# Fondements de la Sécurité (FOSE)

sovanna.tan@u-pec.fr

Source :

*Computer Security Art and Science*,

Matt Bishop, Addison-Wesley, 2002

# Plan du cours

1. Matrices de contrôle d'accès
2. Politiques de sécurité

# Problématique

- Comment formaliser une propriété de sécurité ?
- Comment définir une politique de sécurité ?
- Comment vérifier qu'un système répond à une politique de sécurité ? Dans quelles conditions peut-on le faire ?

# Matrices de contrôle d’accès

Lampton 1971, Graham et Denning 1971

# Matrice de contrôle d'accès

- Soit *O* l'ensemble des objets concernés par les droits.
- Soit $S \subset O$ l'ensemble des sujets ou objets actifs : utilisateur, processus
- Soit *R* l'ensemble des droits d'accès
- Pour chaque $(s,o) \in S \times O, A[s,o] \subset R$
- Le triplet *(S,O,A)* définit le système de protection.

# Le droit own

- Le droit **own** ou créateur propriétaire est souvent spécifique. Il accorde alors des privilèges spéciaux au créateur de l'objet.

Ex : Le créateur d'un fichier dispose du contrôle total sur le fichier. Les autres utilisateurs ont des droits restreints sur le fichier.

# Exemple de matrice 1 (1)

| | *fichier* 1 | *fichier* 2 | *processus* 1 | *processus* 2 |
|---|---|---|---|---|
| *processus* 1 | *read*, *write*, *own*, | *read* | *read*, *write*, *execute*, *own* | *write* |
| *processus* 2 | *append* | *read*, *own* | *read* | *read*, *write*, *execute*, *own* |

# Exemple de matrice 1 (2)

- Le droit *read* entre processus peut prendre diverses significations.

$$read \in A\left[processus\,1, processus\,2\right]$$

  - *processus* 1 peut lire l'état de *processus* 2
  - *processus* 1 accepte les messages provenant de *processus* 2

# Exemple de matrice 2 : Unix

- Droits Unix : $R = \{ r , w , x \}$

(*r* pour read, *w* pour write et *x* pour execute)

- Pour un répertoire *r* signifie lire le contenu et *x* aller dans le répertoire (avec la commande *cd* par exemple).
- Entre processus
  - *r* : recevoir des signaux
  - *w* : envoyer des signaux
  - *x* : exécuter en tant que processus fils

Le super utilisateur *root* a les droits sur tous les fichiers locaux.

# Exemple de matrice 3 : droits sur le réseau local (1)

| | poste de travail *A* | serveur *B* | serveur *C* |
|---|---|---|---|
| poste de travail *A* | *own* | *http* | *http* |
| serveur *B* | | *http, sftp, smtp, own* | *http, sftp, smtp* |
| serveur *C* | | *http, smtp* | *http, sftp, smtp, own* |

# Exemple de matrice 3 (2)

- *http*, *smtp*, et *sftp* représentent des échanges de messages en utilisant le protocole correspondant au nom entre les machines.
- *own* représente la possibilité d'ajouter un service.
- *A* dispose juste d'un client http.
- *B* et *C* sont des serveurs web, mail et sftp. Ils peuvent faire du http ou du smtp entre eux dans les deux sens.
- *C* n'est pas autorisé à faire du sftp sur *B*, mais *B* peut en faire sur C.

# Exemple de matrice 4 : langage de programmation (1)

- Une matrice peut décrire les accès dans un langage de programmation.
- Objets : variables
- Sujets : procédures

Exemple :

- 1 variable *compteur*,
- 3 procédures *inc_cmp*, *dec_cmp*, *manager*

La procédure *manager* est récursive.

# Exemple 4 : langage de programmation (2)

| | *compteur* | *inc_cmp* | *dec_cmp* | *manager* |
|---|---|---|---|---|
| *inc_cmp* | + | | | |
| *dec_cmp* | - | | | |
| *manager* | | *call* | *call* | *call* |

# Changement d'état du système de protection

- Soit $X_0=(S_0,O_0,A_0)$ l'état initial du système.
- L'évolution du système est représenté par des transitions : $X_0 |-_{\tau_1} X_1 |-_{\tau_2} \ldots X_i |-_{\tau_{i+1}} X_{i+1} \ldots$
- L'opération $\tau_i$ implique un changement d'état de la matrice de contrôle d'accès.
- On représente ces changements avec une procédure $c_i(p_{i,1}, p_{i,2}, \ldots, p_{i,n})$.

