Systèmes d'exploitations

Cours 6: Sécurité, virtualisation, cloud

Matthieu Lemerre
CEA LIST

Année 2020-2021

Introduction

- On a vu comment isoler des processus; c'est la première des sécurité.
- ➜ Comment construire un système sécurisé à partir de ca?
  - Comment construire le système pour isoler les parties sensibles?
  - Comment contrôler les communications entre processus?
  - Comment limiter les privilèges d'un processus?

Applications du cours

- Développement de systèmes avec des considérations de sécurité

Applications du cours

- Développement de systèmes avec des considérations de sécurité
  - En réalité: tous les systèmes!

Applications du cours

- Développement de systèmes avec des considérations de sécurité
  - En réalité: tous les systèmes!
- Il faut prendre en compte la sécurité depuis le départ:

  The only sound approach to the provision of secure computer systems is to design security into those systems right from the start.
  – John Rushby

Applications du cours

- Développement de systèmes avec des considérations de sécurité
  - En réalité: tous les systèmes!
- Il faut prendre en compte la sécurité depuis le départ:

  The only sound approach to the provision of secure computer systems is to design security into those systems right from the start. – *John Rushby*
- Exemples de systèmes où la sécurité n'a pas été pensée au départ
  - Microsoft Word 95
  - Serveur mail sendmail
  - Puces Intel (attaques Meltdown et Spectre)
  - La plupart des objets connectés
  - Beaucoup de systèmes multimédia (autoradios...)

Sommaire

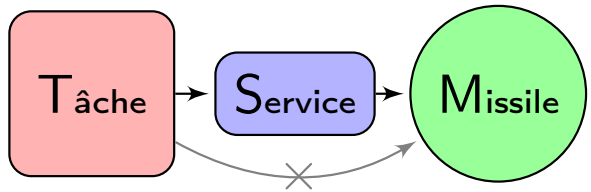
Garantir une propriété de sûreté ou sécurité

- Comment garantir une propriété de sécurité
  - Sans avoir à faire confiance à tout le système?

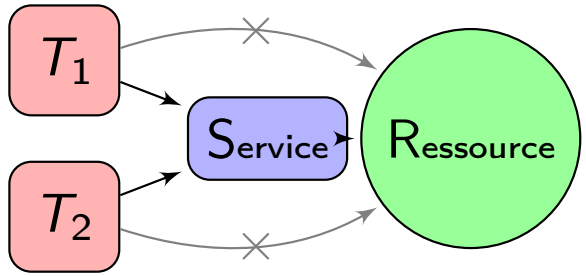
Garantir une propriété de sûreté ou sécurité

- Comment garantir une propriété de sécurité
  - Sans avoir à faire confiance à tout le système?
- Découpage des fonctionnalités en processus indépendants
  - La "partie sensible" (service) doit être *minimale* et *incontournable*

- Exemples:
  - “Pas de tir sans être armé”



  - Partage sécurisé de ressource entre tâches



- La sécurité repose uniquement sur un petit ensemble bien identifié de code

Principes de sécurité de Saltzer et Schroeder (1)

Open design Ne pas se reposer sur le fait que les attaquants ignoreront certains détails.

Psychological acceptability Si la sécurité d'un système le rend trop pénible à utiliser, les utilisateurs contourneront les protections (e.g. politiques de changement de mots de passe, pare-feu trop restrictifs. . .)

Work factor Concevoir le système en évaluant les ressources d'un attaquant.

(Compromise recording) On peut parfois reporter son attention sur la détection d'une compromission, plutôt que sur sa prévention. (mise sous scellé, logs de confiance)

Sécurisation d'un système (avionique): Structuration

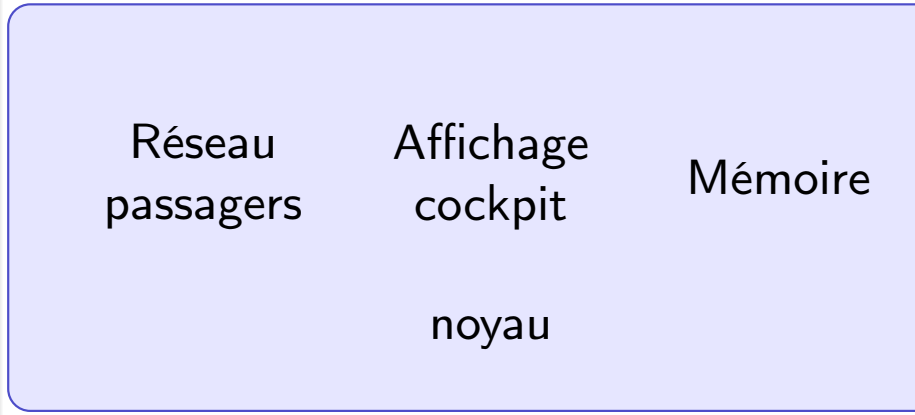
Pilote automatique	Décodage Film	Caméra extérieure
Réseau passagers	Affichage cockpit	Allocation Mémoire

Sécurisation d'un système (avionique): Structuration

- Séparation des privilèges
  - Isole les compromission

Mise en œuvre

- Séparation des tâches en domaines de protection (processus)

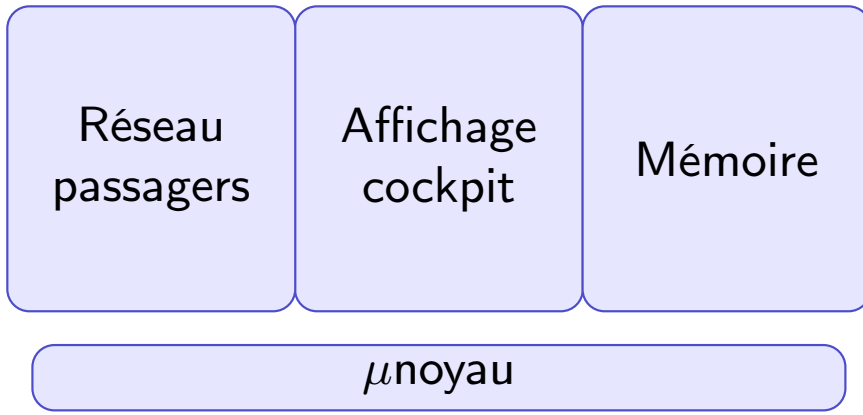


Sécurisation d'un système (avionique): Structuration

- Séparation des privilèges
  - Isole les compromission
- Minimisation des mécanismes communs
  - Minimise l'impact d'une compromission

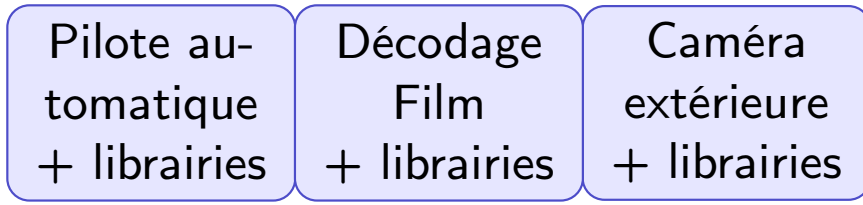
Mise en œuvre

- Séparation des tâches en domaines de protection (processus)
- Séparation des services en domaines de protections (approche micro-noyau)



