id or UID), a separate database associates a username with each UID. Upon account creation, new user information is added to the user database and the user's home directory must be created and populated with some essential files. Command line programs such as **useradd** and **userdel** as well as GUI tools are used for creating and removing accounts.
- For each user, the following seven fields are maintained in the **/etc/passwd** file:

```
student@centos8:/etc$ tail -20 /etc/passwd
radvd:x:75:75:radvd user:/:sbin/nologin
clevis:x:985:984:Clevis Decryption Framework unprivileged user:/var/cache/clevis:/sbin/nologin
cockpit-ws:x:984:982:User for cockpit-ws:/:sbin/nologin
colord:x:983:981:User for colord:/var/lib/colord:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
sssd:x:982:980:User for sssd:/:sbin/nologin
setroubleshoot:x:981:979:/:var/lib/setroubleshoot:/sbin/nologin
pipewire:x:980:978:PipeWire System Daemon:/var/run/pipewire:/sbin/nologin
gdm:x:42:42:/:var/lib/gdm:/sbin/nologin
gnome-initial-setup:x:979:977:/:run/gnome-initial-setup:/sbin/nologin
insights:x:978:976:Red Hat Insights:/var/lib/insights:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
pesign:x:977:975:Group for the pesign signing daemon:/var/run/pesign:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
tcpdump:x:72:72:/:sbin/nologin
student:x:1000:1000:LF Student:/home/student:/bin/bash
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
dovecot:x:97:97:Dovecot IMAP server:/usr/libexec/dovecot:/sbin/nologin
dovenull:x:976:972:Dovecot's unauthorized user:/usr/libexec/dovecot:/sbin/nologin
postfix:x:89:89:/:var/spool/postfix:/sbin/nologin
[student@centos8 etc]$
```

FIELD NAME	DETAILS	REMARKS
Username	User login name	Should be between 1 and 32 characters long
Password	User password (or the character x if the password is stored in the /etc/shadow file) in encrypted format	Is never shown in Linux when it is being typed; this stops prying eyes
User ID (UID)	Every user must have a user id (UID)	UID 0 is reserved for root user UID's ranging from 1-99 are reserved for other predefined accounts UID's ranging from 100-999 are reserved for system accounts and groups Normal users have UID's of 1000 or greater
Group ID (GID)	The primary Group ID (GID); Group Identification Number stored in the /etc/group file	Is covered in detail in the chapter on Processes
User Info	This field is optional and allows insertion of extra information about the user such as their name	For example: Rufus T. Firefly
Home Directory	The absolute path location of user's home directory	For example: /home/rtfirefly
Shell	The absolute location of a user's default shell	For example: /bin/bash

Types of Accounts

➤ By default, Linux distinguishes between several account types in order to isolate processes and workloads. Linux has four types of accounts:

- root
- System
- Normal
- Network

➤ For a safe working environment, it is advised to grant the minimum privileges possible and necessary to accounts, and remove inactive accounts. The **last** utility, which shows the last time each user logged into the system, can be used to help identify potentially inactive accounts which are candidates for system removal.

➤ Keep in mind that practices you use on multi-user business systems are more strict than practices you can use on personal desktop systems that only affect the casual user. This is especially true with security. We hope to show you practices applicable to enterprise servers that you can use on all systems, but understand that you may choose to relax these rules on your own personal system.

```
student@ubuntu: ~  
student@ubuntu:~$ last  
student  tty7      :0                Thu Dec 15 11:51   gone - no logout  
student  tty7      :0                Thu Dec 15 11:49 - 11:51   (00:01)  
reboot   system boot  4.4.0-53-generic Thu Dec 15 11:48   still running  
student  tty7      :0                Wed Dec 14 09:39 - down   (08:29)  
reboot   system boot  4.4.0-53-generic Wed Dec 14 09:39 - 18:09   (08:29)  
student  tty7      :0                Mon Dec 12 09:35 - crash   (2+00:03)  
reboot   system boot  4.4.0-53-generic Mon Dec 12 09:35 - 18:09   (2+08:34)  
student  tty7      :0                Mon Dec 12 09:26 - down   (00:07)  
reboot   system boot  4.9.0           Mon Dec 12 09:26 - 09:34   (00:07)  
student  tty7      :0                Mon Dec 12 09:26 - crash   (00:00)  
reboot   system boot  4.4.0-43-generic Mon Dec 12 09:25 - 09:34   (00:08)  
student  tty7      :0                Mon Dec 12 09:00 - crash   (00:25)  
reboot   system boot  4.4.0-43-generic Mon Dec 12 08:51 - 09:34   (00:43)  
student  tty7      :0                Fri Dec 9 13:09 - crash   (2+19:41)  
student  tty7      :0                Fri Dec 9 12:33 - 12:33   (00:00)  
student  tty2      :0                Fri Dec 9 12:32 - crash   (2+20:19)  
student  tty7      :0                Fri Dec 9 11:42 - 12:31   (00:49)  
reboot   system boot  4.4.0-43-generic Fri Dec 9 11:42 - 09:34   (2+21:52)  
student  tty7      :0                Wed Dec 7 14:11 - crash   (1+21:30)  
  
wtmp begins Wed Dec 7 14:11:29 2016  
student@ubuntu:~$
```

