Computer Security and Privacy
Access Control

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Philippe Oechslin, George Danezis, Ninghui Li

Computer Security and Privacy
Access control Introduction

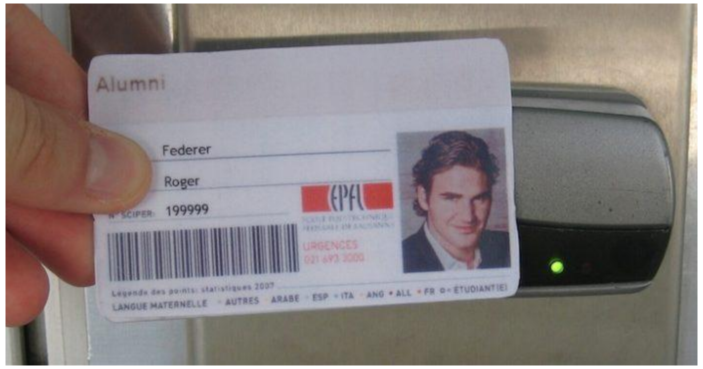
Carmela Troncoso

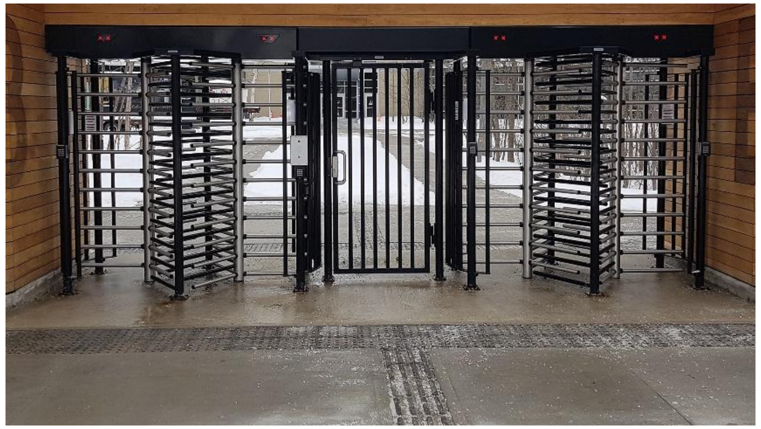
SPRING Lab

carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Philippe Oechslin, George Danezis, Ninghui Li

What is “access control”?





What is "digital" access control?

ACCESS CONTROL: Security mechanism that ensures that
"all accesses and actions on objects by principals are __WITHIN__ the security policy"

Example questions access control systems need to answer:

- Can Alice read file "`/users/Bob/readme.txt`"?
- Can Bob open a TCP socket to "`http://www.abc.com/`"?
- Can Charlie write to row 15 of the table GRADES?

✓

"authorized"
"has *permission"*

✗

"unauthorized"
"access denied"

Only events within the security policy

Why is so important to learn about access control?

Access control is the **first line of defense**. Thus, **it is used everywhere**

Applications

Online Social Networks, Email server, Cloud storage

Middleware

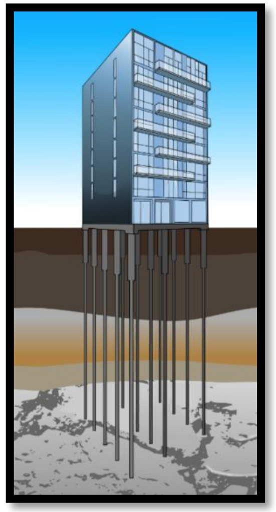
Databases Management Systems (DBMS)

Operating System

control access to files, directories, ports,…

Hardware

Memory, register, privileges



Where does access control (usually) fit?

Authentication (later in course)

Authorization

The system needs to bind the actor to a **principal** before authorization

abstract entity that is authorized to act

(users, connections, processes)

The system needs to decide whether the **principal** is authorized

Access control

The mechanisms that do authentication and authorization are in the **Trusted Computing Base**!

Implementing access control

What NOT to do: “Checks soup”

- All over the program, add checks
    - implementing the decision in-line based on the policy

```
#some code that needs to access file3.txt

if (action == read) and ((userID == Alice) or (userID == Bob) :
    open(file3.txt, ‘r’)
elif (action == write) or (userID == Bob) then:
    open(file3.txt,’w’)
else:
    print(“The user does not have access to file3.txt”)
```

Implementing access control

What you SHOULD DO: Systematic calls to "reference monitor"

- All over the program add checks that call the monitor
    - Checks authorisation required, and provide evidence as to the principals and objects
    - "Central" subsystem establishes whether the checks pass or not

Apache Shiro

https://shiro.apache.org

```
if ( subject.isPermitted("user:delete:jsmith") ){
    //delete the 'jsmith' user
} else {
    //don't delete 'jsmith'
}
```

Least common mechanism??

