Capture the flag @ EPFL

What are CTFs about?

Learn about security

Competitive challenges

Team sport

Develop skills

Understand attacks

Write better code

Join polygl0ts

http://ctf.epfl.ch

Ask Sandra!

Last week

Properties of a computer system must hold
in presence of a resourced **strategic adversary**

THREAT MODEL: what are the resources available to the adversary?

SECURITY MECHANISM: Technical mechanism used to ensure that the security policy is not violated by an adversary <u>within the threat model</u>.

SECURITY ARGUMENT: rigorous argument that the security mechanisms in place are indeed effective in maintaining the security policy (*verbal* or *mathematical*).

<u>Subject to the assumptions of the threat model.</u>

Last week

Principles: Cheat Sheet

1. Economy of mechanism
2. Fail-safe defaults
3. Complete mediation
4. Open Design
5. Separation of Privilege
6. Least Privilege
7. Least Common Mechanism
8. Psychological Acceptability

Last week

Principles: Cheat Sheet

1. Economy of mechanism
2. Fail-safe defaults
3. Complete mediation
4. Open Design
5. Separation of Privilege
6. Least Privilege
7. Least Common Mechanism
8. Psychological Acceptability

Trusted Computing Base (TCB)
Reference monitor

Last week

Principles: Cheat Sheet

1. Economy of mechanism
2. Fail-safe defaults
3. Complete mediation
4. Open Design
5. Separation of Privilege
6. Least Privilege
7. Least Common Mechanism
8. Psychological Acceptability

Trusted Computing Base (TCB)
Reference monitor

2 extra principles
+ Work Factor
+ Compromise Recording

Two extra principles from physical security
9 - Work factor

“Compare the cost of circumventing the mechanism with the resources of a potential attacker” [SS75]

Two extra principles from physical security
9 - Work factor

DIFFICULT TO TRANSPOSE TO COMPUTER SECURITY!!

"Compare the cost of circumventing the mechanism with the resources of a potential attacker" [SS75]

Two extra principles from physical security
9 - Work factor

DIFFICULT TO TRANSPOSE TO COMPUTER SECURITY!!

“Compare the cost of circumventing the mechanism with the resources of a potential attacker” [SS75]

It helps **refining** the threat mode!

Difficult to quantify

Defining **cost?**

- cost of compromising insiders?
- cost of finding a bug?
- monetization?

Two extra principles from physical security
10 - Compromise recording

“Reliably record that a compromise of information has occurred [...] in place of more elaborate mechanisms that completely prevent loss” [SS75]

Two extra principles from physical security
10 - Compromise recording

DIFFICULT TO TRANSPOSE TO COMPUTER SECURITY!!

“Reliably record that a compromise of information has occurred [...] in place of more elaborate mechanisms that completely prevent loss” [SS75]

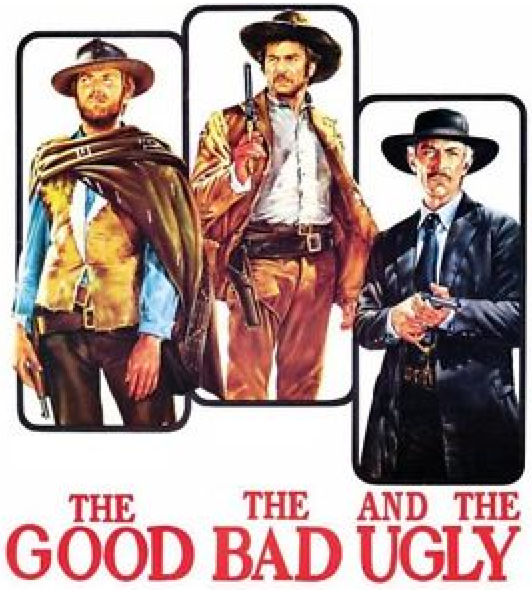
Two extra principles from physical security
10 - Compromise recording

DIFFICULT TO TRANSPOSE TO COMPUTER SECURITY!!

"Reliably record that a compromise of information has occurred [...] in place of more elaborate mechanisms that completely prevent loss" [SS75]

Keep **tamper-evidence logs**

May enable recovery (integrity)



Logs **are not magic**:

- What if you cannot recover? (Confidentiality)
- How to keep integrity? (Blockchain!)
- Logs may be a vulnerability (Privacy)?
- Logging the log? (Availability)

Detecting the compromise may be difficult (or expensive)

Systematic secure system engineering

1.- High-level specification

- Define the **architecture** of the system! (high level block diagram)
- Define the **security policy** (principals, assets, security properties)
- Define the **threat model**

2.- Security design

- Select / Design **security mechanisms**
- State your **security argument**: which controls maintain which properties?

3.- Secure implementation

- **Implement** mechanisms
- Ensure they **conform** to the design model
- Security **testing**

Systematic secure system engineering

1.- High-level specification

- Define the **architecture** of the system! (high level block diagram)
- Define the **security policy** (principals, assets, security properties)
- Define the **threat model**

Threat model != TCB

2.- Security design

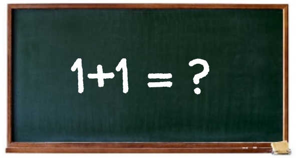
- Select / Design **security mechanisms**
- State your **security argument**: which controls maintain which properties?

3.- Secure implementation

- **Implement** mechanisms
- Ensure they **conform** to the design model
- Security **testing**

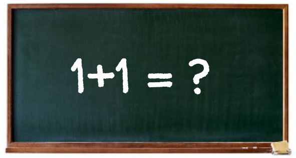
Systems are big! Need security mechanism<u>S</u>

If only composition was linear…



Systems are big! Need security mechanismS

If only composition was linear…

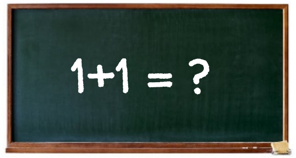


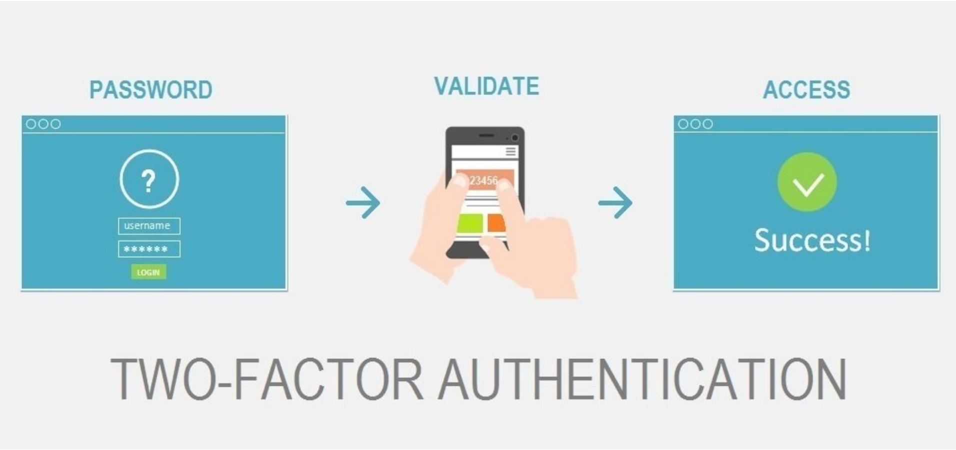
Defence in depth
As long as one remains
Security policy ✔

