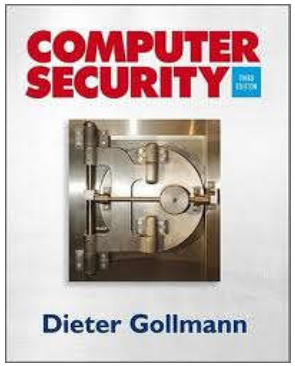
Access Control in Unix and Windows

Nicolas T. Courtois
- University College London



Unix Security

Chapter 7

Windows Security

Chapter 8

Our Objectives

Intended Learning Outcomes:

- short glimpse of how Unix and Windows manage access to files.

Unix:

- Vast topic, not clean, poorly documented, constant mutation…
  - Go to hacker conf, inspect source code, run tests...

Windows:

- did NOT publish all the details…
  - has a lot of added and useful complexity...

Beware:

many versions of Unix….

bottom line: Unix is an old primitive and INADEQUATE operating system

- think about future operating systems…
- how to fix Unix?
- What is missing in our OS?
- Do we trust it?
- Can we add more security like capabilities? Etc…?

*Beware of Hacks

Many things are OMITTED in my course… Technical.

Example: Race conditions, Q:Def?

*Beware of Hacks

Many things are OMITTED in my course… Technical.

Example: Race conditions, real-time hacks:

Exa 1: A privileged program creates a resource and changes permissions. Can we "freeze" it and make it chage permission for sth else?

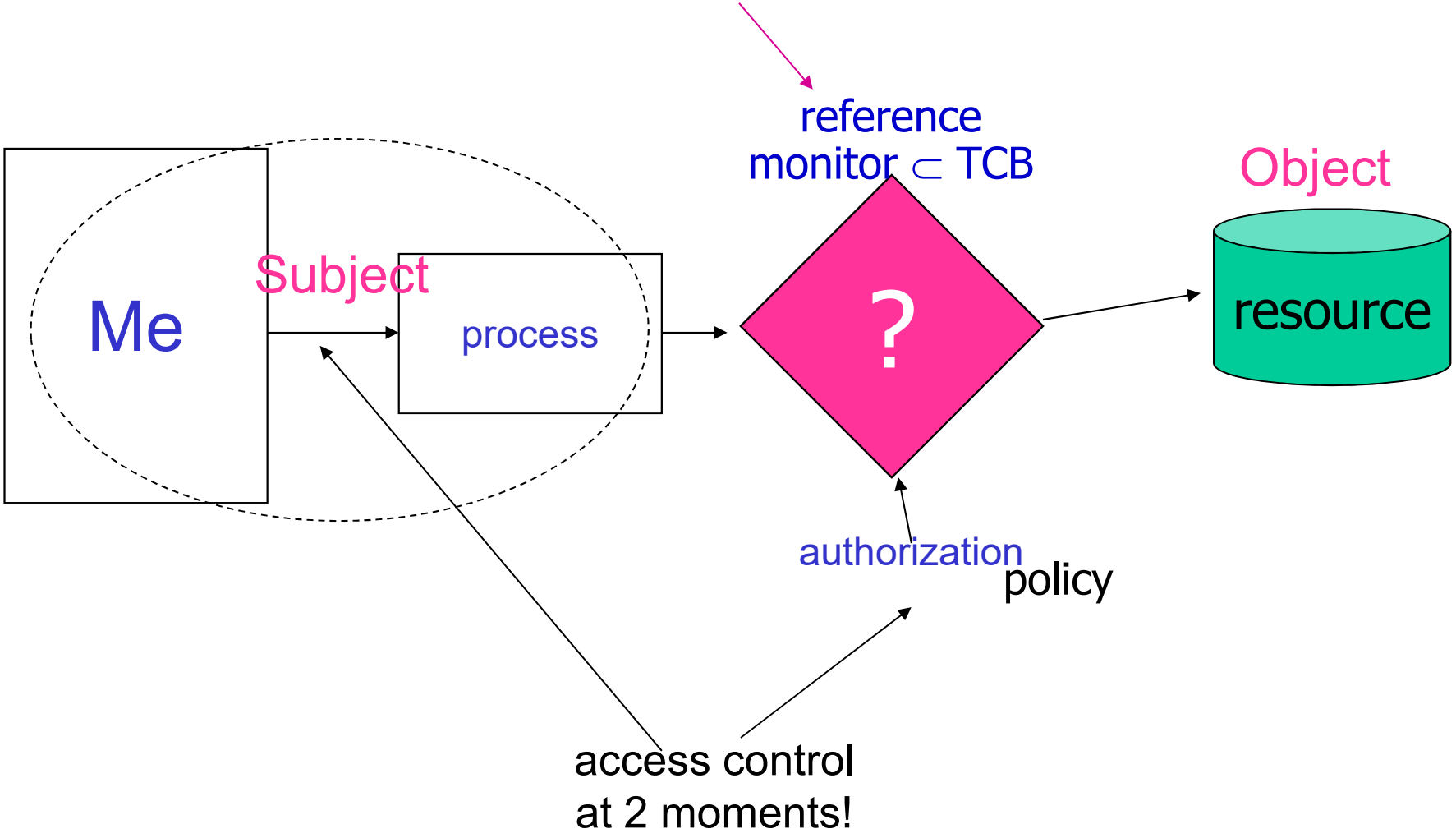
Exa 1b: same by hacking the filesystem or hard drive on the fly?

Exa 3: Scripting/batch/Java/Python: first invoke tool then load+exec.
Can u replace the script in real time???

there are many ways to "freeze" e.g. and there are "system locks" which means that some higher privileges than "admin" must exist…

main thing in part 04:

What Reference Monitor Does?



Q&A

any of our “Q”
could be at the exam..

Basic Principles

Q:

How user privileges are organised and stored?

Basic Principles

Q:

privileges?

They are stored in “user accounts”.

Basic Principles

Q:

In Windows, who decides if I can be logged as Login2?

The LSA = Local Security Authority

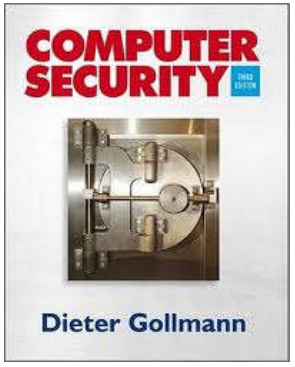


Image Name	PID	User Name
services.exe	732	SYSTEM
lsass.exe	740	SYSTEM

Grant / Remove

Q:

Who can grant / remove user privileges?

Grant / Remove

Q:

Who can grant / remove user privileges?

Any administrator user

Power

Q:

Who is the most powerful user in an OS?

Power

Q:

Who is the most powerful user in an OS?

Auxiliary questions to meditate:

Q: Can Admin user access any file/dir in Unix?

Power

Q:

the most powerful user?

Auxiliary questions to meditate:

Q: Can Admin user access any file/dir in Unix?

Admin Maybe [UID>=500]. Root most likely [UID=0].

Typically /etc/shadow is readable only by root,
stores passwords, see course part 05!)

Unix root vs. Windows

In Windows Admin user and System user are very different.

In Unix root is a [super-]super-user with almost no restrictions… or can go around them.

Example: root can not write a filesystem mounted as read-only, but can dismount and remount…

A process running as root is NOT exactly all powerful or almost is:

- process will run at CPU ring 3 = many CPU restrictions,
  - cannot access the physical RAM directly,
- CAN do almost everything BUT through system/Kernel calls
  - mediated by the system, logged by system (could tamper),
  - only in standard ways allowed by the system and prone to system imperfections…
  - If there is a “rootkit”, you would not notice… Do we understand the source code? Is compiler compromised? Etc.

Power

Q:

the most powerful user?

Q: Can Admin user access any file/dir in Windows?