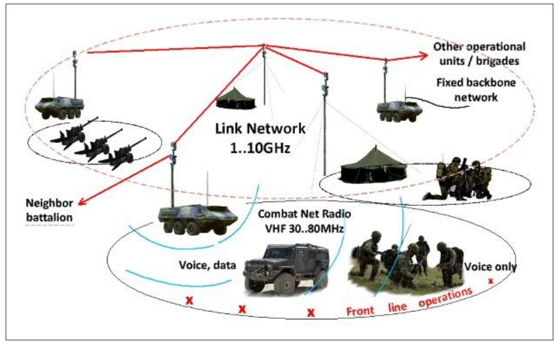
Who decides the access policy? Two approaches

Discretionary Access Control (DAC)

- ***Object owners** assign permissions*
- Ownership of resources
  - Windows, Linux, macOS
  - Social Networks

Mandatory Access Control (MAC)

- ***Central security policy** assigns permissions*
- Organizations with need for central controls
  - Military – focus on confidentiality
  - Hospital environment – focus on confidentiality and integrity
  - Banking – focus on integrity



Computer Security (COM-301)
Discretionary Access control

Carmela Troncoso

SPRING Lab

carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Philippe Oechslin, George Danezis, Ninghui Li

Implementing Discretionary Access Control

- ***Object owners** assign permissions*
- Ownership of resources
    - Windows, Linux
    - Social Networks

How??

Permissions establish which subjects can access which objects

But in a system there are many subjects and objects.

There can be many subjects, many objects, and many combinations of permissions combining subjects and objects, how can we handle?

Discretionary Access Control policies are often conceptualized as an **Access Control Matrix**

The Access Control Matrix

ACCESS CONTROL MATRIX: an *abstract* representation of all **permitted** triplets of (subject, object, access right) within a system

Subjects (principals): entity within an IT system

a user, a process, a service

Objects(assets): resources that (some) subject may access or use

a file, a folder, a row in a database, the system's memory, a machine in the network, a printer, a page in a website

Operations: in abstract, subjects can observe and/or alter objects

read, write, append, execute

B. Lampson. Protection. Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems Rev. 8, 1 (Jan. 1974), pp 18-24.

Access Control Matrix - Example

S … Alice, Bob

O … file1, file2, file3

A … read, write

Can Alice read file1?
Can Bob write file1?
Can file3 be written by Alice?

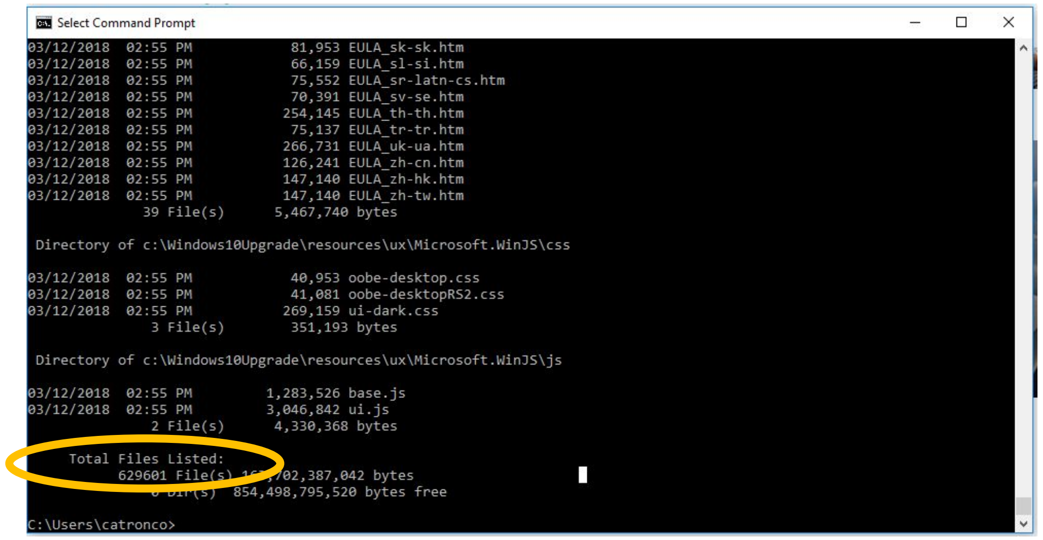
Access control matrix:

	file1	file2	file3
Alice	read write		read
Bob		read write	read write

The Access Control Matrix **is an abstract concept**

Not suitable for direct implementation!

