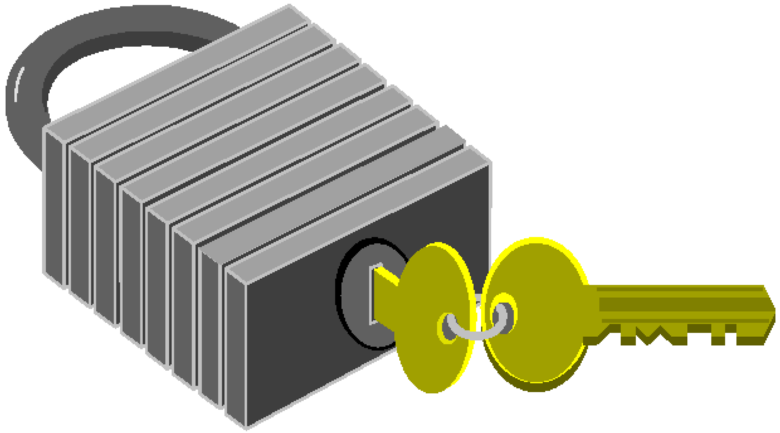
Operating Systems

LS-09. OS Permissions. SUID/SGID/Sticky. Extended Attributes.

Linux/UNIX Security Basics

Agenda

- UID
- GID
- Superuser
- File Permissions
- Umask
- RUID/EUID, RGID/EGID
- SUID, SGID, Sticky bits
- File Extended Attributes
- Mount/umount
- Windows Permissions
- File Systems Restriction



Domain Implementation in Linux/UNIX

- Two types domain (subjects) groups
  - User Domains = User ID (UID>0) or User Group ID (GID>0)
  - Superuser Domains = Root ID (UID=0) or Root Group ID (root can do everything, GID=0)
- Domain switch accomplished via file system.
  - Each file has associated with it a domain bit (SetUID bit = SUID bit).
  - When file is executed and SUID=on, then Effective UID is set to Owner of the file being executed. When execution completes Efective UID is reset to Real UID.
- Each subject (process) and object (file, socket,etc) has a 16-bit UID.
- Each object also has a 16-bit GID and each subject has one or more GIDs.
- Objects have access control lists that specify read, write, and execute permissions for user, group, and world.

Subjects and Objects

Subjects = processes (Effective UID, EGID)	**Objects = files (regular, directory, devices /dev, ram /proc)**
RUID (EUID)	Owner permissions (UID)
RGID-main (EGID) +RGID-list	Group Owner permissions (GID)
Others RUID, RGID	Others ID permissions

The Superuser (root)

- Almost every Unix system comes with a special user in the /etc/passwd file with a UID=0. This user is known as the superuser and is normally given the username root.
- Any process with a EUID=0 runs without security checks and is allowed to do almost anything. Normal security checks and constraints are ignored for the superuser.
- Any Username Can Be a Superuser

  root:x:0:1:Operator:/root:/bin/bash
  student:x:0:1001:Course Student:/home/student:/bin/csh
  trump:x:1002:1001:Donald Trump:/home/trump:/bin/ksh
- Special configured user can switch session account:

switch session to root	switch session to user	execute command as root
$ su	# su user	$ sudo command

- Su switches you to the root user account and requires the root account's password.
- Sudo runs a single command with root privileges – it doesn't switch to the root user and dont requires root user password.

What the Superuser Can Do

- **Device control**
  - Access any working device.
  - Shut down or reboot the computer. Set the date and time.
  - Read or modify any memory location.
  - Create new devices (anywhere in the filesystem) with the mknod command.

The Superuser (root)

What the Superuser Can Do

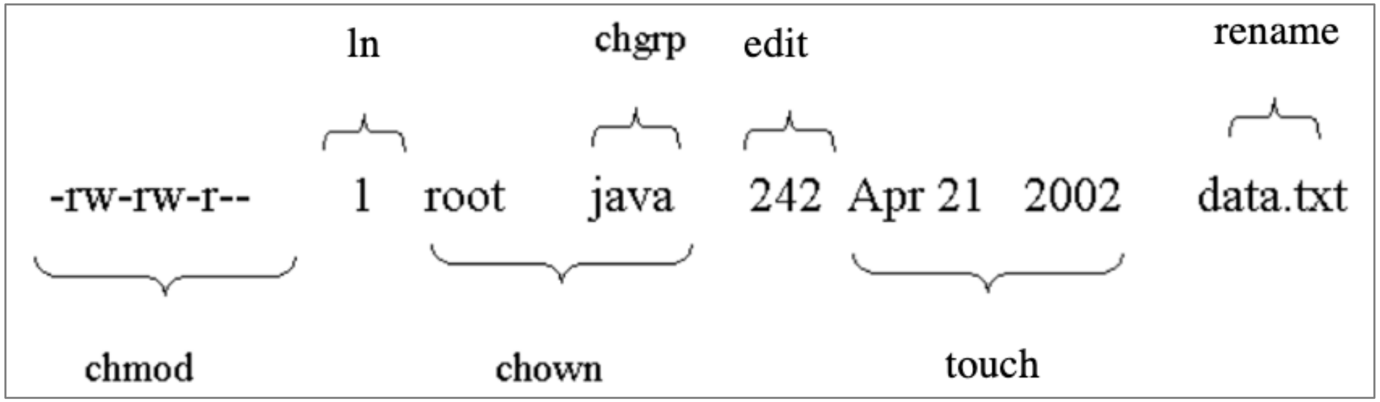
- Process control
  - Change the nice/renice value of any process.
  - Send any signal to any process (see Signals).
  - Alter "hard limits" for maximum CPU time as well as maximum file, data segment, stack segment, and core file sizes (see Limits).
  - Turn accounting and auditing on and off.
  - Bypass login restrictions prior to shutdown. (Note that this may not be possible if you have configured your system so that the superuser cannot log into terminals.)
  - Change his process UID to that of any other user on the system.
  - Log out all users and prevent new logins.

