ACCESS CONTROL I

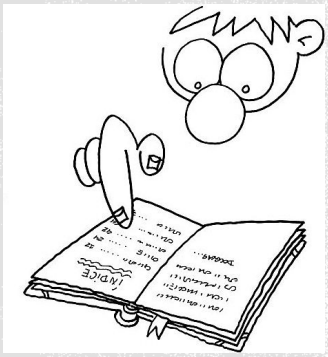
Introduction to Computer and Network Security

Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

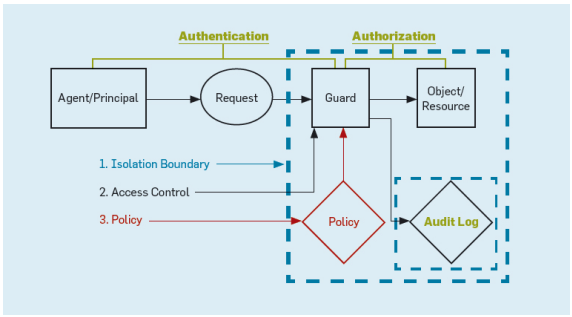
UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

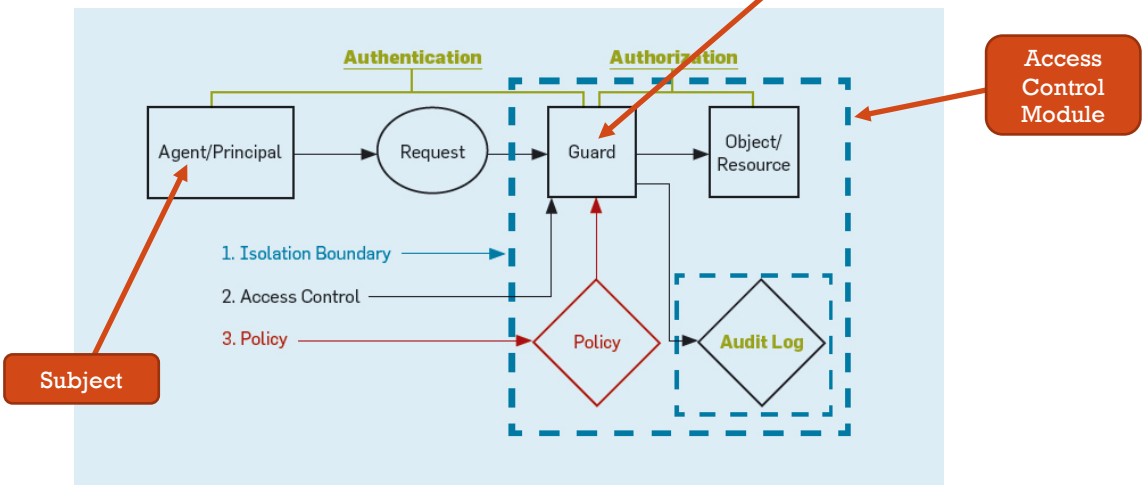


ACCESS CONTROL (AC)



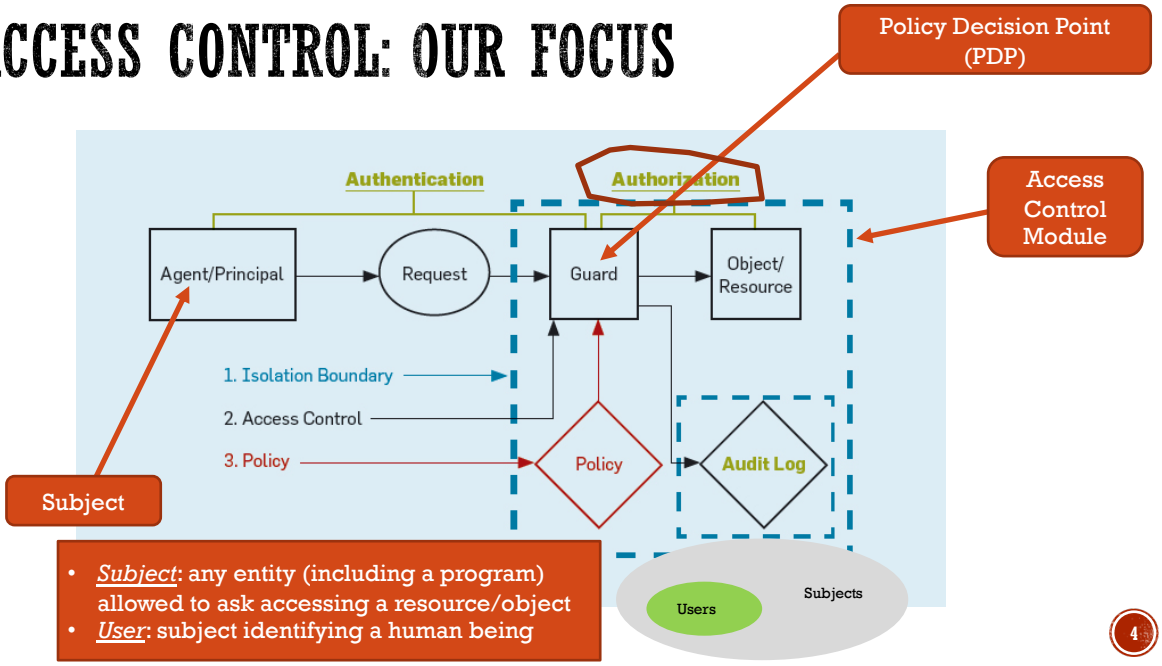
2

ACCESS CONTROL: OUR FOCUS



3

ACCESS CONTROL: OUR FOCUS



- *Subject*: any entity (including a program) allowed to ask accessing a resource/object
- *User*: subject identifying a human being

4

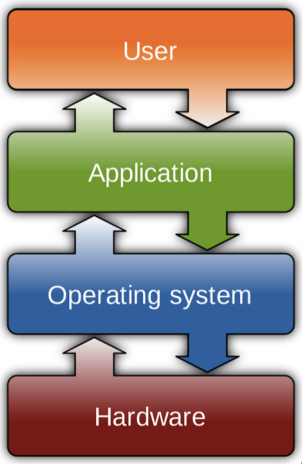
ACCESS CONTROL: REMARKS

- Isolation boundaries play a key role
  - The **outer** one prevents the by-passing of the Guard: all authorization requests go through the Guard, are evaluated according to the policy, and in case they are granted, access is passed to the object or resource
  - The **outer** boundary can be by-passed only by privilege users (such as administrators) that need to install/update policies or perform other routine administrative operations (e.g., updating the software implementing the Guard)
  - The **inner** boundary guarantees the integrity of the audit log so that this can be inspected to understand what happened and why (i.e. to whom access was granted and for which reason)
  - Contrary to the outer boundary, the **inner** boundary cannot be by-passed even by privilege users (such as administrators) as this would give rise to possible insider attacks (to the integrity of the log) that would make auditing meaningless

5

AN EXAMPLE: OPERATING SYSTEM (1)

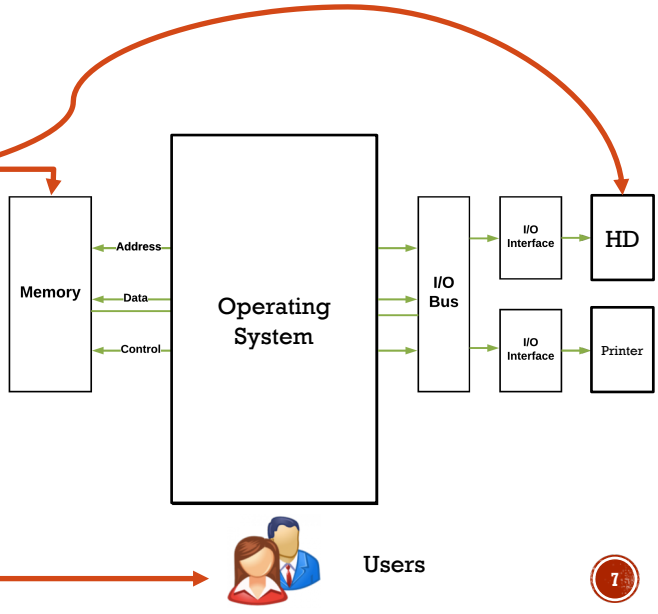
- An Operating System (OS) is a software that manages HW and SW **resources** and provides
  - common services for computer programs and
  - users with the capability of communicating with their computers
- A **multi-user** OS allows for multiple users to use the same computer at the same time and/or different times
- A **multi-tasking** OS is able to allow multiple software processes to run at the same time



6

AN EXAMPLE: OS (2)

- OS must protect users from each other
  - memory protection
  - file protection
  - general control and access to objects
  - user authentication
- The fundamental tradeoff of OS security
  - *Sharing* (desirable)
  - *Protection* (difficult)



7

AN EXAMPLE: OS (3)

- Levels of protection
  - no protection
  - isolation
  - share
    - all or nothing / granular

Designed for just one user, early version of Windows

Processes unaware of other processes
- Each process: own portion of memory (address space), files, *etc.*
- OS provides confinement

Some resources still need to be shared
- Libraries
- Files
- Database

How should we manage the resources?

AN EXAMPLE: OS (4)

Principle of least Priviledge
every **subject** (such as a process, a user, or a program) must be able to **access only the information and resources that are necessary for its legitimate purpose**

- How should we manage resources?
  - share
    - all or nothing / granular
- Useful abstraction: **resource encapsulation**
  - All data and functions required to use the resource are packaged into a single self-contained component
- By using resource encapsulation, the OS can allow access or manipulation of the resource in the way the designer intended
- Besides controlling what operations can be performed on the resource, the OS can also **control which users can perform which operations on the resource**

An **important aspect of privacy protection** is the specification and enforcement of purposes, i.e. **users** should be **confident** that **their data are processed as intended**.
For instance, email addresses are used only for billing but not for marketing purposes.
Checking that information is handled according to legitimate purpose is a difficult task and makes the role of **auditing** crucial for **a posteriori verification**!