what if there are thousands of files or hundreds of users?



```
03/12/2018  02:55 PM            81,953 EULA_sk-sk.htm
03/12/2018  02:55 PM            66,159 EULA_sl-si.htm
03/12/2018  02:55 PM            75,552 EULA_sr-latn-cs.htm
03/12/2018  02:55 PM            70,391 EULA_sv-se.htm
03/12/2018  02:55 PM           254,145 EULA_th-th.htm
03/12/2018  02:55 PM            75,137 EULA_tr-tr.htm
03/12/2018  02:55 PM           266,731 EULA_uk-ua.htm
03/12/2018  02:55 PM           126,241 EULA_zh-cn.htm
03/12/2018  02:55 PM           147,140 EULA_zh-hk.htm
03/12/2018  02:55 PM           147,140 EULA_zh-tw.htm
              39 File(s)      5,467,740 bytes

 Directory of c:\Windows10Upgrade\resources\ux\Microsoft.WinJS\css

03/12/2018  02:55 PM            40,953 oobe-desktop.css
03/12/2018  02:55 PM            41,081 oobe-desktopRS2.css
03/12/2018  02:55 PM           269,159 ui-dark.css
               3 File(s)        351,193 bytes

 Directory of c:\Windows10Upgrade\resources\ux\Microsoft.WinJS\js

03/12/2018  02:55 PM         1,283,526 base.js
03/12/2018  02:55 PM         3,046,842 ui.js
               2 File(s)      4,330,368 bytes

     Total Files Listed:
          629601 File(s) [illegible],702,387,042 bytes
               [illegible]  854,498,795,520 bytes free

C:\Users\catronco>
```

629601

O(f·u)

1 bit per file, 1 user	78KB
3 bits per file, 1 user	236KB
3 bits per file, 10 users	2.36MB

Two more issues:

usually very sparse – extremely inefficient

error prone – hard to have a global view

Access Control Lists (ACLs)

Associate permissions to **objects**

	file1	file2	file3
Alice	read write		read
Bob		read write	read write

Notice blanks are not stored!!

Permissions are associated to **objects**

```
file1: {(Alice,read/write)}
file2: {(Bob, read/write)}
file3: {(Alice,read),(Bob,read/write)}
```

Access Control Lists (ACLs)

Associate permissions to **objects**

	file1	file2	file3
Alice	read write		read
Bob		read write	read write

Notice blanks are not stored!!

Permissions are associated to **objects**

```
file1: {(Alice,read/write)}
file2: {(Bob, read/write)}
file3: {(Alice,read),(Bob,read/write)}
```

✓

can store close/with the resource
easy to determine who can access a resource
easy to revoke rights by resource

✗

difficult to check at runtime
difficult to audit all rights of a user
difficult to remove all permissions from a user
(better remove authentication!)
difficult delegation

Role Based Access Control (RBAC)

Systems have too many subjects! that come and go!

Large dynamic ACLs ☹

Subjects are similar to each other: assign same rights

e.g., a doctor has the same privileges as another doctor

1) assign permissions to *roles*
2) assign *roles* to subjects
3) subjects select an active *role* – they have the permissions of the active role

RBAC Problems

Problem 1: Role Explosion

- Temptation to create fine grained roles, denying benefits of RBAC

Problem 2: Simple RBAC has limited expressiveness

- Problems with implementing least privilege
- Some roles are relative: “Carmela's Doctor” vs. “Any Doctor”

Problem 3: Difficult to implement separation of duty

- “Two doctors are needed to authorize a procedure”
- RBAC Mechanism needs to ensure they are distinct!

Group Based Access Control

Systems have too many subjects! that come and go!