Power

Q:

the most powerful user?

Q: Can Admin user access any file/dir in Windows?

not in default setting…

Power

Q:

the most powerful user in OS?

the ”system” user

Image Name	PID	User Name
VirtualBox.exe	9212	nc
svchost.exe	1004	NETWORK SERVICE
svchost.exe	1724	NETWORK SERVICE
sqlservr.exe *32	2948	NETWORK SERVICE
svchost.exe	2992	NETWORK SERVICE
wmpnetwk.exe	6124	NETWORK SERVICE
System Idle Process	0	SYSTEM
System	4	SYSTEM
smss.exe	428	SYSTEM
svchost.exe	500	SYSTEM
svchost.exe	524	SYSTEM
csrss.exe	572	SYSTEM
wininit.exe	664	SYSTEM
csrss.exe	680	SYSTEM
services.exe	732	SYSTEM
lsass.exe	740	SYSTEM
lsm.exe	748	SYSTEM
winlogon.exe	804	SYSTEM

Have You Noticed Something Special?

?

Image Name	PID	User Name
VirtualBox.exe	9212	nc
svchost.exe	1004	NETWORK SERVICE
svchost.exe	1724	NETWORK SERVICE
sqlservr.exe *32	2948	NETWORK SERVICE
svchost.exe	2992	NETWORK SERVICE
wmpnetwk.exe	6124	NETWORK SERVICE
System Idle Process	0	SYSTEM
System	4	SYSTEM
smss.exe	428	SYSTEM
svchost.exe	500	SYSTEM
svchost.exe	524	SYSTEM
csrss.exe	572	SYSTEM
wininit.exe	664	SYSTEM
csrss.exe	680	SYSTEM
services.exe	732	SYSTEM
lsass.exe	740	SYSTEM
lsm.exe	748	SYSTEM
winlogon.exe	804	SYSTEM

Security Layers

Image Name	PID	User Name
VirtualBox.exe	9212	nc
svchost.exe	1004	NETWORK SERVICE
svchost.exe	1724	NETWORK SERVICE
sqlservr.exe *32	2948	NETWORK SERVICE
svchost.exe	2992	NETWORK SERVICE
wmpnetwk.exe	6124	NETWORK SERVICE
System Idle Process	0	SYSTEM
System	4	SYSTEM
smss.exe	428	SYSTEM
svchost.exe	500	SYSTEM
svchost.exe	524	SYSTEM
csrss.exe	572	SYSTEM
wininit.exe	664	SYSTEM
csrss.exe	680	SYSTEM
services.exe	732	SYSTEM
lsass.exe	740	SYSTEM
lsm.exe	748	SYSTEM
winlogon.exe	804	SYSTEM

Security Layers w.r.t. CPU, RAM, OS

ring 0
physical memory

ring 3
virtual memory

more in part03

Image Name	PID	User Name
VirtualBox.exe	9212	nc
svchost.exe	1004	NETWORK SERVICE
svchost.exe	1724	NETWORK SERVICE
sqlservr.exe *32	2948	NETWORK SERVICE
svchost.exe	2992	NETWORK SERVICE
wmpnetwk.exe	6124	NETWORK SERVICE
System Idle Process	0	SYSTEM
System	4	SYSTEM
smss.exe	428	SYSTEM
svchost.exe	500	SYSTEM
svchost.exe	524	SYSTEM
csrss.exe	572	SYSTEM
wininit.exe	664	SYSTEM
csrss.exe	680	SYSTEM
services.exe	732	SYSTEM
lsass.exe	740	SYSTEM
lsm.exe	748	SYSTEM
winlogon.exe	804	SYSTEM

Ownership

- every object will have an owner

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool

“Ordinary” Rights = rwx

Q:

Who can read / write a file?

9 bits 3 answers:

“Ordinary” Rights = rwx

Q:

Who can r/w/x a file?

-rwx-r-x—-

9 bits 3 answers:

- First we have the rights of the owner

“Ordinary” Rights = rwx

Q:

Who can r/w/x a file?

9 bits 3 answers:

- first we have the rights of the owner
- then rights for the owner group

-rwx-r-x—-

user group world

“Ordinary” Rights = rwx

Q:

Who can r/w/x a file?

-rwx-r-x—-

user group world

9 bits 3 answers:

- first we have the rights of the owner
- then rights for the owner group
- everybody? not recommended, not very secure…

“Ordinary” Rights = rwx

Q:

Who can r/w/x a file?

-rwx-r-x—-

user group world

9 bits 3 answers:

- every object will have an owner
- and also an owner group
- everybody? not recommended, not very secure…

Q: can we have “worse” or “less secure” than everybody???

“Ordinary” Rights = rwx

Q:

Who can r/w/x a file?

-rwx-r-x—-

user group world

9 bits 3 answers:

- every object will have an owner
- and also an owner group
- everybody? not recommended, not very secure…
  - does NOT have to mean a user with an account…
  - DANGER: a remote user with no account…
    - Later about:
      - older WinXP…: ANONYMOUS LOGON
      - Unix: world-writable files in web servers

“World-writable?”

Q: meaning????

“World-writable?”

Means every user can write it…

World-writable directories?

Widely used for public_html directory

Allows the web server to CD and create new files etc…

It is like the web server is NOT trusted,
it could be abused by malicious people out there…

Inadequate

Our operating systems are NOT OK. Inability to to tell apart.

They lack a crucial distinction between […many possibilities…]

- a user with a local account indeed,
- a user with a remote account like say on a corporate network which we have joined an which we trust
- user with Microsoft Google Apple etc] authenticated but not trusted
- local actions of an authenticated web server which runs code on our machine [trusted to be done by a well identified business identity which you do or must trust] or local actions done by an anit-virus
- actions by a remote web site or remote user not authenticated in any way
- actions by a user which is authenticated but protected for anonymity
- etc…

Administrative Rights

Q:

Who can grant / remove permissions on objects/resources?

3+ answers:

Administrative Rights

Q:

Who can grant / remove permissions on objects/resources?

3+ answers:

- every object will have an owner

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool

Administrative Rights

Q:

Who can grant / remove permissions on objects/resources?

3+ answers:

- every object will have an owner
- and also an owner group
  - Yes, also in Windows, see "group SID" later….

Resources

Q:

Who can grant / remove permissions on objects/resources?

3+ answers:

- every object will have an owner
- and also an owner group
  - Yes, also in Windows, see "group SID" later….
- admin user??

Resources

Q:

Who can grant / remove permissions on objects/resources?

3+ answers:

- every object will have an owner
- and also an owner group
  - Yes, also in Windows, see “SID group” later….
- admin user?? - probably, depends on OS…
- the ”system”/Unix root user!

Hidden Powers

Question:

Who is more powerful than ”system” user?

Hidden Powers

Question:

Who is more powerful than ”system” user?

The hardware

CPU+chipset/motherboard+RAM…

Resources

Q:

Who can grant / remove permissions on objects/resources?

3+ answers:

- every object will have an owner
- and typically also an owner group
  - Yes, also in Windows, see “SID group” later….
- admin user?? - probably, depends on OS…
- the ”system”/Unix root user!
- possibly a hardware hack?!

Resources

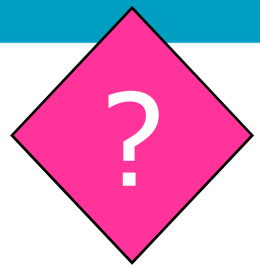
Q:

Who can grant / remove permissions?

3+ answers:

- every object will have an owner
- and typically also an owner group
  - Yes, also in Windows, see “SID group” later….
- admin user?? - probably, depends on OS…
- the ”system”/Unix root user!
- possibly a hardware hack?!
- rootkit which puts the whole OS “in jail”?! FEASIBLE???