AC DEFINITION

- The process of
  - **mediating requests** to resources and data of a system and
  - determining whether a request should be granted or denied

- Flow: subject *s* wants to perform action *a* on resource *r*
  1. Access request (*s*,*a*,*r*) is sent to access control module
  2. The access control module returns **grant/deny**
  3. If answer = grant, then system allows *s* to perform *a* on *r*.
     Else s is informed that cannot perform *a* on *r*

- Paramount importance in system security

10

PURPOSE OF AC

- **Protecting resources from unauthorized accesses**

- This is **very dependent from the context**
  - Operating systems, military systems, online banking, smartphones, …

- **How to specify these security requirements?**

11

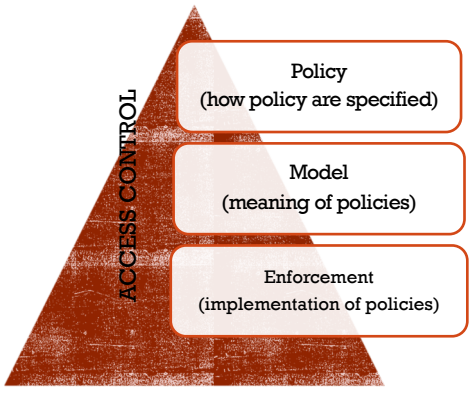
AC POLICY

- Often, security for a system is defined in terms of a policy
- Definition
  - **policy = set of rules to implement specific security properties**
  - Remember: Confidentiality, Integrity and Availability (CIA)
- Alternative definition
  - policy = **contract** between the designer/implementor of a system and customers

12

A STRUCTURED APPROACH TO AC

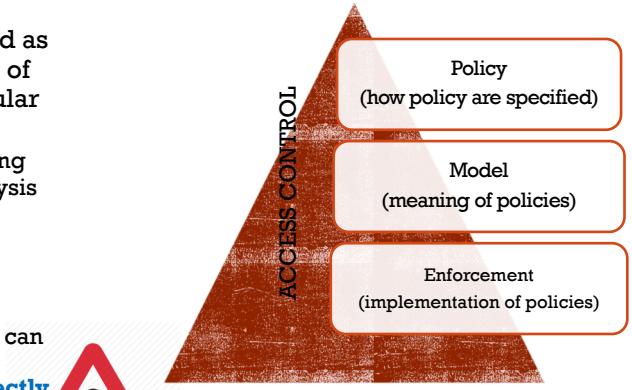
- **Policy** = rules that control what actions, subjects may perform on resources (or objects) containing information
- **Model** = formal (mathematical) representation of the policy and its working
- **Enforcement** = low level (software or hardware) functions that implement the controls imposed by the policy and formally stated in the model



13

STRUCTURED APPROACH: ADVANTAGES

- When policy and model are defined as mathematical objects, the meaning of policies is independent of a particular implementation
  - Possibility to use automated reasoning techniques to perform security analysis of policies
- Separation of concerns
  - When reasoning about policies, one can forget about their enforcement, just **assume that the latter works correctly**



14

WHAT CAN GO WRONG WITH ENFORCEMENT: BACK TO THE OS

- Example from the past: Unix **finger** command (user information lookup program)
  - A widespread implementation would accept an **argument of any length, although only 256 bytes had been allocated** for this argument by the program
  - The result was that when an **attacker used the command with a longer argument, the trailing bytes of the argument ended up being executed by the CPU**
  - In 1988, large numbers of Unix computers were brought down simultaneously by **Morris worm**, which used the finger vulnerability and thus brought **memory-overwriting attacks** to the notice of the mass media
- Programmers are often careless about checking the size of arguments, still now and **holes continue to be found**!

15

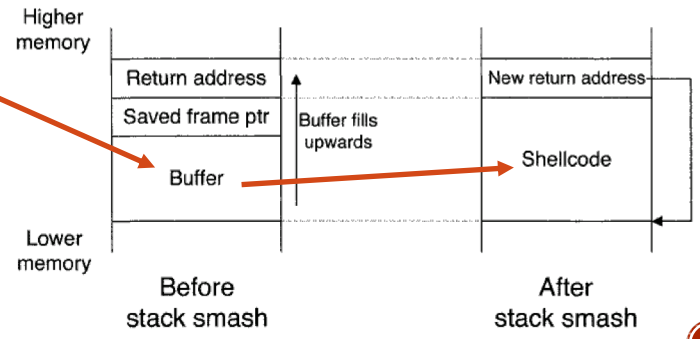
WHAT CAN GO WRONG WITH ENFORCEMENT: BACK TO THE OS (CONT'D)

Smashing the stack

Input string: NOP NOP NOP …" Shellcode" "New return address"

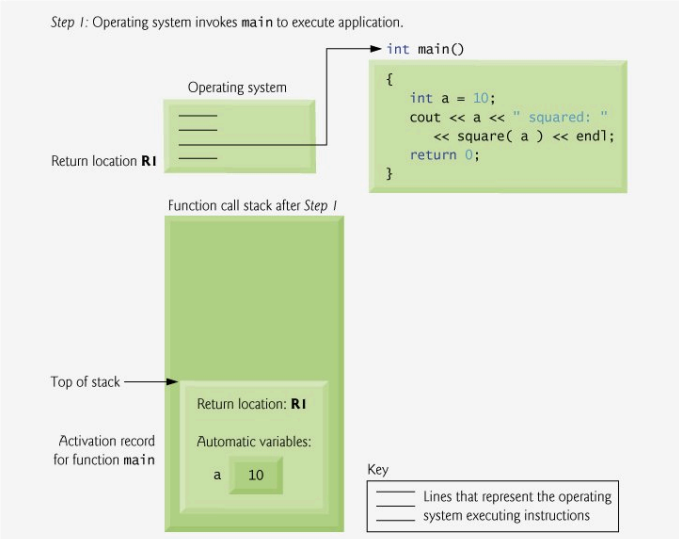
```
def main():
    fill_buffer0
def fill_buffer0 :
    character buffer[100]
    i = 0
    ch = input()
    while (ch != newline) :
        buffer[i] = ch
        ch = input()
        i = i+1
```

No bounds checking on variable i



16

FUNCTION CALL AND THE STACK



17

18

AC MODELS

The models we are going to consider have been designed in different application contexts, namely

- AC Matrix, AC lists, and Capabilities → Operating systems
- Multi Level Security (Bell La Padula) → Military systems
- Role Based AC and Attribute Based AC → Enterprise systems

THE BASIC MODEL: AC MATRIX

- The meaning of policies of the form "*a subject can perform some actions on an object*" can be given in terms of the Access Control Matrix (ACM)
- Given all subjects and objects, the ACM enumerates what actions are allowed for each subject/object pair

o: own
r: read
w:write

	f1	*f2*	*f3*	*f4*	*f5*	*f6*
s1		*o, r, w*	*o, r, w*		*w*	
s2	*o, r, w*	*r*			*o, r, w*	
s3		*r*	*r*	*o, r, w*	*r*	*o, r, w*

Access Matrix

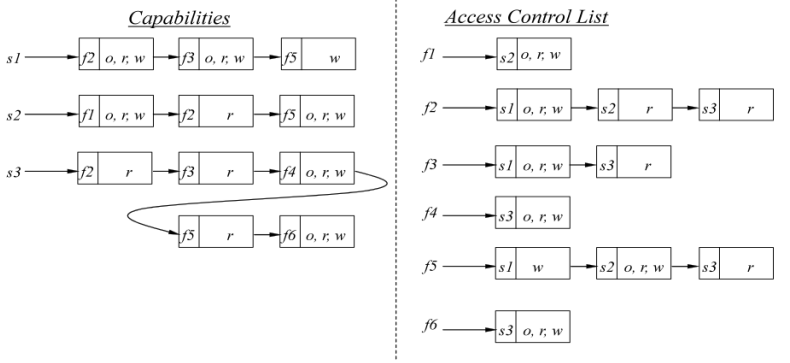
19

ENFORCEMENT MECHANISMS FOR AC MATRIX

o: own
r: read
w:write

	f1	f2	f3	f4	f5	f6
s1		o, r, w	o, r, w		w	
s2	o, r, w	r			o, r, w	
s3		r	r	o, r, w	r	o, r, w

Access Matrix



20

EXERCISE

- **A**lice can **r**ead and **w**rite the file **f1**, can **r**ead the file **f2**, and can e**x**ecute the file **f3**.
- **B**ob can **r**ead **f1**, can **r**ead and **w**rite **f2**, and *cannot access* **f3**.

1. Write a set of access control lists for this situation. Which list is associated with which file?
2. Write a set of capability lists for this situation.

21

EXERCISE: SOLUTION (1)

- **A**lice can **r**ead and **w**rite the file **f1**, can **r**ead the file **f2**, and can e**x**ecute the file **f3**.
- **B**ob can **r**ead **f1**, can **r**ead and **w**rite **f2**, and *cannot access* **f3**.

	f1	f2	f3
Alice			
Bob			

EXERCISE: SOLUTION (2)

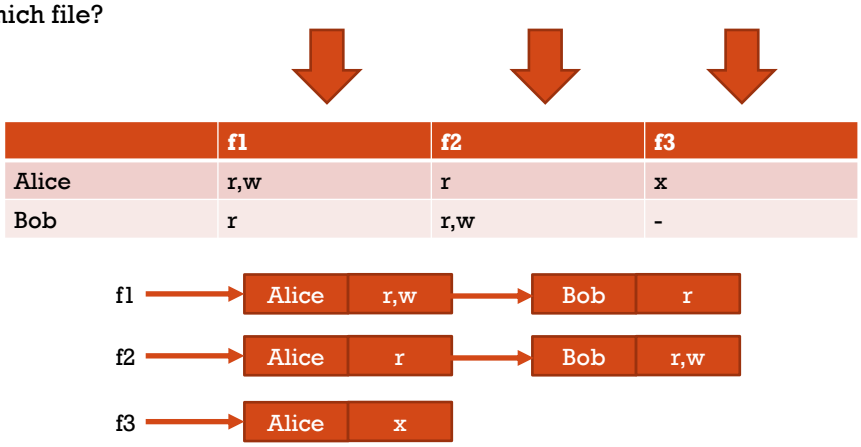
- **A**lice can **r**ead and **w**rite the file **f1**, can **r**ead the file **f2**, and can e**x**ecute the file **f3**.
- **B**ob can **r**ead **f1**, can **r**ead and **w**rite **f2**, and *cannot access* **f3**.

	f1	f2	f3
Alice	r,w	r	x
Bob	r	r,w	-

Sometimes, x implies also r and w

EXERCISE: SOLUTION (3)

1. Write a set of access control lists for this situation. Which list is associated with which file?



	f1	f2	f3
Alice	r,w	r	x
Bob	r	r,w	-

EXERCISE: SOLUTION (4)

2. Write a set of capability lists for this situation.

	f1	f2	f3
Alice	r,w	r	x
Bob	r	r,w	-

DIGRESSION ON PROTECTION IN OPERATING SYSTEMS (1)

- Goals
  - prevent malicious misuse of the system by users or programs
  - ensure that each shared resource is used only in accordance with system policies
- Principle of protections
  - ***Principle of least privilege*** → programs, users, and systems shall be given just enough privileges to perform their tasks
  - Key idea → ensure that failures do the least amount of harm and allow the least of harm to be done
  - Example of application → each user is given their own account, and has only enough privilege to modify their own files
    - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges
    - The root account should not be used for normal day to day activities

DIGRESSION ON PROTECTION IN OPERATING SYSTEMS (2)