- Network control
  - Run network services on "trusted" ports.
  - Reconfigure the network.
  - Put the network interface into "promiscuous mode" and examine all packets on the network (possible only with certain kinds of networks and network interfaces, see Wireshark).

- Filesystem control
  - Read, modify, or delete any file or program on the system.
  - Run any program.
  - Change a disk's electronic label.
  - Mount and unmount filesystems.
  - Add, remove, or change user accounts.
  - Enable or disable quotas and accounting.
  - Use the chroot( ) system call, which changes a process's view of the filesystem root directory.
  - Write to the disk after it is "100 percent" full.

Changing a File's Parameters

- inodes contain a lot of information about a file:
  - mode - type and permissions of file;
  - number of links (names) to the file;
  - owner's UID;
  - owners GID;
  - size - number of bytes in file, blocks in dorectory;
  - last inode accessed, modified, changed, deleted times;
  - physical disk addresses (direct and indirect links to file blocks);
  - number of blocks;
  - shadow inode link with extend access information (ACL);
- Used commands for change file parameters: chmod, ln, chown, chgrp, edit, touch, rename, setacl, getacl, chattr, lsattr. After ls –l you see long files information:



Changing a File's Owner or Group

- The chown and chgrp commands allow you to change the owner/group of a file.
- Only the superuser can change the owner of a file under most modern versions of Unix.
- The chown command has the form:

```
# chown [ -R ] owner filelist
```

  - *-R – recursive change*
  - *owner* - the file's new owner; specify the owner by name or by decimal UID a valid user with an entry in /etc/passwd
  - *filelist* - The list of files whose owner you are changing

- Under most modern versions of Unix, you can change the group of a file if You are:
  - the file's owner and are in the group to which you are trying to change the file,
  - the superuser.

```
$ chgrp [ -R ] group filelist
```

- Some versions of chown can also change a file's group at the same time they change its owner. The syntax is usually:

```
# chown [ -R ] owner:group filelist
```

Objects mode = type + permissions

File type

- : plain file
d : directory
c : character device (tty, printer)
b : block device (disk, CD-ROM)
l : symbolic link
s : socket
=, p : pipe (FIFO)

How to read a –rwxrw-r– permission

r : read / w : write / x : execute

user	group	other
rwx 111	rw- 110	r-- 100
4+2+1 7	4+2+0 6	4+0+0 4

-rwxrw-r--

Access granted to owner

Access granted to others

Access granted to group member

File’s & Directory Permissions

Standard File Permissions Interpretation

- If you have read permission for a file, you can view its contents.
- If you have write permission for a file, you can change its contents.
- If you have execute permission for a file, you can run the file as a program.
- If you have read+execute permission for a file, you can run the file as a script.

Directory Permissions Interpretation

- If you have read+execute permission for a directory, you can list the contents of the directory. Only read is bad permission.
- If you have write+execute permission for a directory, you can create or remove files or sub-directories inside that directory. Only write is bad permission.
- If you have execute permission for a directory, you can change to this directory using the cd command, or use it as part of a pathname.

Special File: Block, Character, Pipe Permissions Interpretation

- If you have read permission for a file, you can view its contents.
- If you have write permission for a file, you can alter its contents.
- Execute permission for a file not interpreted.

Special File: Link Permissions Interpretation

- Read, Write, Execute permissions for a file not interpreted and permission controlled in real file, therefore the rights on the line are displayed as lrwxrwxrwx.

chmod: Changing a File's Permissions

- When you create a file, its initial permissions depend on your umask value (discussed later).
- You can change a file's permissions with the chmod command or the chmod( ) system call.
- If you are logged in as the superuser, you can change the permissions of any file.
- You can change a file's permissions only if you are the file's owner.
- The symbolic form of the chmod command:

```
chmod [-R] [agou][+-=][rwxst] filelist
```

- This command changes the permissions of filelist (a single file or a group of files).
- The letters agou specify whose privileges are being modified. You may provide one or more.
- The symbols +-= specify what is to be done with the privilege. You must type only 1 symbol
- The last letters rwxst specify which privilege will be modified: read, write, execute bits and suid, sgid, sticky bits.
- The -R option causes the chmod command to run recursively. If you specify a directory in filelist, that directory's permissions change, as do all of the files contained in that directory. If the directory contains any subdirectories, the process is repeated.

chmod: Changing a File's Permissions

Letter	Meaning
a	Modifies privileges for all users
g	Modifies group privileges
o	Modifies others' privileges
u	Modifies the owner's privileges

Symbol	Meaning
+	Adds to the current privilege
-	Removes from the current privilege
=	Replaces the current privilege