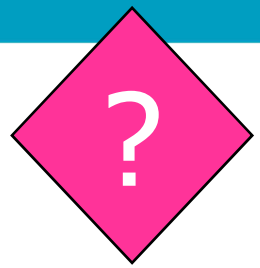
Ref. Monitor



Q:

On what basis it decides whether to grant / deny access?

Ref. Monitor

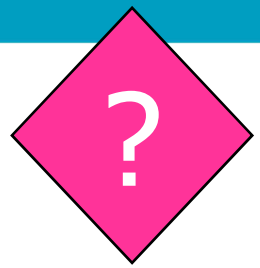


Q:

On what basis it decides whether to grant / deny access?

- User identity and privileges - stored in user profiles
- Process identity and privileges – real time (see effective uid)

Ref. Monitor



Q:

On what basis it decides whether to grant / deny access?

- User+Group identity and privileges - stored in user profiles
- Process identity and privileges – real time (see effective uid)

- Objects permissions: stored with the object,
  - ACL is the most common method (Windows and Unix).

Unix Access Control

Unix vs. Windows

The file system is a central object in Unix,

much more than in Windows:

- in Unix, files are not only files but also an abstraction for most other system resources (e.g. devices).

Ownership and Groups

Users in Linux (and many other Unix)

A user is identified by a User ID (UID)
= non-negative

- UID=0 == root
- 32 bits
- Low numbers <500 or 1000 reserved for programs and services,
- Human users usually start at 500 or 1000.
- the file /etc/passwd contains the login name for this UID

A group is identified by a Group ID (GID)
= non-negative int

Process Ownership

In Unix each process has 3[or 4] user IDs:

- Real User ID == ruid,
  - identifies the "Owner of this process"
    - user who has created the process,
    - or inherited from father process
    - does not matter a lot except f you want to change the effective user ID of an already running process, the kernel looks at the real user ID as well as the effective user ID

"admin part"

- Effective User ID == euid,
  - determines current access rights

"daily practice"

- Saved User ID == suid,
  - the previous one, used to store the UID when dropping the privileges, and to restore it later.

File Ownership

Change with chown

How To Determine Them (from the inside)

- Real User ID == ruid, == the owner
  - the process itself can get it through getuid() system call
- Effective User ID == euid, == current rights
  - read it by geteuid() system call

Groups

Groups - Users

A user can belong to many groups.
But at a given moment one is active. Change with chgrp

Intended usage:

- One unique primary group:
  - like “Bob” belongs to group “lecturers”.
- Member of other groups in order to:
  - This allows to implement various sorts of file privileges, that the process or a user can acquire and drop, making it harder to attack (the user / programmer is somewhat helped or forced to paying attention to security), the group must be “activated”.

Special "System Groups"

Special groups with gid<100
which partition the space of privileges…

Can be used to limit certain resources to a particular set of users.

User 'root' will be a member of many system groups

Examples:

- in Mac OS the primary group for root is wheel.
- www = the group that runs the Apache web server processes.
- mysql = the group that runs the MySQL database server processes

Groups - Files

important limitation of most UNIX systems…

- a file will be a member of ONLY ONE group.
- a process can at one moment be a member of ONLY ONE group.
  - (needed to set gid of files it creates!)

Unix File Permissions

Unix File System -

first letter

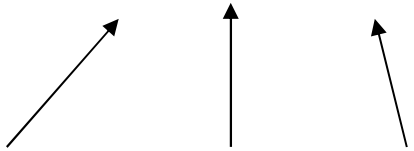
ls -l => -rwx-r-x—-

- file
d directory
l soft link

Unix File System -

the famous “9 bits”:

ls -l => -rwx-r-x—-



user group world

==other
==everybody
with an account

What’s inside each group?

A={read,write,exec}.

Octal: r=4 w=2 e=1, e.g. 775.

Changing Permissions

Q1: what is the command for changing permissions on Unix?

Changing Permissions

Q1: what is the command for changing permissions on Unix? We use chmod,

Q2: who has the right to do it?

Changing Permissions

chmod,

Q2: who has the right to do it?

Answer: the owner and root.

When rwx Means Something Special

A={read,write,exec}.

1. For directories, already quite special:
   - read – means list files, (does NOT mean you can read files)
   - write – means add/remove files and subdirs
   - exe (also called 'search') – means one can CD to that dir, and traverse a directory to access subdirectories.

Notes:

In order to read any file you MUST have 'exe' access to ALL directories on the path starting from the root directory /.

X not R for a directory will allow already to read files with known names which are R

rwx for a Process

- read – receive signals
- write – send signals
- exe –execute as a sub-process

Seen that?

-r-sr-sr-T

setuid,setgid,sticky bit

General idea:

By default an executable acts as the person who calls it and as the group of this person.
Except if:

setuid = can act as another user

setgid = can act as another group

sticky bit = related to “world”: about sharing…

Invocation

By default, programs run with the permissions of their caller.

Related question:

Q: why the current directory "."
isn't in UNIX PATH by default?

Invocation

By default, programs run with the permissions of their caller.

Related question:

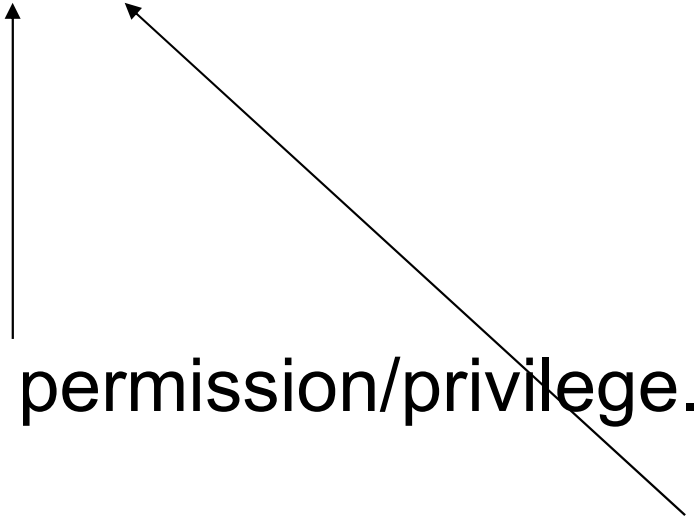
Q: why the current directory "."
isn't in UNIX PATH by default?

A: If one could fool a system process running as root into calling your program system("hack.exe"), hack.exe will be running with root privileges!

setuid permission

setuid

ls -l => -r-sr-sr-x



Occurs for exe files.

For user part: setuid permission/privilege..

For owner group: setgid permission.
Occurs for exe files and dirs.

What is setuid permission?

This process has the access rights of the owner of the file (owner on the disk), even if another user is running the process (the caller <> the owner).

The program starts with Effective User ID == euid, which can be high, for example root, and can be changed during the execution (more about this later).

Unix Password Storage

Password Storage in Linux

- Old old times:

  in /etc/passwd, readable by all.

- Now:

  in /etc/shadow, read-protected file, only accessible to passwd program and only to user=root.

How this is implemented?

Using this setuid permission!

setuid permission

access rights of the owner!

Example:

-r-sr-sr-x 3 root sys 28144 Jun 17 12:02 /usr/bin/passwd
(it is an executable file)

s makes (indirectly) that it can change the protected password file, a file that ordinary users cannot even read in most current Unix systems (owner=root can).

Technically, here the

1. Effective User ID == euid, will be root when you start the program
2. And can be changed later… (happily, more about this later)

Q: From 2012 Exam

4. Consider the following Unix file listing:

```
bash> ls -l /etc/shadow
-rw-r-----  1 root shadow  680 Dec  16 22:02 shadow
```

Is it possible to be able to run a dictionary attack on the passwords of all users, this without obtaining root access? Explain why.