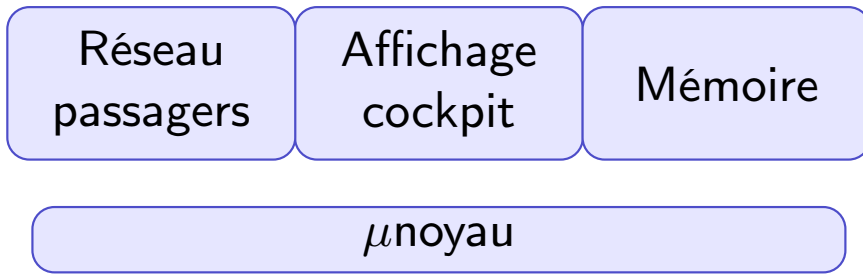
Sécurisation d'un système (avionique): Structuration

- Séparation des privilèges
  - Isole les compromission
- Minimisation des mécanismes communs
  - Minimise l'impact d'une compromission

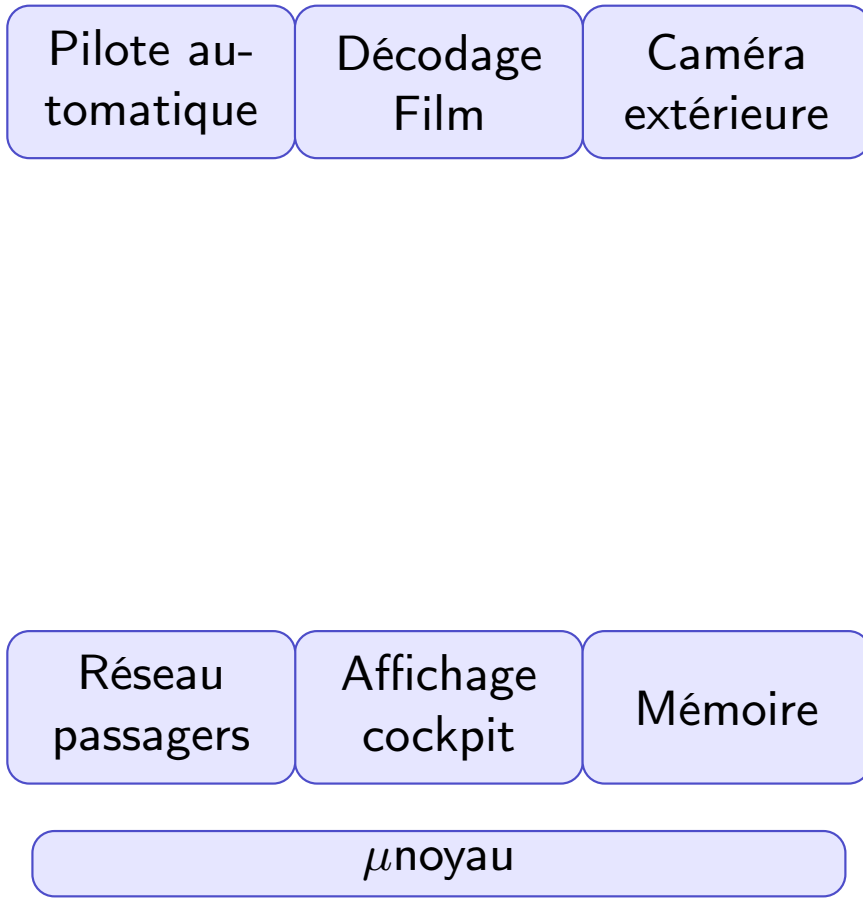


Mise en œuvre

- Séparation des tâches en domaines de protection (processus)
- Séparation des services en domaines de protections (approche micro-noyau)
- Minimisation des noyau et services (approche exo-noyau et hyperviseur)



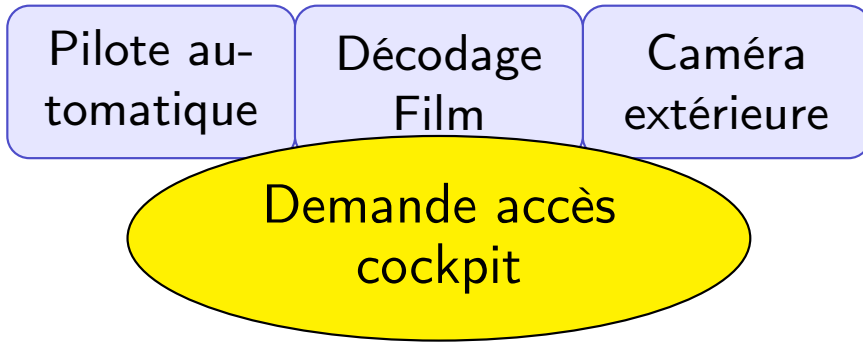
Sécurisation d'un système (avionique): Gestion des privilèges



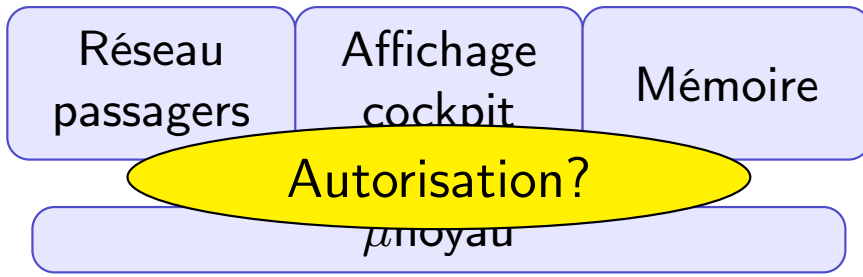
Mise en œuvre

Sécurisation d'un système (avionique): Gestion des privilèges

- Médiation complète
  - Vérification systématique des privilèges

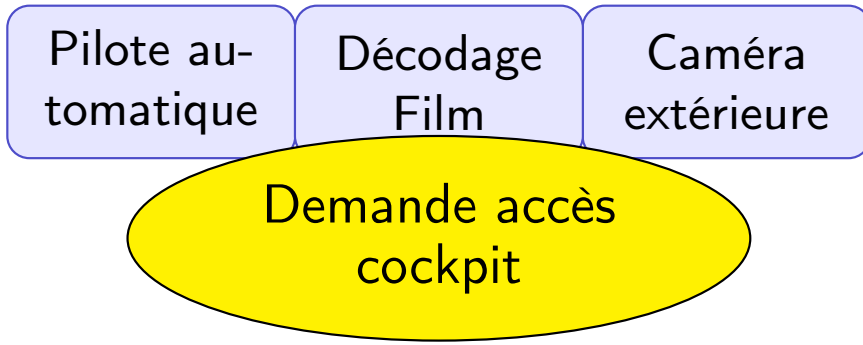


Mise en œuvre

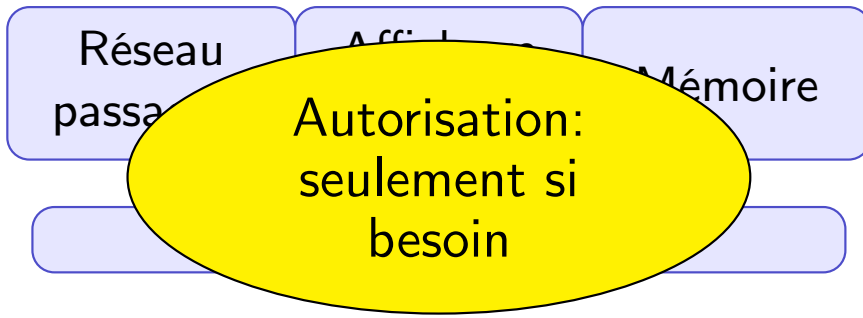


Sécurisation d'un système (avionique): Gestion des privilèges

- Médiation complète
  - Vérification systématique des privilèges
- Principe du moindre privilège
  - Autoriser seulement le nécessaire