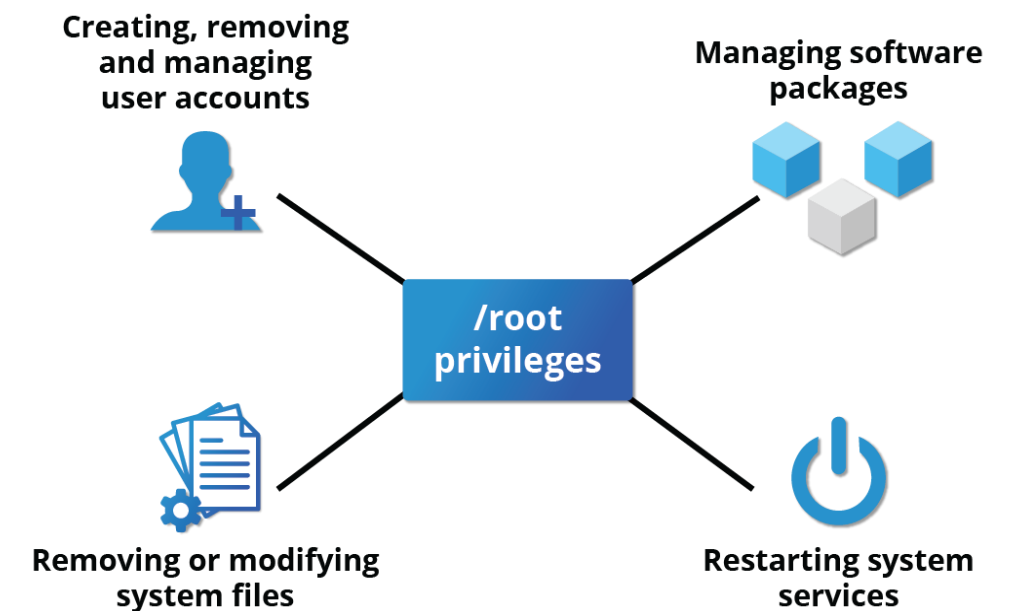
Understanding the root Account

- ✎ **root** is the most privileged account on a Linux/UNIX system. This account has the ability to carry out all facets of system administration, including adding accounts, changing user passwords, examining log files, installing software, etc. Utmost care must be taken when using this account. It has no security restrictions imposed upon it.
- ✎ When you are signed in as, or acting as root, the shell prompt displays '#' (if you are using **bash** and you have not customized the prompt, as we have discussed previously). This convention is intended to serve as a warning to you of the absolute power of this account.



Operations Requiring root Privileges

- root privileges are required to perform operations such as:
 - Creating, removing and managing user accounts
 - Managing software packages
 - Removing or modifying system files
 - Restarting system services.
- Regular account users of Linux distributions might be allowed to install software packages, update some settings, use some peripheral devices, and apply various kinds of changes to the system. However, root privilege is required for performing administration tasks such as restarting most services, manually installing packages and managing parts of the filesystem that are outside the normal user's directories.



Operations Not Requiring root Privileges

- ✚ A regular account user can perform some operations requiring special permissions; however, the system configuration must allow such abilities to be exercised.
- ✚ **SUID** (**S**et owner **U**ser **ID** upon execution – similar to the Windows "run as" feature) is a special kind of file permission given to a file. Use of SUID provides temporary permissions to a user to run a program with the permissions of the file owner (which may be root) instead of the permissions held by the user.
- ✚ The table provides examples of operations which do not require root privileges:

OPERATIONS THAT DO NOT REQUIRE ROOT PRIVILEGE	EXAMPLES OF THIS OPERATION
Running a network client	Sharing a file over the network
Using devices such as printers	Printing over the network
Operations on files that the user has proper permissions to access	Accessing files that you have access to or sharing data over the network
Running SUID-root applications	Executing programs such as passwd