# Mise à jour de la matrice de contrôle d'accès

- Dans $c_i(p_{i,1}, p_{i,2}, \ldots, p_{i,n})$ les paramètres sont des sujets, des objets ou d'autres entrées du système .
- On définit des commandes primitives que l'on peut combiner pour former un commande (Harrison, Ruzzo et Ullman,1976).
- Appelé modèle HRU.

# Commandes primitives

- **create subject** *s* (*s* ne doit pas être déjà dans *S*), ajoute une ligne et un colonne à la matrice.
- **create object** *o* (*o* ne doit pas être déjà dans *O*), ajoute une colonne à la matrice.
- **enter** *r* **into** *A*[*s*,*o*] (rien si *r* est déjà dans *A*[*s*,*o*])
- **delete** *r* **from** *A*[*s*,*o*] (rien si *r* n'est pas dans *A*[*s*,*o*])
- **destroy subject** *s* (supprime la ligne et la colonne *s* dans la matrice)
- **destroy object** *o* (supprime la colonne *o*)

# Exemple de commande : Unix (1)

- Le processus *p* crée un fichier *f* qui lui appartient et dans lequel il peut lire et écrire :

**command** create_file(*p*,*f*)

**create object** *f*;

**enter** *own* **into** *A*[*p*,*f*];

**enter** *r* **into** *A*[*p*,*f*];

**enter** *w* **into** *A*[*p*,*f*];

**end**

# Exemple de commande : Unix (2)

- Le processus *p* crée un processus fils *q* :

**command** spawn_process(*p*,*q*)
  **create subject** *q*;
  **enter** *own* **into** *A*[*p*,*q*];
  **enter** *r* **into** *A*[*p*,*q*];
  **enter** *w* **into** *A*[*p*,*q*];
  **enter** *r* **into** *A*[*q*,*p*];
  **enter** *w* **into** *A*[*q*,*p*];
**end**

- Les quatre dernières primitives permettent à *p* et *q* de communiquer entre eux.

# Commande mono-opérationnelle

- Une commande *mono-opérationnelle* est une commande qui invoque une seule commande primitive.

# Commande conditionnelle

- Dans certaines commandes, des pré-conditions doivent être vérifiées pour qu'une primitive soit exécutée.
- On peut mettre une conjonction (and) de n'importe quel nombre de conditions.
- Une commande *mono-conditionnelle* (resp. *bi-conditionnelle*) comporte une seule (resp. deux ) condition
- La disjonction (ou) est inutile, elle se simule avec plusieurs commandes.
- La négation n'est pas permise. On ne peut pas tester l'absence d'un droit dans une commande.

# Exemple de commande mono-conditionnelle

- Si le processus *p* est propriétaire de *f*, *p* peut donner le droit de lire le fichier *f* à un processus *q*.

**command** grant_read_file(p,f,q)

**if** *own* **in** *A*[*p*,*f*]

**then**

**enter** *r* **into** *A*[*q*,*f*];

**end**

Cette commande est mono-opérationnelle.

# Exemple de commande bi-conditionnelle

- Un sujet *p* doit posséder le droit *r* et le droit spécial *copy* pour le transmettre à un sujet *q*.

**command** grant_read_file_2(*p*,*f*,*q*)
    **if** r **in** *A*[*p*,*f*] **and** *copy* **in** *A*[*p*,*f*]
    **then**
        **enter** *r* **into** *A*[*q*,*f*];
**end**

# Principe de l’atténuation des privilèges

- *Principe de l’atténuation des privilèges* : un sujet ne peut pas donner un droit qu’il ne possède pas.