[5 marks]

Q: From 2012 Exam

4. Consider the following Unix file listing:

```
bash> ls -l /etc/shadow
-rw-r-----  1 root shadow  680 Dec  16 22:02 shadow
```

Is it possible to be able to run a dictionary attack on the passwords of all users, this without obtaining root access? Explain why.

[5 marks]

A program run by a person who is not root,
but is a member of the group 'shadow' can read this file,
but cannot write it.

So he can run a dictionary attack (see slides part 05!!!).

**for directories

*Setuid / setgid for directories:

Special meaning,

- DISABLER not enabler

Any user who has write and execute permissions in the directory can create a file there.

However, the file belongs to the user/group that owns the directory, not to the creator user / group.

Makes these directories more protected, “more secure”.

Exercise

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Q: Which files alice can write?

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Q: Which files alice can write?

Q: Which information is missing?

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.

User dave is a member of groups dave and gdev.

Q: Which files alice can write?

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: Which files alice can write?

setup; sourcg; hosts;

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: alice executes gtool.
Can it execute 'hosts'?

Step by step.

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: alice executes gtool.

Can she?

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

Q: ruid=____ euid=____ for gtool?

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

A: ruid=alice euid=dave due setuid perm. for gtool!

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

A: ruid=alice euid=dave due setuid perm. for gtool!

Q: Can gtool execute ‘hosts’?

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

A: ruid=alice euid=dave due setuid perm. for gtool!

Q: Can gtool execute 'hosts'?

Q: Which information is missing?

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

A: ruid=alice euid=dave due setuid perm. for gtool!

A: gtool can execute:

A: setup; gtool; AND NOT hosts!; ARE WE SURE?

Must Also Check GID!

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

A: ruid=alice euid=dave due setuid perm. for gtool!

A: rgid=____ egid=___ for gtool!

Must Also Check GID!

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

A: ruid=alice euid=dave due setuid perm. for gtool!

A: rgid=alice egid=alice because alice's primary group is alice AND there is NO setgid privilege for gtool!

A: setup; gtool; AND NOT hosts!;

Exercise

Permissions	Owner	Group	Size	Last Update	File Name
-rws-----x	dave	gdev	1452306	Nov 03 21h11	gtool
drwxrwxrwt	dave	gdev	1452306	Nov 03 21h11	gdata
-rwx--x--x	alice	alice	214768	Nov 03 09h36	setup
-rw-r-----	alice	pcrack	12486	Dec 04 11h00	sourcg
-rw-r--r--	dave	pcrack	14257	Oct 02 18h44	config
-rw--wxr--	root	pcrack	176704	Nov 01 12h23	hosts

Suppose that user alice is a member of groups alice and pcrack.
User dave is a member of groups dave and gdev.

Q: which files can gtool execute, run by alice?

A: ruid=alice euid=dave due setuid perm. for gtool!

A: gtool can execute: (alice primary group: alice!)

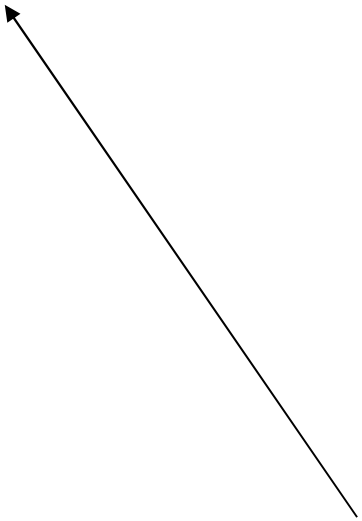
setup; gtool; AND NOT hosts!;

sticky bit

What about this one?

In last group now:

ls -l => drwxrwxrwt



Occurs for directories

called “sticky bit”

Extra Features of Unix

The famous sticky bit = save text image bit.

Example:

drwxrwxrwt 104 bin bin 14336 Jun 7 00:59 /tmp

Replaces the last “other” x. Means last x is present.

Capital T means last x is absent.

Usage: can be set using chmod +t command or chmod 1XXX.

It is peculiar and very useful feature.

Again not the same in every version of Unix.

Sticky bit for Directories

If it is 1:

“MAKES FILES STICK TO YOU”

Makes it harder to remove or rename files in this dir.

Even if the directory is world writable (everybody can create files), still

- only the owner of the file,
- or owner of the directory,
- or root
- [frequently also a superuser]

can remove or rename files contained in the directory.

Sticky bit for Directories

Application:

Typically used for /tmp directory.

Writable by all,
yet people can only remove or rename their own files.

Why Do We Have So Many UIDs?

Process Ownership

In Unix each process has

- Real User ID == ruid, identifies the owner
- Effective User ID == euid, determines current access rights
- Saved User ID == suid,

Why?

Because it allows one to implement security much closer to the least privilege principle… as we will see.

Though the Unix security seems simple and clear on the first day, it is neither simple, nor very easy to understand.

Because:

The Effective User ID can be both higher and lower than Real User ID. (both can be arbitrary)

Why it would be higher?

- This Allows Dropping of Privileges as we will see.
- A program can start in such a privileged (for now) state.

Can A Process be More Privileged than the User that Calls It?

YES!

- Happens all the time.

- History:
  in 1973 Denis Ritchie @ Bell Labs
  have patented this mechanism!

Example of application:

Install Programs

Example:

click on SETUP.exe for an anti-virus software.

This will install a system-level driver (a system service)

which is a very highly privileged piece of software.

- An escalation of privileges clearly occurs here.
- In Vista, if the name contains setup or install, the process already acquires many administrator privileges.
- you may need a digital signature from Microsoft to install such a sensitive system driver…

Another Important Example

Old example:

sendmail 8.10.1 program:

- when you run the sendmail program,
  executed by a non-root user, it has:
  - ruid=user, euid=root, suid=root.
  - this allowed the program to write to the mail queue.

OK, but isn't it very dangerous to have root privileges?

Yes, and once its write access to the mail queue is open, it can permanently drop the euid=root privilege.

set-uid programs

Definition:

A "set-uid program"
(property acquired @ creation and installation)
is the program that assumes the identity of the owner of the program, and runs as the owner, even though a different user uses it.

Examples:

- passwd
- su, sudo

BTW: if copied to a "user" directory, it stops working! (set-uid/gid programs are usually FORBIDDEN in home directories, no legitimate reason to have any!)

setuid system call

Inside the program source code.

General rule:

setuid(integer):

- IF euid == 0,
  - one can set effective UID to any value
- IF euid <> 0,
  - one can only set effective UID to ruid or suid

A set-uid program can drop root privileges by calling setuid(getuid()), which sets all three user IDs to the non-root user ID. Should be PERMAMENT. Except in older Unix versions…

Troubles with Sendmail

(history of Unix OS)

Dropping Privileges in Unix

sendmail 8.10.1

- ruid=user, euid=root, suid=root.
- this allowed the program to write to the mail queue.

- however, before users can request anything, the program permanently? dropped root privileges by calling setuid(getuid()), which sets all three user IDs to the non-root user ID.
  - Except with Linux Kernel <2.2.16.
  - it was NOT permanent, just did not work (bug)
  - it was possible later to become root again by “restoring” the saved uid (suid)…

Trouble: setuid()

- Different behaviour depending if euid=0 or not (!).
- Inconsistent behaviour in different versions of Unix.
- Sometimes man pages gave the wrong answer…
- Many attacks on Linux and Solaris in the past…
  - more secure in FreeBSD.