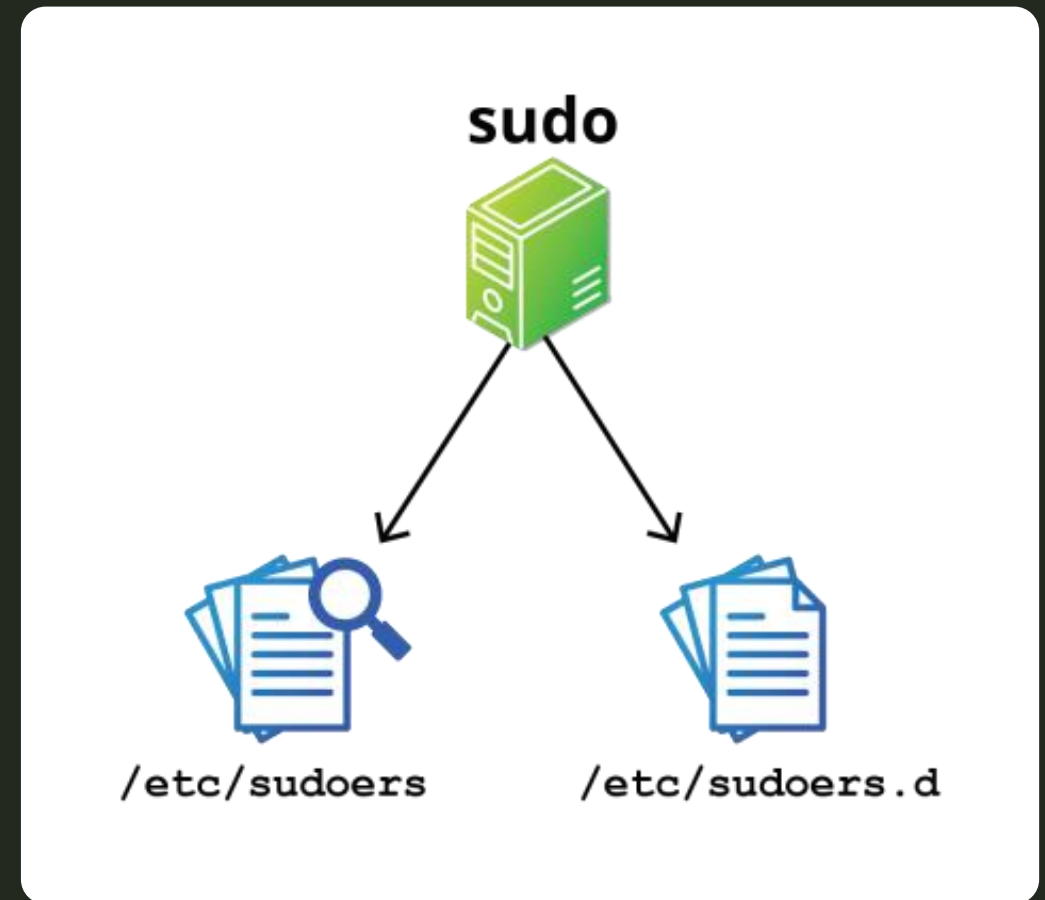
Comparing sudo and su

➤ In Linux you can use either **su** or **sudo** to temporarily grant root access to a normal user. However, these methods are actually quite different. Listed below are the differences between the two commands.

SU	SUDO
When elevating privilege, you need to enter the root password. Giving the root password to a normal user should never, ever be done.	When elevating privilege, you need to enter the user’s password and not the root password.
Once a user elevates to the root account using su , the user can do anything that the root user can do for as long as the user wants, without being asked again for a password.	Offers more features and is considered more secure and more configurable. Exactly what the user is allowed to do can be precisely controlled and limited. By default the user will either always have to keep giving their password to do further operations with sudo , or can avoid doing so for a configurable time interval.
The command has limited logging features.	The command has detailed logging features.