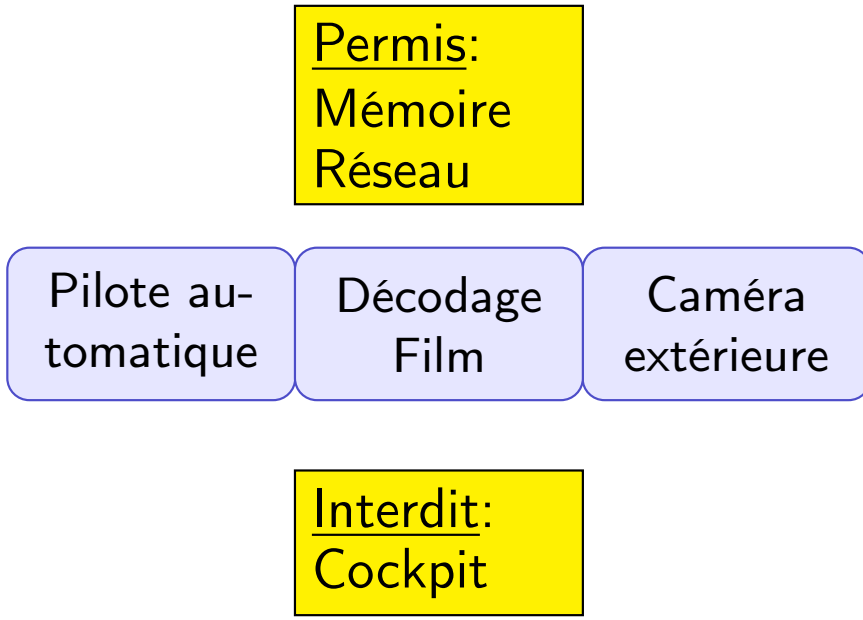


Mise en œuvre



Sécurisation d'un système (avionique): Gestion des privilèges

- Médiation complète
  - Vérification systématique des privilèges
- Principe du moindre privilège
  - Autoriser seulement le nécessaire
- Sûr par défaut
  - Tout ce qui n'est pas explicitement autorisé est interdit

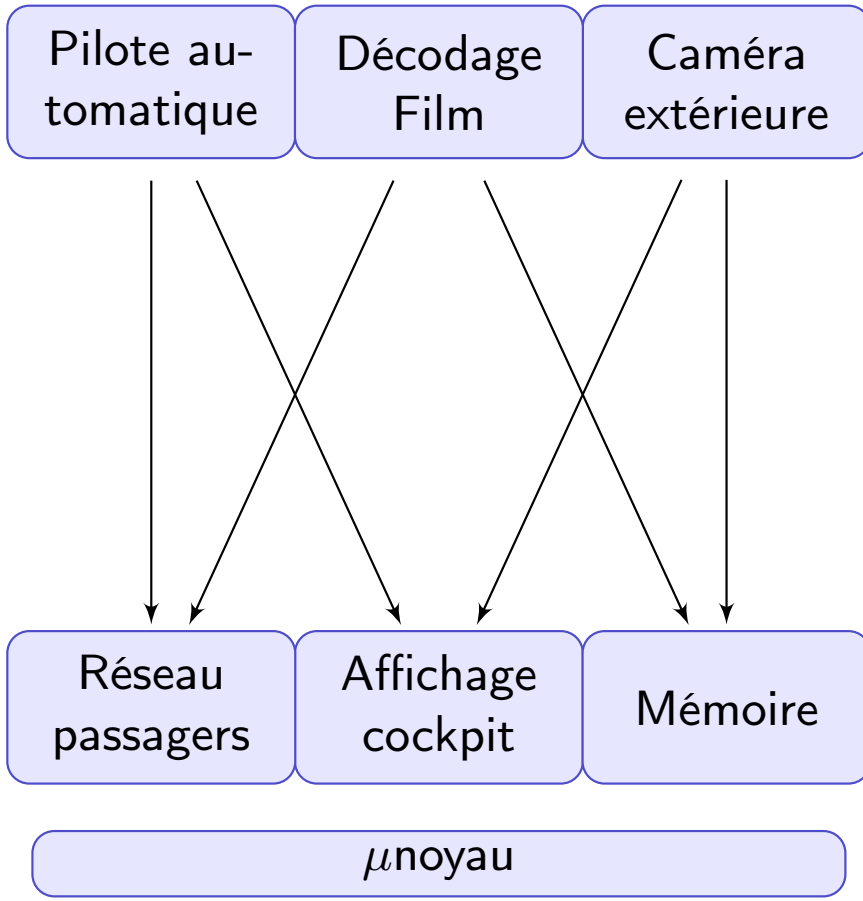


Mise en œuvre

Sécurisation d'un système (avionique): Gestion des privilèges

- Médiation complète
  - Vérification systématique des privilèges
- Principe du moindre privilège
  - Autoriser seulement le nécessaire
- Sûr par défaut
  - Tout ce qui n'est pas explicitement autorisé est interdit
- Économie de mécanismes
  - Nécessité d'un design simple, surtout pour la protection

Mise en œuvre

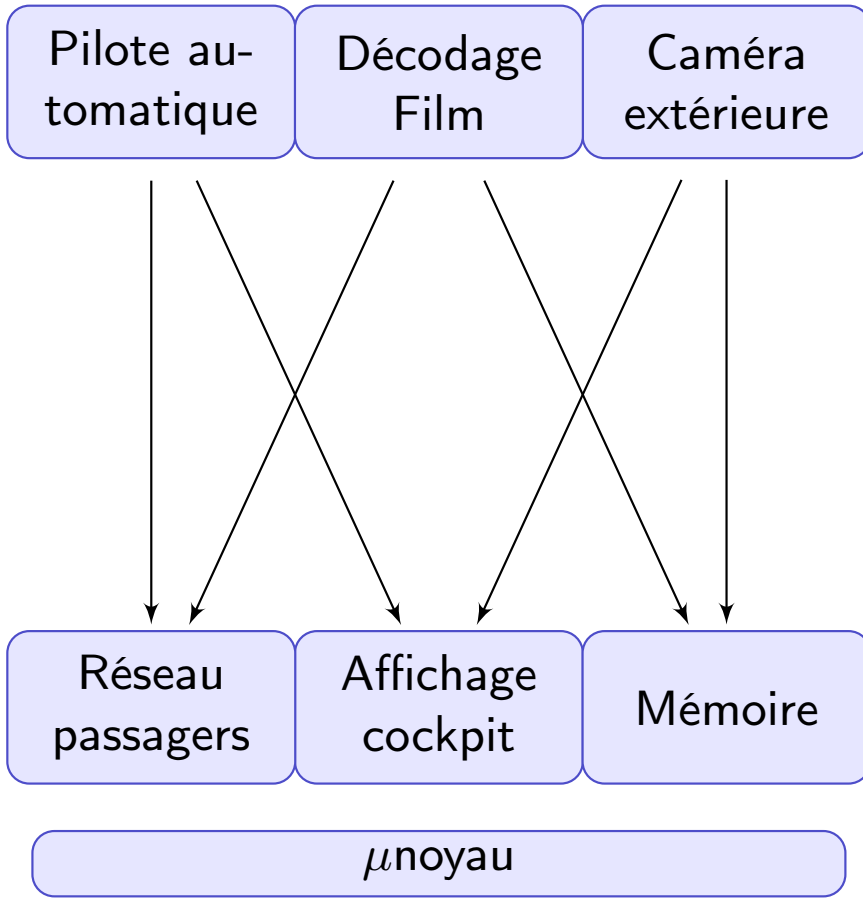


Sécurisation d'un système (avionique): Gestion des privilèges

- Médiation complète
  - Vérification systématique des privilèges
- Principe du moindre privilège
  - Autoriser seulement le nécessaire
- Sûr par défaut
  - Tout ce qui n'est pas explicitement autorisé est interdit
- Économie de mécanismes
  - Nécessité d'un design simple, surtout pour la protection