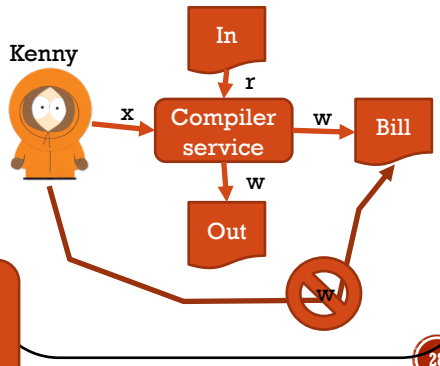
- A computer can be viewed as a collection of processes and objects (hardware or software)
- **Need to know principle** → a process should only have access to those objects it needs to accomplish its task and only for the operations for which it needs access and only during the time frame when it needs access
- A **domain of protection** is the set of objects and the access rights, i.e. the ability to execute certain operations on selected objects
  - Formally, a domain of protection is a set of pairs of the following form
    < object, { access right set } >
- Domains of protection can be implemented as users, processes, or procedures
  - Example
    - In Unix/Linux, each user corresponds to a domain
    - A domain defines the access rights of a user
    - Changing domains involves changing user identifiers
    - **Programs running on behalf of a user inherit the user identifier and thus are associated to the same domain of protection**
    - **When an executable is given permissions to access certain (e.g., shared) resources that the user executing the program cannot access, then security issues may arise as we will see next…**

ACL VS CAPABILITIES

- The compiler service takes as inputs the names of the source file to be compiled and an output file
- The compiler service compiles the source file, writes the compiled binary to the output file and-on the side-writes a billing file that records how much the caller must pay for using the compiler
- **The caller tricks the compiler by passing to it the name of the billing file in place of the output file**, which causes the compiler to overwrite the billing file with a binary, thus destroying the billing file's integrity

Confused deputy: privilege escalation attack where the adversary who does not have direct access to some sensitive resource, indirectly writes the resource by confusing a subject (called deputy) who can access the resource.

Standard example: **compiler service**



For instance, in Linux…
- Every process executes on behalf of a given user
- The process may thus access the files that its owner may access…
- … and more that have been given to it by system administrators

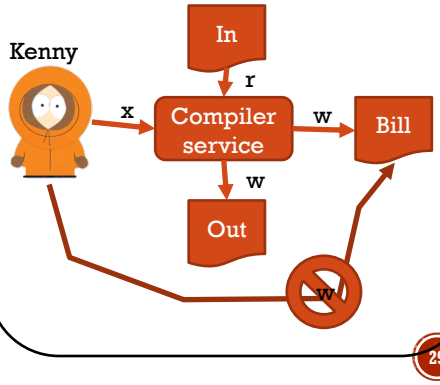
28

ACL VS CAPABILITIES (2)

- The compiler to execute with In as the source file and Bill as the output file needs to perform the following authorization queries
  - (Kenny, x, Compiler)
  - (Compiler, r, In)
  - (Compiler, w, Bill)
- These are all granted assuming the following ACLs
  - Compiler -> [Kenny|x]
  - In -> [Kenny|o,r,w]
  - Bill -> [Compiler|w]
- The confusion arises by the fact that the Compiler is delegated
  - **by Kenny to read its source file In** and
  - **by the OS to write the billing file Bill**
- During execution, the compiler cannot distinguish the source from which each right derives from and uses them both

Confused deputy: privilege escalation attack where the adversary who does not have direct access to some sensitive resource, indirectly writes the resource by confusing a subject (called deputy) who can access the resource.

Standard example: **compiler service**

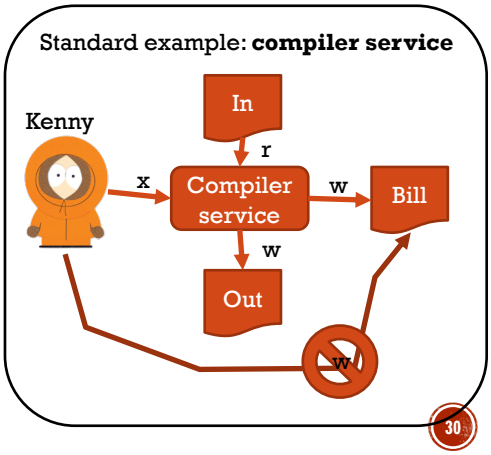


29

ACL VS CAPABILITIES (3)

Confused deputy: privilege escalation attack where the adversary who does not have direct access to some sensitive resource, indirectly writes the resource by confusing a subject (called deputy) who can access the resource.

- The compiler to execute with In as the source file and Bill as the output file needs to perform the following authorization queries
  - (Kenny, x, Compiler)
  - (Compiler, r, In)
  - (Compiler, w, Bill)
- Now, only the first two are granted but the third is denied assuming the following capabilities
  - Kenny -> [Compiler|x] -> [In|r]
  - Compiler -> [Bill|w]
- There is no more confusion as **Kenny needs to explicitly pass its capabilities to the Compiler**:
  - it can do this for reading the file In but
  - it cannot do it for writing to the billing file Bill



30

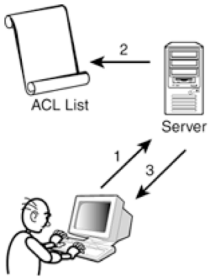
AC MODELS

- Several models exist, e.g.
  - **DAC**: subjects can give rights to other subjects (DISCRETIONARY)
  - **MAC**: system enforces mandatory rules (MANDATORY)
- Can you give an example of DAC policies? And one of MAC policies?
- Usually, DAC is too flexible whereas MAC is too restrictive
- Search for flexible and expressive models
  - Role Based Access Control (RBAC) is considered superior to both DAC and MAC
  - More on RBAC later…

31

DAC POLICY EXAMPLES

- **Owner of a resource decides how it can be shared**
  - NO central entity deciding who can do what on which resources
  - Owner can choose to give read/write/… access to other users on the resources they own
- Typically used in operating systems
- Usually enforced by ACLs
  - To simplify administration, users can be put in **groups**



32

DAC AND GROUPS

- Large organizations usually group (most of the) employees according to their profiles and form small number of categories (usually called **groups**)
- Example: a bank might have 40 or 50 such categories: teller, chief teller, branch accountant, branch manager, and so on
- The remainder (such as the security manager, and chief foreign exchange dealer,…), who need to have their access rights defined individually, may amount to only a few dozen people
- These observations can be exploited to simplify AC by identifying a small number of predefined **groups** to which (almost) each employee can be assigned
  - A **group may appear in** access control matrices or **ACLs**
  - Checking that a subject has a permission amounts to finding a group to which the user belongs and that appears in a ACL with associated that permission on a given resource

33

DAC: PROS AND CONS

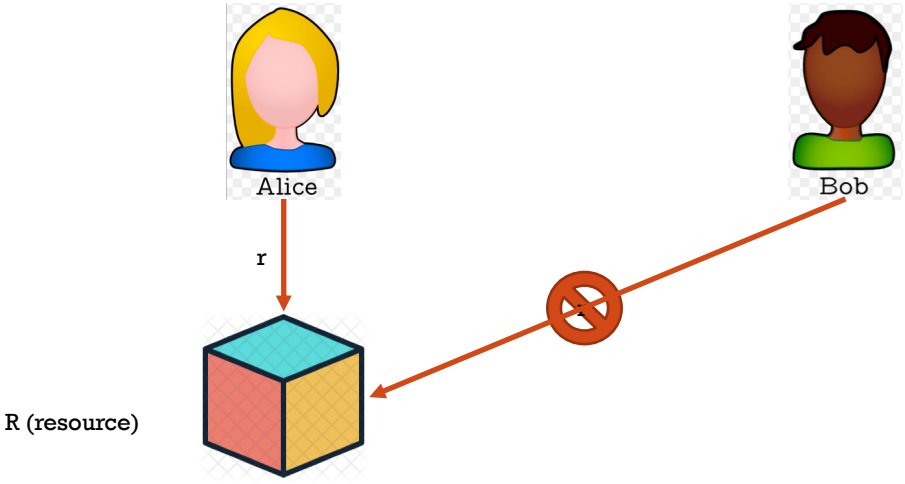
- Pros
  - **Flexible** - the main reason of its popularity in OSes
  - Implementation: well understood (e.g., in Unix)
  - Intuitive

- Cons
  - **Subjective**: rely on owner's judgement to modify the protection state of a resource
  - It works if users make no mistakes (impossible): loosing control of the situation is very easy!
  - Vulnerable to **trojans** (*information leakage*)
    - Resource R only readable by Alice
    - Bob induces Alice to run a trojan that can read R and copy information to resource R' readable by Bob
    - Bob can read R' and thus also the information in R

Trojan: any malicious computer program which misleads users of its true intent.
They are generally spread by some form of social engineering, for example where a user is duped into executing an e-mail attachment disguised to be unsuspicious

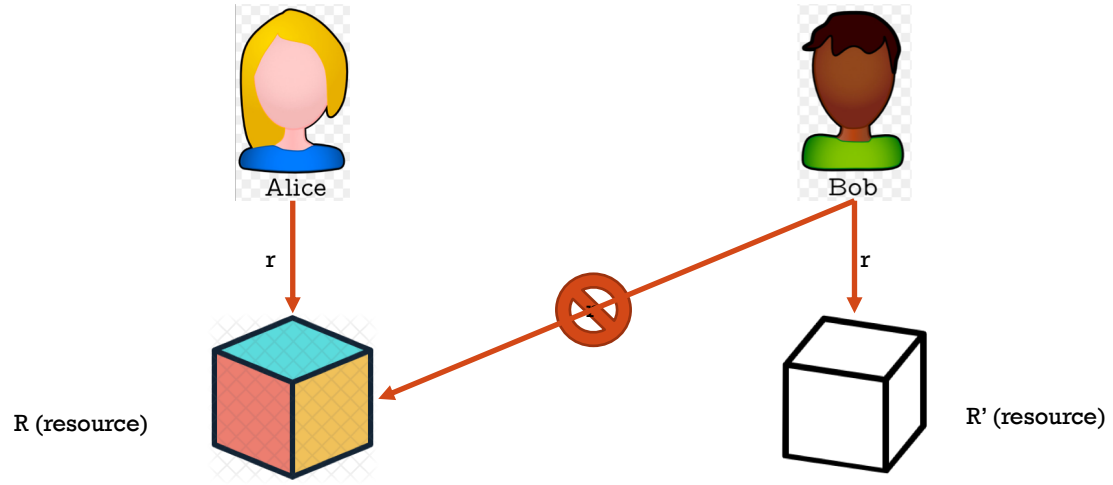
34

TROJAN (1)