Large dynamic ACLs ☹

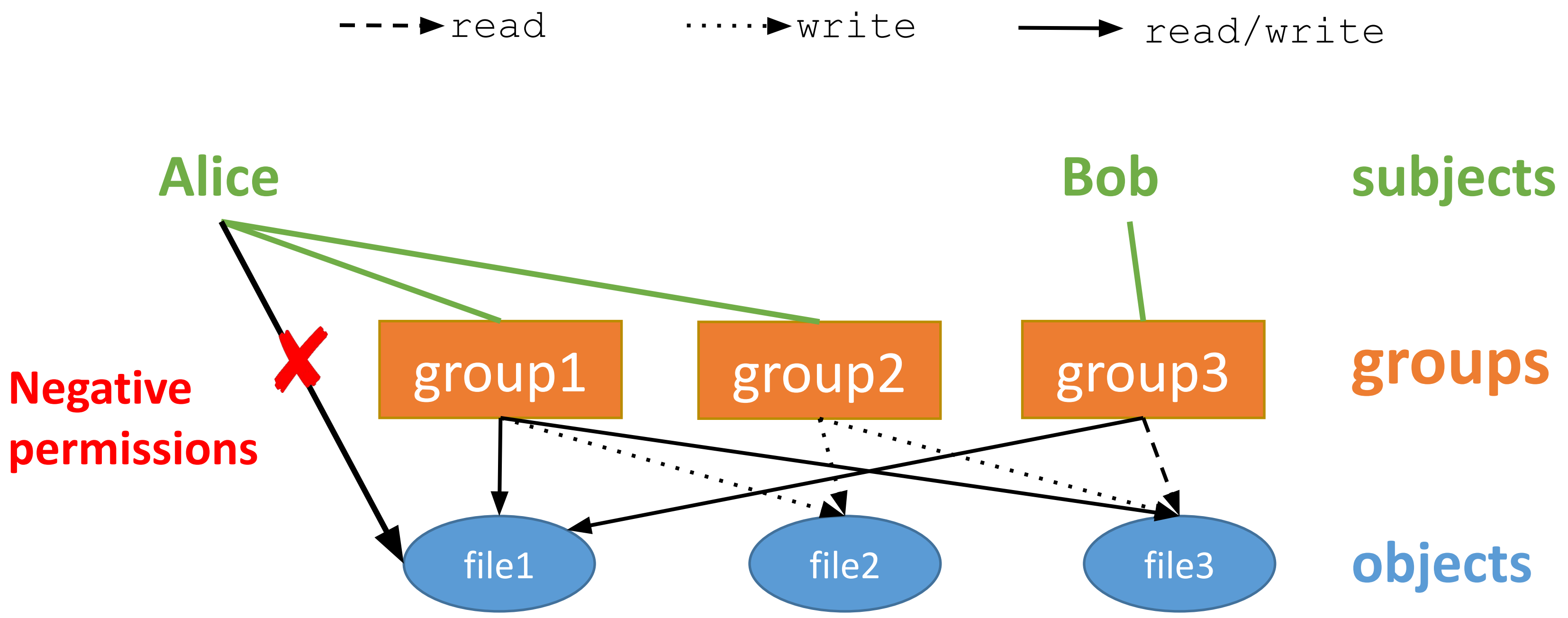
Observation: Some permissions are always needed together

e.g., access to sockets and network interface always go hand in hand

1) assign permissions to access objects to *groups*
2) assign subjects to *groups*
3) subjects have the permissions of *all* their groups

Group based access control

Negative permissions to implement fine-grained policies



What would you check first: Negative permissions or group permissions?

Capabilities

Associate permissions to **subjects**

	file1	file2	file3
Alice	read write		read
Bob		read write	read write

Notice blanks are not stored!!

Permissions associated to **subjects**

```
Alice:{(file1,read/write),(file3,read)}
Bob:{(file2,read/write),(file3,write)}
```

Mark S. Miller , Ka-Ping Yee, and Jonathan Shapiro. Capability myths demolished. Technical Report SRL2003-02, Johns Hopkins University Systems Research Laboratory, 2003

Capabilities

Associate permissions to **subjects**

	file1	file2	file3
Alice	read write		read
Bob		read write	read write

Permissions associated to **subjects**

```
Alice:{(file1,read/write),(file3,read)}
Bob:{(file2,read/write),(file3,write)}
```

Notice blanks are not stored!!

✓

can store with the subject (portable!)
easy to audit all subject permissions
delegating is “simple”

✗

revoking permission on one object is hard
transferability, once the capability is given
how can we prevent sharing?
authenticity, how to check?

Mark S. Miller , Ka-Ping Yee, and Jonathan Shapiro. Capability myths demolished. Technical Report SRL2003-02, Johns Hopkins University Systems Research Laboratory, 2003

A recurrent problem in access control

AMBIENT AUTHORITY is used by a subject if for an action to succeed it **only** needs to specify the **operation** and the **names** of the involved object(s)

In these cases the **subject** (with authority) is implicit

```
open("file1", "rw")
```

The program cannot check permissions!