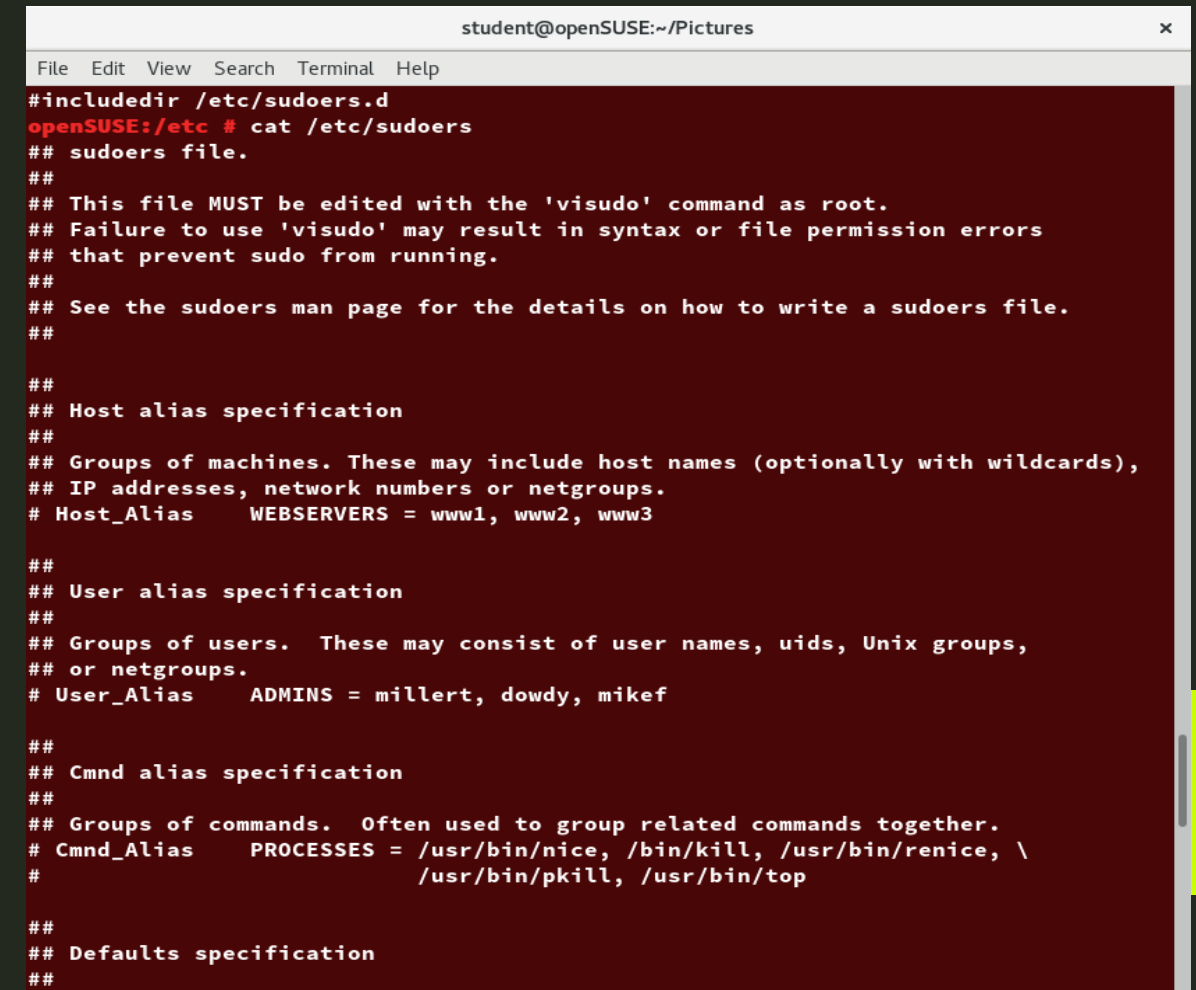
sudo Features

- ✎ **sudo** has the ability to keep track of unsuccessful attempts at gaining root access. Users' authorization for using **sudo** is based on configuration information stored in the **/etc/sudoers** file and in the **/etc/sudoers.d** directory.
- ✎ A message such as the following would appear in a system log file (usually **/var/log/secure**) when trying to execute **sudo** for **badperson** without successfully authenticating the user:
badperson : user NOT in sudoers ; TTY=pts/4 ; PWD=/var/log ; USER=root ; COMMAND=/usr/bin/tail secure



The sudoers File

- Whenever **sudo** is invoked, a trigger will look at `/etc/sudoers` and the files in `/etc/sudoers.d` to determine if the user has the right to use **sudo** and what the scope of their privilege is. Unknown user requests and requests to do operations not allowed to the user even with sudo are reported. The basic structure of entries in these files is:
 - **who where = (as_whom) what**
 - `/etc/sudoers` contains a lot of documentation in it about how to customize. Most Linux distributions now prefer you add a file in the directory `/etc/sudoers.d` with a name the same as the user. This file contains the individual user's **sudo** configuration, and one should leave the main configuration file untouched except for changes that affect all users.
 - You should edit any of these configuration files by using **visudo**, which ensures that only one person is editing the file at a time, has the proper permissions, and refuses to write out the file and exit if there are syntax errors in the changes made. The editing can be accomplished by doing a command such as the following ones:
 - **# visudo /etc/sudoers**
 - **# visudo -f /etc/sudoers.d/student**
 - The actual specific editor invoked will depend on the setting of your **EDITOR** environment variable.



```
student@openSUSE:~/Pictures
File Edit View Search Terminal Help
#includedir /etc/sudoers.d
openSUSE:/etc # cat /etc/sudoers
## sudoers file.
##
## This file MUST be edited with the 'visudo' command as root.
## Failure to use 'visudo' may result in syntax or file permission errors
## that prevent sudo from running.
##
## See the sudoers man page for the details on how to write a sudoers file.
##
##
## Host alias specification
##
## Groups of machines. These may include host names (optionally with wildcards),
## IP addresses, network numbers or netgroups.
# Host_Alias   WEBSERVERS = www1, www2, www3
##
## User alias specification
##
## Groups of users. These may consist of user names, uids, Unix groups,
## or netgroups.
# User_Alias   ADMINS = millert, dowdy, mikef
##
## Cmnd alias specification
##
## Groups of commands. Often used to group related commands together.
# Cmnd_Alias   PROCESSES = /usr/bin/nice, /bin/kill, /usr/bin/renice, \
#                                     /usr/bin/pkill, /usr/bin/top
##
## Defaults specification
##
```

Command Logging

➤ By default, **sudo** commands and any failures are logged in **/var/log/auth.log** under the Debian distribution family, and in **/var/log/messages** and/or **/var/log/secure** on other systems. This is an important safeguard to allow for tracking and accountability of sudo use. A typical entry of the message contains:

- Calling username
- Terminal info
- Working directory
- User account invoked
- Command with arguments
- Running a command such as **sudo whoami** results in a log file entry such as:
- **Dec 8 14:20:47 server1 sudo: op : TTY=pts/6 PWD=/var/log USER=root COMMAND=/usr/bin/whoami**