# Système sûr pour un droit

- On dit qu'un système laisse filtrer le droit *r*, si *r* est ajouté à un élément de la matrice de contrôle d'accès, alors qu'aucun élément ne contenait *r*.
- Un système qui ne laisse jamais filtrer le droit *r* est *sûr* vis à vis de *r*.
- Un système qui laisse filtrer le droit *r n'est pas sûr* vis à vis de *r*.

  Le terme sûr s'applique au modèle abstrait et l'attribut sécurisé à l'implémentation.

# Résultats théoriques

- Dans le cas général, il n'existe pas d'algorithme qui étant donné un système de protection peut décider s'il est sûr ou non vis à vis d'un droit. Ce problème est indécidable.
- Le problème de savoir si un système qui n'a que des commandes mono-opérationnelles est sûr pour un droit donné est lui décidable.

# Théorème 1

- Théorème : *Le problème de savoir si un système qui n'a que des commandes mono-opérationnelles est sûr pour un droit donné est décidable.*
- Idée de la preuve :
  - Transformer le système général en un système plus simple, équivalent au niveau des droits laissés filtrés.
  - Tester toutes les suites de commandes d'une certaine longueur pour voir si le nouveau système laisse filtrer un droit donné ou non .

# Preuve du théorème 1 (1)

- Preuve :

Soit $S=(S_0, O_0, A_0, C)$ un système de protection où *C* est un ensemble fini de commandes mono-opérationnelles.

Soit $c_0, c_1, \ldots, c_k$ une suite de commandes de longueur minimale qui laisse filtrer le droit *r* à partir de l'état initial.

Si on omet toutes les commandes **destroy** ou **delete,** on obtient encore une suite de commande qui laisse filtrer *r*. La suite $c_0, c_1, \ldots, c_k$ est minimale, elle ne contient donc pas de telles commandes.

Si l'état initial ne contient pas de sujet, soit *p* le premier sujet créé. Si l'état initial contient des sujets, soit *p* un de ces sujets.

# Preuve du théorème 1 (2)

On construit un système équivalent en transformant les commandes de la manière suivante :

- Supprimer les **create** (sauf éventuellement pour *p*)
- Remplacer les sujets et les objets qui auraient dû être créés par les commandes par *p* dans les **enter** et dans les tests.

Le système obtenu laisse filtrer exactement les mêmes droits que le système de départ.

Lorsque le système évolue, il a au plus $|S_0|+1$ sujets et $|O_0|+1$ objets.

# Preuve du théorème 1 (3)

Soit $c'_0, c'_1, \ldots, c'_{k'}$ la séquence équivalente à $c_0, c_1, \ldots, c_k$ dans le nouveau système, de longueur minimale.

Cette séquence ne contient que des ajouts de droits et éventuellement un **create subject**. Sa longueur *k'* est donc majorée par $K = \left(|S_0| + 1\right)\left(|O_0| + 1\right)|R| + 1$ .

Pour savoir si le nouveau système est sûr pour un droit donné, il suffit donc d'énumérer toutes les suites de commandes de longueur $K$ . Le problème est donc décidable. □

# Théorème 2

- Théorème : *Le problème de savoir si un système de protection est sûr pour un droit donné est indécidable dans le cas général.*
- Idée de la preuve : réduire le problème de l'arrêt d'une machine de Turing au problème de sûreté pour un droit donné (i.e. construire un système de protection qui simule une machine de Turing. )

# Machine de Turing (1)

Une machine de Turing possède

- un ruban infini à droite avec des cases numérotées 1, 2, 3, … ;
- une tête de lecture qui pointe sur une des cases ;
- un alphabet fini doté d'un caractère spécial *blank*, au départ un nombre fini de cases ne contient pas *blank* ;
- un ensemble d'états fini dont un état initial $q_0$ et un état final $q_f$ ;
- Une fonction de transition qui décrit l'état de la bande

$$\delta : Q \times A \rightarrow Q \times A \times \{left, right\}.$$

# Machine de Turing (2)

Si $\delta(q,a)=(q',c,left)$ (resp. *right*) et si la machine est dans l'état *q* et la tête de lecture se trouve sur *a*, alors *a* est remplacé par *c* et la tête de lecture se déplace d'une case vers la gauche (resp. droite). Dans le cas d'un déplacement à gauche, si la tête se trouve dans la case la plus à gauche, elle ne bouge pas.