(the subject is missing, it is understood it is the process owner)

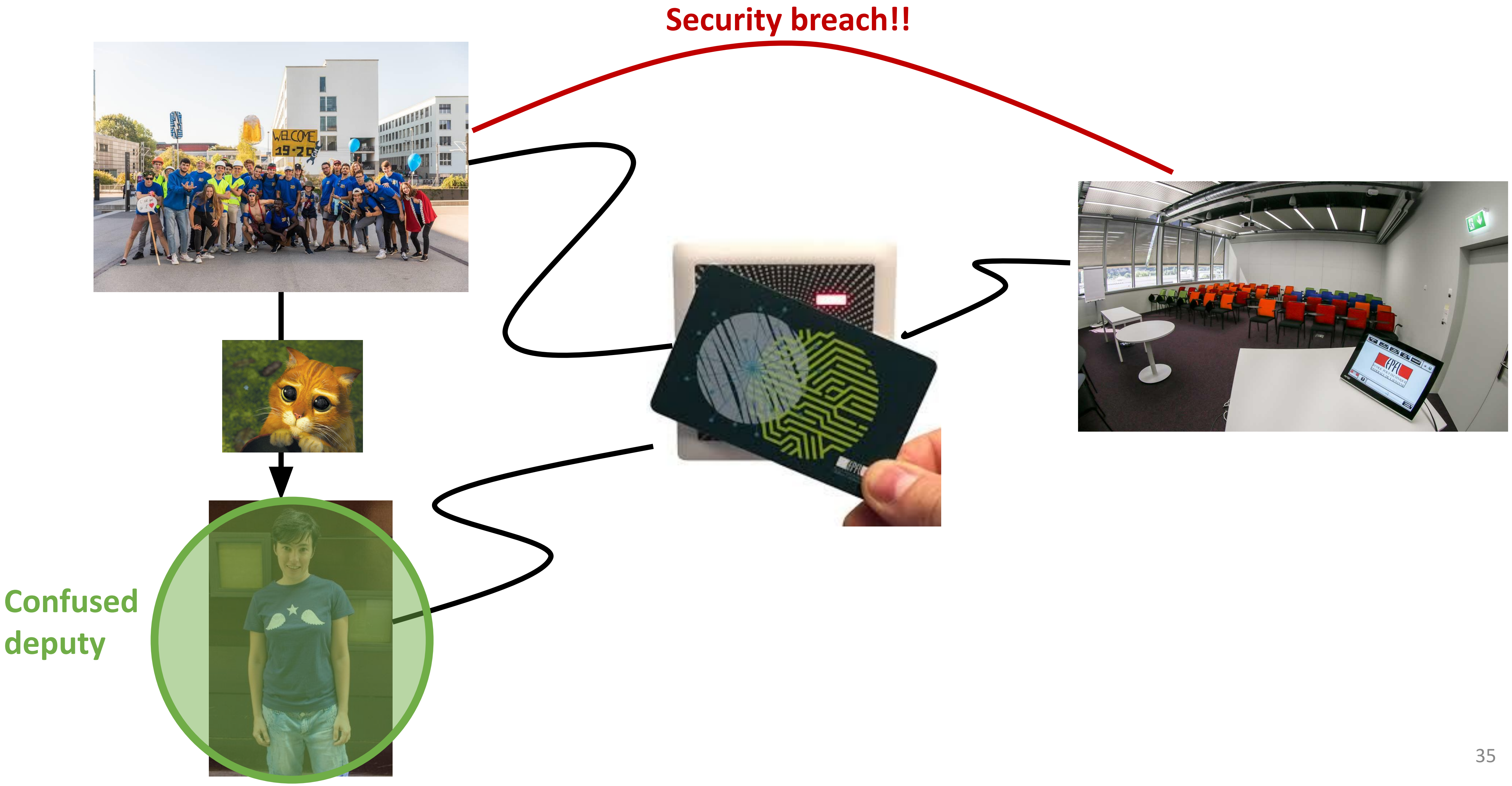
✓ no need to repeat the subject all the time (usability)

✗ least privilege becomes harder to enforce
confused deputy problem!

The confused deputy problem

Problem with ambient authority:
A privileged program can be tricked to misuse its authority
(Confused deputy problem)

The confused deputy problem



The confused deputy

PAY-PER-USE COMPILER

- Compiler receives (`input,output`)
- Compiles the program `input` and:
    - writes usage into `bill`
    - writes errors into `output`

	`input`	`output`	`bill`
Alice	write	read	read
Compiler	read	read write	read write

```
ACL
input: {(Alice, write), (Compiler, read)}
output: {(Alice, read), (Compiler, read/write)}
bill: {(Alice, read), (Compiler, read/write)}
```

CAN ALICE CHANGE THE `bill`?