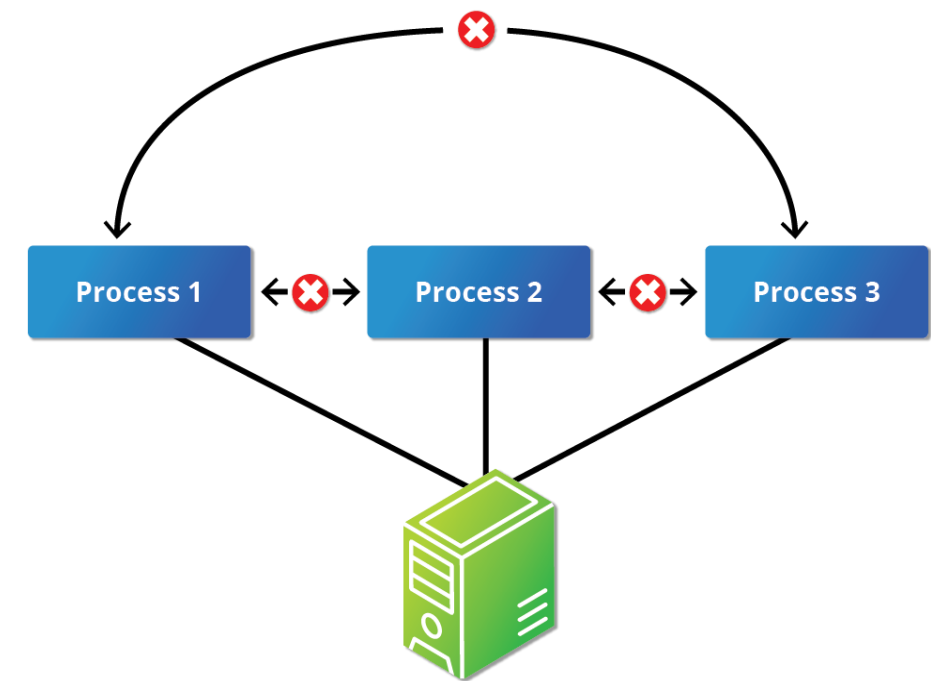
```
coop@r9:/tmp
r9:/tmp>sudo tail /var/log/secure
Feb 16 10:40:51 r9 sshd[182788]: Connection closed by 192.168.1.200 port 36463 [preauth]
Feb 16 10:41:02 r9 sudo[182796]:      coop : TTY=pts/0 ; PWD=/usr/local/teaching/LFCW ; USER=root ; COMMAND=/usr/b
in/systemctl status nfs-server
Feb 16 10:41:02 r9 sudo[182796]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Feb 16 10:41:07 r9 sudo[182796]: pam_unix(sudo:session): session closed for user root
Feb 16 10:41:12 r9 sudo[182803]:      coop : TTY=pts/0 ; PWD=/usr/local/teaching/LFCW ; USER=root ; COMMAND=/usr/b
in/systemctl restart nfs-server
Feb 16 10:41:12 r9 sudo[182803]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Feb 16 10:41:12 r9 sudo[182803]: pam_unix(sudo:session): session closed for user root
Feb 16 10:58:53 r9 sudo[184257]:      coop : TTY=pts/1 ; PWD=/usr/local/teaching/LFS101x/2023-PICS/CH18 ; USER=roo
t ; COMMAND=/usr/bin/tail /var/log/secure
Feb 16 10:58:53 r9 sudo[184257]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Feb 16 10:58:53 r9 sudo[184257]: pam_unix(sudo:session): session closed for user root
r9:/tmp>
```

Process Isolation

- Linux is considered to be more secure than many other operating systems because processes are naturally isolated from each other. One process normally cannot access the resources of another process, even when that process is running with the same user privileges. Linux thus makes it difficult (though certainly not impossible) for viruses and security exploits to access and attack random resources on a system.
- More recent additional security mechanisms that limit risks even further include:
- Control Groups (cgroups)
- Allows system administrators to group processes and associate finite resources to each cgroup.

Containers

- Makes it possible to run multiple isolated Linux systems (containers) on a single system by relying on cgroups.
- Virtualization
- Hardware is emulated in such a way that not only can processes be isolated, but entire systems are run simultaneously as isolated and insulated guests (virtual machines) on one physical host.



Hardware Device Access

- Linux limits user access to non-networking hardware devices in a manner that is extremely similar to regular file access. Applications interact by engaging the filesystem layer (which is independent of the actual device or hardware the file resides on). This layer will then open a device special file (often called a device node) under the `/dev` directory that corresponds to the device being accessed. Each device special file has standard owner, group and world permission fields. Security is naturally enforced just as it is when standard files are accessed.
- Hard disks, for example, are represented as `/dev/sd*`. While a root user can read and write to the disk in a raw fashion, for example, by doing something like:
 - `# echo hello world > /dev/sda1`
- The standard permissions, as shown in the figure, make it impossible for regular users to do so. Writing to a device in this fashion can easily obliterate the filesystem stored on it in a way that cannot be repaired without great effort, if at all. The normal reading and writing of files on the hard disk by applications is done at a higher level through the filesystem and never through direct access to the device node.

```
coop@r9:/tmp
r9:/tmp>ls -lF /dev/sd* /dev/nvme*
crw----- 1 root root 246, 0 Feb 16 07:33 /dev/nvme0
brw-rw---- 1 root disk 259, 0 Feb 16 07:33 /dev/nvme0n1
brw-rw---- 1 root disk 259, 1 Feb 16 07:33 /dev/nvme0n1p1
brw-rw---- 1 root disk 259, 2 Feb 16 07:33 /dev/nvme0n1p2
brw-rw---- 1 root disk 8, 0 Feb 16 07:33 /dev/sda
brw-rw---- 1 root disk 8, 1 Feb 16 07:33 /dev/sda1
brw-rw---- 1 root disk 8, 16 Feb 16 07:33 /dev/sdb
brw-rw---- 1 root disk 8, 17 Feb 16 07:33 /dev/sdb1
brw-rw---- 1 root disk 8, 18 Feb 16 07:33 /dev/sdb2
brw-rw---- 1 root disk 8, 32 Feb 16 07:33 /dev/sdc
brw-rw---- 1 root disk 8, 35 Feb 16 07:33 /dev/sdc3
brw-rw---- 1 root disk 8, 37 Feb 16 07:33 /dev/sdc5
brw-rw---- 1 root disk 8, 38 Feb 16 07:33 /dev/sdc6
brw-rw---- 1 root disk 8, 39 Feb 16 07:33 /dev/sdc7
brw-rw---- 1 root disk 8, 48 Feb 16 07:33 /dev/sdd
r9:/tmp>
```