Weakest Link
If any one fails
Security policy ✘

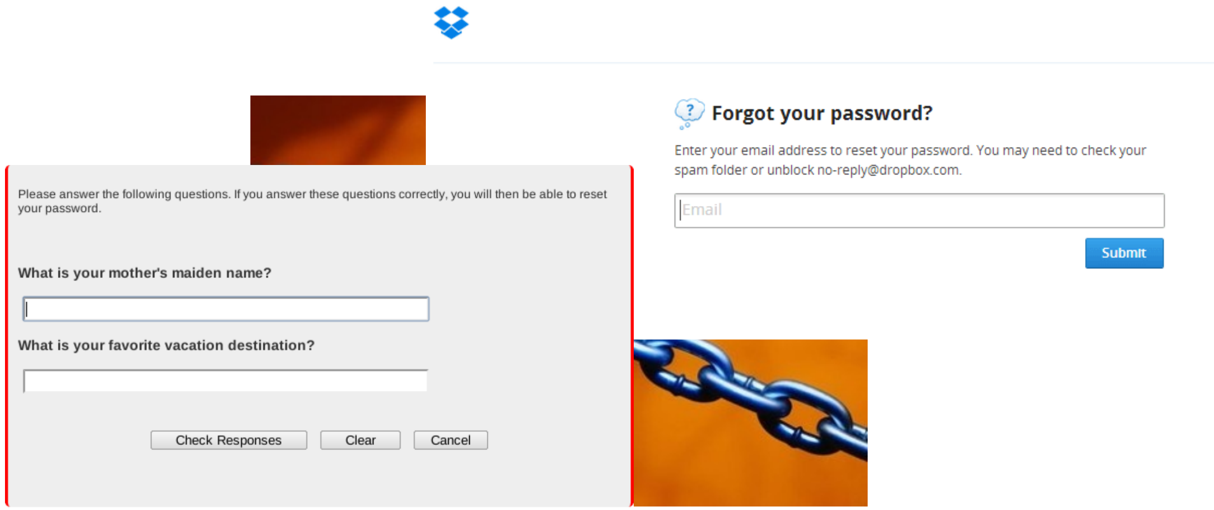
Systems are big! Need security mechanismS

If only composition was linear…





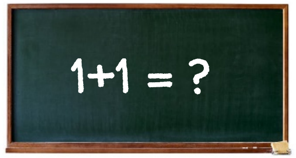
Defence in depth
As long as one remains
Security policy ✔

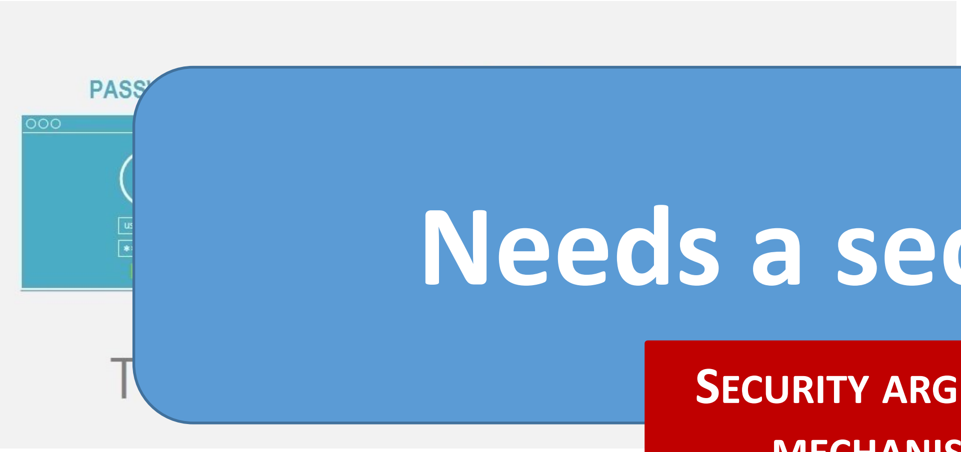


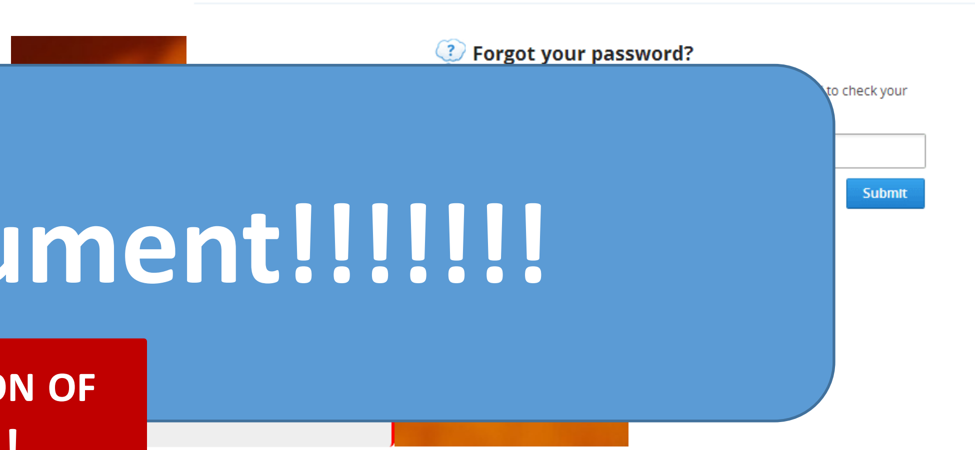
Weakest Link
If any one fails
Security policy ✘

Systems are big! Need security mechanismS

If only composition was linear…

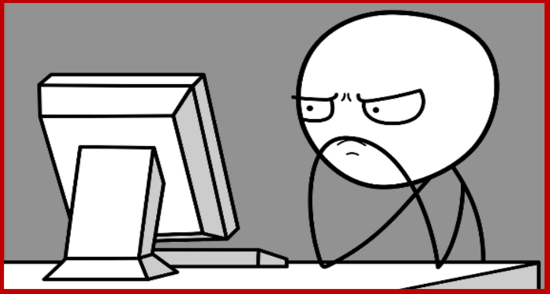






Needs a security argument!!!!!!!

SECURITY ARGUMENTS FOR COMPOSITION OF MECHANISMS IS A VERY HARD TASK!!



Defence in de
As long as one re
Security polic

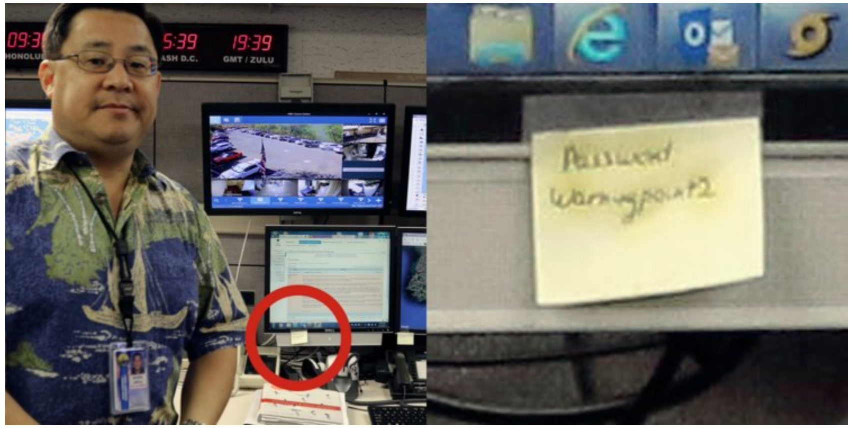
Weakest Link
If any one fails
Security policy ✘

Humans are also a weak link

Social Engineering

Weak passwords

Phishing attacks

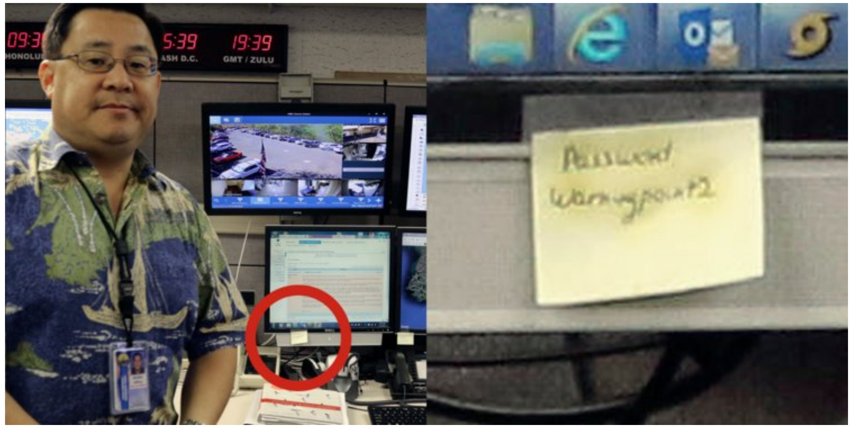


Humans are also a weak link

Social Engineering

Phishing attacks

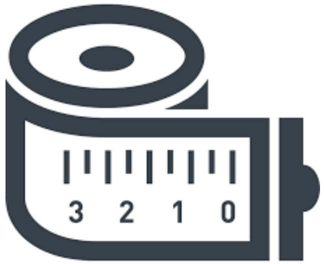
Weak passwords



It does not mean you should not care!!