Une configuration de la machine est définie par son état, la position de la tête et la partie utile du ruban qui est toujours finie si on omet la partie droite constituée uniquement de *blank*.

# Arrêt d’une machine de Turing

- On dit qu’une machine s’arrête si elle est dans une configuration où l’état est terminal.
- Le *problème de l’arrêt d’une machine* de Turing est le suivant : la machine peut-elle atteindre une configuration où elle s’arrête ?
- Ce problème est indécidable.

# Preuve du théorème 2 (1)

- Preuve :

Etant donné une machine de Turing *T*, on construit un système de protection *S* dont l'ensemble des droits est $Q \cup A \cup \{owns, end\}$ tel que la machine s'arrête si et seulement si *S* est sûr pour le droit $q_f$. A un instant donné la case *i* correspond au sujet $S_i$. Dans le système, $S_i$ *owns* $S_{i+1}$.

Si la tête est sur la case *i* et si la case *i* contient *a* et *T* est dans l'état *q* alors $S_i$ a les droits *a* et *q* sur lui même.

# Preuve du théorème 2 (2)

Les commandes qui simulent les transitions sont:

- Pour une transition $\delta(p,a)=(q,b,left)$ et $i>1$

**command** $c_{p,a}(s_i, s_{i-1})$
**if** *owns* **in** $A[s_{i-1},s_i]$ **and** *p* **in** $A[s_i,s_i]$ **and** *a* **in** $A[s_i,s_i]$
**then**
    **delete** *p* **from** $A[s_i,s_i]$;
    **delete** *a* **from** $A[s_i,s_i]$;
    **enter** ***b*** **into** $A[s_i,s_i]$;
    **enter** ***q*** **into** $A[s_{i-1},s_{i-1}]$;
**end**

# Preuve du théorème 2 (3)

- Pour une transition $\delta(p,a)=(q,b,left)$ et $i$=1(la tête pointe sur la case la plus à gauche.)

**command** $c_{p,a}(s_1, s_1)$

**if** *owns* **in** $A[s_1,s_1]$ **and** *p* **in** $A[s_1,s_1]$ **and** *a* **in** $A[s_1,s_1]$

**then**

**delete** *p* **from** $A[s_1,s_1]$;

**delete** *a* **from** $A[s_1,s_1]$;

**enter** ***b*** **into** $A[s_1,s_1]$;

**enter** ***q*** **into** $A[s_1,s_1]$;

**end**

# Preuve du théorème 2 (4)

- Pour une transition $\delta(p,a)=(q,b,right)$ et la tête ne pointe pas sur la case la plus à droite :

**command** $c_{p,a}(s_i, s_{i+1})$
**if** *owns* **in** $A[s_i,s_{i+1}]$ **and** *p* **in** $A[s_i,s_i]$ **and** *a* **in** $A[s_i,s_i]$
**then**
    **delete** *p* **from** $A[s_i,s_i]$;
    **delete** *a* **from** $A[s_i,s_i]$;
    **enter** ***b*** **into** $A[s_i,s_i]$;
    **enter** ***q*** **into** $A[s_{i+1},s_{i+1}]$;
**end**

# Preuve du théorème 2 (5)

- Pour une transition $\delta(p,a)=(q,b,right)$ et la tête pointe sur la case la plus à droite, il faut alors ajouter un sujet correspondant à la case blanche qui sera remplie à la transition suivante :

**command** $C_{p,a}(s_i, s_{i+1})$
**if** ***end*** **in** $A[s_i,s_i]$ **and** *p* **in** $A[s_i,s_i]$ **and** *a* **in** $A[s_i,s_i]$
**then**
**delete** ***end*** **from** $A[s_i,s_i]$;
**create subject** $s_{i+1}$;
**enter** ***owns*** **in** $A[s_i,s_{i+1}]$;
**enter** ***end*** **in** $A[s_{i+1},s_{i+1}]$;
**delete** *p* **from** $A[s_i,s_i]$;
**delete** *a* **from** $A[s_i,s_i]$;
**enter** ***b*** **into** $A[s_i,s_i]$;
**enter** ***q*** **into** $A[s_{i+1},s_{i+1}]$;
**end**