AND AVOID PAYING?

The confused deputy

Pay-per-use compiler

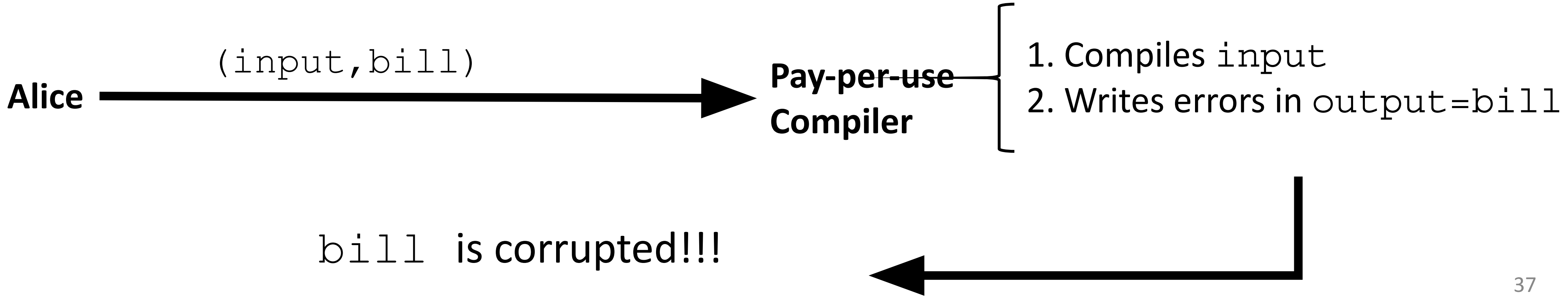
- Compiler receives (`input,output`)
- Compiles the program `input` and:
    - writes usage into `bill`
    - writes errors into `output`

	input	**output**	**bill**
Alice	write	read	read
Compiler	read	read write	read write

```
ACL
input: {(Alice, write), (Compiler, read)}
output: {(Alice, read), (Compiler, read/write)}
bill: {(Alice, read), (Compiler, read/write)}
```

Can Alice change the `bill`?

And avoid paying?



`bill` is corrupted!!!

How to avoid confused deputies

Real problem. Ambient authority is used for convenience in many real systems, OS services, web servers,…

Solutions:

1) Re-implement access control in the privileged process
2) Let privileged process check authorization for Alice.
3) Capabilities can help!

In the previous example…

- Compiler has capabilities to the file `bill`.
- To compile Alice must give access to the debugging file `output`
  - Cannot give a capability for writing on `bill`!
  - Cannot confuse anyone!

Summary

Discretionary Access Control: owners establish permissions

Conceptualized as an Access Control Matrix

Implemented in two flavors:

- Access control list: permission associated to objects
- Capabilities: permission associated to owners

When relying on access control, it is always important to think about confused deputies

Computer Security and Privacy
Discretionary Access Control Examples: Linux & Windows

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Philippe Oechslin, George Danezis, Ninghui Li

Discretionary Access Control in Real life

We saw that many of the systems we use nowadays rely on discretionary access control:

- Social networks
- Cloud file sharing systems
- **Operative systems**

Unix: Principals & Groups

- User Identities (UIDs) and Group Identities (GIDs)
  - Originally 16-bit (now 32-bit) numbers.
  - Special UIDs: -2, 0, 1, …
- User Information
  - Each user has own directory `/home/username`
  - User accounts: `/etc/passwd`

    ```
    username:password:UID:GID:info:home:shell
    ```
- Users belong to one or more groups
  - Primary group (`/etc/passwd`), other groups (`/etc/group`)

Security Architecture

- Everything is a file

- Each user “owns” a set of files

- Each file as a simple **A**ccess **C**ontrol **L**ist to express the access control policy to the file
  - System files are owned by special users that can make system operations

- All user processes run by a user run with that user’s privileges

  Ambient authority!!