Letter	Meaning
r	Read access
w	Write access
x	Execute access
s	SUID or SGID
t	Sticky bit[9]

Examples:

```
$ chmod a+rwx file
$ chmod u=rw file
$ chmod ag-r,o+wx f1 f2
$ chmod u=s file
$ chmod ug+wxs,o-t file
$ chmod a+rwx,go-wx f1
$ chmod a-rwx,u=rwxs f1
```

```
% ls -l notes
-rw-r--r-- 1 sian     biochem    4320 Feb  9 13:20 notes
% chmod g+w notes
% ls -l notes
-rw-rw-r-- 1 sian     biochem    4320 Feb  9 13:20 notes
%
```

chmod: Changing a File's Permissions

- You can also use the chmod command to set a file's permissions, without regard to the settings that existed before the command was executed.
- This format is called the absolute (or numeric, or octal) form of the chmod command:

`chmod [-R] mode` ***`filelist`***

- The mode to which you wish to set the file, expressed as an octal value with 3 octal numerals. Every octal numeral is interpreted as 3 binary bits (rwx)

Example. Octal, binary and symbolic permissions:

$ chmod 000 file	000 000 000	---------
$ chmod 640 file	110 100 000	rw-r-----
$ chmod 123 file	001 010 011	--x-w--wx
$ chmod 777 file	111 111 111	rwxrwxrwx

chmod: Changing a File's Permissions

Common directory permissions

Octal number	Directory	Permission
0755	/	Anybody can view the contents of the directory, but only the owner or superuser can make changes.
1777	/tmp	Any user can create a file in the directory, but a user cannot delete another user's files.
0700	$HOME	A user can access the contents of his home directory, but nobody else can.

Common file permissions

Octal	Binary	Symbolic	Allowed file accesses
700	111 000 000	rwx------	Owner can read, write and execute
770	111 111 000	rwxrwx---	Owner and group can read, write, execute
640	110 100 000	rw-r-----	Owner can read and write; group can read
644	110 100 100	rw-r--r--	Owner can read and write; all other can read
655	110 101 101	rwxr-xr-x	Owner can do everything, rest can read & execute
000	000 000 000	---------	Nobody has any access
007	000 000 111	------rwx	Only other have access (strange, but legal)

chmod: Changing a File's Permissions

Common file permissions

Octal number	File	Permission
0755	*/bin/ls*	Anybody can copy or run the program; the file's owner can modify it.
0711	*$HOME*	Locks a user's home directory so that no other users on the system can display its contents, but allows other users to access files or subdirectories contained within the directory if they know the names of the files or directories.
0700	*$HOME*	Locks a user's home directory so that no other users on the system can access its contents, or the contents of any subdirectory.
0600	*/usr/mail/$USER* and other mailboxes	The user can read or write the contents of the mailbox, but no other users (except the superuser) may access it.
0644	Any file	The file's owner can read or modify the file; everybody else can only read it.
0664	*groupfile*	The file's owner or anybody in the group can modify the file; everybody else can only read it.
0666	*writable*	Anybody can read or modify the file.
0444	*readable*	Anybody can read the file; only the superuser can modify it without changing the permissions.

ACL

Access Control Lists

ACLs are a mechanism for providing fine-grained control over the access to files.

Without ACLs, the only way that you can grant permission to a single person or a group of people to access a specific file or directory is to create a group for that person or group of people.

With ACLs you can grant the access directly. For example, you can allow four different groups to a read a file without making it world-readable, or allow two users to create files in a directory without putting them in a group together.

Commands:

```
$ getfacl abc.txt
# file: abc.txt
# owner: student
# group: users
user::rw-
group::rw-
other::r--
$ setfacl [-bkndRLP] { -m|-M|-x|-X ... } file ...
```

Example:

```
user::rw-
user:lisa:rw-
user:vasja:rwx
group::r--
group:toolies:rw-
other::r--
```

umask

- The umask (Unix shorthand for "user file-creation mode mask") is a 3 or 4 octal number that Unix uses to determine the file permission for newly created files and directory.

$ umask NNN

- Every process has its own umask, inherited from its parent process.
- By default, Linux/Unix specify an octal standard mode of 666 (any user can read or write the file) when they create new files.
- By default, Linux/Unix specify an octal standard mode of 777 (any user can read, write, or look the directory) when they create new directory.
- For Result Permissions using bitwise AND with the default permissions and the complement of the umask value (the bits that are not set in the umask).
- Normally, you or your system administrator set the umask in your .login, .cshrc, or .profile files, or in the system /etc/profile or /etc/cshrc file. For example, you may have a line that looks like this in one of your startup files:

```
# Set the user's umask
umask 033
```

- The most common umask values are 022, 027, and 077. A umask value of 022 lets the owner both read and write all newly created files, but everybody else can only read them.

umask

Rules.

File Result Permissions Bits = NOT(umask Bits) AND (File Standard Permissions Bits)
Directory Result Permissions Bits = NOT(umask Bits) AND (Directory Standard Permissions Bits)

Example.