# Preuve du théorème 2 (6)

- Dans la matrice de S, les droits représentent exactement une configuration de la machine de Turing T.

| ruban | a | b | c | d | *blank* |
|---|---|---|---|---|---|
| case n° | 1 | 2 | 3 tête ↑ | 4 | … |

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|
| $s_1$ | *a,owns* | *owns* | | |
| $s_2$ | | *b* | *owns* | |
| $s_3$ | | | *c,p* | *owns* |
| $s_4$ | | | | *d,end* |

# Preuve du théorème 2 (7)

- A chaque configuration de T, il y a au plus une seule commande applicable. Le système S simule exactement T.
- La machine T entre dans l'état terminal $q_f$ si et seulement si S laisse filtrer le droit $q_f$. Le problème de l'arrêt de T est indécidable, la sûreté de S pour $q_f$ l'est donc aussi. □

# Les Listes de Contrôle d'Accès (Access Control Lists)

- Colonnes d'une matrice de contrôle d'accès

| | *file1* | *file2* | *file3* |
|---|---|---|---|
| *Andy* | rx | r | rwo |
| *Betty* | rwxo | r | |
| *Charlie* | rx | rwo | w |

ACLs pour les droits dans les systèmes de fichiers :

- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

# Les listes de capacités

- Liste de capacité : ligne d'une matrice de contrôle d'accès.
- Elle donne une liste d'objets et droits associés à un agent.

# Politiques de sécurité

# Politique de sécurité

- Une *politique de sécurité* partage les états d'un automate fini en un ensemble d'états *autorisés* ou *sécurisés* et un ensemble d'états *interdits*.
- Un *système sécurisé* est un système qui commence dans un état autorisé et ne peut pas aller dans un état interdit.
- Il se produit une *brèche de sécurité* lorsqu'un système entre dans un état interdit.

# Confidentialité, intégrité

- Soit *X* un ensemble d'entités et *I* un ensemble d'information. L'ensemble *I*, a la propriété de *confidentialité, est confidentiel* par rapport à *X* si aucun élément de *X* ne peut obtenir d'information sur *I*.
- L'ensemble *I* a la propriété *d'intégrité, est intègre* vis à vis de *X* , si tous les éléments de *X* font confiance à *I*.

# Disponibilité

- Soit *X* un ensemble d'entités et *I* une ressource. La ressource *I* a la propriété de *disponibilité*, *est disponible* pour *X* si tous les éléments de *X* peuvent accéder à *I*.

# Mécanisme de sécurité, modèle de sécurité

- Un *mécanisme de sécurité* est une entité ou une procédure destinée à assurer une partie de la politique de sécurité.
- Un *modèle de sécurité* est un modèle qui représente une politique particulière ou un ensemble de politiques.

# Types de politique de sécurité

- Une *politique de sécurité militaire ou gouvernementale* est une politique conçue pour assurer essentiellement la confidentialité.
- Une *politique de sécurité commerciale* est une politique de sécurité qui se préoccupe principalement de l'intégrité.
- Une *politique de confidentialité* ne s'occupe que de la confidentialité.
- Une *politique d'intégrité* ne s'occupe que d'intégrité.

# Le rôle de la confiance

- Maxime : En matière de sécurité, toutes les théories et tous les mécanismes reposent sur des hypothèses. Il faut être conscient de ces hypothèses et les assimiler pour bien comprendre l'efficacité des politiques, des procédures et des mécanismes de sécurité.
- Si ces hypothèses sont faussent, elles détruisent la superstructure sur laquelle est bâtie la sécurité.

# Exemple : installer un patch pour un système d'exploitation

Lors de l'installation d'un patch sur un OS (Operating System), on suppose que :

1. le patch provient du vendeur et est intègre ;
2. le vendeur a testé son patch de façon approfondie (développés sous pression, les patchs peuvent introduire d'autres trous de sécurité);
3. l'environnement du vendeur est le même que le sien (en particulier au niveau des droits) ;
4. le patch a été installé correctement.