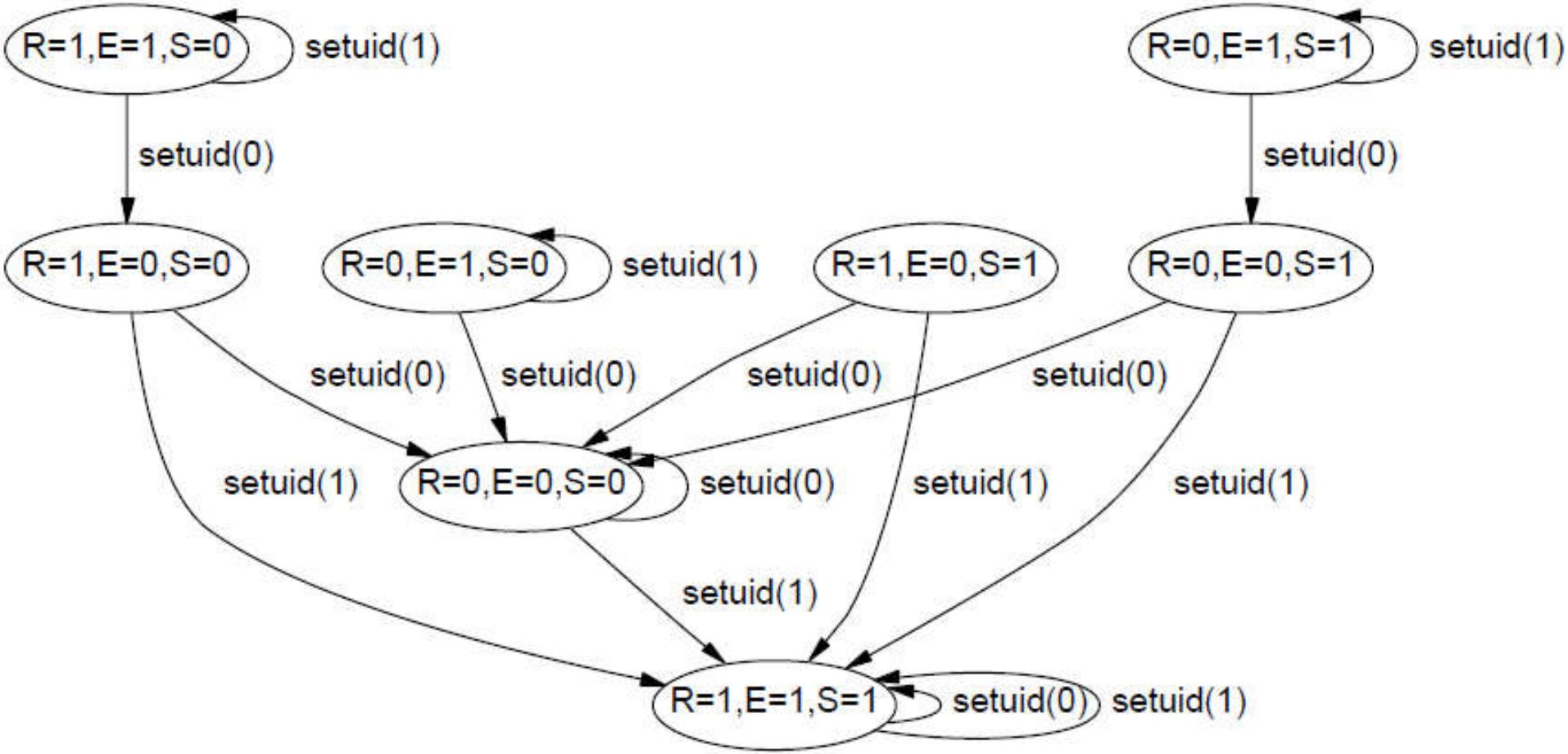
Conclusion:

NEVER use the ambiguous **setuid(.)**,

Instead:

**Trouble…

This one is for Linux 2.4.18. [cf. Setuid Demystified, Chen-Dean-Wagner]



Legend: 0 indicates root, 1 indicates a positive value $\neq 0$

Correct Ways To Drop Privileges in Unix

1. Permanent: changing all the three UIDs.
   - **setresuid(. . .) is used to set all the three ids.**
     - works if the process has appropriate privileges,
     - all 3 or nothing changed, clear-cut behaviour
     - use –1 at one param. to keep one of these unchanged.

2. Temporary - operational
   - can be later restored from saved uid
   - just set the effective uid by **seteuid(.)**
     - changes the "operational" one: **euid**.

No need to ever use the dangerous **setuid(.)** .

Extra Security Features in Unix

File Systems *ext2* and *ext3*

Have very important extra security functionalities

-works only if a file is stored on a volume using *ext2* or *ext3*.

Two important bits:

- Immutable - the file can never be changed.
  - However root is able to reset this parameter.
- Append-only - equivalent of a Write Once Read Many times mechanism.
  - For log files etc.

Useful commands are: lsattr and chattr.

AC in Unix – Is It Good?

Is It Excellent?

Not really:

- Access control is by user ID,
  - users don't want to give all their privileges to programs they run!

Is It Excellent?

Not really:

- Access control is by user ID,
  - users don't want to give all their privileges to programs they run!
- Software bugs do break some defences, and there never was enough defences in the system (many more layers would be needed).
  - More human ingenuity goes into attacks than into defences…
  - Attacks and malware are not local and propagate worldwide.

Is It Excellent?

Not really:

- Access control is by user ID,
  - users don't want to give all their privileges to programs they run!
- Software bugs do break some defences, and there never was enough defences in the system (many more layers would be needed).
  - More human ingenuity goes into attacks than into defences…
  - Attacks and malware are not local and propagate worldwide.
- All powerful root. Too dangerous.

Unix is by no means a reference w.r.t. security.
Both Linux and Windows remain in fact rather underdeveloped…

Fixing Unix - Ideas

- ?

Fixing Unix - Ideas

- Virtualization/confinement
- Breaking up the power of root
- Adding some MAC controls to remove many existing privileges and have much finer granularity.

root privileges
called capabilities

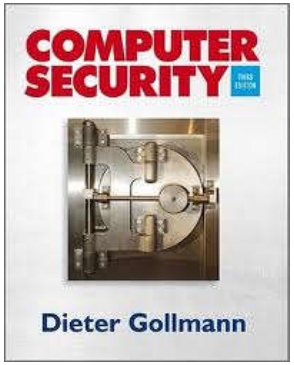
Breaking Up the root

All powerful root is too dangerous.

- 31 different capabilities are defined in capability.h in Linux kernel 2.6.11.
- it would be better to manage these separately…
  - and drop some capabilities at system boot already,
  - the future?

Windows File Access

Cf. Chapter 8



History

Initially Microsoft OS were not designed for multi-user environments.

- lack of OS support
- lack of file system support.

Windows NT, 2000, XP and Better

At the end

- OS support: becomes MUCH MORE complex than Unix,
- All these features require the NTFS file system.

DOS, old MACs

There was nothing except “read-only” or “protected” attribute (1 bit).

In great simplification…

Windows 95, 98

FAT32 system.

- Attributes: read-only, hidden, system, archive.
- not multi-user, no user permissions…

NT, XP, Vista, W7

All depends if your OS supports NTFS,

NTFS allows compression and encryption of individual files.

Q: How to lose access to encrypted files forever?

NT, XP, Vista, W7

Q: How to lose access to encrypted files forever?

- forget the password
- + delete the account
  - so that the password cannot even be cracked anymore

Windows Access Control

Windows NT, 2000, XP etc.

Windows was developed in C++.

Object oriented programming.

An object has attributes and methods.

Objects can be "securable" (or not).