http://www.slate.com/blogs/future_tense/2016/01/22/calling_humans_the_weakest_link_in_computer_security_is_dangerous.html

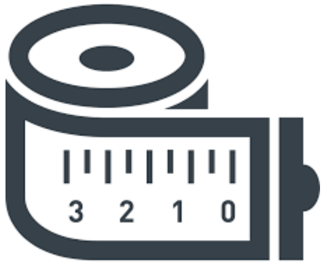
How secure is the system?
Worse Case vs. Average Case Security



How to measure the degree of protection afforded by a security system

open question!

How secure is the system?
Worse Case vs. Average Case Security



How to measure the degree of protection afforded by a security system

open question!

Worst case
worst user input / worst adversary

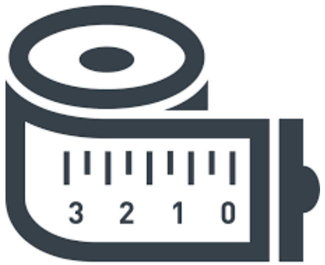
No assumptions on user behaviour in the security policy.

Strong guarantee

Pessimistic – low performance.

Cryptographic primitives

How secure is the system?
Worse Case vs. Average Case Security



How to measure the degree of protection afforded by a security system

open question!

Worst case
worst user input / worst adversary

No assumptions on user behaviour in the security policy.

Strong guarantee

Pessimistic – low performance.

Cryptographic primitives

Average Case
typical users / worst adversary

What is a typical user?

Which actions are more important to protect?

More fragile but better performance

Data anonymization

Computer Security (COM-301)
Access control

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

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

What is “access control”?

Security mechanism that ensures that
“all accesses and actions on system 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 table GRADES?

✓

“authorized”
“has *permission*”

✗

“unauthorized”
“access denied”

Only events within the security policy

What is “access control”?

Security mechanism that ensures that
“all accesses and actions on system 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 table GRADES?

Implementing this should be easy…

✔

“*authorized*”
“has *permission*”

✘

“*unauthorized*”
“*access denied*”

Only events within the security policy

Access control is everywhere

Operating System

control access to files, directories, ports,…

Middleware

Databases Management Systems (DBMS)

Hardware

Memory, register, privileges

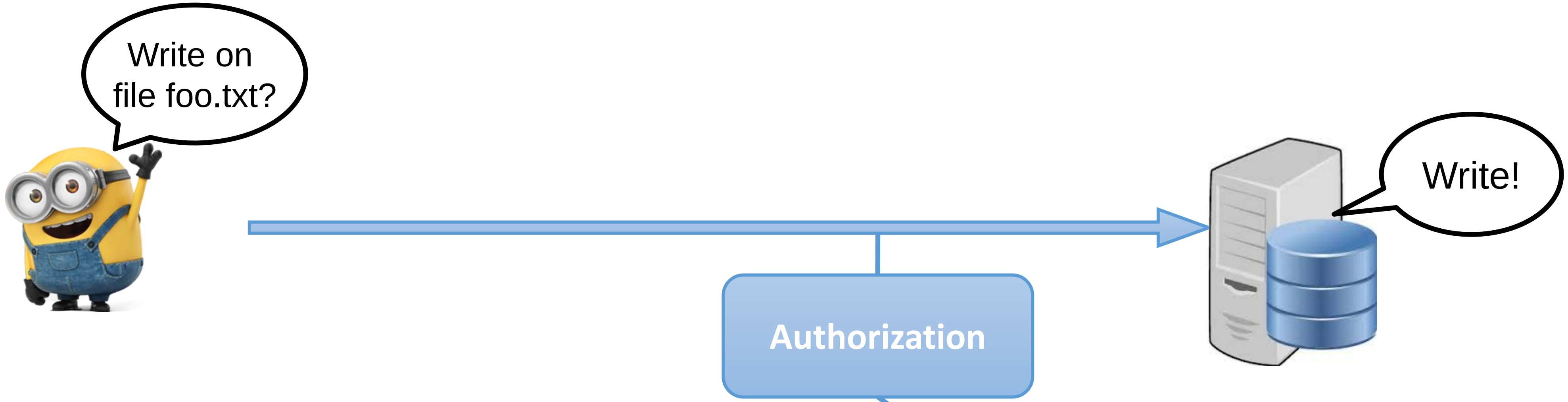
Applications

Online Social Networks

"Access control is the traditional center of gravity of computer security. It is where security engineering meets computer science"

Ross Anderson
Security Engineering

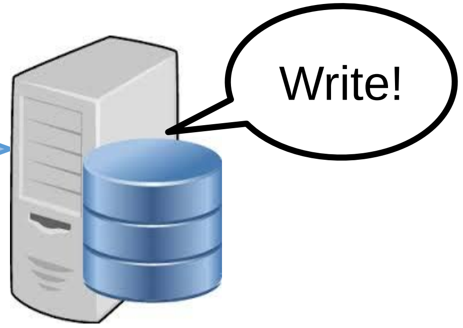
Where does access control (usually) fit?



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

Access control

Where does access control (usually) fit?



Authentication (week 5)

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

abstract entity that is authorized to act

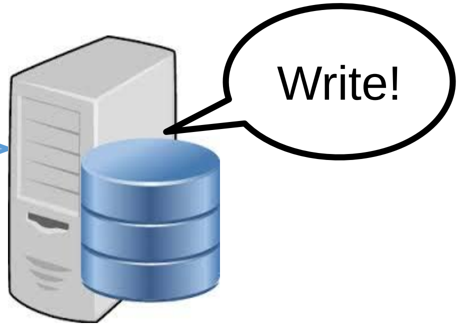
(users, connections, processes)

Authorization

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

Access control

Where does access control (usually) fit?



Authentication (week 5)

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

abstract entity that is authorized to act

(users, connections, processes)

Authorization

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

Access control

The mechanisms that do authentication and authorization are in the **TCB**!

Access Control Matrix

The **Access Control Matrix** represents all **permitted** triplets of (subject, action, access right)

S … set of subjects

O … set of objects

A … set of access operations

Access control matrix: $\boldsymbol{M} = (M_{so})_{s \in S, o \in O}$, $M_{so} \subseteq A$

M_{so} specifies the operations subject s may perform on object o

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.

A refresher on terminology

S … set of subjects

O … set of objects

A … set of access operations

Access control matrix: $\boldsymbol{M} = (M_{so})_{s \in S, o \in O}$, $M_{so} \subseteq A$

M_{so} specifies the operations subject *s* may perform on object *o*

***S* -** A **subject**: entity within an IT system

- *A user*
- *A process*
- *A service*

What Properties? – The security policy

A high level description of the **principals**, **assets** and **security properties** that must hold in the system.

- Principals (subjects): people, computer programs, services (entities that can be authenticated) *(may not contain the adversary)*

- Assets (objects): anything with value that needs to be protected.

- Properties: usually defined in relation to Principals + Assets

A refresher on terminology

What **Properties**? – The security policy

A high level description of the **principals**, **assets** and **security properties** that must hold in the system.

- **Principals (subjects)**: people, computer programs, services (entities that can be authenticated) *(may not contain the adversary)*

- **Assets (objects)**: anything with value that needs to be protected.