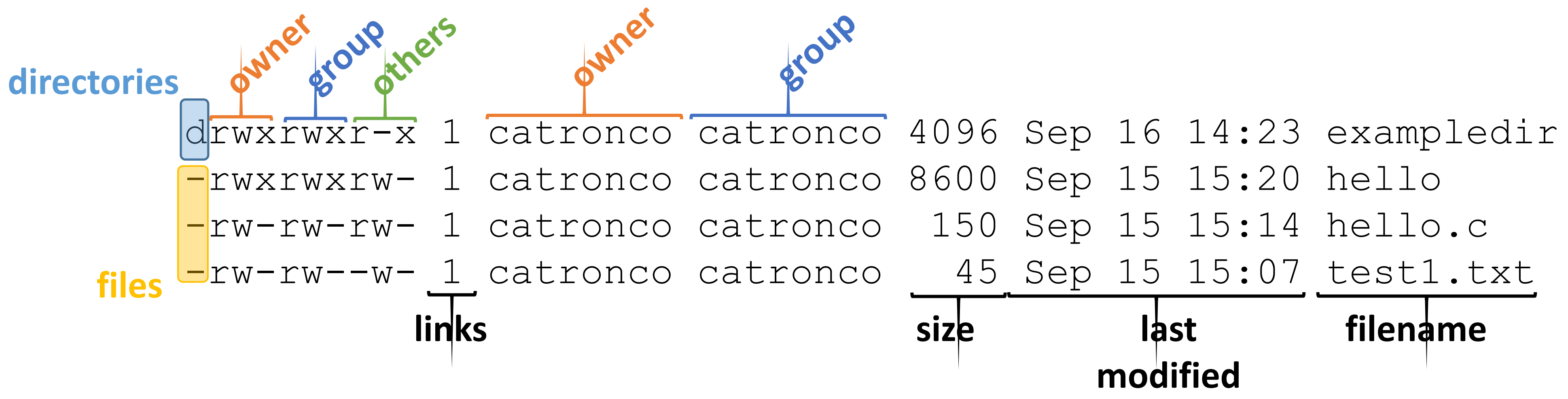
File Access Control Lists

- Files have **ACLs** attached to them
    - Each file is assigned an **owner UID** and **GID**
    - Each file has 9 permission bits
        – 3 actions: Read, write, execute
        – 3 subjects *owner, group, other*

- Different semantics between files and directories
    - *Directories*: Read → List files, Write → Add file, Exec → “cd”

- 3 attributes: “`suid`”, “`sgid`”, and “`sticky`”

Example of UNIX ACLs



```
drwxrwxr-x 1 catronco catronco 4096 Sep 16 14:23 exampledir
-rwxrwxrw- 1 catronco catronco 8600 Sep 15 15:20 hello
-rw-rw-rw- 1 catronco catronco  150 Sep 15 15:14 hello.c
-rw-rw--w- 1 catronco catronco   45 Sep 15 15:07 test1.txt
```

Owner can change permissions on files

chmod [+r, -w, / 666, 662 / +t or **1**666, +s or **4**666] filename

UNIX Access control in action

Compare:

`UID` / `GID` of process trying to perform action

with:

state of file (Owner, Group, mode bits)

Order matters in the comparison

1. If `UID` says you are owner: check bits for owner.
2. If not owner, but your group is owner, check `GID` with bits for group.
3. Otherwise check bits for “other”

`root` user is never denied access

Super users

Never login as root!

- Some distributions assign no password
- Use "`sudo`" or "`su`" command
- Difference?

  (`$ sudo su catronco`)

Special "`root`" user account

- User ID 0
- Access system files and special operations
- Can access anything: (almost all) security checks disabled
- `root` is in the **TCB**!!

Super users

Never login as root!

- Some distributions assign no password
- Use "`sudo`" or "`su`" command
- Difference?

(`$ sudo su catronco`)

Special "`root`" user account

- User ID 0
- Access system files and special operations
- Can access anything: (almost all) security checks disabled
- `root` is in the **TCB**!!

Normal users also need to access system services
but these services need to run with system privileges

`suid` / `sgid` mechanism

Special rights: `suid/sgid`

Setuid and setgid bits serve to indicate that a file is not run with the privileges of the launcher, but **with the privileges of the owner** user/group

Specially useful to run programs that require root, respecting the least privilege principle, e.g., to change a password:

```
ls -l /bin/passwd
-rwsr-xr-x. 1 root root 27768 Aug 20 2020 /bin/passwd
```

Special rights: `suid/sgid`

How do you know if a `suid` program does what it is meant to do? and only what it is meant to do?

```
-rwxr-xr-x 1 root root 3492656 Dec  4  2017 python2.7
```