35

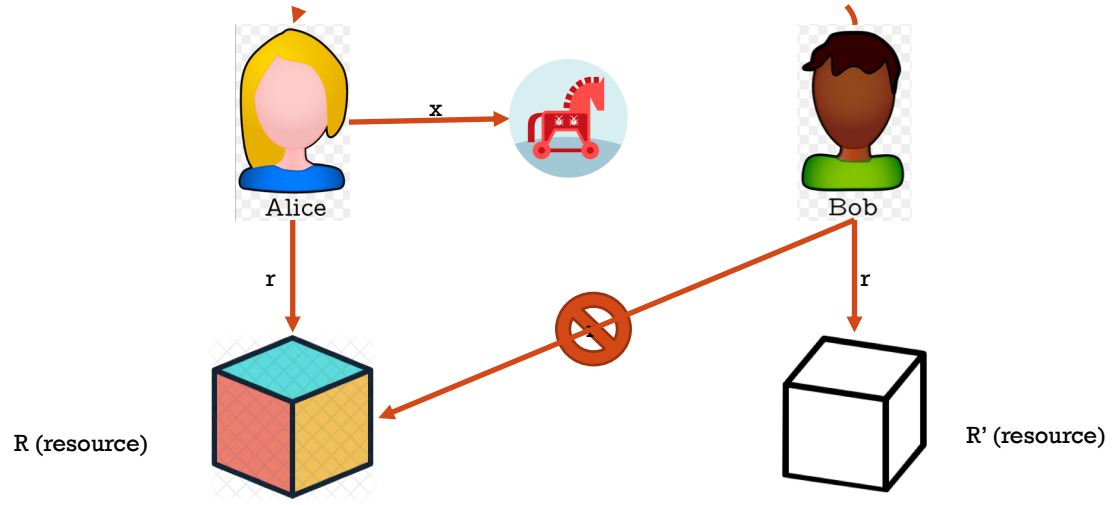
TROJAN (2)



36

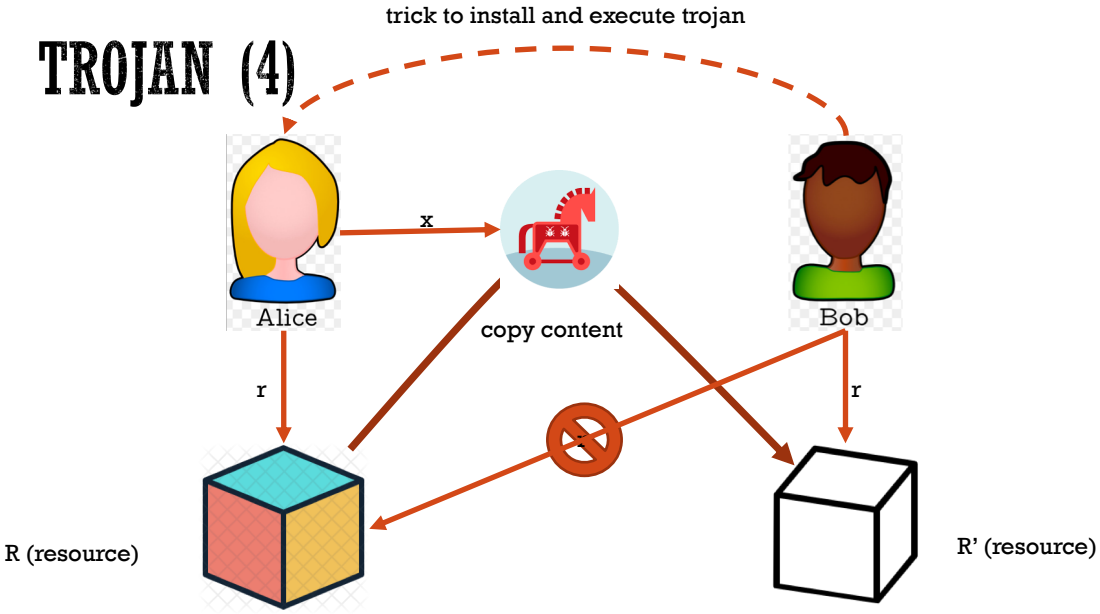
trick to install and execute trojan

TROJAN (3)



37

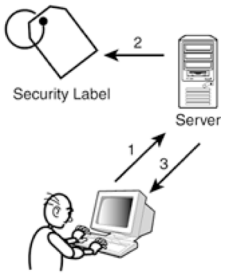
TROJAN (4)



38

MAC POLICY EXAMPLES

- Users operates in an **organization** and it is the latter that **decides how data should be shared**
  - Central entity deciding who can do what on which resources
- Hospital owns patients records and limits their sharing
  - Regulatory/legal requirements may limit sharing
- Usually enforced by using **security labels**
  - This leads to *multi-level security models*



39

MULTI-LEVEL SECURITY

- Early security problem: protection of confidentiality in a military setting
  - Given **information at various sensitivity levels** and **users having various degrees of trustworthiness**, how do we control access to information within the system to protect confidentiality?
  - Relevant even before computers were invented!
- Information with different "sensitivity" levels
  - war plan, defense budget, football schedule, cafeteria menu, …
- Users with different degrees of trustworthiness:
  - generals, privates, colonels, secretaries, janitors)
- GOAL: define policy to **prevent the release of sensitive information** (e.g., war plans) **to untrusted users** (e.g., janitors)

40

MULTI-LEVEL SECURITY (2)

Linear order: binary relation on a set which satisfies
- Antisymmetry
  - If a>= b & b >= a then a=b
- Transitivity
  - If a>=b & b>= c then a>= c
- Connex property
  - a>= b or b>=a

- How can we **categorize information wrt sensitivity**?
- Information is compartmentized into separate containers (resources such as documents, folders, or files) labeled according to their *sensitivity label* **L = (S,N)**
  - S takes values over a **linearly ordered set** such as
    - Top Secret >= Secret >= Confidential >= Unclassified
  - N is a **set of "need-to-know" categories** from an **unordered set** expressing membership within some interest group (e.g., Crypto, Nuclear, Janitorial, Personnel, etc).
- Examples
  - label **(Secret, {Nuclear, Crypto})** indicates a resource containing sensitive information related to the categories "Nuclear" and "Crypto"
  - label **(TopSecret, {Crypto})** indicates a resource containing very sensitive information related to the category "Crypto"

41

MULTI-LEVEL SECURITY (3)

- At creation time, each **resource** is **associated** to a sensitivity **label** by resource originator according to some criteria (not made explicit in the MLS model)
- When a document contains both sensitive and non-sensitive information, the **originator needs to use the highest appropriate level**
- When some conditions change, e.g. time, it may happen that the associated security label should be downgraded (**de-classified**) as the content has become less sensitive

42

MULTI-LEVEL SECURITY (4)

- After associating sensitivity labels to resources, we must establish which users are authorized to access which resources
- For this, assign **clearances** (or *authorization levels*), which have the same structure of the sensitivity levels associated to resources, i.e. **each user is associated to a clearance C = (S, N)**
  - S is a **hierarchical security level** indicating the degree of trustworthiness to which the user has been vetted
  - N is a **set of "need-to-know categories"** indicating domains of interest in which the user is authorized to operate

- Sensitivity labels (on resources) indicate the sensitivity of the contained information whereas clearances (on users) indicate classes of information that users are authorized to access
- Need-to-know categories reflect the *Principle of Least Privilege*: even within a given security level (such as Top Secret) not everyone needs to know everything

43

MULTI-LEVEL SECURITY (5)

- Access control policies defined by using sensitivity labels and clearances
- *No Read Up Property*
  - subject s can read resource r if (Ss, Ns) **dominates** (Sr, Nr)
  - subject s asking to read the content of a resource r must show that its **clearance dominates the sensitivity label** of the resource
- *No Write Down Property*
  - subject s can write to resource r if (Sr, Nr) **dominates** (Ss, Ns)
  - subject s asking to write to a resource must show that its **clearance is dominated by the sensitivity label** of the resource

(S1,N1) dominates (S2,N2) iff
- S1 >= S2
- N1 ⊇ N2

Examples:
- (Secret, {Crypto}) dominates (Confidential, {Crypto}) since
  - Secret >= Confidential
  - {Crypto} ⊇ {Crypto}
- (Secret, {Crypto, Nuclear}) does not dominate (Top Secret, {Crypto}) since
  - Secret />= Top Secret despite the fact that {Crypto, Nuclear} ⊇ {Crypto}

44

MULTI-LEVEL SECURITY (6)

- Access control policies defined by using sensitivity labels and clearances
- *No Read Up Property*
  - subject s can read resource r if (Ss, Ns) **dominates** (Sr, Nr)
  - subject s asking to read the content of a resource r must show that its **clearance dominates the sensitivity label** of the resource
- *No Write Down Property*
  - subject s can write to resource r if (Sr, Nr) **dominates** (Ss, Ns)
  - subject s asking to write to a resource must show that its **clearance is dominated by the sensitivity label** of the resource
- Notice that the *No Write Down Property* aims to prevent that a subject with access to a Top Secret file may copy the information into an Unclassified file

45

MULTI-LEVEL SECURITY (7)

- Implicit assumption: **Tranquility Principle** prevents the ability to change security labels arbitrarily as this can subvert security
- Example
  - Subject s1 with highest clearance
  - Subject s2 with lowest clearance
  - Resource r1 with highest sensitivity label
  - Resource r2 with lowest sensitivity label
  - After reading the content of r1 (thereby satisfying the *No Read Up Property*), s1 change his/her clearance level to lowest and write to resource r2 (thereby satisfying the *No Write Down Property*) the content he/she has read from r1
  - While both the basic properties of MLS are satisfied, **confidentiality is not preserved** as now s2 (with lowest clearance) can read resource r2 that also contain the information from r1

MULTI-LEVEL SECURITY (8)

- Now Read Up, No Write Down and the Tranquility Principle are at the heart of the **Bell-La Padula security model** introduced in **1973**
- Despite its age, it is a cornerstone of modern computer security and widely used in military applications
- Shortcoming of Bell-La Padula
  - Can a corporal with no clearance overwrite the war plan (associated to the highest sensitivity level)?
  - The No Write Down Property does not prevent this but Bell-La Padula is concerned with confidentiality while the question points to an integrity issue (writing to a resource with high sensitivity by a user without enough clearance is likely to make the content of the resource useless)
  - Use the Biba model, 1977 based on integrity labels (similar mathematical model, different meaning)
    - integrity label of a resource characterizes the degree of "trustworthiness" of the information contained in that resource
    - integrity label of a subject measures the confidence one places in its ability to produce or handle information

EXERCISE

- Security levels: Top Secret >= Secret >= Confidential >= Unclassified
- Two categories: Nuclear and Army
- Four subjects:
  - President with Top Secret clearance for Nuclear and Army
  - Colonel has SECRET clearance for Army and Nuclear
  - Major has only CONFIDENTIAL clearance for Army
  - Soldier has only UNCLASSIFIED clearance for Nuclear
- Resources
  - Army position at sec lev Secret
  - Number of army units at sec lev Confidential
  - Number of nuclear units at sec lev Confidential
  - Costs of the nuclear program at sec lev Unclassified
  - Costs of the army at sec lev Unclassified
  - Nuclear code at sec lev Top Secret

48

EXERCISE (CONT'D)

1. Can the president compute the overall defense costs (army + nuclear)?
2. Can the major compute the total number of nuclear and army units?
3. Can the colonel compute the total number of nuclear and army units?
4. Can the colonel change the army position?
5. Can the major change the nuclear code?
6. Can the soldier change the nuclear code?
7. What problem is raised by previous question?