- **Properties**: usually defined in relation to Principals + Assets

S … set of subjects

O … set of objects

A … set of access operations

Access control matrix: $\boldsymbol{M} = (M_{so})_{s \in S, o \in O}$, $M_{so} \subseteq A$

M_{so} specifies the operations subject s may perform on object o

***S* -** A **subject**: entity within an IT system
- *A user*
- *A process*
- *A service*

***O* -** An **object** is a resource 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 website*

A refresher on terminology

What **Properties**? – The security policy

A high level description of the **principals**, **assets** and **security properties** that must hold in the system.

- **Principals (subjects)**: people, computer programs, services (entities that can be authenticated) *(may not contain the adversary)*

- **Assets (objects)**: anything with value that needs to be protected.

- **Properties**: usually defined in relation to Principals + Assets

S … set of subjects

O … set of objects

A … set of access operations

Access control matrix: $\boldsymbol{M} = (M_{so})_{s \in S, o \in O}$, $M_{so} \subseteq A$

M_{so} specifies the operations subject s may perform on object o

***S* -** A **subject**: entity within an IT system

- *A user*
- *A process*
- *A service*

***O* -** An **object** is a resource 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 website*

A – access operation ???

A refresher on terminology

What **Properties**? – The security policy

A high level description of the **principals**, **assets** and **security properties** that must hold in the system.

- **Principals (subjects)**: people, computer programs, services (entities that can be authenticated) *(may not contain the adversary)*

- **Assets (objects)**: anything with value that needs to be protected.

- **Properties**: usually defined in relation to Principals + Assets

S … set of subjects

O … set of objects

A … set of access operations

Access control matrix: $\boldsymbol{M} = (M_{so})_{s\in S, o \in O}$, $M_{so} \subseteq A$

M_{so} specifies the operations subject *s* may perform on object *o*

***S* -** A **subject**: entity within an IT system

- *A user*
- *A process*
- *A service*

***O* -** An **object** is a resource 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 website*

A – access operation ???

observe

alter

A refresher on terminology

What **Properties**? – The security policy

A high level description of the **principals**, **assets** and **security properties** that must hold in the system.

- **Principals (subjects)**: people, computer programs, services (entities that can be authenticated) *(may not contain the adversary)*
- **Assets (objects)**: anything with value that needs to be protected.
- **Properties**: usually defined in relation to Principals + Assets

S … set of subjects

O … set of objects

A … set of access operations

Access control matrix: $\boldsymbol{M} = (M_{so})_{s \in S, o \in O}$, $M_{so} \subseteq A$

M_{so} specifies the operations subject s may perform on object o

S - A **subject**: entity within an IT system

- *A user*
- *A process*
- *A service*

O - An **object** is a resource 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 website*

A – access operation ???

observe	**read**
	write/append
alter	**execute**

A refresher on terminology

What **Properties**? – The security policy

A high level description of the **principals**, **assets** and **security properties** that must hold in the system.

- **Principals (subjects)**: people, computer programs, services (entities that can be authenticated) *(may not contain the adversary)*
- **Assets (objects)**: anything with value that needs to be protected.
- **Properties**: usually defined in relation to Principals + Assets

S … set of subjects

O … set of objects

A … set of access operations

Access control matrix: $\boldsymbol{M} = (M_{so})_{s \in S, o \in O}$, $M_{so} \subseteq A$

M_{so} specifies the operations subject s may perform on object o

***S* -** A **subject**: entity within an IT system

- *A user*
- *A process*
- *A service*

Sometimes you will find Subject != Principal

For instance in UNIX documentation and forums:
- User: one or more principals (authenticated)
- Process: subject (not authenticated)

Windows / Java make different distinction!

t may access or use

- *A printer*
- *A website*

A – access operation ???

observe	**read**
	write/append
alter	**execute**

Access Control Matrix - Example

S … Alice, Bob

O … file1, file2, file3

A … read, write

Can Alice read file1?
Can Bob write file1?
Can Alice write file3?

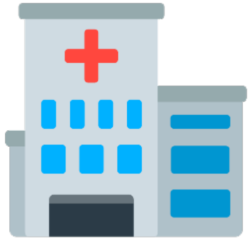
Access control matrix:

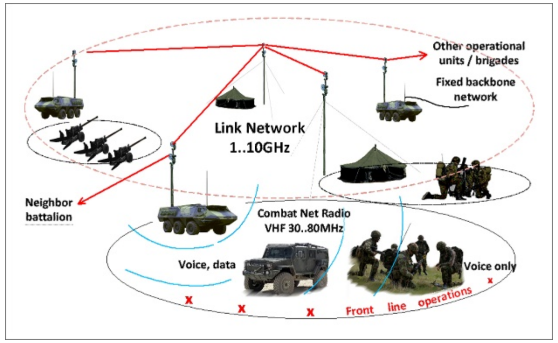
	file1	**file2**	**file3**
Alice	read write		read
Bob		read write	read write

Who sets the policy?
Mandatory vs Discretionary

Mandatory Access Control (MAC)

- ***Central security policy*** *assigns permissions*
- Organizations with need for central controls
  - Military (confidentiality, integrity)
  - Hospital environment (confidentiality, integrity)
  - Banking (integrity, confidentiality)
- (Week 3 Security Models)





Discretionary Access Control (DAC)

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

Beyond “static” Access Control Matrix

MAC: system-wide policy vs. **DAC**: the owners of objects set the permissions

Note access control matrix has two roles:

- Establish rights of **subjects** to perform actions on **objects**.
- Establish rights **subjects** can give to (or take from) other **subjects**

It can (**must**) change! But under which rules?

Access Control Matrix “Safety”

ACM is **NOT** the policy,
ACM is a **Security Mechanism**

The Access Control Matrix needs to implement the security policy

It **cannot** evolve in a way that violates the policy

There exist models to formalize the evolution of rights

- creates/destroy object/subject
- grant/transfer/delete right on object to/from subject
- check rights of subject

There exist models to reason about their safety

G. Scott Graham and Peter J. Denning. Protection: principles and practice. AFIPS '72.
M.A. Harrison, W.L. Ruzzo, and J.D. Ullman: Protection in Operating Systems. Communications of the ACM'76.

Implementing access control

Option 1: “Checks soup”

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

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

```
#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

Option 1: “Checks soup”

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

Problems

- How to update the policy?
- How to convince yourself the checks are correct?
- How to ensure no checks are missing?
- How to audit the policy?

Implementing access control

Option 1: “Checks soup”

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

Problems

- How to update the policy?
- How to convince yourself the checks are correct?
- How to ensure no checks are missing?
- How to audit the policy?

DO NOT DO THIS!!!

Implementing access control

Option 2: 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

Implementing access control

Option 2: 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’
}
```

Implementing access control

Option 2: 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

DO THIS!!!

Apache Shiro

https://shiro.apache.org

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

Implementing access control

Option 2: 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

DO THIS!!!

Apache Shiro

https://shiro.apache.org

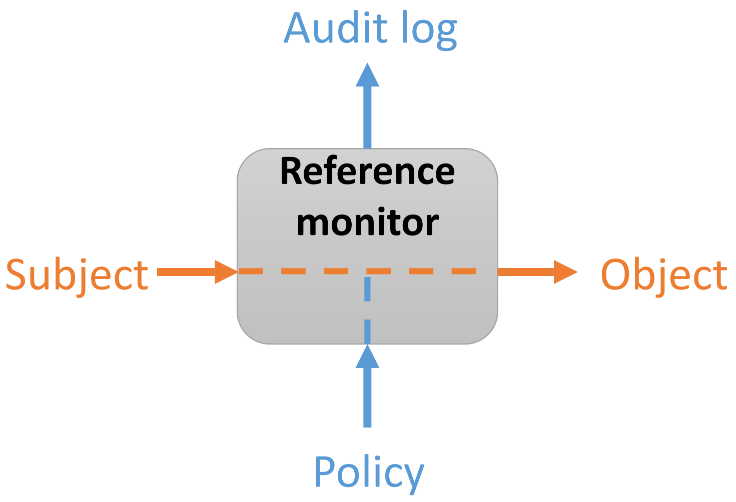
```
if ( subject.isPermitted("user:delete:jsmith") ){
    //delete the 'jsmith' user
} else {
    //don't delete 'jsmith'
}
```

Least common mechanism??