Most objects are "securable":

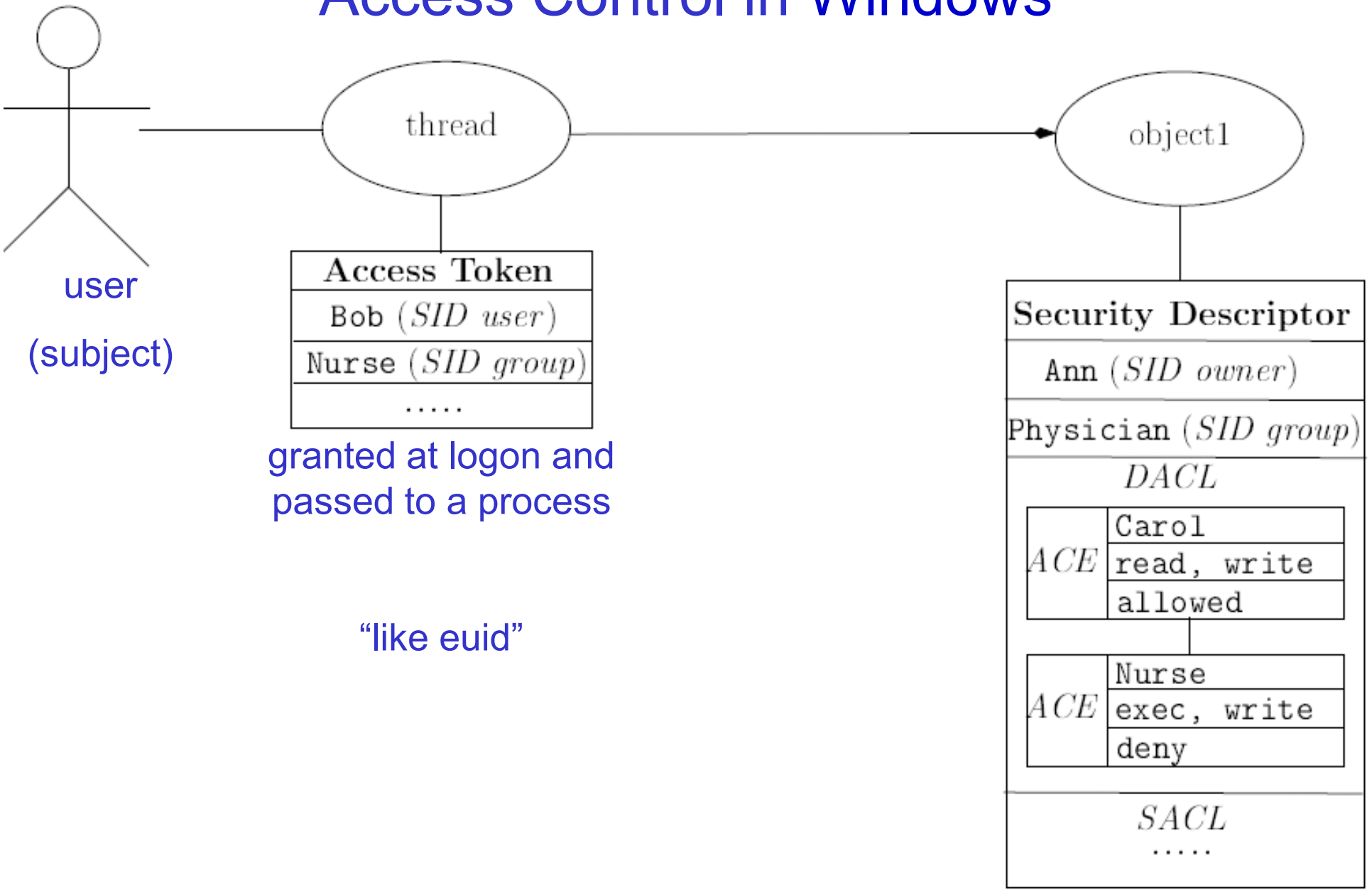
- files,
- processes,
- threads,
- named pipes,
- shared memory areas,
- registry entries
- etc.

Security Descriptors

Every "securable" object has a security descriptor.

=> The Microsoft equivalent of the 9-bit wrx_rx___x string
+ much more.

Access Control in Windows

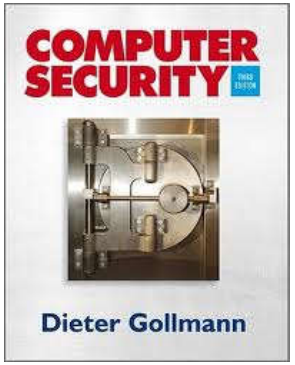


Revision:

Q:

who makes logon decisions?

The LSA =

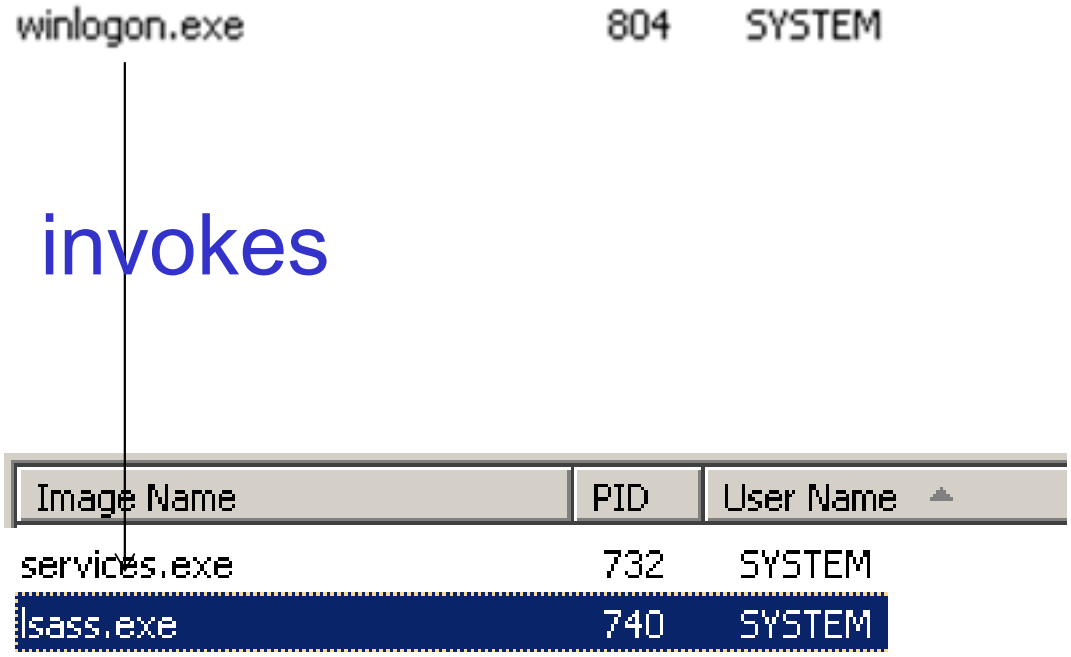


Revision:

Q:

who makes logon decisions?