# Exemple : vérification formelle de programme

Un programme a été prouvé correct à l'aide d'un démonstrateur de théorème (theorem prover). Cela implique que l'on suppose que:

1. la preuve est correcte, que la partie automatisé du démonstrateur est correcte ;
2. Les hypothèses fournies au démonstrateur correspondent aux pré-conditions dans lesquelles le programme sera exécuté ;
3. le compilateur, l'éditeur de liens, le chargeur, les bibliothèques sont corrects ;
4. Le hardware ne génère pas d'erreur.

# Contrôle d’accès discrétionnaire

- Si une personne peut mettre en œuvre un mécanisme pour permettre ou interdire l’accès à un objet, on dit que le mécanisme est un mécanisme de *contrôle d’accès discrétionnaire* (DAC pour Discretionary Access Control) ou *fondé sur l’identité* (IBAC pour Identity-Based Access Control).

# Contrôle d’accès obligatoire

- Lorsqu’un mécanisme sur lequel les utilisateurs ne peuvent pas influer, contrôle l’accès à un objet on dit que le contrôle d’accès est *obligatoire* ou *mandataire* (MAC Mandatory Access Control) ou encore *fondé sur des règles (RBAC Rule Based Access Control)..*

# Le modèle de Bell-LaPadula

1973

# Présentation du modèle de Bell-LaPadula

- Modèle de politique de confidentialité qui a influé le livre orange sur la sécurité du Dod.
- .Ce modèle se fonde sur le principe militaire « need to know ». Les sujets n'ont accès qu'aux informations dont ils ont besoin.
- Il combine un mode de contrôle d'accès MAC et DAC. Quand le MAC autorise l'accès, on regarde les droits DAC.

# Habilitation, sensibilité

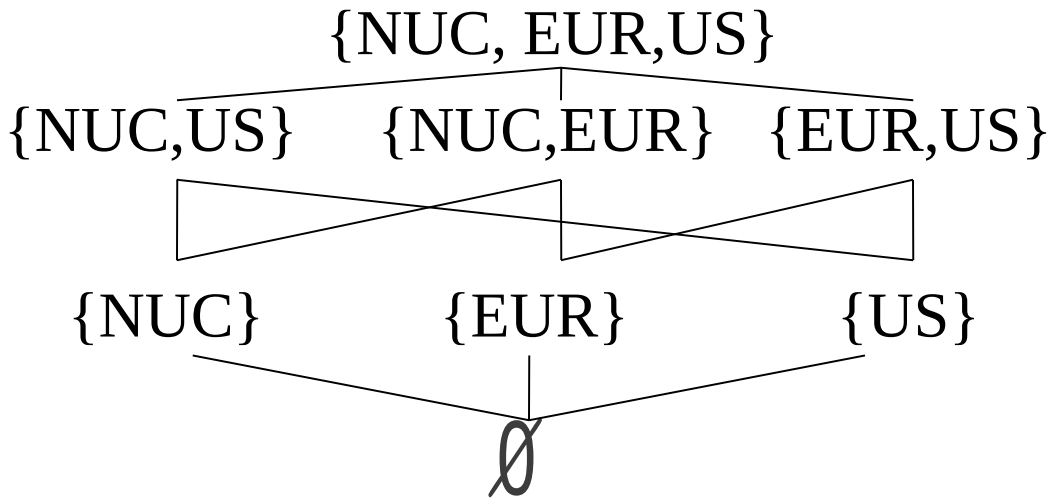
- Les sujets et les objets sont classés par niveau, niveau d'*habilitation* pour les sujets et niveau de *confidentialité* pour les objets.
- L'ensemble des niveaux est muni d'un ordre partiel. C'est un treillis : toute paire d'éléments possède une borne inférieure et une borne supérieure.
- Le but du modèle est d'empêcher un sujet d'accéder aux objets dont la confidentialité est plus grande que son niveau d'habilitation.