49

EXERCISE: SOLUTION (1)

- Security levels: Top Secret >= Secret >= Confidential >= Unclassified
- Two categories: Nuclear and Army
- Four subjects:
  - President with Top Secret clearance for Nuclear and Army
  - Colonel has Secret clearance for Army and Nuclear
  - Major has only Confidential clearance for Army
  - Soldier has only Unclassified clearance for Nuclear

- President <- (Top Secret, {Nuclear, Army})
- Colonel <- (Secret, {Nuclear, Army})
- Major <- (Confidential, {Army})
- Soldier <- (Unclassified, {Nuclear})

50

EXERCISE: SOLUTION (2)

- Resources
  - Army position at sec lev Secret
  - Number of army units at sec lev Confidential
  - Number of nuclear units at sec lev Confidential
  - Costs of the nuclear program at sec lev Unclassified
  - Costs of the army at sec lev Unclassified
  - Nuclear code at sec lev Top Secret

- Army position <- (Secret, {Army})
- Number of army units <- (Confidential, {Army})
- Number of nuclear units <- (Confidential, {Nuclear})
- Cost of nuclear program <- (Unclassified, {Nuclear})
- Cost of army <- (Unclassified, {Army})
- Nuclear code <- (Top Secret, {Nuclear})

51

EXERCISE: SOLUTION (4)

Check the answers by applying the definition of the dominates relation

1. Can the president compute the overall defense costs (army + nuclear)? Yes according to Read Down
2. Can the major compute the total number of nuclear and army units? No, failing need to know
3. Can the colonel compute the total number of nuclear and army units? Yes according to Read Down
4. **Can the colonel change the army position? No, failing need to know although same level (secret)**
5. **Can the major change the nuclear code? No, failing need to know although higher level (top secret and confidential)**
6. Can the soldier change the nuclear code? Yes, according to Write Up
7. What problem is raised by previous question? Integrity issue, not in the scope of Bell-La Padula

- President <- (Top Secret, {Nuclear, Army})
- Colonel <- (Secret, {Nuclear, Army})
- Major <- (Confidential, {Army})
- Soldier <- (Unclassified, {Nuclear})

- Army position <- (Secret, {Army})
- Number of army units <- (Confidential, {Army})
- Number of nuclear units <- (Confidential, {Nuclear})
- Cost of nuclear program <- (Unclassified, {Nuclear})
- Cost of army <- (Unclassified, {Army})
- Nuclear code <- (Top Secret, {Nuclear})

MAC: PROS AND CONS

Covert channel: a path for the flow of information between subjects within a system, utilizing system resources that were not designed to be used for inter-subject communication

- Pros
  - Not vulnerable to trojans because of the No Write Down property
  - Rigid: it is easy to keep the situation under control!
- Cons
  - Rigid: may hinder business continuity
  - Information leakage still possible by **covert channel**
    - In the Bell-La Padula model it is **possible to encode a covert channel by using** the two key properties No Read Up and No Write Down (see next slide)

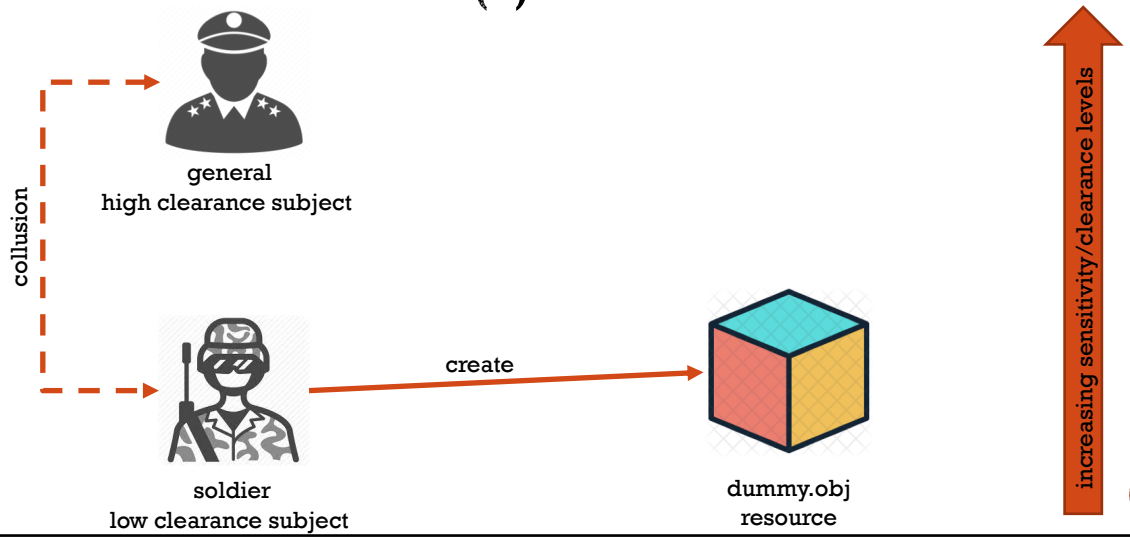
53

A COVERT CHANNEL IN BELL-LA PADULA

- A low level subject makes an object *"dummy.obj"* at its own level
- Its high level accomplice either upgrades the security level of *dummy.obj* to high or leaves it unchanged
- Later, the low level subject tries to read *dummy.obj*.
  - Success or failure of this request disclose the action of the high-level subject.
    - One bit of information has flown from high to low
      - Failure means *dummy.obj* has be upgraded
      - Success means *dummy.obj* has not been changed
- So, *the high level accomplice can send one bit of information to the low level subject*
- If they can repeat this multiple times, *the high level accomplice can send any amount of information to the low level accomplice!*

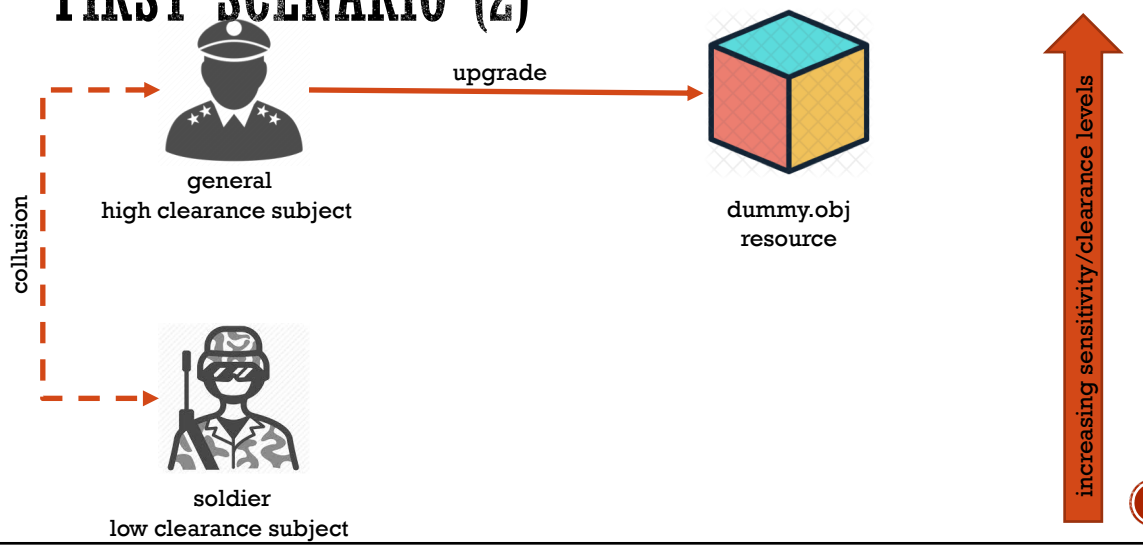
54

A COVERT CHANNEL IN BELL-LA PADULA: FIRST SCENARIO (1)

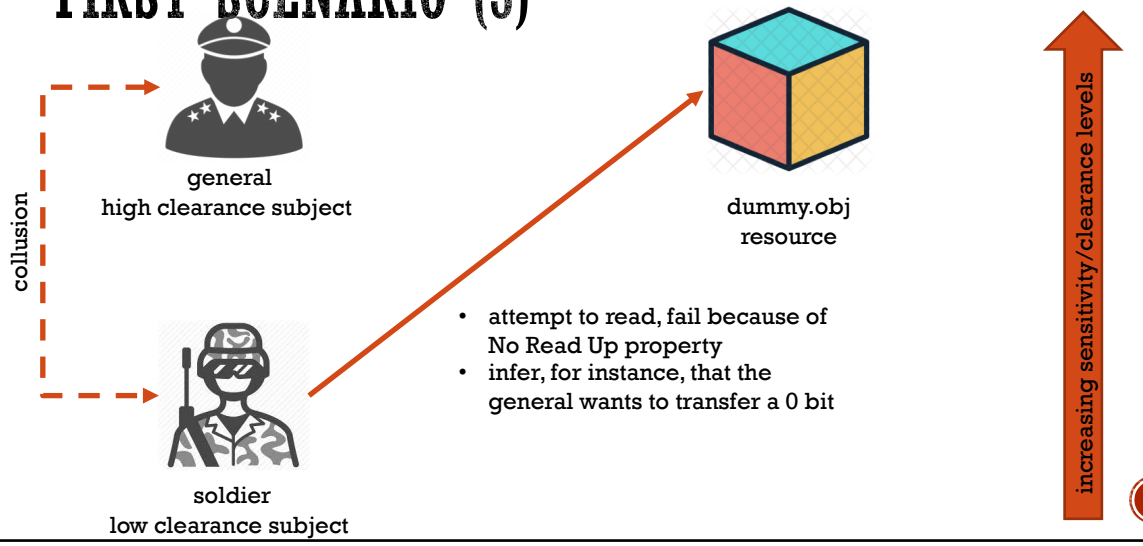


55

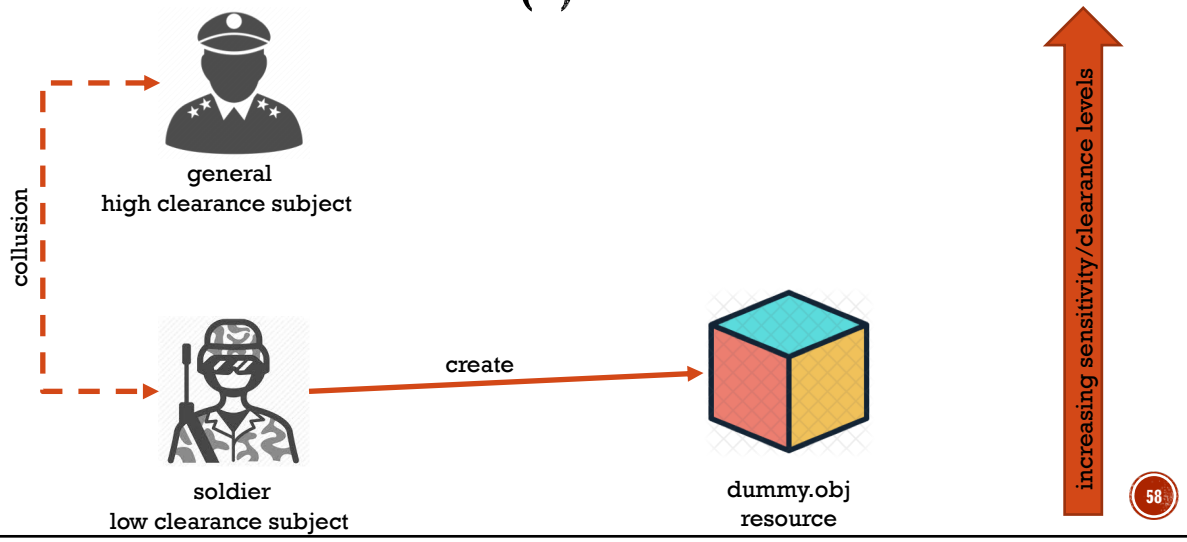
A COVERT CHANNEL IN BELL-LA PADULA: FIRST SCENARIO (2)