After umask 174

174	**(001 111 100)**	**Umask**
- 603	(110 000 011)	NOT(Umask)
* 666	(110 110 110)	Default file-creation mode
602	(110 000 010)	Result mode for new file

174	**(001 111 100)**	**Umask**
- 603	(110 000 011)	NOT(Umask)
* 777	(111 111 111)	Default directory-creation mode
603	(110 000 011)	Result mode for new directory

Set and test umask value.

```
$ umask              # current umask
0002
$ umask 174          # new umask
$ mkdir dir1         # create new directory
$ touch file1        # create new file
$ ls -l              # list permissions
drw-----wx 2 std std 512 Sep 1 20:59 dir1
-rw-----w- 1 dave dave 0 Sep 1 20:59 file1
```

Task 2.

Find result permissions for new files and directory after command:

$ umask 123
$ umask 325
$ umask 547
$ umask 406
$ umask 737
$ umask 100
$ umask 372
$ umask 077
$ umask 345
$ umask 704

RUID & EUID

- When a process executes, it has more values related to file permission
  - a Process ID, an Parent Process ID (PID-PPID)
  - a Real User ID, an Effective User ID (RUID-EUID)
  - a Real Group ID, an Effective Group ID (RGID-EGID)
- When you login, your login shell process' values are your user ID and group ID

- The UID of the user who started the program is used as its RUID and EUID.
- The EUID affects what the program can do (e.g. create, delete files).
- For example, the owner of /usr/bin/passwd and nano programms is root:

```
$ ls -l /etc/shadow /usr/bin/cat /usr/bin/passwd
-rw------- 1 root root   4270 ноя 11 13:09 /etc/shadow
-rwxr-xr-x 1 root root 246160 июн 12  2019 /usr/bin/cat
-rwsr-xr-x 1 root root 63736 июл 27  2018 /usr/bin/passwd
```

- But when we use nano, its RUID=EUID=UID is user (not root), so we can only edit user files.

- Programs can change to use the EUID the UID of the program owner if SUID bit enables for program,
  - e.g. the /usr/bin/passwd program changes to use its EUID (root) so that it can edit the /etc/passwd file (EUID=0 (root), RUID=UID (user).

SUIDSGID/sticky bits other interpretations

- SUID (set uid) --s------
  - Processes are granted access to system resources based on user who owns the program-file.
- SGID (set gid) -----s--
  - (For program-file) Same with SUID except group is affected.
  - (For directory) Files created in that directory will have their group set to the directory's group.
- Sticky bit --------t
  - This is obsolete with files, but is used for directories.
  - If set on a directory, then a user may only delete files that he owns or for which he has explicit write permission granted, even when he has write access to the directory. (e.g. /tmp) # **chmod 1777 /tmp**

SUID/SGID/sticky bits

Problems with SUID

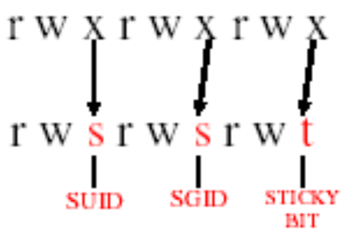
Any program can be SUID, SGID, or both SUID and SGID. Because this feature is so general, SUID/SGID can open up some interesting security problems.

Task. Finding all of the SUID and SGID files

find / \(-perm -004000 -o -perm -002000 \) -type f -print

$ find / \(-perm -004000 -o -perm -002000 \) -type f –print 2>/dev/null

- The access permission status that is displayed using the “ls –l” command does not have a section for special permissions
- However, since special permissions required “execute”, they mask the execute permission when displayed using the “ls –l” command.




```
/bin/su
/bin/ping
/bin/eject
/bin/mount
/bin/ping6
/bin/umount
/opt/kde2/bin/kreatecd
/opt/kde2/bin/konsole_grantpty
/opt/kde3/bin/artswrapper
/opt/kde3/bin/konsole_grantpty
/usr/bin/lpq
/usr/bin/lpr
/usr/bin/rcp
/usr/bin/rsh
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/lprm
/usr/bin/sudo
/usr/bin/crontab
/usr/bin/chage
/usr/bin/mandb
/usr/bin/vmware-ping
/usr/bin/expiry
/usr/bin/lpstat
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/gpasswd
/usr/bin/rlogin
/usr/bin/vmware
/usr/bin/cdda2cdr
```

SUID/SGID/sticky bits

Use the chmod command with 4 numerals octal mode

suid	sgid	stb	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1
7 Special			7 user			7 group			7 others		

Example. Octal, binary and symbolic permissions:

```
$ chmod 7000  file    111 000 000 000      --S--S--T
$ chmod 5740  file    101 111 100 000      rwsr----T
$ chmod 1123  file    001 001 010 011      --s-w--wx
$ chmod 0777  file    000 111 111 111      rwxrwxrwx
```

Files Extended Attributes

- File Attributes Alongside the standard permissions there is another system that can be used to change the way a file can be used.
- Extended attributes are supported by all major Linux file systems, including Ext2, Ext3, Ext4, Btrfs, JFS, XFS, and Reiserfs.
- Attributes do not show up in the 'ls' command. The lsattr and chattr command is used to show, set and drop these attributes.
- The symbolic form of the chattr command:

chattr [-R] [+-=][AacDijsSTtu] *filelist*

Examples:

```
# lsattr testfile
------------- testfile
# chattr +i testfile
# lsattr testfile
----i-------- testfile
# rm -f testfile
rm: cannot remove `testfile': Operation not permitted
# chattr -i testfile
# rm -f testfile
# ls testfile
ls: testfile: No such file or directory
# chattr +Si test.txt
# chattr -ai test.txt
# chattr =aiA test.txt
```