# Exemple Bell-LaPadula (1)

- Niveaux de confidentialité :
  - TOP SECRET (TS)
  - SECRET (S)
  - CONFIDENTIAL  (C)
  - UNCLASSIFIED (UC)
- Catégories ou sujets d’intérêt
  - NUC
  - EUR
  - US

# Exemple Bell-LaPadula (2)

- L’ensemble des parties de l’ensemble des catégories muni de la relation d’inclusion forme un treillis.



# Exemple Bell-LaPadula (3)

- Un *niveau de sécurité* est un couple (confidentialité,ensemble de sujets).
- On définit la relation d'ordre partiel *dom* (domine) par *(L,P) dom (L',P')* si et seulement si $L' \leq L$ et $P' \subseteq P$.
- L'ensemble des niveaux de sécurité muni de la relation *dom* est un treillis.

# Exemple Bell-LaPadula (4)

- Georges a le niveau (S,{NUC,EUR}).
- DocA est (C,{NUC}).
- DocB est (S,{EUR,US}).
- DocC est (S,{EUR}).

On a Georges *dom* DocA, Georges *dom* DocC.

Mais la relation Georges *dom* DocB est fausse.

# No reads up

- *No reads up* : Un sujet *s* peut lire un objet o si et seulement si *s dom o* et *s* a le droit discrétionnaire *de lire o*.
- Cette règle ne suffit pas pour assurer la sécurité.
  - Si Paul est habilité (S,{NUC,EUR,US}), il peut lire DocB et le copier dans DocA ce qui permet alors à Georges de lire DocB.

# No writes down

- Pour qu'un système évolue en respectant les droits d'accès, il faut ajouter la règle suivante :
- *No writes down* : Un sujet *s* peut écrire sur un objet *o* si et seulement si *o dom s* et si *s* a le droit discrétionnaire d'écrire sur *o.*
  - Paul ne peut plus écrire dans DocA.
- Cette règle sert à neutraliser les chevaux de Troie.

# Flot d'information

- Un flot d'information consiste en une suite dans laquelle alternent lectures et écritures.
- Théorème : *Une information o de haut niveau de confidentialité C(o) ne peut pas être transmise dans un objet o' de niveau plus bas C(o') (C(o)>C(o')).*
- Pour que le sujet *s* de niveau d'habilitation *C(s)* puisse lire *o*, il faut $C(o) \leq C(s)$.
- Pour que *s* écrive dans *o'*, il faut $C(s) \leq C(o')$.
- Il faut donc $C(o) \leq C(o')$. □

# Théorème de sécurité de base

- Théorème : *Soit* $\Sigma$ *un système avec un état initial sécurisé* $\sigma_0$. *Si toutes les transformations du système respectent les règles « no reads up » et « no writes down » alors tous les états* $\sigma_i, i \geq 0$ *sont sécurisés.*

# Remarque

- Dans ce modèle, un sujet ne peut pas écrire un document destiné à un sujet avec un niveau d'habilitation plus bas.
- Pour cela, on permet à un sujet de diminuer son niveau d'habilitation et de passer de son niveau d'habilitation maximum à un niveau d'habilitation plus bas.

# Le principe de tranquillité

- Augmenter le niveau de sécurité d'un objet
  - Une information disponible à un moment donné pour certains sujets n'est plus disponible.
  - Cela n'a pas d'effet si les sujets ont déjà pris connaissance de l'information.
- Diminuer le niveau de sécurité d'un objet
  - Correspond à un « write down » c'est le *Problème de déclassification.*
  - On remédie au problème en définissant un ensemble de sujets de confiance qui vont enlever les informations sensibles avant la déclassification.

# Types de tranquillité

- Le *principe de tranquillité forte* stipule que les niveaux de sécurité (niveaux de confidentialité et habilitations) ne changent pas pendant la vie du système.
- Le *principe de tranquillité faible* stipule que les changements de niveaux de sécurité n'affectent pas les règles « no reads up » et « no writes down ».

# Modèles de politique d’intégrité

# Requis pour assurer l'intégrité (Lipner 1982)

1. Les utilisateurs n'écrivent pas leurs programmes, ils utilisent un système d'information en production.
2. Les développements s'effectuent sur un système hors production.
3. Une procédure spécifique assure la mise en production.
4. Cette procédure doit être contrôlée et auditée.
5. Les responsables et les auditeurs doivent avoir accès au système et à ses logs.