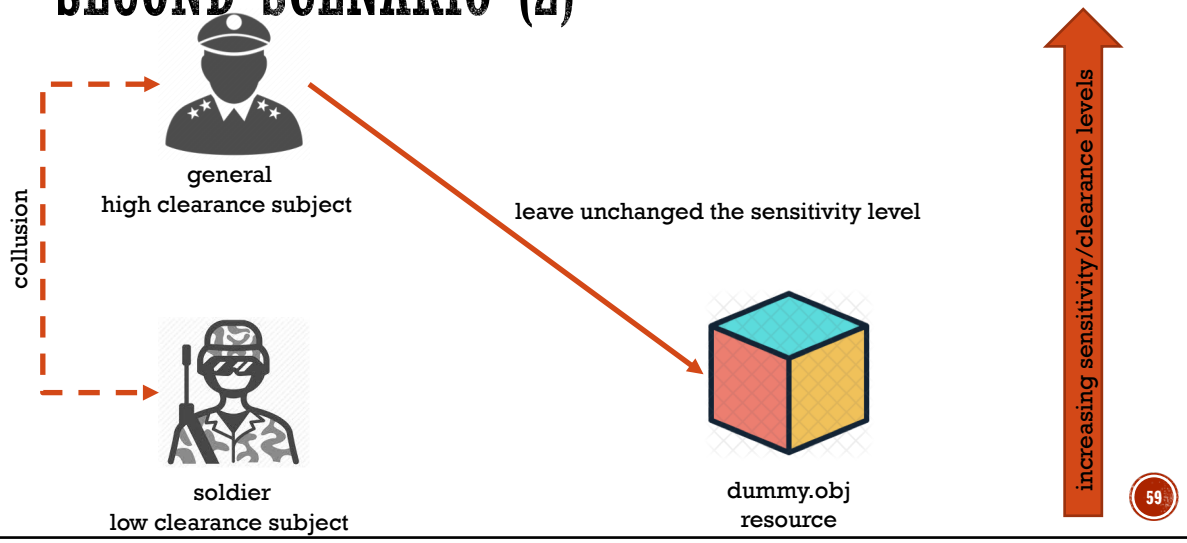
A COVERT CHANNEL IN BELL-LA PADULA: FIRST SCENARIO (3)



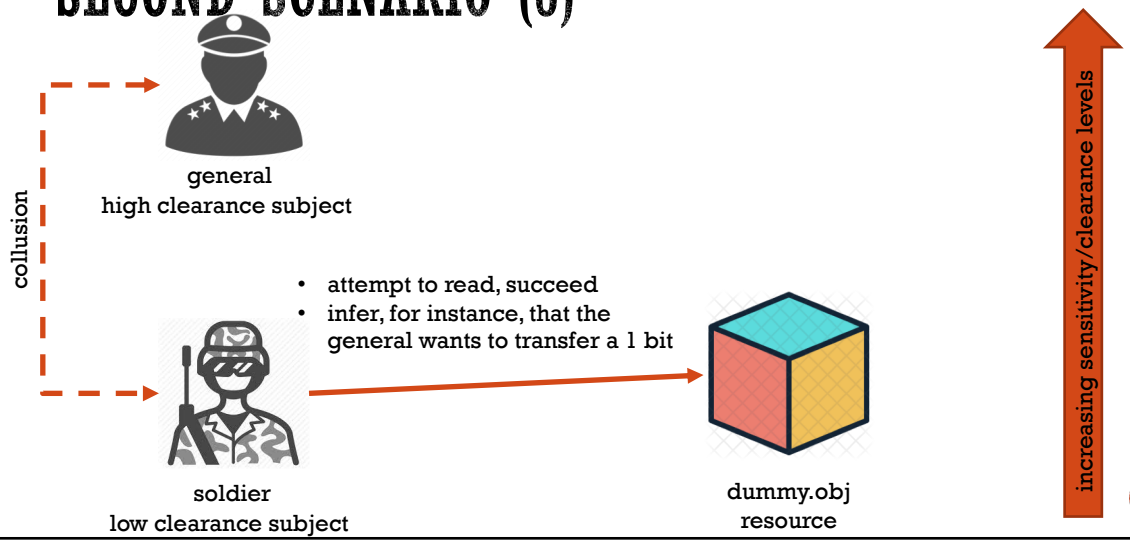
A COVERT CHANNEL IN BELL-LA PADULA: SECOND SCENARIO (1)



A COVERT CHANNEL IN BELL-LA PADULA: SECOND SCENARIO (2)



A COVERT CHANNEL IN BELL-LA PADULA: SECOND SCENARIO (3)



60

WHY DAC & MAC ARE NOT ADEQUATE?
EX: AT A SMALL UNIVERSITY

- Users: Alice, Bob, Charlie, David, Eve, Fred, Greg
- Permissions: GrantTenure, AssignGrades, ReceiveBenefits, **UseGym**, Register4Courses
  - A **permission** is an abstraction of (action,resource)
- Policy rules:
  - Alice has permissions: GrantTenure, AssignGrades, *ReceiveBenefits*, **UseGym**
  - Bob and Charlie have permissions: GrantTenure, AssignGrades, **UseGym**
  - David has permissions: AssignHWScores, Register4Courses, **UseGym**
  - Eve has permissions: *ReceiveBenefits*, **UseGym**
  - Fred has permissions: Register4Courses, **UseGym**
  - Greg has permission: **UseGym**

61

EX: AT A SMALL UNIVERSITY HOW TO TAKE AC DECISIONS?

- Look at the permissions
- It is possible to group them according to the profile of users or the set of functionalities users need to carry out their work!

User	Permission	
Alice	GrantTenure	Promotion Committee Member
Alice	AssignGrades	
Alice	*ReceiveBenefits*	
Alice	UseGym	
Bob	GrantTenure	Faculty Member
Bob	AssignGrades	
Bob	UseGym	
Charlie	GrantTenure	Faculty Member
Charlie	AssignGrades	
Charlie	UseGym	
David	AssignHWScores	Teaching Assistant
David	Register4Courses	
David	UseGym	
Eve	*ReceiveBenefits*	University Employee
Eve	UseGym	
Fred	Register4Courses	Student
Fred	UseGym	
Greg	UseGym	University Member

62

EX: AT A SMALL UNIVERSITY HOW TO TAKE AC DECISIONS?

- When a user is promoted or demoted (i.e. changes job and thus changes the set of functions needed to carry out its work)…
- … Administrators need to update the table very carefully for each permission!

User	Permission	
Alice	GrantTenure	Promotion Committee Member
Alice	AssignGrades	
Alice	*ReceiveBenefits*	
Alice	UseGym	
Bob	GrantTenure	Faculty Member
Bob	AssignGrades	
Bob	UseGym	
Charlie	GrantTenure	Faculty Member
Charlie	AssignGrades	
Charlie	UseGym	
David	AssignHWScores	Teaching Assistant
David	Register4Courses	
David	UseGym	
Eve	*ReceiveBenefits*	University Employee
Eve	UseGym	
Fred	Register4Courses	Student
Fred	UseGym	
Greg	UseGym	University Member

63

EX: AT A SMALL UNIVERSITY
A 2ND LOOK AT THE POLICY

1. Alice is a member of the **Promotion Committee** and has permissions GrantTenure, AssignGrades, *ReceiveBenefits*, **UseGym**
2. Bob and Charlie are **Faculty Members** and have permissions GrantTenure, AssignGrades, **UseGym**
3. David is a **Teaching Assistant** and has permissions AssignHWScores, Register4Courses, **UseGym**
4. Eve is a **University Employee** and has permissions *ReceiveBenefits*, **UseGym**
5. Fred is a **Student** and has permissions Register4Courses, **UseGym**
6. Greg is a **University Member** and has permission **UseGym**

- Words in boldface identify **roles** to which both *users* and *permissions* are associated

EX: AT A SMALL UNIVERSITY
HOW TO TAKE AC DECISIONS AGAIN?

User Assignment (UA)

User	Role
Alice	**PCMember**
Bob	**Faculty**
Charlie	**Faculty**
David	**TA**
David	**Student**
Eve	**UEmployee**
Fred	**Student**
Greg	**UMember**

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
PCMember	AssignGrades
PCMember	*ReceiveBenefits*
PCMember	**UseGym**
Faculty	AssignGrades
Faculty	GrantTenure
Faculty	**UseGym**
TA	AssignHWScores
TA	Register4Courses
TA	**UseGym**
UEmployee	*ReceiveBenefits*
UEmployee	**UseGym**
Student	Register4Courses
Student	**UseGym**
UMember	**UseGym**

User *u* has permission *p* iff there exists a role *r* s.t. $(u, r) \in UA$ and $(r, p) \in PA$

ADMIN IS SIMPLIFIED!

User Assignment (UA)

User	Role
Alice	**PCMember**
Bob	**Faculty**
Charlie	**Faculty**
David	**TA**
David	**Student**
Eve	**UEmployee**
Fred	**Student**
Greg	**UMember**

Reconsider the situation in which Alice resigns from being member of the Promotion Committee and becomes a Faculty Member

1. Delete the association (Alice, **PCMember**) from UA
2. Add the association (Alice, **Faculty**) to UA
3. PA is left unchanged*!*

Can we do better*?*

User *u* has permission *p* iff there exists a role *r* s.t. $(u, r) \in UA$ and $(r, p) \in PA$

EX: AT A SMALL UNIVERSITY A 3RD LOOK AT THE POLICY

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
PCMember	AssignGrades
PCMember	*ReceiveBenefits*
PCMember	**UseGym**
Faculty	AssignGrades
Faculty	GrantTenure
Faculty	**UseGym**
TA	AssignHWScores
TA	Register4Courses
TA	**UseGym**
UEmployee	*ReceiveBenefits*
UEmployee	**UseGym**
Student	Register4Courses
Student	**UseGym**
UMember	**UseGym**

Set of permissions associated to **PCMember** is a superset of Set of permissions associated to **Faculty**

Set of permissions associated to **TA** is a superset of Set of permissions associated to **Student**

There are other relationships, find them out!