“Central” subsystem: The reference monitor

A system component (usually OS component) that enforces access control decisions

- Complete mediation
  - Principle 3 (week 1): “Every access to every object must be checked for authority”
- Tamper proof: adversary cannot influence it (in the TCB!)
- **Small!!!** to verify its correctness

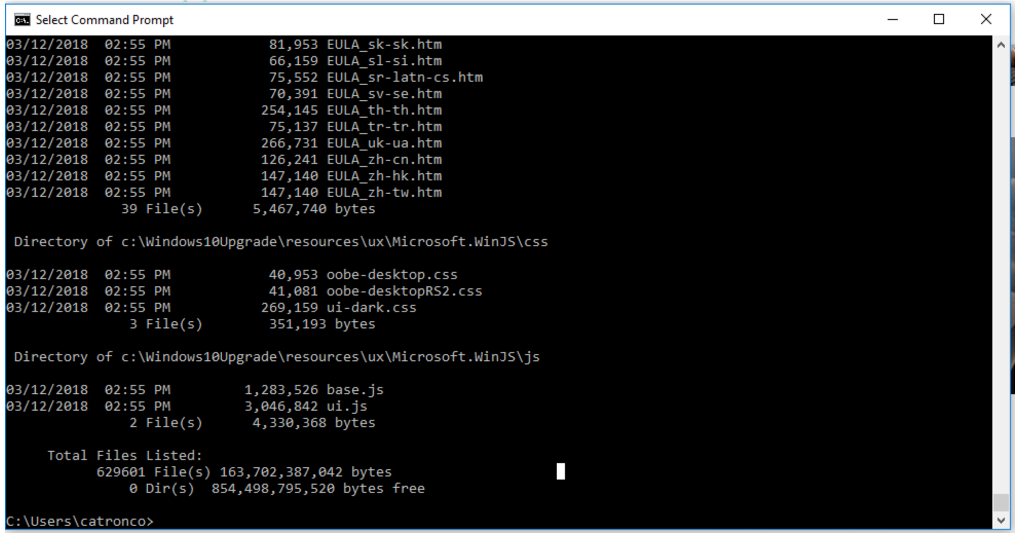


Anderson, J. 'Computer Security Technology Planning Study', ESD-TR-73-51, US Air Force Electronic Systems Division (1973).

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

Not suitable for direct implementation!

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



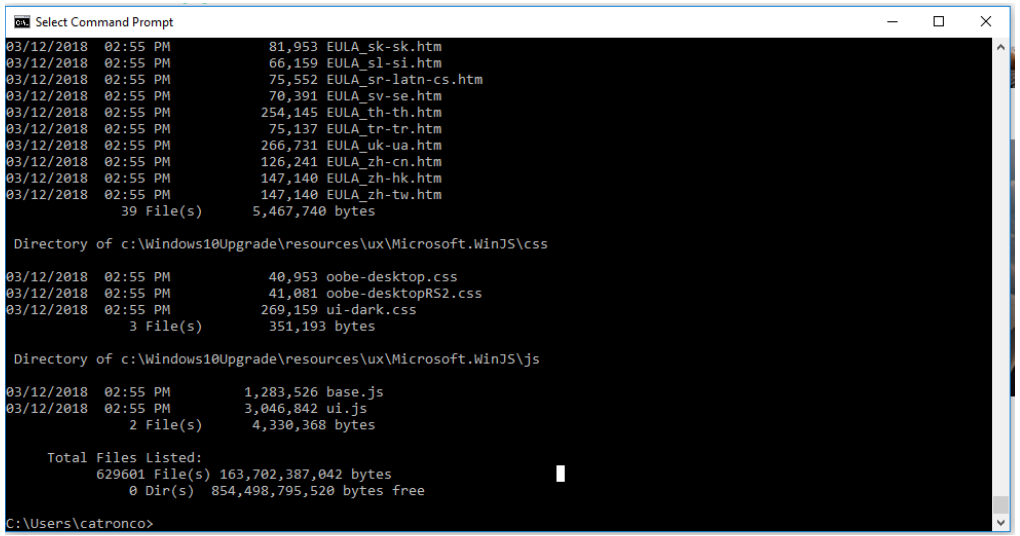
O(f·u)

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

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

Not suitable for direct implementation!

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



$O(f \cdot u)$

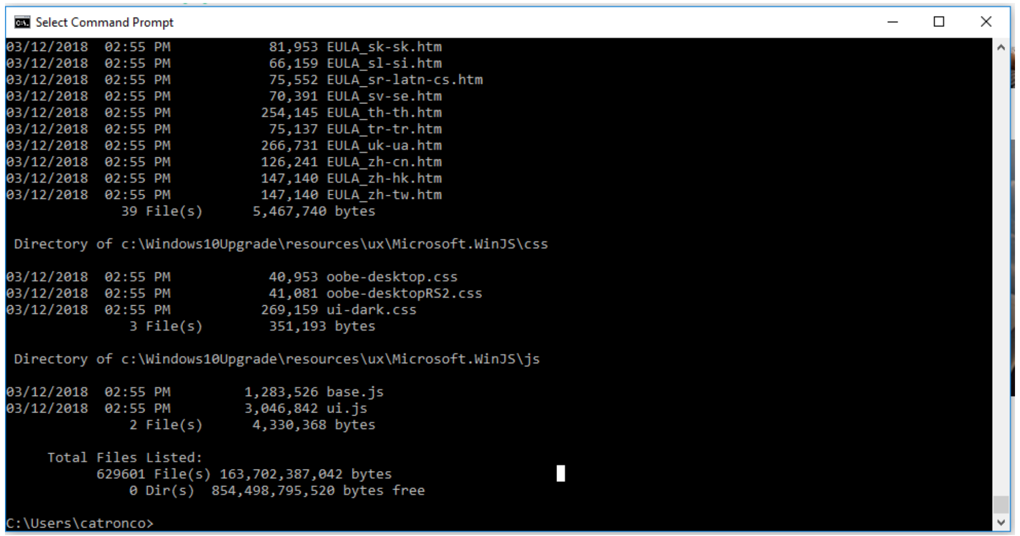
1 bit per file, 1 user	78KB
3 bits per file, 1 user	236KB
3 bits per file, 10 users	2.36MB

ERROR PRONE!

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

Not suitable for direct implementation!

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



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

ERROR PRONE!

- usually very sparse – extremely inefficient

How to “store” the Access Control Matrix?

(1) Store by column: “**Access control List**” (ACL)

	file1	file2	file3
Alice	read write		read
Bob		read write	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 delegation
difficult to remove all permissions from a user
(better remove authentication!)

How to “store” the Access Control Matrix?

(2) Store by row: “**Capability**”

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

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

✗ revoking permission on one object is hard

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

How to “store” the Access Control Matrix?

(2) Store by row: “**Capability**”

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

✓
can store with the user (portable!)
easy to audit all user 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?

Capabilities as tickets

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

Capability vs. ACL: Ambient Authority

A subject uses **ambient authority** if for an action to succeed it **only needs** to specify the **names** of the involved object(s) and the **operation** to be performed

The “principal” (authority) is implicit from some global property of process.

```
open(“file1”, “rw”)
```

(the subject is missing, but inferred from the process owner)

Capability vs. ACL: Ambient Authority

A subject uses **ambient authority** if for an action to succeed it **only needs** to specify the **names** of the involved object(s) and the **operation** to be performed

The "principal" (authority) is implicit from some global property of process.

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

The program cannot check permissions!

(the subject is missing, but inferred from the process owner)

Capability vs. ACL: Ambient Authority

A subject uses **ambient authority** if for an action to succeed it **only needs** to specify the **names** of the involved object(s) and the **operation** to be performed

The “principal” (authority) is implicit from some global property of process.

```
open(“file1”, “rw”)
```

The program cannot check permissions!