Keeping Current

- When security problems in either the Linux kernel or applications and libraries are discovered, Linux distributions have a good record of reacting quickly and pushing out fixes to all systems by updating their software repositories and sending notifications to update immediately. The same thing is true with bug fixes and performance improvements that are not security related.
- However, it is well known that many systems do not get updated frequently enough and problems which have already been cured are allowed to remain on computers for a long time; this is particularly true with proprietary operating systems where users are either uninformed or distrustful of the vendor's patching policy as sometimes updates can cause new problems and break existing operations. Many of the most successful attack vectors come from exploiting security holes for which fixes are already known but not universally deployed.
- So the best practice is to take advantage of your Linux distribution's mechanism for automatic updates and never postpone them. It is extremely rare that such an update will cause new problems.



Timely System Update

Sample

- Create a new user, using `useradd`, and give the user an initial password with `passwd`.
- Configure this user to be able to use `sudo`.
- Login as or switch to this new user and make sure you can execute a command that requires root privilege.
- For example, a trivial command requiring root privilege could be:

➤ `$ ls /root`

➤ `sudo useradd newuser`

`sudo passwd newuser`

(give the password for this user when prompted)

- With root privilege, (use `sudo visudo`) add this line to `/etc/sudoers`:

```
newuser  ALL=(ALL)  ALL
```

- Alternatively, create a file named `/etc/sudoers.d/newuser` with just that one line as content.

- You can login by doing:

```
sudo su newuser
```

or

```
ssh newuser@localhost
```

which will require giving `newuser`'s password, and is probably a better solution. Instead of `localhost` you can give your hostname, IP address

or `127.0.0.1`. Then as `newuser` just type:

```
sudo ls /root
```

SUID, SGID, and Sticky Bit in Linux

➤ These are **special permission bits** in Linux, in addition to the usual rwx (read, write, execute) permissions. They control how programs and files behave when executed or modified by users.

➤ **SUID (Set User ID)**

➤ Applies to **executables**.

➤ When a file with SUID is executed, the process **runs with the file owner's privileges** (usually root), not the user's privileges.

```
ls -l /usr/bin/passwd
```

```
-rwsr-xr-x 1 root root 54256 Jul 20 12:30 /usr/bin/passwd
```

➤ Notice the s in place of x for the **user permission**: rws.

➤ The passwd program needs root privileges to update /etc/shadow, but normal users can still run it safely because of **SUID**.

➤ **Set SUID manually:**

```
chmod u+s file
```

➤ **Remove SUID:**

```
chmod u-s file
```


SGID (Set Group ID)

- ✎ Applies to **executables** and **directories**.
- ✎ For executables: The process runs with the **group ID of the file** instead of the user's group.
- ✎ For directories: Files created inside inherit the **group ownership** of the directory.

✎ Example (directory SGID):

```
mkdir /sharedchgrp developers /sharedchmod 2775 /sharedls -ld /shared
```

✎ Output:

```
drwxr-sr-x 2 root developers 4096 Aug 18 12:00 /shared
```

- ✎ New files created inside /shared will belong to the developers group automatically.

✎ Set SGID manually:

```
chmod g+s file_or_directory
```

✎ Remove SGID:

```
chmod g-s file_or_directory
```


Sticky Bit

- ✎ Applies to **directories**.
- ✎ Users can **only delete their own files** inside the directory, even if others have write permission.
- ✎ **Example:**

```
ls -ld /tmp
```

- ✎ Output:

```
drwxrwxrwt 10 root root 4096 Aug 18 12:00 /tmp
```

- ✎ Notice the **t** at the end.

/tmp allows everyone to create files, but the **sticky bit** prevents one user from deleting another user's files.