EX: AT A SMALL UNIVERSITY
HOW TO TAKE AC DECISIONS YET AGAIN?

User Assignment (UA)

User	Role
Alice	**PCMember**
Bob	**Faculty**
Charlie	**Faculty**
David	**TA**
David	**Student**
Eve	**UEmployee**
Fred	**Student**
Greg	**UMember**

Permission Assignment (PA)

Role	Permission
PCMember	*ReceiveBenefits*
Faculty	AssignGrades
TA	AssignHWScores
UEmployee	*ReceiveBenefits*
Student	Register4Courses
UMember	**UseGym**
Faculty	GrantTenure

Role hierarchy ($\succeq$)

Least reflexive, antisymmetric, and transitive relation containing (**PCMember**, **Faculty**), (**Faculty**, **UEmployee**), (**TA**, **Student**), (**Student**, **UMember**), i.e.

Hasse diagram of partial order $\succeq$

User u has permission p iff there exist roles r, r' s.t. $(u, r') \in UA$ and $r' \succeq r$ and $(r, p) \in PA$

DIGRESSION

- What is a binary relation? Let S be a set, $R \subseteq S \times S$ is a binary relation over S
- What is a reflexive relation? A binary relation R over S is reflexive iff $\forall e \in S : (e, e) \in R$
- What is an antisymmetric relation? A binary relation R over S is antisymmetric iff $\forall e, e' \in S :$ if $(e, e') \in R$ and $(e', e) \in R$ then $e = e'$
- What is a transitive relation? A binary relation R over S is transitive iff $\forall e, e', e'' \in S :$ if $(e, e') \in R$ and $(e', e'') \in R$ then $(e, e'') \in R$

EXERCISE

- On the university scenario, check that

1. the access control module based on the first table (user-permission) takes the same decisions as the module based on relations *UA* and *PA*
2. the access control module based on relations *UA* and *PA* takes the same decisions as the module based on relations *UA*, *PA*, and the role hierarchy

3. **Note**: is the role hierarchy shown above complete*?* What about the role **UEmployee***?* How does it relate to **UMember***? (see next slide)*

EX: AT A SMALL UNIVERSITY HOW TO TAKE AC DECISIONS YET AGAIN?

User Assignment (UA)

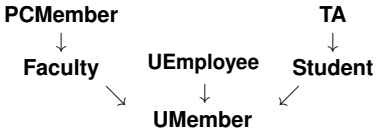
User	Role
Alice	**PCMember**
Bob	**Faculty**
Charlie	**Faculty**
David	**TA**
David	**Student**
Eve	**UEmployee**
Fred	**Student**
Greg	**UMember**

Permission Assignment (PA)

Role	Permission
PCMember	*ReceiveBenefits*
Faculty	AssignGrades
TA	AssignHWScores
UEmployee	*ReceiveBenefits*
Student	Register4Courses
UMember	UseGym
Faculty	GrantTenure
UEmployee	*ReceiveBenefits*

Role hierarchy ($\succeq$)

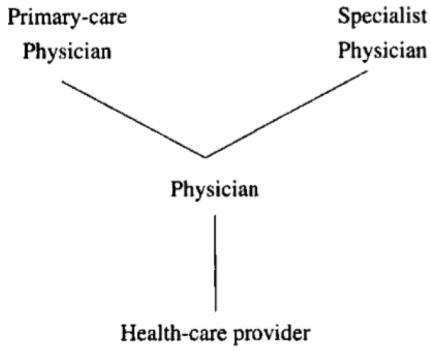
Least reflexive, antisymmetric, and transitive relation containing (**PCMember**, **Faculty**), (**Faculty**, **UEmployee**), (**TA**, **Student**), (**Student**, **UMember**), i.e.

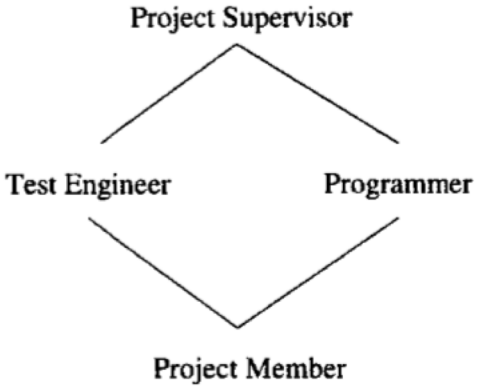


Hasse diagram of partial order $\succeq$

User u has permission p iff there exist roles r, r' s.t. $(u, r') \in UA$ and $r' \succeq r$ and $(r, p) \in PA$

ROLE HIERARCHY: EXAMPLES

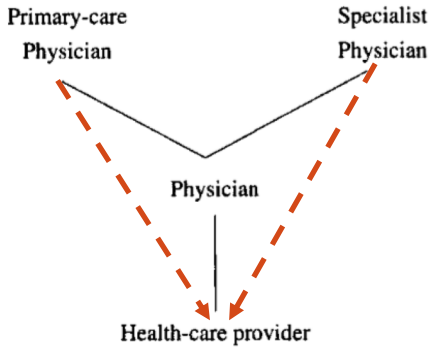




By convention more powerful (or senior) roles are shown toward the top of these diagrams, and less powerful (or junior) roles toward the bottom

72

ROLE HIERARCHY: EXAMPLES



Hasse diagram
type of mathematical diagram used to represent a finite partially ordered set, in the form of a drawing of its transitive reduction.

The **transitive reduction** of a directed graph D is another directed graph with the same vertices and as few edges as possible, such that if there is a (directed) path from vertex v to vertex w in D, then there is also such a path in the reduction

Project Member

By convention more powerful (or senior) roles are shown toward the top of these diagrams, and less powerful (or junior) roles toward the bottom

73

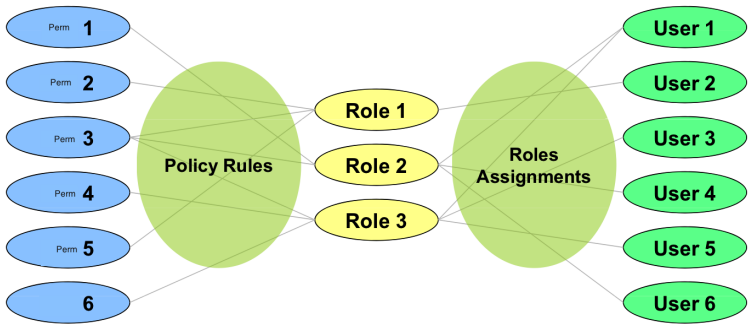
ROLE BASED ACCESS CONTROL

- Permissions are assigned to roles rather than to individual users
- Users are assigned to roles rather than directly to permissions
- This level of indirection facilitates user-permission management and provides additional security benefits

74

ROLE

- *A role is a job function within the context of an organization, with some associated semantics regarding the authority and responsibility conferred on the subjects to whom the role is assigned* [from ANSI RBAC Standard]
- Roles allow to manage permissions indirectly through roles



75

ROLES VS GROUPS

- Groups are collections of users
- A role is
  - a collection of users and
    a collection of permissions
- Sometimes role defined simply as a collection of permissions

USERS/SUBJECTS/PRINCIPALS

- Users are
  - human beings or
    other active agents (as programs, applications, …)
- Each individual should be known as exactly one user
  - That's why authentication is pre-requisite!

USER-ROLE ASSIGNEMENT (UA RELATION)

- A user can be a member of many roles
- Each role can have many users as members

78

PERMISSION-ROLE ASSIGNEMENT (PA RELATION)

- A permission can be a member of many roles
- Each role can have many associated permissions

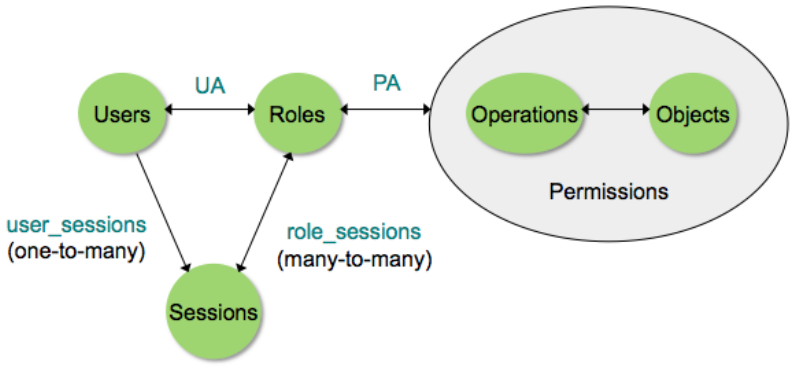
79

SESSIONS

- A user can invoke multiple sessions
- In each session a user can invoke any subset of roles that the user is a member of
  - Help implementing the **Principle of least Priviledge**

80

CORE RBAC



81

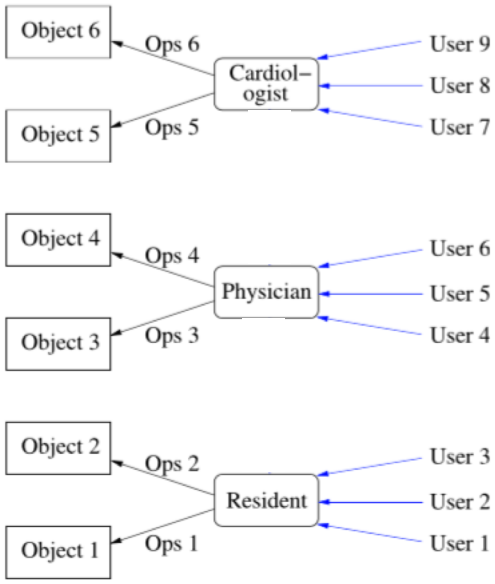
CORE RBAC

- U, R, P, S: users, roles, permissions, sessions (respectively)
- U x R ⊇ UA: many-to-many user-role assignment binary relation
- P x R ⊇ PA: many-to-many permission-role assignment binary relation
- *user* : S -> U: function mapping each session s to a user user(s)
  - function *user* does not change during the session's lifetime
- *roles* : S-> 2^R: function mapping each session s to a set of roles roles(s) such that {r | (user(s),r) in UA } ⊇ roles(s)
  - function *roles* may change during the session's lifetime since the user may decide to activate or deactivate some of the roles he/she is associated with in the relation UA
- *permissions*: S -> 2^P: function mapping each session s to a set *permissions*(s) =union of the set {p | (p,r) in PA } for each r in roles(s)