(the subject is missing, but inferred from the process owner)

✓ no need to repeat all the time the subject

✗ least privilege harder to enforce

confused deputy problem!

Which mechanism considers an ambient authority: ACL or Capabilities?

Capability vs. ACL: Ambient Authority

A subject uses **ambient authority** if for an action to succeed it **only needs** to specify the **names** of the involved object(s) and the **operation** to

The "principal" (authority) is implicit from some global property

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

The program cannot chec

(the subject is missing, but inferred from the process owner)

✓ no need to repeat all the time the subject

✗ least privilege harder to enforce

confused deputy problem!

ACL generally considers ambient authority, since permissions are usually checked for the user running the program (the ambient authority)

In **Capabilities** the capability itself contains the identity of the principal. Thus, there is no ambient authority

Which mechanism considers an ambient authority: ACL or Capabilities?

Capability vs. ACL: Ambient Authority

A subject uses **ambient authority** if for an action to succeed it **only needs** to specify the **names** of the involved object(s) and the **operation** to be performed

The "principal" (authority) is implicit from some global property of process.

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

The program cannot check permissions!

(the subject is missing, but inferred from the process owner)

✓ no need to repeat all the time the subject

✗ least privilege harder to enforce

confused deputy problem!

Which mechanism considers an ambient authority: ACL or Capabilities?

The confused deputy

Pay-per-use compiler
This program compiles files for users for a fee.
It works as follows:

- It receives (`&input`,`&output`)
    - `input`: file to compile
    - `output`: file to hold the compilation infor
- It compiles `&input` and returns the compiled executable to the user. It writes the compilation debugging information in `&output`.
- After compiling, it records who compiled in a file `/HOME/BILL` used to compute the users' bill

ACL

	`&input`	`&output`	`BILL`
Alice	write	read	read
Pay-per-use Compiler	read	read write	read write

The confused deputy

PAY-PER-USE COMPILER

- Compiler receives (`&input`,`&output`)
- Compiler writes stat compilation `/HOME/BILL`
- Compiler writes debugging info in `&output`

ACL

	`&input`	`&output`	`BILL`
Alice	write	read	read
Pay-per-use Compiler	read	read write	read write

CAN ALICE CHANGE HER BILL?

The confused deputy

PAY-PER-USE COMPILER

- Compiler receives (`&input,&output`)
- Compiler writes stat compilation `/HOME/BILL`
- Compiler writes debugging info in `&output`

ACL

	`&input`	`&output`	`BILL`
Alice	write	read	read
Pay-per-use Compiler	read	read write	read write

CAN ALICE CHANGE HER BILL?

AND AVOID PAYING?

The confused deputy

PAY-PER-USE COMPILER

- Compiler receives (`&input,&output`)
- Compiler writes stat compilation `/HOME/BILL`
- Compiler writes debugging info in `&output`

ACL

	`&input`	`&output`	`BILL`
Alice	write	read	read
Pay-per-use Compiler	read	read write	read write

CAN ALICE CHANGE HER BILL?

AND AVOID PAYING?

`(&input,&/HOME/BILL)`

Alice → **Pay-per-use Compiler**

Compiles &input
Writes debugging in
&output = &HOME/BILL

`/HOME/BILL is corrupted!!!`

How to avoid confused deputies

PAY-PER-USE COMPILER

- Compiler receives (`&input,&output`)
- Compiler writes stat compilation `/HOME/BILL`
- Compiler writes debugging info in `&output`

Real problem with ambient authority: system 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!

How to avoid confused deputies

Pay-per-use compiler

- Compiler receives (`&input,&output`)
- Compiler writes stat compilation `/HOME/BILL`
- Compiler writes debugging info in `&output`

Real problem with ambient authority: system 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!

- Compiler has capabilities to the stats file.
- To compile Alice must give access to the debugging file
  - Cannot give a capability to `/HOME/BILL`!
  - Cannot confuse anyone!

How to “store” the Access Control Matrix? Role Based Access Control (RBAC)

Problem with ACLs: too many subjects! that come and go!

Large dynamic ACLs ☹

R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-Based Access Control Models. IEEE Computer, 29(2):38--47, 1996

How to “store” the Access Control Matrix?
Role Based Access Control (RBAC)

Problem with ACLs: too many subjects! that come and go!

Large dynamic ACLs ☹

Observation: Subjects are similar to each other

a doctor has the same privileges as another doctor

- Assign Roles to subjects
- Subjects select an active role (implicit or explicit)
- Assign permissions to roles

Subject can only access a resource if they are taking a role that is permitted to access the resource

R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-Based Access Control Models. IEEE Computer, 29(2):38--47, 1996

How to “store” the Access Control Matrix?
Role Based Access Control (RBAC)

Problems with Role Based Access Control

Problem 1: Role Explosion

- Temptation to create fine grained roles, denying benefits of RBAC
- Not that small and simple

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

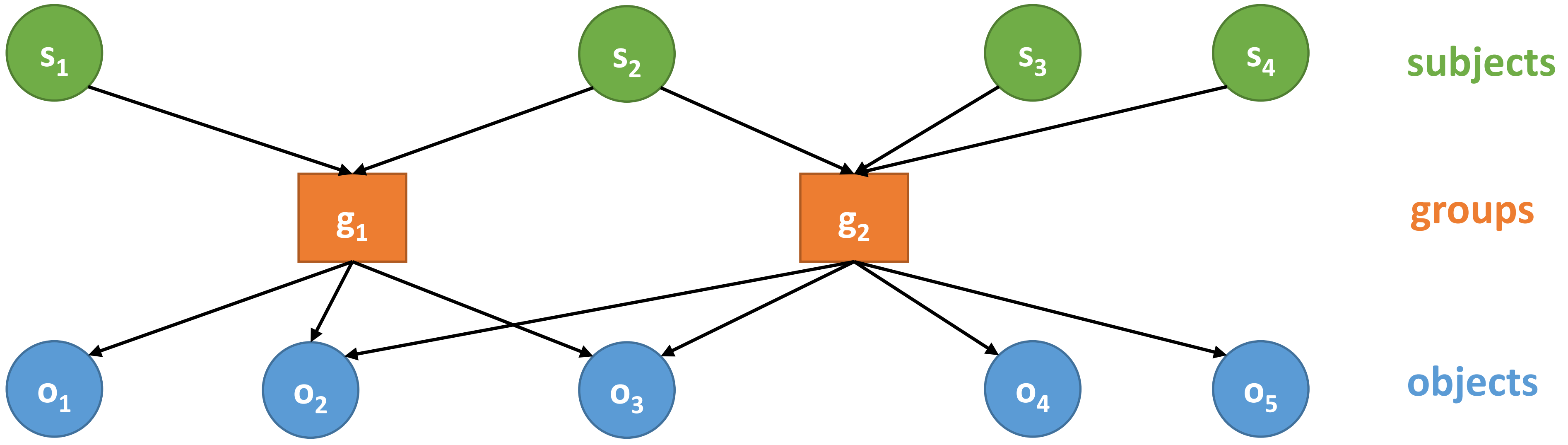
- “Two doctors are needed to authorize a procedure”
- RBAC Mechanism needs to ensure they are distinct!

How to "store" the Access Control Matrix?