The LSA = Local Security Authority compare: Shadow group in our exercise

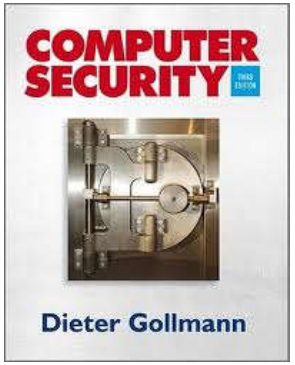


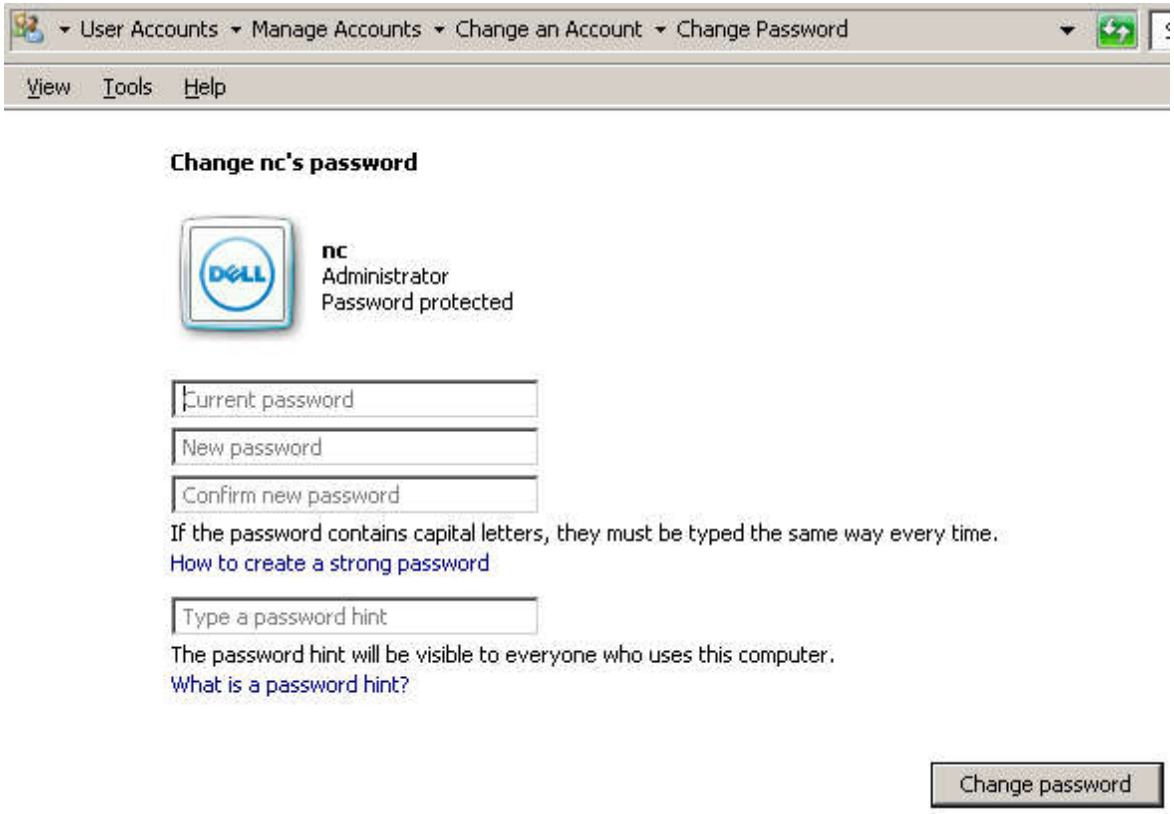
Accounts

Q:

who maintains the user accounts database?

The SAM =





Accounts

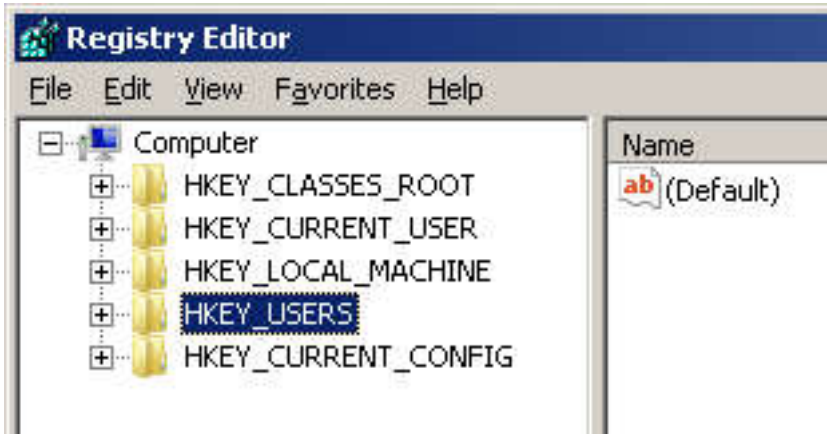
Q:

who maintains the user accounts database?

The SAM =

Security Account Manager

"visible" data for apps



Hidden Data?
Like Encrypted/Hashed Passwords?

Q:

Unix: /etc/shadow

Windows: ??????????????????

Windows Password Storage

The SAM file

Unix: /etc/shadow

Windows XP: C:\Windows\System32\config\SAM

At all times Win Kernel keeps an exclusive filesystem lock on this file, one cannot read it…

Access Control in Windows

MUCH more complex than is Unix…

Less power to “admin”. More security at system levels (more privileged than admin).

Let’s just study the very basic key elements involved in Windows Access Control Lists = ACL’s.

Subjects

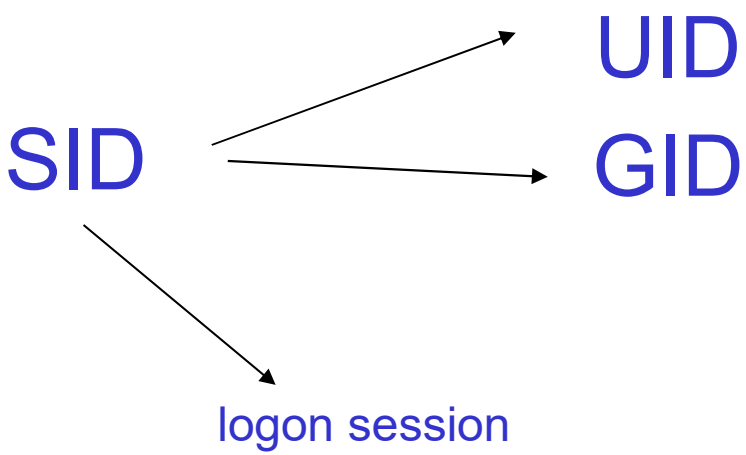
In Windows NT, subjects, that can operate on objects can be:

- users,
- groups,
  - example: “Authenticated Users” as opposed to “Anonymous Users”.
- “logon sessions”, confused and complicated concept.
  - for example ANONYMOUS LOGON. occur with remote logon
    - allow a remote machine to act as a secondary graphical "terminal" to a Windows NT.

Historically hackers used a lot of anonymous logon sessions to hack windows: many version of Windows allow to enumerate user accounts to a remote user which is anonymous…

For a long time group Everyone included ANONYMOUS LOGON, since XP SP2 it doesn’t and ANONYMOUS LOGON unless explicitly added.

Windows vs. Unix



SIDs

In Windows NT, subjects, that can operate on objects can be:

- users,
- groups,
- “logon sessions”

Have each a unique SID = Security Identifier.

- Example: "S-1-5-21-XXXXXXXXX..

*Nested Groups?

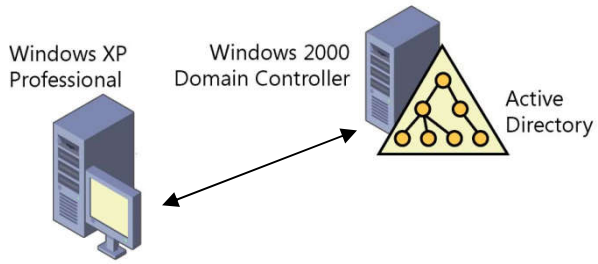
In Windows a group is a collection of SIDs.

Two sorts:

- local group = a.k.a. alias, managed by LSA.
  - similar to UNIX:
    - can be used to grant access to resources
    - can NOT be nested

lsass.exe

- global group – managed by domain controller
  == another computer
  - CAN be nested!!!!



Question for 1$:

relates to ACLs and file access control:

Q: Why windows computers using a workgroup to share files are potentially less secure than using a domain?

Question for 1$:

Q: Why windows computers using a workgroup to share files are potentially less secure than using a domain?