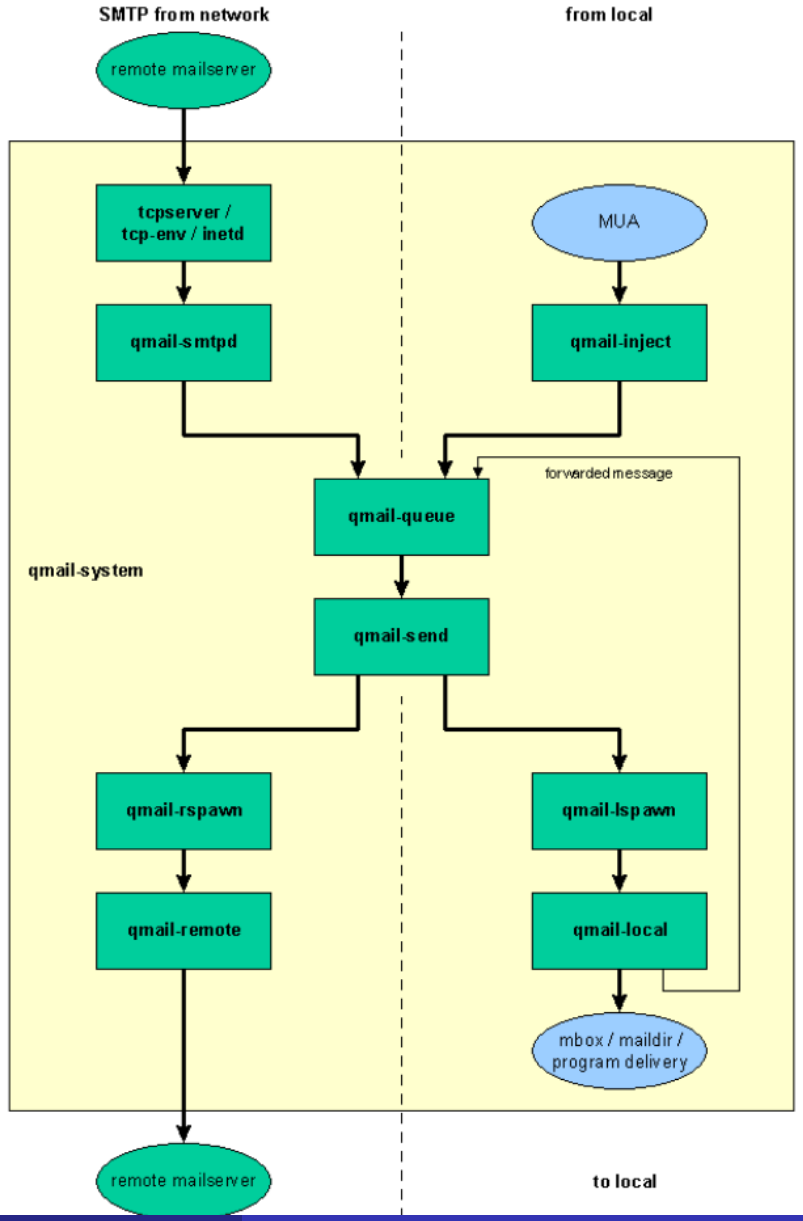
Mise en œuvre

- Contrôle d'accès
- Deux modèles principaux:
  - Les listes de contrôle d'accès (ACL)
  - Les capacités



Un exemple: qmail

- Implantation d'un serveur mail (SMTP)
- Différents programmes avec différentes permissions (accès réseau, disque)
  - Réception de mails d'un serveur distant
  - Réception de mails d'un utilisateur local
  - Envoi de mail à distance
  - Écriture de mail en local
  - ...



Sommaire

Contrôle d'accès: capacité et listes de contrôle d'accès

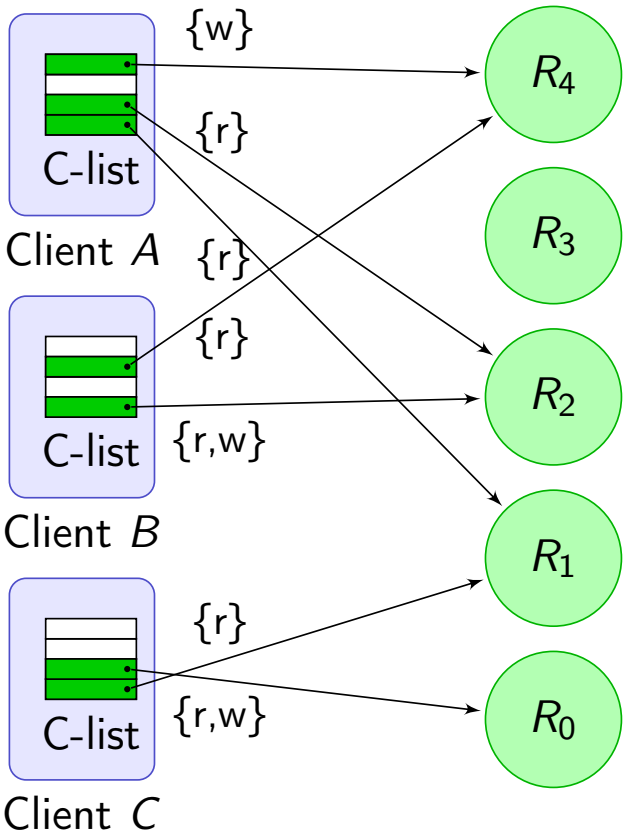
Tableau des permissions:

(Processus) client \ Ressources	R_0	R_1	R_2	R_3	R_4
A	{}	{r}	{r}	{}	{w}
B	{}	{}	{r,w}	{}	{r}
C	{r,w}	{r}	{}	{}	{}

Contrôle d'accès: capacité et listes de contrôle d'accès

Tableau des permissions:

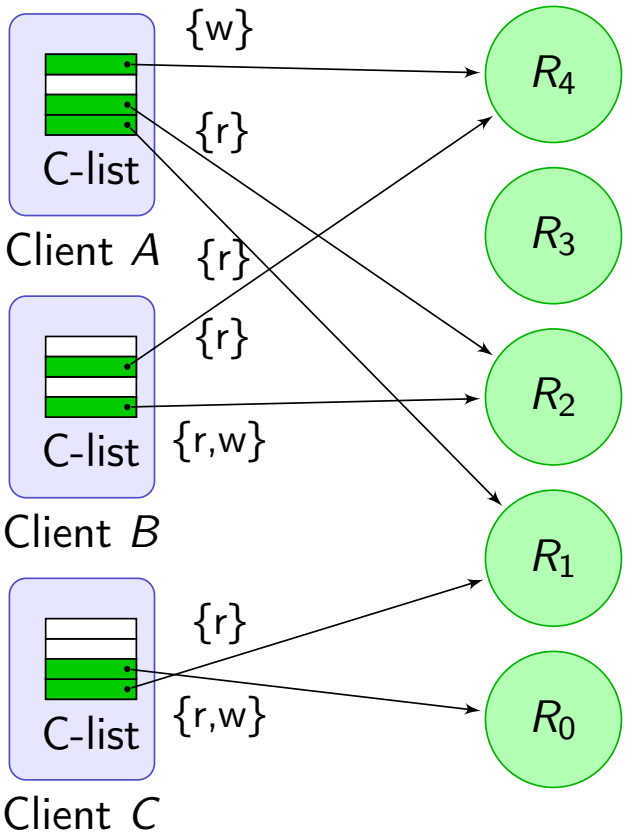
(Processus) client \ Ressources	R_0	R_1	R_2	R_3	R_4
A	{}	{r}	{r}	{}	{w}
B	{}	{}	{r,w}	{}	{r}
C	{r,w}	{r}	{}	{}	{}



Contrôle d'accès: capacité et listes de contrôle d'accès

Tableau des permissions:

Ressources / (Processus) client	R_0	R_1	R_2	R_3	R_4
A	{}	{r}	{r}	{}	{w}
B	{}	{}	{r,w}	{}	{r}
C	{r,w}	{r}	{}	{}	{}

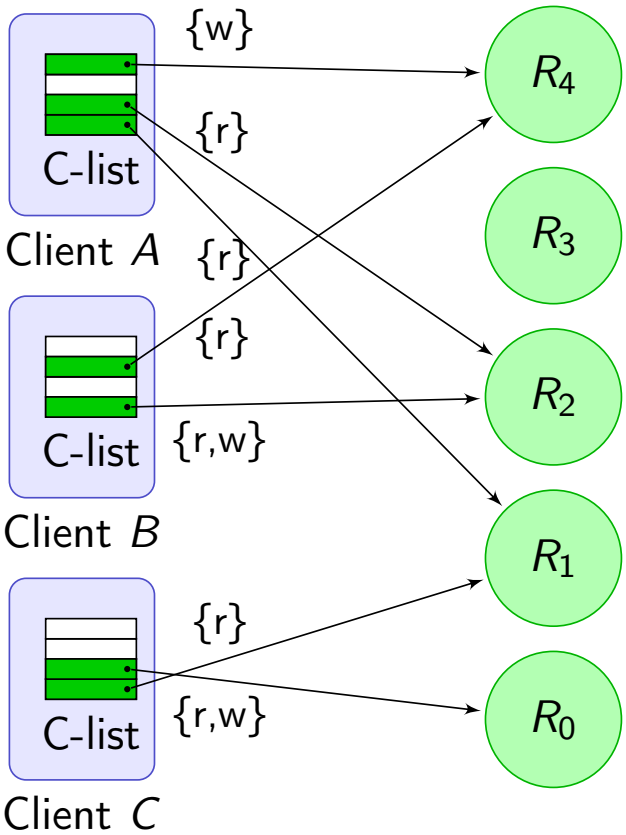


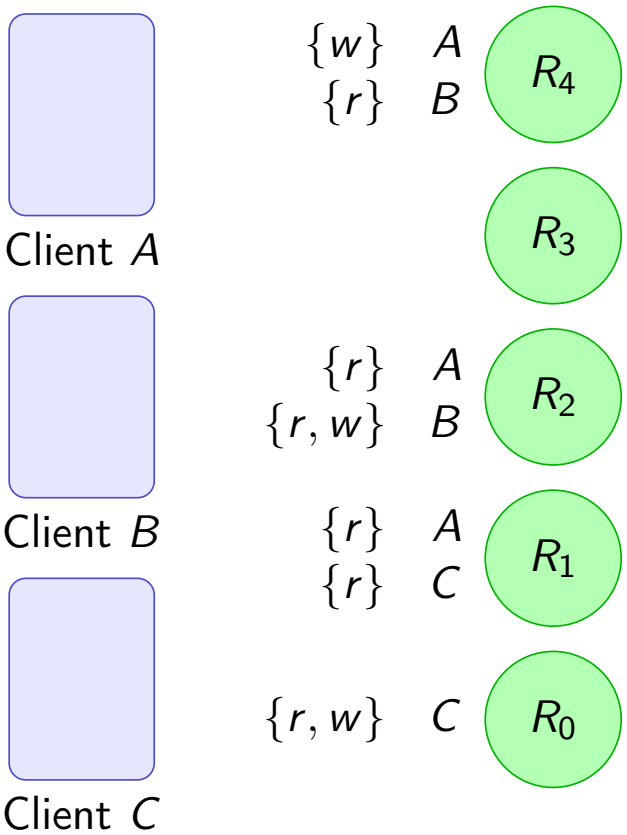
Capacités (Analogie: clé)

Contrôle d'accès: capacité et listes de contrôle d'accès

Tableau des permissions:

(Processus) client \ Ressources	R_0	R_1	R_2	R_3	R_4
A	{}	{r}	{r}	{}	{w}
B	{}	{}	{r,w}	{}	{r}
C	{r,w}	{r}	{}	{}	{}



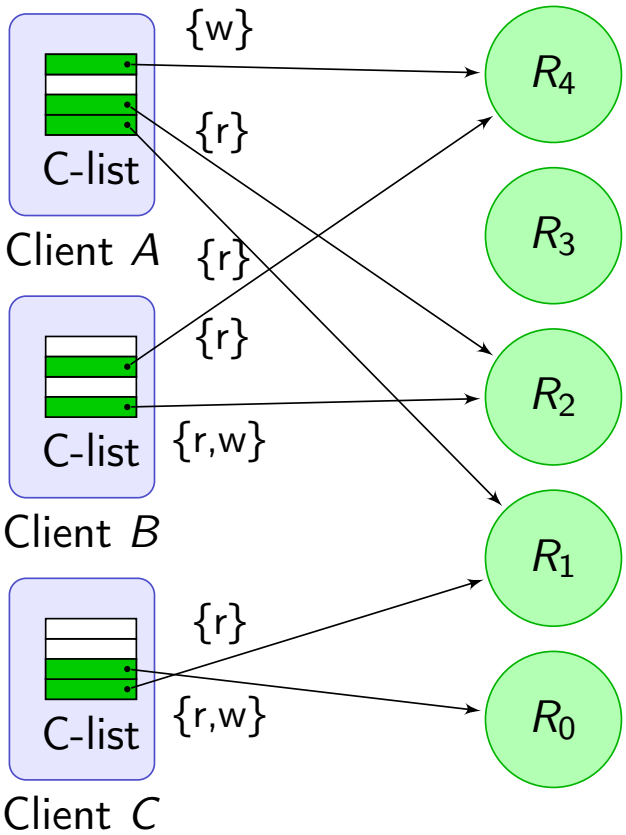


Capacités (Analogie: clé)