# Principes à mettre en œuvre

- Séparation des tâches
  - L’installation ne doit pas être effectuée par un développeur.
- Séparation des fonctions
  - Le développement ne doit pas s’effectuer sur un système en production.
- Audit

En outre, s’ajoute une problématique de confidentialité :

- vérifier que les informations mises à la disposition du public ne permettent pas de déduire des informations sensibles.

# Niveaux d’intégrité

- Les sujets et les objets ont des niveaux d’intégrité, qui diffèrent des niveaux de sécurité et sont maintenus séparément. Ils servent à empêcher les modifications par des sujets auxquels on ne ne fait pas confiance.
- Plus le niveau est élevé, plus la confiance en la validité des informations est grande.

# Le modèle de Biba (1977)

- Les règles :
  - Un sujet *s* peut lire un objet *o* ssi $i(s) \leq i(o)$.
  - *s* peut écrire sur *o* ssi $i(s) \geq i(o)$.
  - *s* peut exécuter un sujet *s'* ssi $i(s) \geq i(s')$.
- Lorsqu'une information est transférée d'un objet o à un objet o', $i(o) \geq i(o')$.
- Ce modèle est le dual du modèle de Bell-LaPadula.

# Exercice 1

- Soit un ordinateur à trois utilisateurs Alice Bob et Cyndy. Alice possède le fichier *alicerc*. Bob et Cyndy peuvent le lire. Cyndy peut lire et écrire dans le fichier *bobrc* dont Bob est le propriétaire. Alice peut seulement le lire.Seule Cyndy peut lire et écrire dans le fichier *cyndyrc* qu'elle possède. Le propriétaire de chaque fichier peut l'exécuter.
  - Créer la matrice de contrôle d'accès.
  - Cyndy donne à Alice la permission de lire *cyndyrc* et Alice supprime le droit de lire *alicerc* pour Bob. Donner la nouvelle matrice de contrôle d'accès.

# Exercice 2

- Soit l'ensemble de droits *{read, write, execute, append, list, modify, own }.*
  - Ecrire la commande *delete_all_rights(p,q,s)* qui fait que le sujet *p* efface tous les droits que le sujet *q* a sur l'objet *s*.
  - Modifier la commande de sorte que l'effacement ne puisse avoir lieu que si *p* possède le droit *modify* sur *s*.
  - Modifier la commande de sorte que l'effacement ne puisse avoir lieu que si *p* possède le droit *modify* sur *s* et si *q* ne possède pas le droit *own* sur *s*.

# Exercice 3

- Soit les niveaux de confidentialité TOP SECRET, SECRET, CONFIDENTIAL et UNCLASSIFIED ordonnés du plus haut au plus bas et les catégories A, B et C. Les contrôles d'accès discrétionnaires autorisent l'accès. Donner dans chaque cas, le cas échéant, le ou les types d'accès permis read, write :
  - Paul (TOP SECRET,{A,C}) veut accéder à un document (SECRET, {B,C}) ;
  - Anna (CONFIDENTIAL, {C}) veut accéder à un document (CONFIDENTIAL, {B}) ;
  - Jesse (SECRET,{C}) veut accéder à un document (CONFIDENTIAL,{C}) ;
  - Sammi (TOP SECRET, {A,C}) veut accéder à un document (CONFIDENTIAL,{A}) ;
  - Robin qui n'a pas d'habilitation veut accéder à un document (CONFIDENTIAL, {B}).

# Exercice 4

Dire dans chacun des cas suivants s'il s'agit d'une politique d'accès discrétionnaire, mandataire ou une combinaison de ces politiques.

1. Le système de droits des fichiers sous Unix.
2. Un espace militaire réservé aux généraux.
3. La base de données des notes des étudiants d'une université sachant qu'un enseignant ne peut consulter l'ensemble des notes d'un étudiant que si l'étudiant a accordé nominativement par écrit la permission à l'enseignant.