The following attributes are available Linux:

a: append only
c: compressed
d: no dump
e: extent format
i: immutable
j: data journalling
s: secure deletion
t: no tail-merging
u: undeletable
A: no Access time updates
C: no copy on write
D: synchronous directory updates
S: synchronous updates
T: top of directory hierarchy

Task:

```
$ chattr +AcstuS testfile
$ lsattr testfile
suS----Ac--t---- b
$ lsattr -l testfile
…?
```

Files Extended Attributes

- 'A' When a file with the ’A’ attribute set is accessed, its atime (access time) record is not modified. This avoids a certain amount of disk I/O, typically for temporary files.
- 'a' A file with the ‘a’ attribute set can only be open in append mode for writing. Only the superuser or a process possessing the CAP_LINUX_IMMUTABLE capability can set or clear this attribute. This is probably most effectively used on system log files, to prevent intruders removing evidence of their passage.
- 'c' A file with the ‘c’ attribute set is automatically compressed on the disk by the kernel. A read from this file returns uncompressed data. A write to this file compresses data before storing them on the disk.
- 'D' When a directory with the ‘D’ attribute set is modified, the changes are written synchronously on the disk; this is equivalent to the ‘dirsync’ mount option applied to a subset of the files. When this attribute is set the file system work on down speed run.
- 'i' A file with the ‘i’ (immutable) attribute cannot be modified: it cannot be deleted or renamed, no link can be created to this file and no data can be written to the file. Only the superuser or a process possessing the CAP_LINUX_IMMUTABLE capability can set or clear this attribute.
- 'j' A file with the ‘j’ attribute has all of its data written to the ext3 journal before being written to the file itself. Only the superuser or a process possessing the CAP_SYS_RESOURCE capability can set or clear this attribute.
- 's' When a file with the ‘s’ attribute set is deleted, its blocks are zeroed and written back to the disk. When this attribute is set the file system work on down speed run.

Files Extended Attributes

- 'S' When a file with the 'S' attribute set is modified, the changes are written synchronously on the disk; this is equivalent to the 'sync' mount option applied to a subset of the files. It is most often used for the 'cooked files' used by database programs to hold their data.
- 'T' The 'T' attribute is only usable when using the 2.6.x kernel. The 'T' attribute is designed to indicate the top of directory hierarchies, this is designed for use by the Orlov block allocator. The newer file allocation policies of the ext2 and ext3 filesystems place subdirectories closer together allowing faster use of a directory tree if that directory tree was created with a 2.6 kernel.
- 't' A file with the 't' attribute will not have a partial block fragment at the end of the file merged with other files (for those filesystems which support tail-merging). This is necessary for applications such as LILO which read the filesystems directly, and which don't understand tail-merged files.
- 'u' When a file with the 'u' attribute set is deleted, its contents are saved. This allows the user to ask for its undeletion. This is another attribute that is supported by everything except the kernel itself.

Common use Extended Attributes:

```
# chattr -R +i /bin /boot /etc /lib /sbin
# chattr -R +i /usr/bin /usr/include /usr/lib /usr/sbin
# chattr -R +a /var/log/messages /var/log/secure
```

Never use chattr +i for directory /, /dev, /tmp, /var !!!

Linux Capabilities. The root can set or clear [immutable/append-only attributes]. With a Linux kernel, you can prevent clearing these flags by dropping CAP_LINUX_IMMUTABLE from the Capability Bounding Set by doing flags:

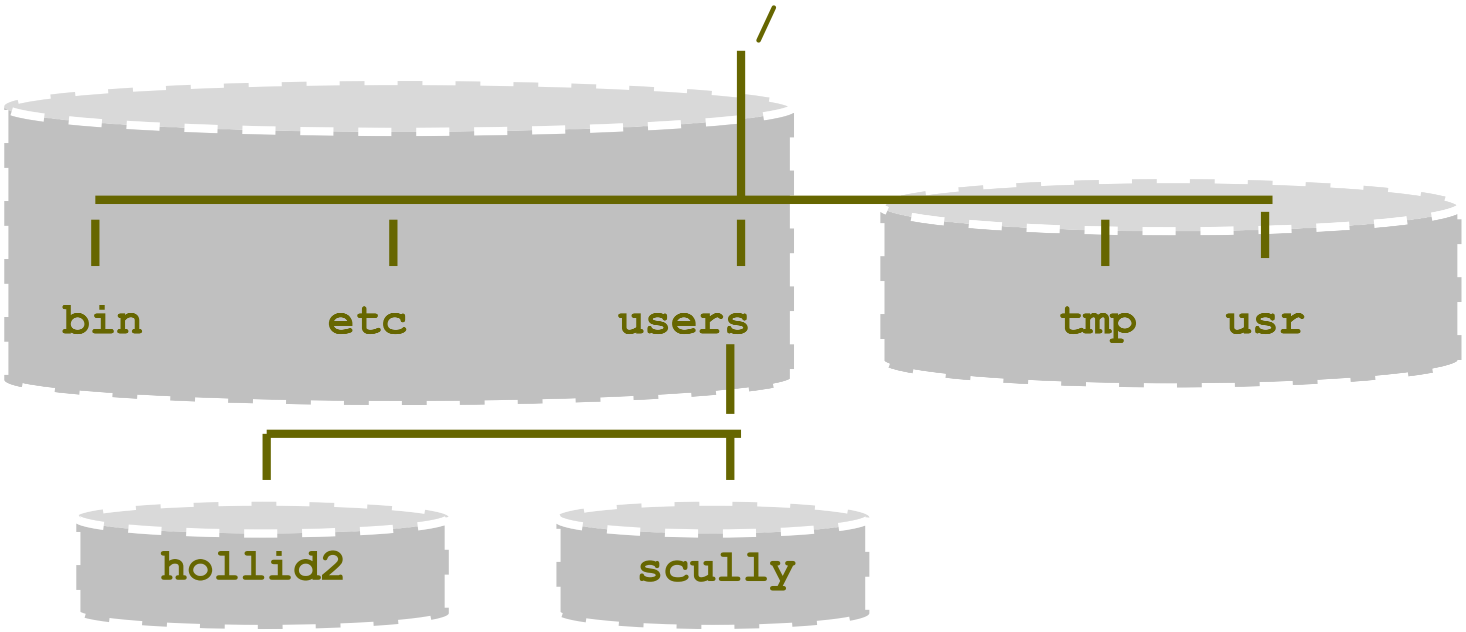
```
# echo 0xFFFFFDFF ?> /proc/sys/kernel/cap-bound
```