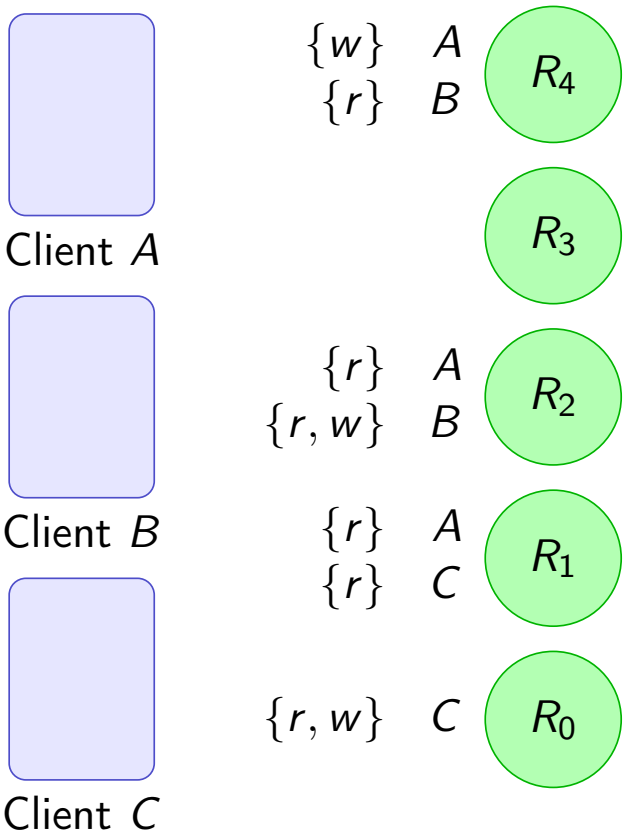
Contrôle d'accès: capacité et listes de contrôle d'accès

Tableau des permissions:

(Processus) client \ Ressources	R_0	R_1	R_2	R_3	R_4
A	{}	{r}	{r}	{}	{w}
B	{}	{}	{r,w}	{}	{r}
C	{r,w}	{r}	{}	{}	{}



Capacités (Analogie: clé)



Access-control list (Analogie: videur)

Un exemple de système de type ACL: les permissions de fichiers UNIX

- À chaque fichier est associé un utilisateur et un groupe
- À chaque fichier est associé les permissions (Read,Write,eXecute) pour l'utilisateur, le groupe, et les autres
- Un fichier central associe les utilisateurs au groupe.
- À chaque processus est associé un "user id" et un "group id"

➜ Ensemble, ce système détermine la liste des processus qui peuvent accéder à un fichier donné.

Un exemple de système de type capacité: les file descriptors UNIX

- Lorsqu'un fichier est ouvert (`open`), le noyau renvoie un numéro: le file descriptor
- Ce file descriptor est utilisé pour les opérations sur le fichier:
  - Envoi et reception de données (`read` et `write`)
  - Fermeture de l'accès au file descriptor (`close`)

➜ Le nom du fichier n'est plus utilisé

  - Le fichier peut même être supprimé/remplacé alors qu'un processus l'utilise

- Le file descriptor correspond à tous types de ressources:
  - Vrai fichiers
  - Pipes et communication inter-processus
  - Socket réseaux...

Comparaison: ACL et capacité

- Efficacité

Comparaison: ACL et capacité

- Efficacité
  - ACL: regarder dans une liste
    - Nécessite des mécanismes supplémentaires:
    - Un espace de nommage (nommer la ressource à laquelle on veut accéder)
    - Un mécanisme d'identification (identifier le client)
  - Capacité: vérification en temps constant
    - La capacité sert à nommer l'objet (c'est un pointeur vers la ressource)

Comparaison: ACL et capacité

- Efficacité
  - ACL: regarder dans une liste
    - Nécessite des mécanismes supplémentaires:
    - Un espace de nommage (nommer la ressource à laquelle on veut accéder)
    - Un mécanisme d'identification (identifier le client)
  - Capacité: vérification en temps constant
    - La capacité sert à nommer l'objet (c'est un pointeur vers la ressource)
  - ➜ Sous UNIX:
    - Les permissions ne sont utilisées que pour l'ouverture du fichier
    - Ensuite, toutes les opérations sont faites en utilisant le file descriptor

Comparaison: ACL et capacité

- Efficacité
  - ACL: regarder dans une liste
    - Nécessite des mécanismes supplémentaires:
    - Un espace de nommage (nommer la ressource à laquelle on veut accéder)
    - Un mécanisme d'identification (identifier le client)
  - Capacité: vérification en temps constant
    - La capacité sert à nommer l'objet (c'est un pointeur vers la ressource)
  - ➜ Sous UNIX:
    - Les permissions ne sont utilisées que pour l'ouverture du fichier
    - Ensuite, toutes les opérations sont faites en utilisant le file descriptor
- Responsabilité (savoir qui peut avoir accès a une ressource)

Comparaison: ACL et capacité

- Efficacité
  - ACL: regarder dans une liste
    - Nécessite des mécanismes supplémentaires:
    - Un espace de nommage (nommer la ressource à laquelle on veut accéder)
    - Un mécanisme d'identification (identifier le client)
  - Capacité: vérification en temps constant
    - La capacité sert à nommer l'objet (c'est un pointeur vers la ressource)
  - ➜ Sous UNIX:
    - Les permissions ne sont utilisées que pour l'ouverture du fichier
    - Ensuite, toutes les opérations sont faites en utilisant le file descriptor
- Responsabilité (savoir qui peut avoir accès a une ressource)
  - ACL: La liste donne l'information directement
  - Capability: Peut être plus difficile

Comparaison: ACL et capacité

- Efficacité
  - ACL: regarder dans une liste
    - Nécessite des mécanismes supplémentaires:
    - Un espace de nommage (nommer la ressource à laquelle on veut accéder)
    - Un mécanisme d'identification (identifier le client)
  - Capacité: vérification en temps constant
    - La capacité sert à nommer l'objet (c'est un pointeur vers la ressource)
  - ➜ Sous UNIX:
    - Les permissions ne sont utilisées que pour l'ouverture du fichier
    - Ensuite, toutes les opérations sont faites en utilisant le file descriptor
- Responsabilité (savoir qui peut avoir accès a une ressource)
  - ACL: La liste donne l'information directement
  - Capability: Peut être plus difficile
- Revocation (supprimer l'accès)

Comparaison: ACL et capacité

- Efficacité
  - ACL: regarder dans une liste
    - Nécessite des mécanismes supplémentaires:
    - Un espace de nommage (nommer la ressource à laquelle on veut accéder)
    - Un mécanisme d'identification (identifier le client)
  - Capacité: vérification en temps constant
    - La capacité sert à nommer l'objet (c'est un pointeur vers la ressource)
  - ➜ Sous UNIX:
    - Les permissions ne sont utilisées que pour l'ouverture du fichier
    - Ensuite, toutes les opérations sont faites en utilisant le file descriptor
- Responsabilité (savoir qui peut avoir accès a une ressource)
  - ACL: La liste donne l'information directement
  - Capability: Peut être plus difficile
- Revocation (supprimer l'accès)
  - ACL: supprimer de la liste
  - Capacité: plus difficile (nécessité de mettre en place un proxy...)

Sommaire

Cloud computing: définition (NIST Special Publication 800)

Définition (Cloud computing)

Le Cloud Computing (infonuagique) est un modèle pour permettre

- *l'accès à la demande*
- à l'aide d'*un réseau* (comme Internet)
- à un *groupe de ressources* (e.g. réseaux, serveurs, stockage, applications ou services)
- qui peut être rapidement provisionné et libéré
- avec un effort de gestion minimal ou une interaction minimale avec le fournisseur.

Ce modèle présentent 5 caractéristiques essentielles, 3 modèles de services et 4 modèles de déploiements.

Cloud computing: caractéristiques essentielles

On demand self service Le consommateur accède au service selon ses besoins sans interaction avec un humain.