Simplifying the matrix: Groups

- Cluster principals with similar access rights in **groups**
  - Users may belong to more than one group

- Give permissions to groups

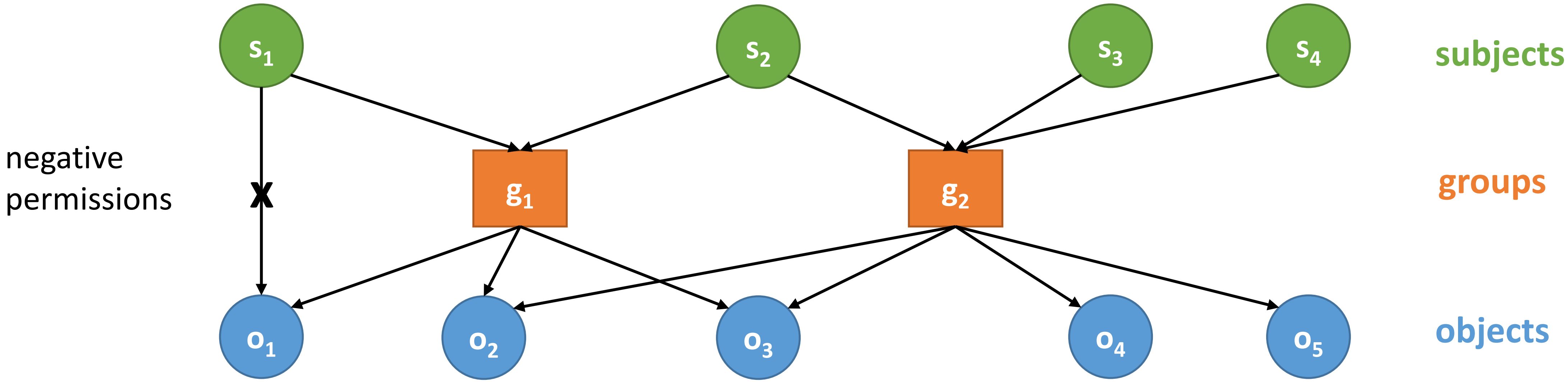


How to "store" the Access Control Matrix?

Simplifying the matrix: Groups

- Cluster principals with similar access rights in **groups**
    - Users may belong to more than one group

- Give permissions to groups

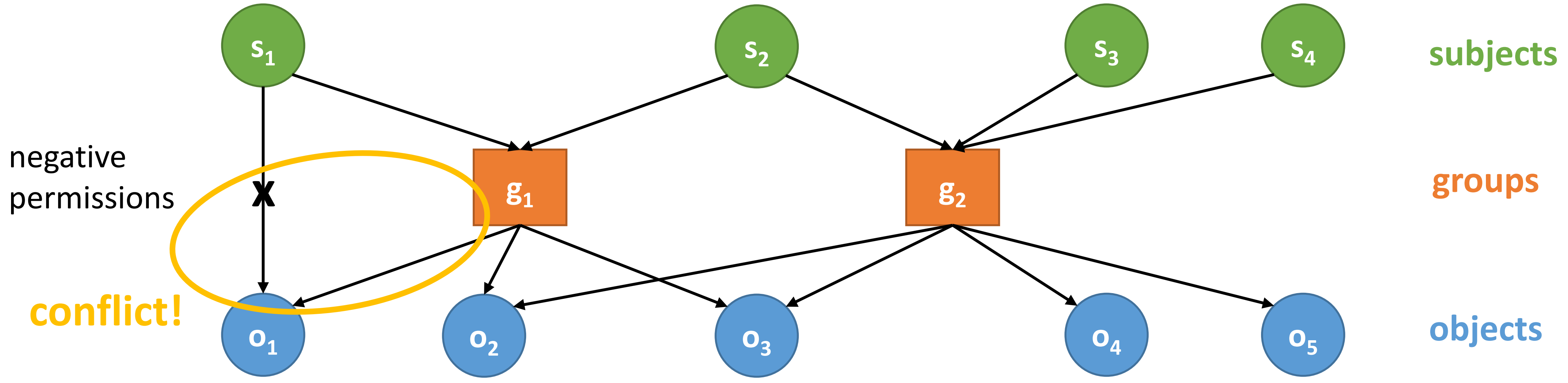


How to “store” the Access Control Matrix?

Simplifying the matrix: Groups

- Cluster principals with similar access rights in **groups**
  - Users may belong to more than one group

- Give permissions to groups



Access control example: UNIX / Linux
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`)

      `groupname:password:GID:userlist`

Group membership gives additional permissions!!

Access control example: UNIX / Linux Security Architecture

- Everything is a file

- Discretionary access control

- Each user “owns” a set of files
    - Simple way to express who else can access
    - All user processes run with that user privileges
    - Ambient authority!!

Access control example: UNIX / Linux Security Architecture

- Everything is a file

- Discretionary access control

- Each user "owns" a set of files
  - Simple way to express who else can access
  - **All user processes run with that user privileges**
  - Ambient authority!!

- Special super-users and programs

Access control example: UNIX / Linux Security Architecture

- Everything is a file

- Discretionary access control

- Each user “owns” a set of files
    - Simple way to express who else can access
    - **All user processes run with that user privileges**
    - Ambient authority!!

- Special super-users and programs

3 GROUPS

owner – owner of a file

group – grouper of the owner

other – rest

Access control example: UNIX / Linux Super users

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**!!

Access control example: UNIX / Linux
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**!!

Access control example: UNIX / Linux
Super users

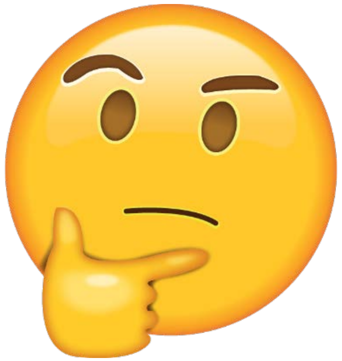
Never login as root!

- 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

Access control example: UNIX / Linux ACL = control bits

- Files have **ACLs** attached to them
    - Each file is assigned an **owner UID** and **GID**
    - Each file has 9 permission bits
        – Read, write, execute
        – User, group, others

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

- 3 attributes: "`suid`", "`sgid`", and "`sticky`"

Access control example: UNIX / Linux
ACL = control bits

- Files have **ACLs** attached to them
  - Each file is assigned an **owner UID** and **GID**
  - Each file has 9 permission bits
    - –Read, write, execute
    - –User, group, others

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

- 3 attributes: "`suid`", "`sgid`", and "`sticky`"

STICKY BIT
on a **directory**

only the owner of a file,
the owner of the directory,
or the super-user
will be able to remove or
rename a file/directory.

Example

directories owner group others owner group

```
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
```

files

links size last modified filename

Owner can change permissions on files

Example

directories owner group others owner group

```
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
```

files

links **size** **last modified** **filename**

Owner can change permissions on files

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

Access control example: UNIX / Linux
Access control implementation

Compare:

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

with:

state of file (Owner, Group, mode bits)

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

`root` user is never denied access

Access control example: UNIX / Linux
Why `suid/sgid` ?

Simple service: should deliver messages:

`$msg Alice “Hello Alice”`

The parameter sentence is appended to a file `msgfile` owned by Alice

Two options:

```
-rwx--x---  Alice Alice+Bob msg
-rwx------  Alice Alice+Bob msgfile

-rwx--x---  Alice Alice+Bob msg
-rwx-w----  Alice Alice+Bob msgfile
```

Access control example: UNIX / Linux
Why `suid/sgid` ?

Simple service: should deliver messages:

`$ msg Alice "Hello Alice"`

The parameter sentence is appended to a file `msgfile` owned by Alice

If Bob wants to send a message to Alice…

```
-rws--x---     Alice Alice+Bob msg
-rwx------     Alice Alice+Bob msgfile
```

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

Access control example: UNIX / Linux
Why `suid/sgid` ?