File system mount options

- The entire hierarchy can actually include many disk drives.
  - some directories can be on other computers
- Turning off SUID / SGID / EXECUTE in mounted file system
  - use nosuid (nosgid, noexec and nodev if possible) when mounting remote file system or allowing users to mount floppies or CD-ROMs (ReadOnly –ro)
- See /etc/fstab

```
LABEL=/home          /home        ext3     defaults,nosuid 1 2
LABEL=/tmp           /tmp         ext3     defaults,nosuid,noexec 1 2
```

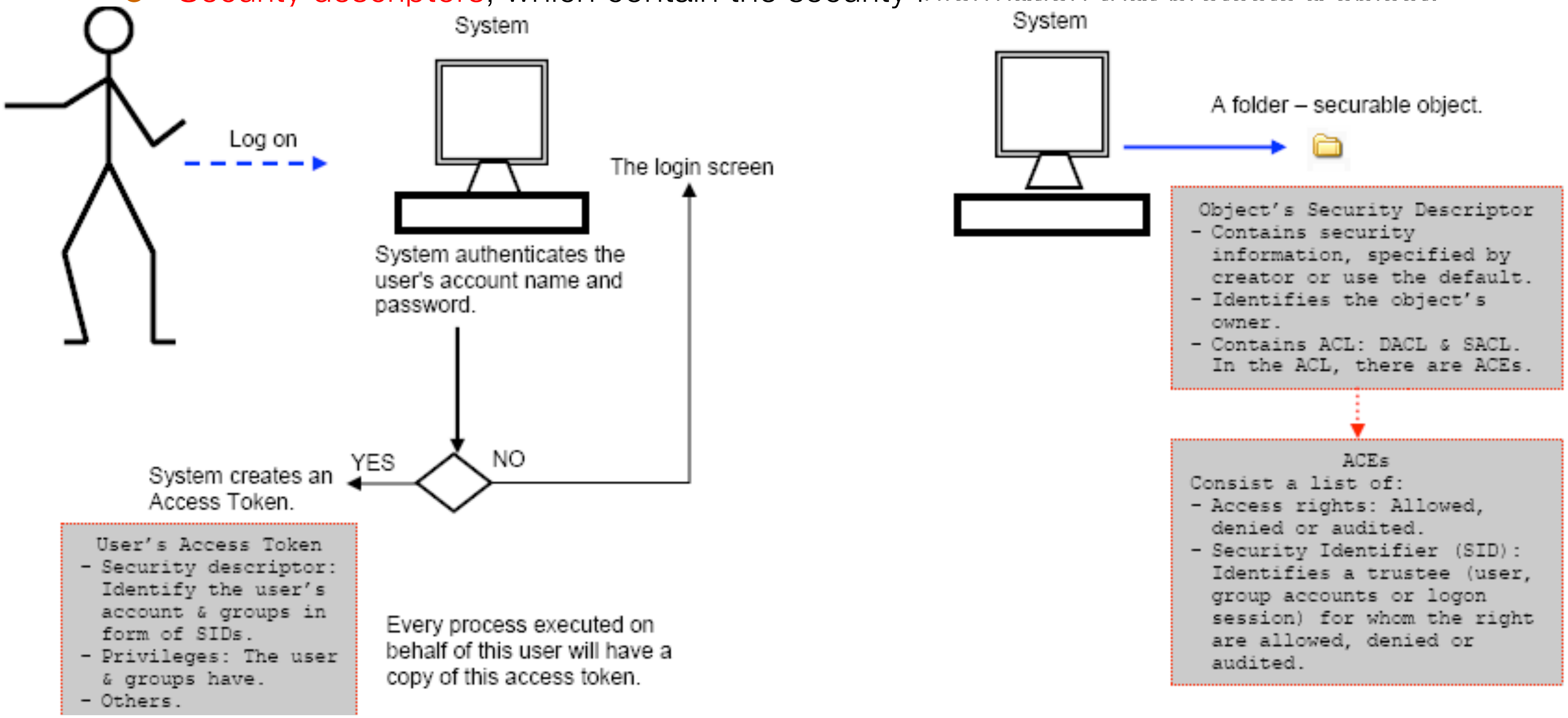
```
LABEL=/usr          /usr     ext3      defaults,ro 1 2
```



Windows Permissions 1/5

https://www.installsetupconfig.com/win32programming/accesscontrollistacl1.html

- 1. Two Basic Parts of the Access Control Model:
  - Access tokens, which contain information about a logged-on user (process, threads)
  - Security descriptors, which contain the security information that protects a object.