Broad network access Les ressources doivent être accessibles à travers le réseau à l'aide de mécanismes "standards"

Ressource pooling Les ressources doivent être agrégées pour servir différents clients suivant un modèle "multi-tenant". Les ressources sont dynamiquement ré-assignées en fonction des demandes des utilisateurs.

Rapid elasticity Les ressources doivent être allouées et désallouées rapidement.

Mesured service La consommation des ressources doit être mesurée et contrôlée à l'aide d'un mécanisme transparent.

Cloud computing: Modèles de déploiement

Cloud privé L'infrastructure est provisionnée pour les besoins exclusifs d'une seule organisation qui comprend plusieurs entités (e.g. départements, filiales...). L'infrastructure peut être possédée par l'organisation elle-même, un tiers, ou une combinaison des deux.

Cloud communautaire infrastructure est provisionnée pour les besoins exclusifs d'une communauté de clients (exemple: boostaerospace)

Cloud public L'infrastructure est ouverte à tout le monde

Cloud hybride Combinaison des déploiements précédents; par exemple utilisation d'un cloud public pour absorber des pics de charge.

Cloud computing: Modèles de service

IaaS (Intrastructure as a service) fournit à l'utilisateur des resources de calcul fondamentales (bande passante, stockage, temps de calcul). Exemple: machines virtuelles

Cloud computing: Modèles de service

IaaS (Intrastructure as a service) fournit à l'utilisateur des resources de calcul fondamentales (bande passante, stockage, temps de calcul). Exemple: machines virtuelles

SaaS (Software as a service) fournit à l'utilisateur l'accès à une application, installée chez le fournisseur; accessible par un navigateur ou une interface dédiée. Exemple: webmail, hébergement de vidéos, hébergement de code, d'organisation de conférences, traitement des logs...

Cloud computing: Modèles de service

IaaS (Intrastructure as a service) fournit à l'utilisateur des resources de calcul fondamentales (bande passante, stockage, temps de calcul). Exemple: machines virtuelles

SaaS (Software as a service) fournit à l'utilisateur l'accès à une application, installée chez le fournisseur; accessible par un navigateur ou une interface dédiée. Exemple: webmail, hébergement de vidéos, hébergement de code, d'organisation de conférences, traitement des logs...

Paas (Platform as a service) fournit des services pour le développement d'applications SaaS. Exemple: serveurs webs, serveurs SQL pré-configurés, runtimes java

Cloud computing: Modèles de service

IaaS (Intrastructure as a service) fournit à l'utilisateur des resources de calcul fondamentales (bande passante, stockage, temps de calcul). Exemple: machines virtuelles

SaaS (Software as a service) fournit à l'utilisateur l'accès à une application, installée chez le fournisseur; accessible par un navigateur ou une interface dédiée. Exemple: webmail, hébergement de vidéos, hébergement de code, d'organisation de conférences, traitement des logs...

Paas (Platform as a service) fournit des services pour le développement d'applications SaaS. Exemple: serveurs webs, serveurs SQL pré-configurés, runtimes java

---

Le IaaS abstrait le matériel

le PaaS l'OS et des services de base,

le SaaS fournit des applications qui n'ont pas besoin d'être installées chez l'utilisateur.

Sommaire

Cloud computing: conséquences pour la sécurité des serveurs (1)

- Les systèmes d'exploitation prépondérants pour accéder à Internet sont de type UNIX (et Windows).
  - La sécurité est en concurrence avec la facilité d'usage
    - Windows: pour les ordinateurs personnels
    - UNIX: pour faciliter la collaboration sur mini-ordinateurs
  - Niveau de sécurité suffisant pour un serveur internet classique (mail, web, base de donnée...)
  - La sécurité du système repose pour beaucoup sur le code du programme serveur.

Cloud computing: conséquences pour la sécurité des serveurs (2)

- Mécanismes de sécurité insuffisant pour le cloud computing et autres applications de sécurité
  - Cloud computing = Permettre l'exécution de code arbitraire sur des machines partagées entre plusieurs clients
  - UNIX/Windows: Principe du moindre privilège insuffisant
    - L'administrateur a tous les droits (charger du code dans le noyau)
    - Les users ont beaucoup de droits (accès au réseau, aux programmes partagés).
  - Noyau monolithique qui offre une surface d'attaque importante
  - Nombreux modes de communication inter-processus (FIFO, sockets, système de fichier...)
  - Pas de moyen de comptabiliser précisément les ressources utilisées par chaque processus/user.

Cloud computing: conséquences pour la sécurité des serveurs (2)

- Mécanismes de sécurité insuffisant pour le cloud computing et autres applications de sécurité
  - Cloud computing = Permettre l'exécution de code arbitraire sur des machines partagées entre plusieurs clients
  - UNIX/Windows: Principe du moindre privilège insuffisant
    - L'administrateur a tous les droits (charger du code dans le noyau)
    - Les users ont beaucoup de droits (accès au réseau, aux programmes partagés).
  - Noyau monolithique qui offre une surface d'attaque importante
  - Nombreux modes de communication inter-processus (FIFO, sockets, système de fichier...)
  - Pas de moyen de comptabiliser précisément les ressources utilisées par chaque processus/user.

➜ Nécessité de créer des mécanismes d'encapsulation (sandbox) plus sécurisés que les processus traditionnels

Encapsulation (sandboxing)

Définition (Encapsulation (*sandbox*))

Une encapsulation (*sandbox*) permet l'exécution d'un ou plusieurs programme de manière à limiter les dommages liés à leurs défauts ou vulnérabilité.

- Notamment: limite la liste des privilèges accessible à ces programmes

Encapsulation (sandboxing)

Définition (Encapsulation (*sandbox*))

Une encapsulation (*sandbox*) permet l'exécution d'un ou plusieurs programme de manière à limiter les dommages liés à leurs défauts ou vulnérabilité.

- Notamment: limite la liste des privilèges accessible à ces programmes
- Note: Dans un système sécurisé, tous les programmes sont "encapsulés"!

Implantation de l'encapsulation

- On peut fournir une encapsulation au niveau d'un language de programmation:
  - Exemples: Java (Android), C#, Berkeley Packet Filter

Implantation de l'encapsulation

- On peut fournir une encapsulation au niveau d'un language de programmation:
  - Exemples: Java (Android), C#, Berkeley Packet Filter
- Pour pallier à leurs manques, les OS classiques fournissent de nouveaux mécanismes de sandboxing:
  - Seccomp (filtre d'appel système)
  - Containers/OS-level virtualization (Docker, OpenVZ)
    - Plusieurs systèmes Linux qui partagent le même noyau

Implantation de l'encapsulation

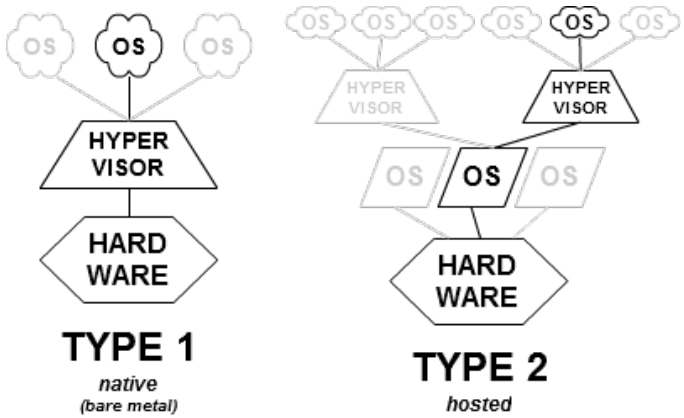
- On peut fournir une encapsulation au niveau d'un language de programmation:
  - Exemples: Java (Android), C#, Berkeley Packet Filter
- Pour pallier à leurs manques, les OS classiques fournissent de nouveaux mécanismes de sandboxing:
  - Seccomp (filtre d'appel système)
  - Containers/OS-level virtualization (Docker, OpenVZ)
    - Plusieurs systèmes Linux qui partagent le même noyau
- Lorsqu'un mécanisme de sandboxing fait croire à un programme que celui-ci s'exécute tout seul sur la machine, on parle de virtualisation.