- ✎ **Set Sticky Bit manually:**

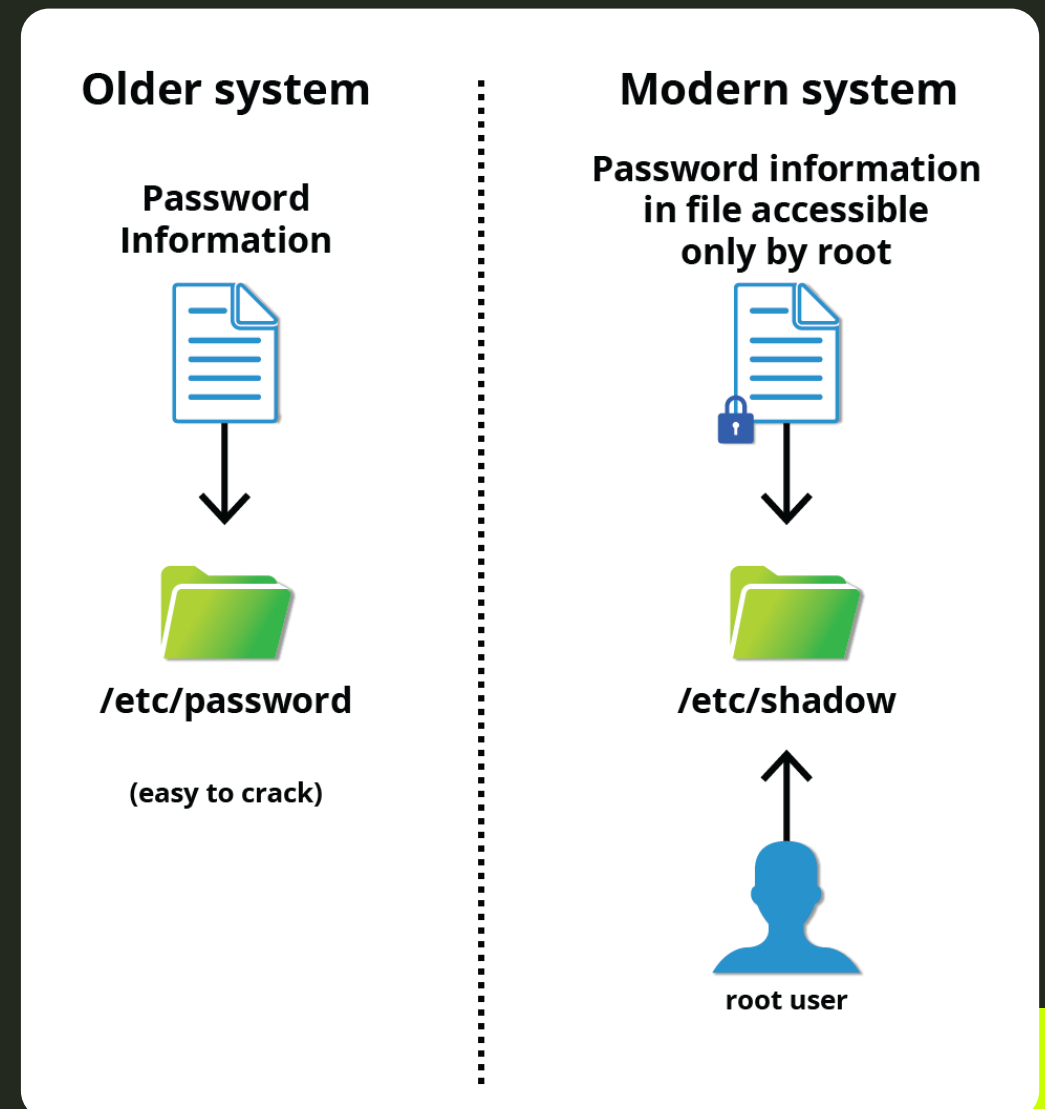
```
chmod +t directory
```

- ✎ **Remove Sticky Bit:**

```
chmod -t directory
```

How Passwords Are Stored

- ✎ The system verifies authenticity and identity using user credentials.
- ✎ Originally, encrypted passwords were stored in the `/etc/passwd` file, which was readable by everyone. This made it rather easy for passwords to be cracked.
- ✎ On modern systems, passwords are actually stored in an encrypted format in a secondary file named `/etc/shadow`. Only those with root access can read or modify this file.
- ✎ Change password with `passwd` command



Password Encryption

- Protecting passwords has become a crucial element of security. Most Linux distributions rely on a modern password encryption algorithm called SHA-512 (Secure Hashing Algorithm 512 bits), developed by the U.S. National Security Agency (NSA) to encrypt passwords.
- The SHA-512 algorithm is widely used for security applications and protocols. These security applications and protocols include TLS, SSL, PHP, SSH, S/MIME and IPSec. SHA-512 is one of the most tested hashing algorithms.
- For example, if you wish to experiment with **SHA-512** encoding, the word "test" can be encoded using the program sha512sum to produce the **SHA-512** form (see graphic):

```
File Edit View Search Terminal Help
c7:/tmp>echo -n test | sha512sum
ee26b0dd4af7e749aa1a8ee3c10ae9923f618980772e473f8819a5d4940e0db27ac
185f8a0e1d5f84f88bc887fd67b143732c304cc5fa9ad8e6f57f50028a8ff -
c7:/tmp>
```

Good Password Practices

- IT professionals follow several good practices for securing the data and the password of every user.
- Password aging is a method to ensure that users get prompts that remind them to create a new password after a specific period. This can ensure that passwords, if cracked, will only be usable for a limited amount of time. This feature is implemented using **chage**, which configures the password expiry information for a user.
- Another method is to force users to set strong passwords using **Pluggable Authentication Modules (PAM)**. PAM can be configured to automatically verify that a password created or modified using the **passwd** utility is sufficiently strong. PAM configuration is implemented using a library called **pam_cracklib.so**, which can also be replaced by **pam_passwdqc.so** to take advantage of more options.
- One can also install password cracking programs, such as John The Ripper, to secure the password file and detect weak password entries. It is recommended that written authorization be obtained before installing such tools on any system that you do not own.

```
student@openSUSE:~  
File Edit View Search Terminal Help  
student@openSUSE:~> chage --list student  
Password:  
Last password change           : Dec 14, 2016  
Password expires               : never  
Password inactive              : never  
Account expires                : never  
Minimum number of days between password change : 0  
Maximum number of days between password change : 99999  
Number of days of warning before password expires : 7  
student@openSUSE:~>
```

Sample

- With the newly created user from the previous exercise, look at the password aging for the user.
- Modify the expiration date for the user, setting it to be something that has passed, and check to see what has changed.
- When you are finished and wish to delete the newly created account, use **userdel**, as in:

```
$ sudo userdel newuser
```

```
chage --list newuser
```

```
sudo chage -E 2014-31-12 newuser
```

```
chage --list newuser
```

```
sudo userdel newuser
```

- **NOTE:** The example solution works in the US. **chage -E** uses the default date format for the local keyboard setting.