82

EXERCISE

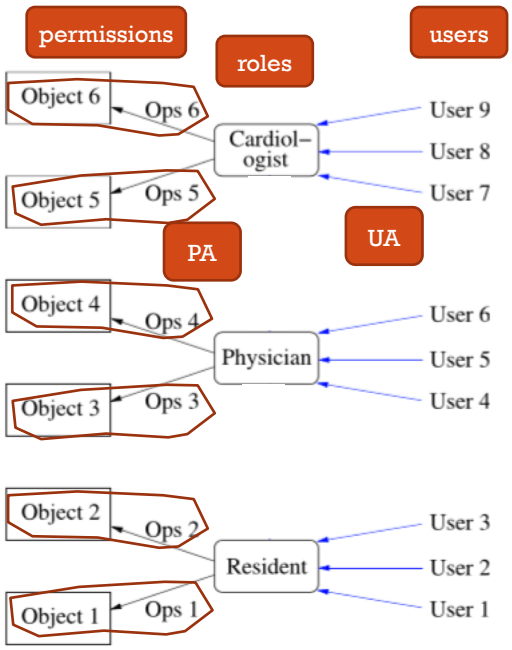
- Consider the RBAC model depicted on the right
- Write the UA relation
- Write the PA relation
- Assume the set S = {s1,s2} of sessions and define
  - users(s1) = User5 and
  - users(s2) = User9
- Define the upper bounds for
  - roles(s1)
  - permissions(s2)



83

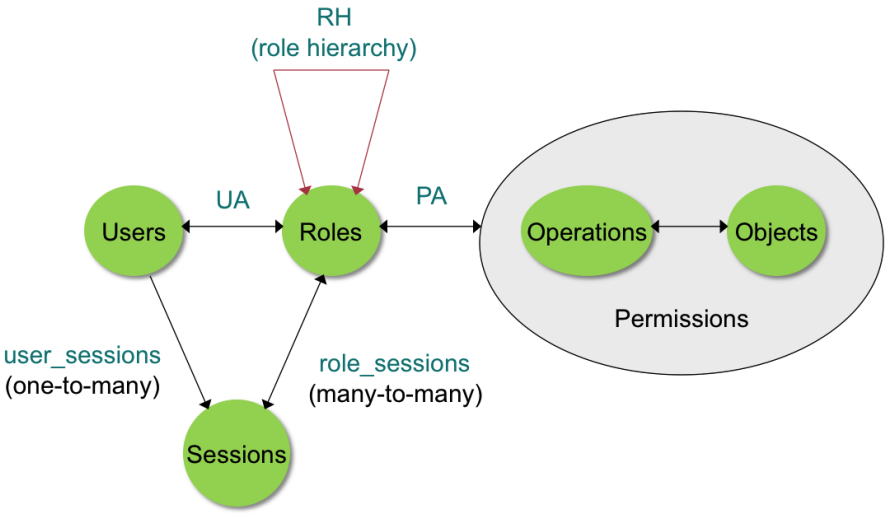
EXERCISE

- Draw the access matrix that corresponds to the RBAC model depicted on the right
- Hint



84

RBAC



85

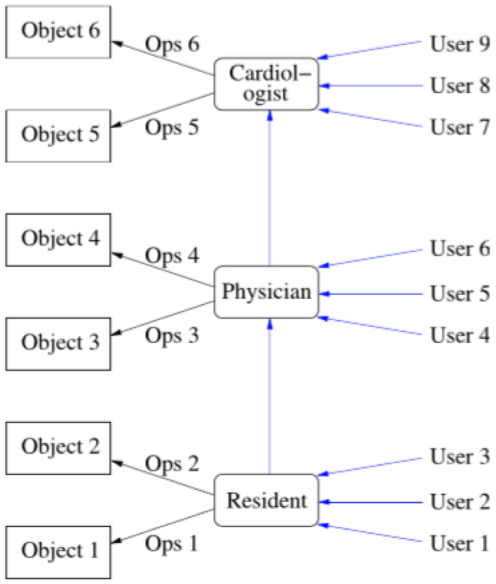
RBAC

- U, R, P, S, UA, PA, user: same as for core RBAC
- R x R ⊇ RH: partial order on R
  - (r,r') in RH also written as r >= r'
- *roles* : S-> 2^R: function mapping each session s to a set of roles roles(s) such that {r | there exists r' s.t. r'>=r and (user(s),r') in UA } ⊇ roles(s)
  - function *roles* is modified to consider the role hierarchy, i.e. all the roles that are associated to user(s) both in an explicit way in UA and in an implicit way through the role hierarchy
- *permissions*: S -> 2^P: function mapping each session s to a set *permissions*(s) =union of the set {p | there exists r' s.t. r>=r' and (p,r') in PA } for each r in roles(s)
  - also the function *permissions* is modified to consider the role hierarchy, i.e. all permissions associated to the roles that are below the role hierarchy wrt the roles that are active in the session under consideration

86

EXERCISE

- Consider the RBAC model depicted on the right
- Write the UA relation
- Write the PA relation
- Assume the set S = {s1,s2} of sessions and define
  - users(s1) = User5 and
  - users(s2) = User9
- Write
  - roles(s1)
  - permissions(s2)



87

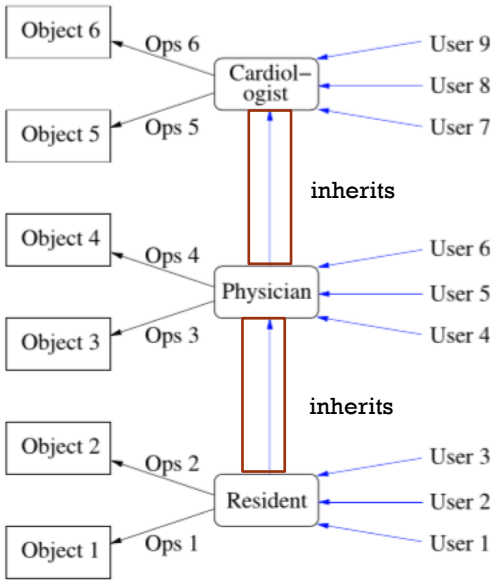
EXERCISE

- Draw the access matrix that corresponds to the RBAC model depicted on the right
- Hint

- Cardiologist >= Physician
- Physician >= Resident

Equivalently

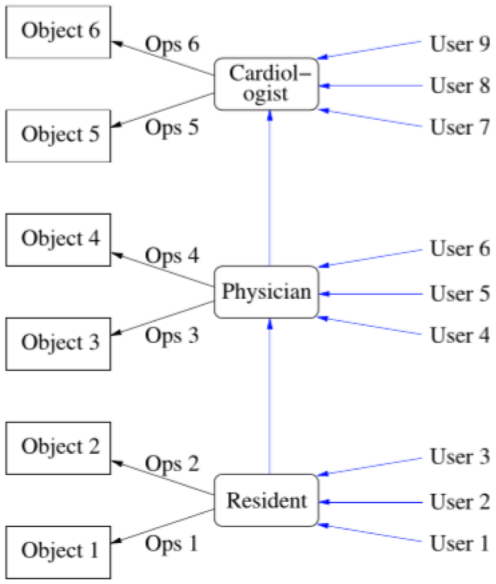
- Permissions associated to Resident are inherited by role Physician
- Permissions associated to Physician are inherited by role Cardiologist



88

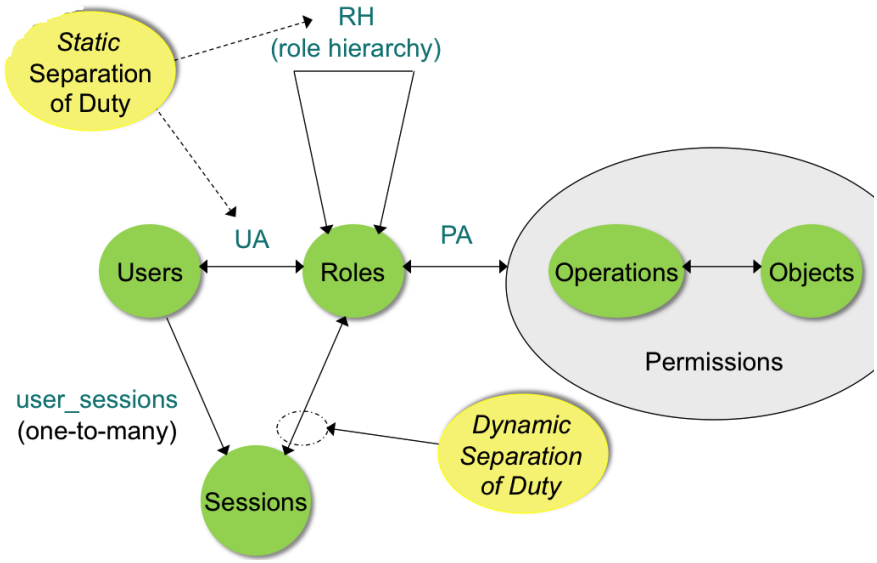
EXERCISE

- Draw the access matrix that corresponds to the RBAC model depicted on the right
- Hints
  - Write the relations UA and PA
  - Derive from UA and PA the relation associating users with permissions
  - From the previous relation derive the access matrix



89

CONSTRAINED RBAC



90

CONSTRAINED RBAC: SOD

- Separation of Duty (SoD) aka 4 eyes principle
  - Widely recognized
  - Captures conflict of interest policies to restrict authority of a single entity
    - Key to **fraud prevention**
- Example
  - A single person should not be allowed to both
    - Approve a check
    - Cash it

91

THE REFERENCE FOR RBAC

- *D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn and R. Chandramouli.*
  *"**Proposed NIST Standard for Role-Based Access Control**."*
  ACM Transactions on Information and System Security, Volume 4, Number 3, August 2001, pages 224-274.

92

RBAC: PROS AND CONS

- Pros
  - Easy to grasp the idea of roles
  - Easy to manage in principle
    - Roles decouple digital identities from permissions
    - Simply assign roles to a new subject, instead of deciding on access for each resource
    - Easy to revoke authorization for a subject by removing roles
  - Easy to tell through roles which permissions a subject has, and why

- Cons
  - Difficult to decide on the granularity of roles
    - Should we create separate roles for modifying client information and for deleting a client, or not?
    - Is authorization too broad / still up-to-date for all subjects having this role?
  - Role meaning is fuzzy
    - Employee position in company may be different from RBAC roles
    - Source of misunderstanding between admins and managers, for instance who assign them

93

RECAP QUESTIONS

- What is access control and what is its basic architecture?
- What is an access control matrix? ACLs? Capabilities?
- What is DAC?
- What is MAC? Define No Read Up and the No Write Down principles.
- What are the differences, advantages and disadvantages of DAC and MAC?
- What is RBAC? How does it simplify administration?

RECAP QUESTIONS

- Define the Principle of Least Priviledge
- What is a confused deputy?
- What is a trojan?
- What is a covert channel? How can a covert channel be created in MAC?
- How access control can mitigate command injection attacks?