Sécurité et virtualisation

Il y a beaucoup de manière d'exécuter des tâches de manière isolée (sandbox) et la virtualisation n'est que l'une d'entre elles.

Virtualisation: implantation

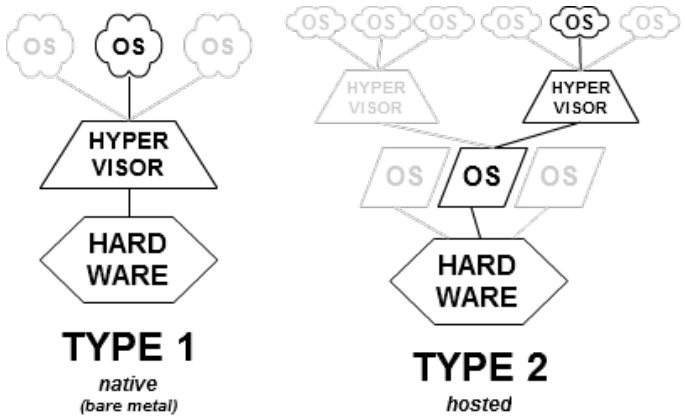
- Le programme permettant d'émuler une machine s'appelle un hyperviseur.



- Émulation: écrire un interpréteur du code machine de la machine virtualisée
  - Permet d'exécuter un système conçu pour une architecture différente

Virtualisation: implantation

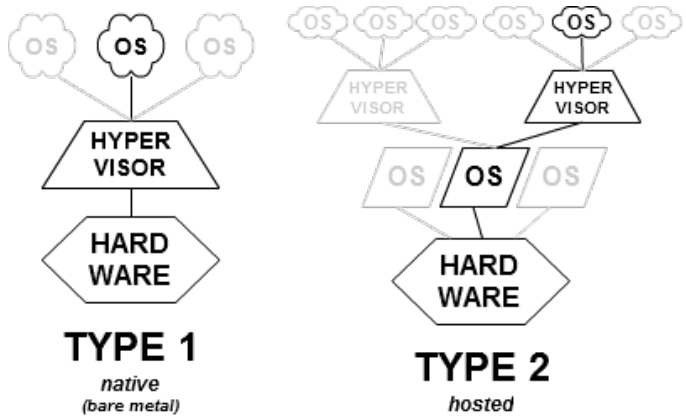
- Le programme permettant d'émuler une machine s'appelle un hyperviseur.



- Émulation: écrire un interpréteur du code machine de la machine virtualisée
  - Permet d'exécuter un système conçu pour une architecture différente
- Implantation efficace (Popek&Goldberg)
  - Séparation entre instructions systèmes et non privilégiées

Virtualisation: implantation

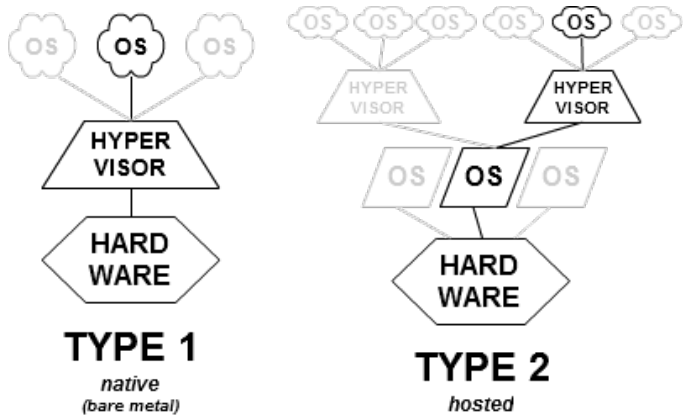
- Le programme permettant d'émuler une machine s'appelle un hyperviseur.



- Émulation: écrire un interpréteur du code machine de la machine virtualisée
  - Permet d'exécuter un système conçu pour une architecture différente
- Implantation efficace (Popek&Goldberg)
  - Séparation entre instructions systèmes et non privilégiées
  - L'utilisation d'instructions privilégiée par du code non-privilégié doit provoquer une interruption
  - Il faut simuler l'exécution réelle de l'instruction dans les routines de ces interruptions

Virtualisation: implantation

- Le programme permettant d'émuler une machine s'appelle un hyperviseur.



- Émulation: écrire un interpréteur du code machine de la machine virtualisée
  - Permet d'exécuter un système conçu pour une architecture différente
- Implantation efficace (Popek&Goldberg)
  - Séparation entre instructions systèmes et non privilégiées
  - L'utilisation d'instructions privilégiée par du code non-privilégié doit provoquer une interruption
  - Il faut simuler l'exécution réelle de l'instruction dans les routines de ces interruptions
  - Optimisations matérielles (e.g. 2ème niveau de mémoire virtuelle)
  - On peut modifier l'OS invité pour améliorer la communication avec l'hyperviseur (para-virtualisation)
  - ➜ Un hyperviseur de type 1 est une sorte d'OS où les processus sont des machines virtuelles.

Virtualisation: applications

- Rétro-compatibilité; exécuter des applications conçues pour un autre matériel ou OS
  - Les mainframes IBM d'aujourd'hui peuvent exécuter les programmes écris dans les années 60.

Virtualisation: applications

- Rétro-compatibilité; exécuter des applications conçues pour un autre matériel ou OS
  - Les mainframes IBM d'aujourd'hui peuvent exécuter les programmes écris dans les années 60.
- Multiplexer les ressources d'une machine entre différents OS (e.g. cloud IaaS)
  - Permet de faire des *snapshots*, des migrations automatique

Virtualisation: applications

- Rétro-compatibilité; exécuter des applications conçues pour un autre matériel ou OS
  - Les mainframes IBM d'aujourd'hui peuvent exécuter les programmes écris dans les années 60.
- Multiplexer les ressources d'une machine entre différents OS (e.g. cloud IaaS)
  - Permet de faire des *snapshots*, des migrations automatique
- Gérer des applications qu'un OS classique ne sait pas bien gérer
  - Tâches temps-réel dur
  - Tâches avec un haut niveau de sûreté et sécurité
- Mise en place d'architectures de sécurité (separation kernel)
  - Les hyperviseurs sont (souvent!) plus petits et simples qu'un OS
    - ➜ moins de risque de compromission
  - Contrôle des liens de communications entre les machines virtuelles

Sommaire

Résumé du cours

- Les grands principes (Saltzer et Schroeder) permettant de réaliser une architecture sécurisée.
- Les notions de contrôle d'accès, capacité et ACL.
- Le cloud : la location d'infrastructure informatique
- Le sandboxing: le moindre privilège appliqué aux OS mainstreams
- La virtualisation de type 1: un OS sous l'OS

Le sandboxing cycle https://xkcd.com/2044/