Simple service: should deliver messages:

`$msg Alice "Hello Alice"`

The parameter sentence is appended to a file `msgfile` owned by Alice

If Bob wants to send a message to Alice…

```
-rws--x---     Alice Alice+Bob msg
-rwx------     Alice Alice+Bob msgfile
```

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
```

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

Access control example: UNIX / Linux Nobody

Special user (User ID -2)

- owns no files
- belongs to no user

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

What about Windows?

1985 1990 1995 2001 2006-2009 2006 2012

Principals = users, machines, groups,…

Objects = files, Registry keys, printers, …

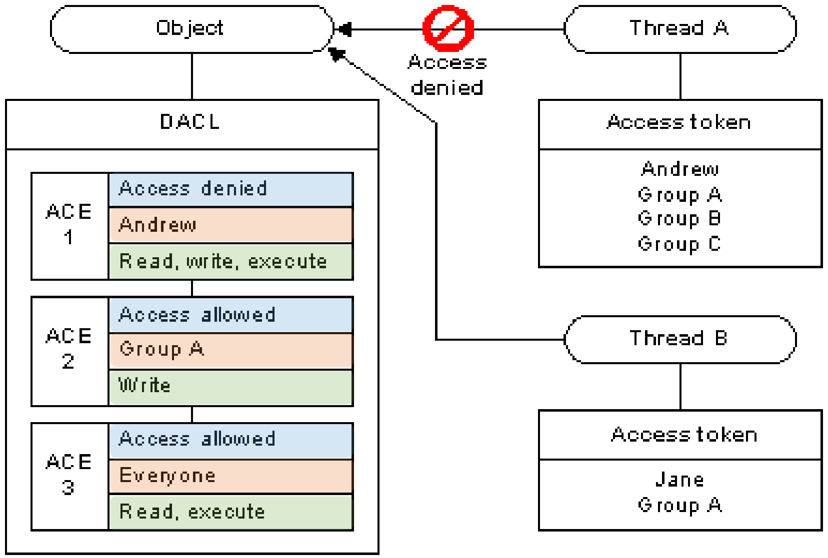
Access control:

- Each object has a discretionary access control list (DACL)
- Each process (or thread) has an access token:
  - Login user account (process "runs as" this user)
  - All groups in 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)



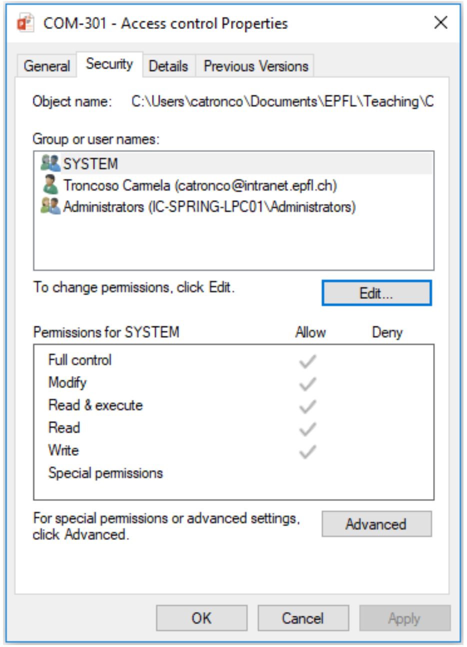
Type: negative / positive

Why negative first?

Principal

Permissions: more fine grained than UNIX

+ Flags and others…



Least Privilege by default

Run as administrator

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

Final note
Access control is domain specific!

Operating System

- Objects: files, devices, OS operations, ...
- Subjects: principals are processes, pipes, ...

Middleware

- Objects: tables, records, rows, columns, …
- Subjects: DB specific, e.g. stored in USERS table

Hardware

- Objects: Memory pages, privileged instructions
- Subjects: processor mode, protection domains

Applications

- Objects: Photos, posts, messages
- Subjects: users, groups

Final note
Access control is domain specific!

Operating System

- Objects: files, devices, OS operations, ...
- Subjects: principals are processes, pipes, ...

Middleware

- Objects: tables, records, rows, columns, …
- Subjects: DB specific, e.g. stored in USERS table

Hardware

- Objects: Memory pages, privileged instructions
- Subjects: processor mode, protection domains

Applications

- Objects: Photos, posts, messages
- Subjects: users, groups

Mixing domains is meaningless!!

OS access control cannot restrict access to a certain row of a Database.

but they build on top of each other:

OS access control required to restrict access to the *whole* DB file.

Final note
Access control is domain specific!

Operating System

- Objects

- Subject

Middleware

- Objects

- Subject

Hardware

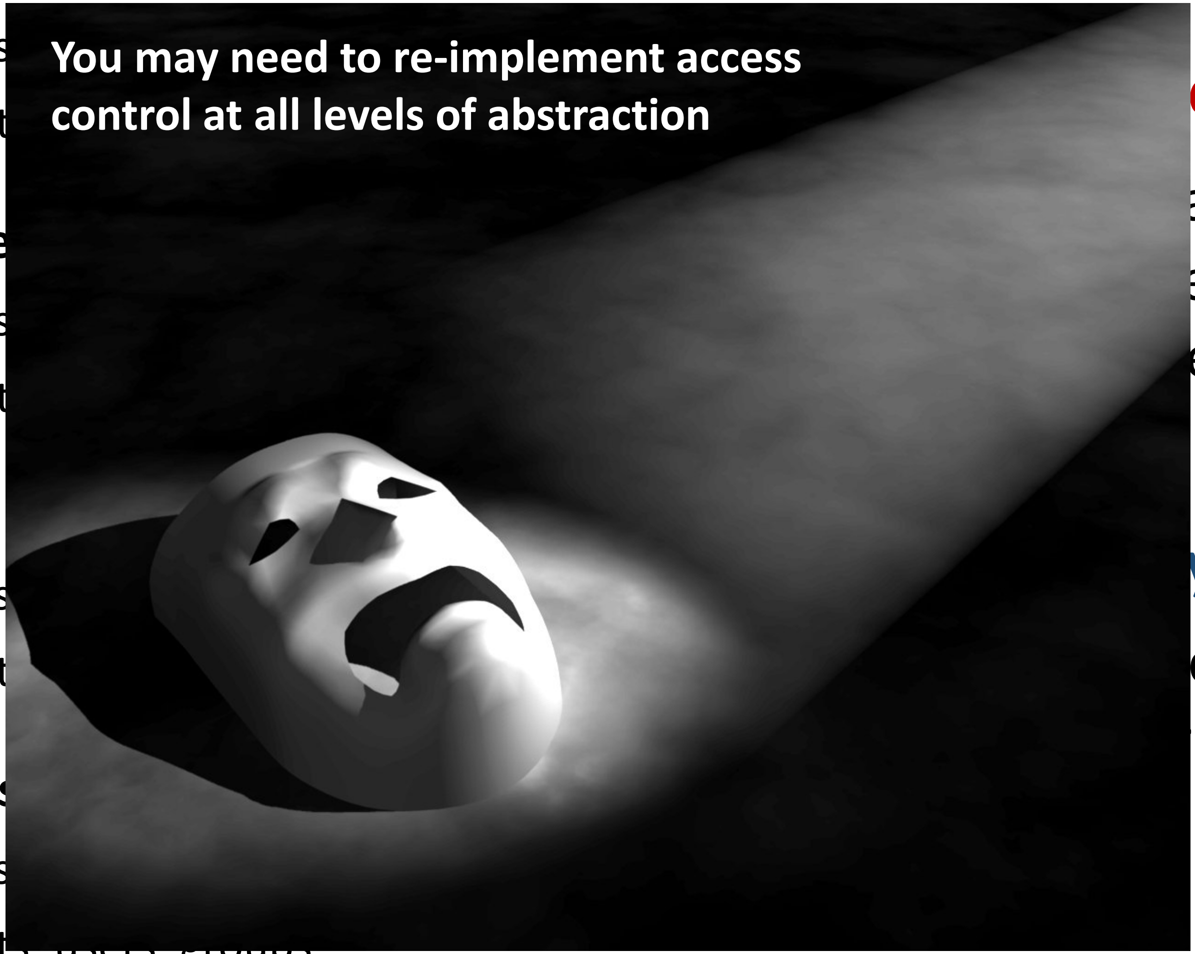
- Objects

- Subject

Applications

- Objects

- Subjects: users, groups



domains is meaningless!!

access control cannot restrict ess to a certain row of a e.

y build on top of each other:

ccess control required to ict access to the *whole* DB file.

Summary of the day

- Access control is a **backbone** for computer security

- The **Access Control Matrix** is a useful **abstraction**
  - Difficult to implement

- Access control **is far from trivial**
  - The UNIX example