Many problems,

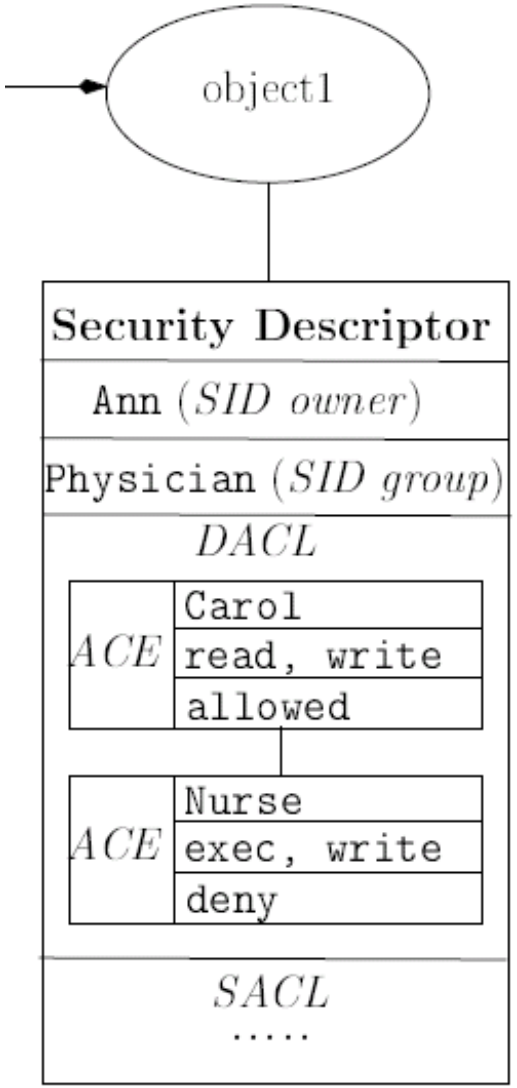
- lack of user authentication, trusted machines
- quite technical:
  - if the user, or a group on a second computer has the same SID (because the machine SID is the same for example), which can happen, then an access can be mistakenly granted by the system (!)

DAC and ACL in Windows

Windows has

- Discretionary Access Control Lists = **D**ACL.
- System Access Control Lists = **S**ACL.

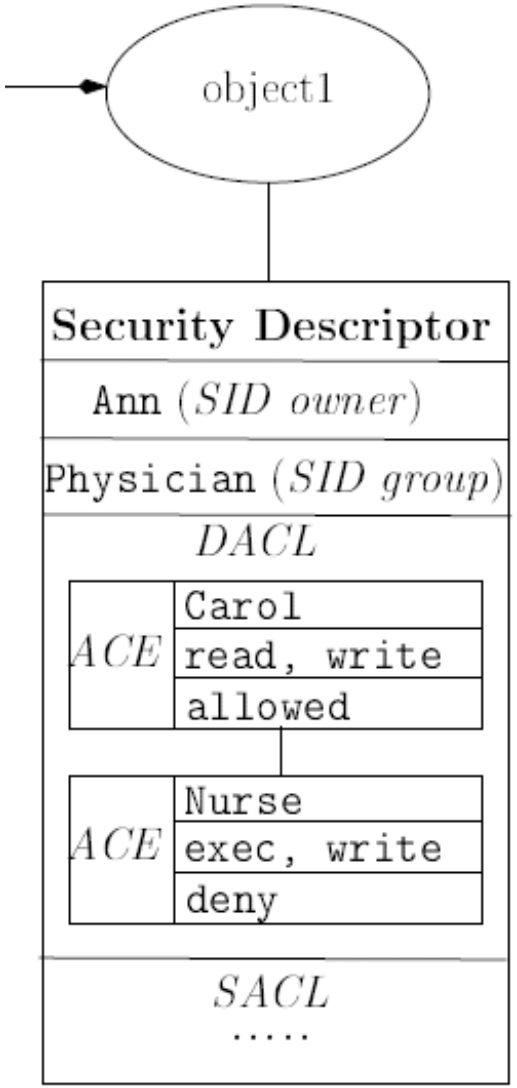
Each DACL is a collection of Access Control Elements = ACE.



ACE - Elements

For each (subject, object) we have various privileges
divided into three groups, see winnt.h:

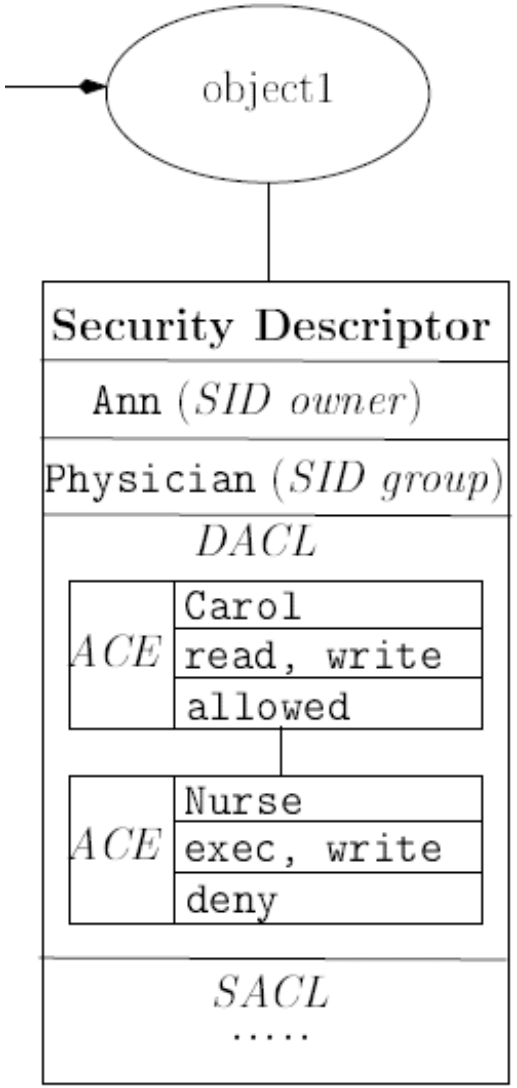
- generic:
  - read, write, execute
- standard:
  - delete,
  - read_control (right to read the security descriptor),
  - synchronize (right to wait for some signal from the object),
  - write_dac (right to change the DACL),
  - write_owner (right to change the object's owner);
- SACL: system privileges



ACE - Directories

Microsoft has several special attributes that exist only for directories:

- open,
- create_child,
- delete_child,
- list,
- read_prop,
- write_prop,

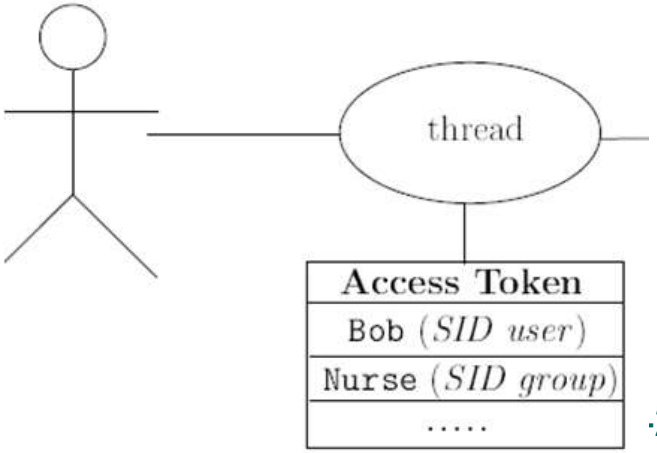


The Windows Equivalent of euid change with setuid()?

Windows have the concept of **impersonation token** (a second token).

Threads can have three access tokens:

- the primary access token (e.g. from parent process)
- the impersonation access token that contains the security context of a different user, can contain more privileges.
- a saved access token = like suid.



**The Windows Equivalent of set-uid privilege?

SE_IMPERSONATE_NAME in SACL

User Right: Impersonate a client after authentication.

A server program can run with an access token of the logged on user and calls two functions

- **ImpersonateLoggedOnUser**
- **RevertToSelf – uses the** saved access token…

Inheritance

In Windows NT, permissions propagate through inheritance.

Application:

1. for all sub-directories
2. for all sub-keys in the registry

Automatic propagation since Windows 2000, not in NT.