- 2. How Windows Access Check Works
  - When a process take to access securable object, the system looks for Access Control Entries (ACEs) in the object's DACL (discretionary access control list) that apply to the process.
  - Each ACE in the object's DACL specifies the access rights allowed or denied for a trustee, which can be a user account, a group account, or a logon session SIDs (security identifiers).

Windows Permissions 2/5

- **3. Access tokens contain:**

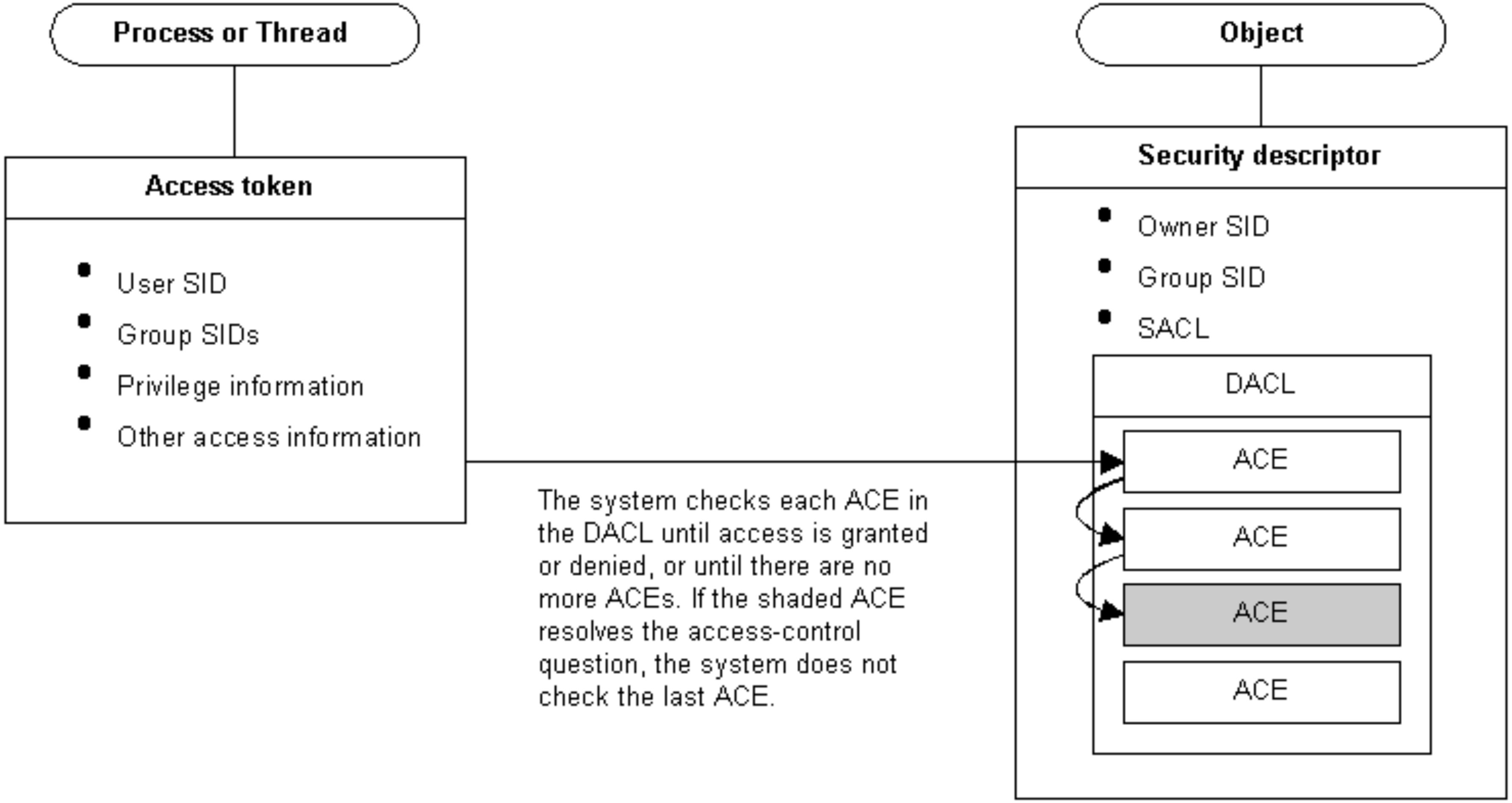
User SID (Secure ID) for the user's account.
Primary Group SID for the primary group.
Groups SIDs for the groups of which the user is a member.
A list of the Privileges held by either the user or the user's groups.
Logon SID that identifies the current logon session.
An owner SID for process.
The default DACL that the system uses when the user creates a securable object without specifying a security descriptor.
The source of the access token.
Whether the token is a primary or impersonation token (олицетворение).
An optional list of restricting SIDs.
Current impersonation levels.
Other statistics.