Requiring Boot Loader Passwords

- You can secure the boot process with a secure password to prevent someone from bypassing the user authentication step. This can work in conjunction with password protection for the BIOS. Note that while using a bootloader password alone will stop a user from editing the bootloader configuration during the boot process, it will not prevent a user from booting from an alternative boot media such as optical disks or pen drives. Thus, it should be used with a BIOS password for full protection.
- For the older GRUB 1 boot method, it was relatively easy to set a password for **grub**. However, for the GRUB 2 version, things became more complicated. However, you have more flexibility, and can take advantage of more advanced features, such as user-specific passwords (which can be their normal login ones).
- Furthermore, you never edit **grub.cfg** directly; instead, you can modify the configuration files in **/etc/grub.d** and **/etc/defaults/grub**, and then run **update-grub**, or **grub2-mkconfig** and save the new configuration file.
- To learn more, read the following post: "**GRUB 2 Password Protection**".



Hardware Vulnerability

- When hardware is physically accessible, security can be compromised by:
 - Key logging
 - Recording the real-time activity of a computer user, including the keys they press. The captured data can either be stored locally or transmitted to remote machines.
 - Network sniffing
 - Capturing and viewing the network packet level data on your network.
 - Booting with a live or rescue disk
 - Remounting and modifying disk content.
- Your IT security policy should start with requirements on how to properly secure physical access to servers and workstations. Physical access to a system makes it possible for attackers to easily leverage several attack vectors in a way that makes all operating system level recommendations irrelevant.
- The guidelines of security are:
 - Lock down workstations and servers.
 - Protect your network links such that it cannot be accessed by people you do not trust.
 - Protect your keyboards where passwords are entered to ensure the keyboards cannot be tampered with.
 - Ensure a password protects the BIOS in such a way that the system cannot be booted with a live or rescue DVD or USB key.
 - For single-user computers and those in a home environment, some of the above features (like preventing booting from removable media) can be excessive, and you can avoid implementing them. However, if sensitive information is on your system that requires careful protection, either it shouldn't be there, or it should be better protected by following the above guidelines.



Software Vulnerability

✚ Like all software, hackers occasionally find weaknesses in the Linux ecosystem. The strength of Linux (and open source community in general) is the speed with which such vulnerabilities are exposed and remediated. Specific coverage of vulnerabilities is beyond the scope of this course, but the Discussion Board can be used to carry out further discussion.

File Integrity Monitoring (FIM) — AIDE & Tripwire

- What is FIM?
- File Integrity Monitoring (FIM) is a security technique to detect changes to critical files, directories, or binaries. It's essential for:
 - Detecting unauthorized changes (intrusion detection)
 - Ensuring system configuration compliance
 - Monitoring tampering of logs, binaries, or configuration files
- How it works:
 - Baseline Creation – Scan files, record hashes, permissions, timestamps.
 - Regular Checks – Compare current state vs baseline.
 - Alerts – Report any unauthorized changes.

How FIM Works (Conceptually)

- Baseline Database File Scan → Hashes/Perms/Size Compare Current State ↔ Baseline Report Changes → Alerts
- Hash algorithms: SHA256, SHA1, MD5 (depending on tool)
- Monitored attributes: size, hash, owner, permissions, ACL, modification time

TOOL	TYPE	NOTES
AIDE	Open-source	Lightweight, easy setup, generates reports
Tripwire	Commercial/open-source	More features, compliance-ready, can integrate with SIEM

File Integrity

- ✎ FIM detects unauthorized file changes in real-time or via scheduled scans.
- ✎ **AIDE**: Lightweight, open-source, easy baseline creation.
- ✎ **Tripwire**: More enterprise-ready, includes compliance auditing.
- ✎ Key steps:
 - ✎ Install
 - ✎ Initialize database (baseline)
 - ✎ Run periodic checks
 - ✎ Respond/update database for legitimate changes

Summary

✎ The root account has authority over the entire system.

root privileges may be required for tasks, such as restarting services, manually installing packages and managing parts of the filesystem that are outside your home directory.

✎ In order to perform any privileged operations such as system-wide changes, you need to use either su or sudo.

✎ Calls to **sudo** trigger a lookup in the **/etc/sudoers** file, or in the **/etc/sudoers.d** directory, which first validates that the calling user is allowed to use **sudo** and that it is being used within permitted scope.

✎ One of the most powerful features of **sudo** is its ability to log unsuccessful attempts at gaining root access. By default, **sudo** commands and failures are logged in **/var/log/auth.log** under the Debian family and **/var/log/messages** in other distribution families.

✎ One process cannot access another process' resources, even when that process is running with the same user privileges.

✎ Using the user credentials, the system verifies the authenticity and identity.

✎ The SHA-512 algorithm is typically used to encode passwords. They can be encrypted, but not decrypted.

✎ Pluggable Authentication Modules (PAM) can be configured to automatically verify that passwords created or modified using the passwd utility are strong enough (what is considered strong enough can also be configured).

✎ Your IT security policy should start with requirements on how to properly secure physical access to servers and workstations.

✎ Keeping your systems updated is an important step in avoiding security attacks.

AI STUDIO

info@[aistudio.com.tr](mailto:info@aistudio.com.tr)

aistudio.com.tr

Thank you!