$\xrightarrow{\quad}$ **Setuid Root programs are dangerous! (in TCB)**

⚠

Chen, H., Wagner, D., and Dean, D. "Setuid Demystified". In USENIX Security Symposium 2002
Kamp, P.H., and Watson, R.N. "Jails: Confining the omnipotent root". Proceedings of SANE 2000

Special rights: `sticky bit`

“Restricted deletion bit” (chmod +t)

Directories:

prevents unprivileged users from removing or renaming a file in the directory
unless they own the file

Example: /tmp folder. Users can only edit their own files

Files:

historically prevented program from being moved from swap for fast load
current: linux ignores the bit

Special rights: Nobody

Special user (User ID -2)

- owns no files
- belongs to no user

- Safer user to execute code you do not know, particularly obfuscated code
- Limits damages if they misbehave / get compromised

What about Windows?

Principals = users, machines, groups,…

Objects = files, Registry keys, printers, …

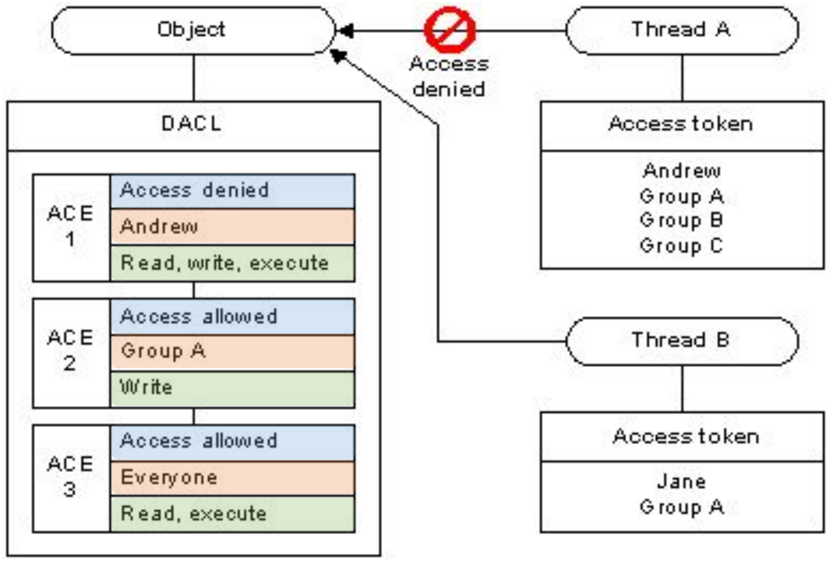
Access control:

- Each object has a discretionary access control list (DACL)
- Each process (or thread) has an access token with
  - Login user account (process "runs as" this user)
  - All groups of which the user is a member(recursively!)
  - All privileges assigned to these groups

Compare DACL with the process' access token when creating a handle to the object

What about Windows? DACL

List of Access Control Entries (ACEs)



Why negative first?

🟦 **Type**: negative / positive

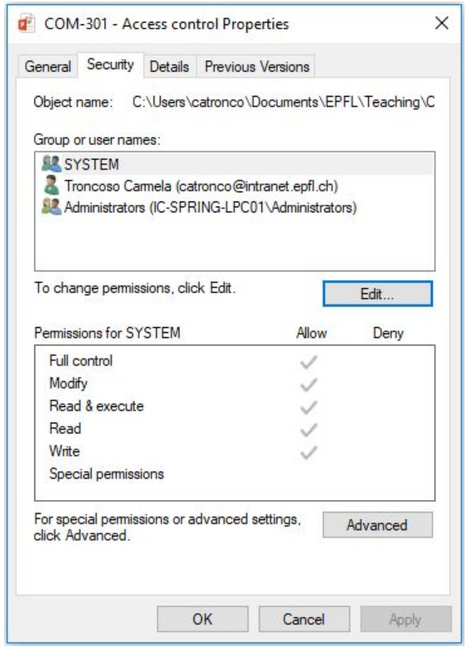
🟧 **Principal**

🟩 **Permissions:** more fine grained than UNIX

+ Flags and others…

Least Privilege by default

Run as administrator


COM-301 - Access control Properties

General | Security | Details | Previous Versions

Object name: C:\Users\catronco\Documents\EPFL\Teaching\C

Group or user names:

SYSTEM
Troncoso Carmela (catronco@intranet.epfl.ch)
Administrators (IC-SPRING-LPC01\Administrators)

To change permissions, click Edit. Edit...

Permissions for SYSTEM	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced. Advanced

OK Cancel Apply

https://docs.microsoft.com/en-us/windows/desktop/secauthz/dacls-and-aces