- **4. Security Descriptor contain:**

Object SID object owner
Primary Group SID object primary group
SACL (Secure ACL) that specifies the types of access attempts that generate audit records for the object.
DACL that specifies the access rights allowed or denied to particular users or groups in ACEs.
ACEs (Access Control Entry), each ACE controls or monitors access to an object by a specified SIDs.

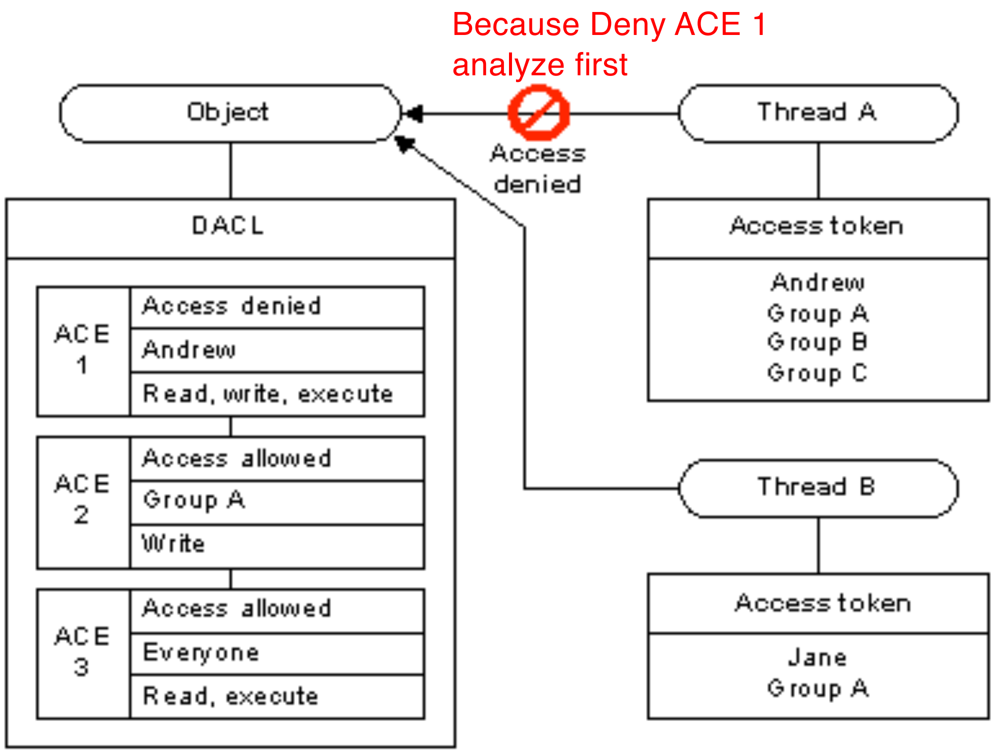
Windows Permissions 3/5

- 5. Interaction Between Process and Securable Objects



Windows Permissions 4/5

- 6. DACLs and ACEs



- 7. Securable Object Types
  - Files or directories on an NTFS
  - Named pipes, Anonymous pipes
  - File-mapping objects
  - Access tokens
  - Window-management objects
  - Registry keys
  - Windows services
  - Local or remote printers
  - Network shares
  - Interprocess synchronization objects
  - Job objects
  - Active Directory service objects

- 8. ACE structure:
  - SID,
  - Access Mask,
  - Type bits,
  - Child Inherit bits

- 9. Have 6 ACE types
  - Deny DACL
  - Allow DACL
  - SACL
  - 3 ACE types for Active Directory specific

Windows Permissions 5/5

10. Access Rights and Access Masks

- An access mask is a 32-bit value, whose bits correspond to the access rights supported by an object.
- These rights are used in ACEs and are the primary means of specifying the requested or granted access to an object.
- High-order 4 bits are used to specify generic access rights that each object type can map to a set of standard and object-specific rights.
- The AS-bit corresponds to the right to access the object's SACL, (manage auditing and security log privilege),
- Next 8 bits are for standard access rights, which apply to most types of objects,
- Low-order 16 bits are for object-specific access rights.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
GR	GW	GX	GA	R	R	MA	AS	R	R	R	SY	WO	WD	RC	DE	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

0 – Generic Read
1 – Generic Write
2 – Generic eXecute
3 – Generic All
4, 5 – Reserved
6 – Maximum Allowed
7 – Access System security (SACL)

Standard Access Rights
8, 9, 10 – Reserved
11 – SYnhronize
12 – Write Owner
13 – Write Dacl
14 – Read Control
15 – DElete

X - Object-specific Access Rights
16–31 – bits

Access control in Linux and Windows

Characteristic	Linux	Windows
Access rights	Read, write, execute for U/G id, Extended atributes	Support up to 32 different access rights (16 specific)
Inheritance	Mainly umask, but with SGID the objects inside can inherit	Support explicitly specified inheritance
ACE Types	Only have Allow in ACL	Allow, deny, audit
Access control granularity	User level, controlled by UID	Thread level, controlled by restricted context in Access Token
	There is an extended implementation on special UNIX / Linux	

File Systems Restrictions

Comparison of file systems
https://en.wikipedia.org/wiki/Comparison_of_file_systems

The End