Une approche de virtualisation assistée par le matériel pour protéger l'espace noyau d'actions malveillantes

Eric Lacombe[1]

Ph.D Supervisors: Yves Deswarte and Vincent Nicomette

[1]eric.lacombe@security-labs.org
LAAS - CNRS
Toulouse (France)

Contexte

- Complexité du matériel/logiciel ⇒ Failles de sécurité
- Exploitation de failles ⇒ Attaquants accomplissent leur but malveillant

- Kernel (noyau) = Coeur de l'OS
  - Point de passage obligatoire de toute application
  - Gère les ressources matérielles

- Faille de sécurité du noyau ⇒ Critique
- En particulier dans les pilotes de périphériques (qualité inférieure au reste du noyau)

- Kernel : Cible privilégiée pour les attaquants

Problématique

Deux problèmes sur lesquels nous souhaitons travailler :

1. Comment empêcher un maliciel d'entrer dans le noyau ?
2. Comment protéger le système quand le noyau est compromis ?

Plan

Hypothèses

- Focus sur les attaques visant l'intégrité du noyau
- Perte d'intégrité du noyau
  → Modification inappropriée de :
  - Structure du noyau = code du noyau
  - ou État du noyau = données utilisée par le noyau (ex : données dans la mémoire, valeurs des registres du processeur)

  → Aboutit à :
  - Rien (injection/modification de données/code non utilisés)
  - ou Crash du système
  - ou Exécution d'une action malveillante
- Les failles du matériel ne sont pas prises en compte

Plan

Vecteur d'accès de type CPU

1° catégorie : fonctionnalités système

- Fonctionnalités logicielles : chargeur de module noyau, `/dev/kmem`, etc.
- Fonctionnalités matérielles : System Management Mode des CPU x86

2° catégorie : failles de sécurité du système

- Débordement de tampons
- Chaînes de format
- Emploi de données incorrectes
- Emploi de données périmées
- etc.

Vecteur d'accès de type DMA

DMA = Direct Memory Access sans l'implication directe de la CPU

- Périphérique malveillant connecté à un bus d'E/S, maître du DMA (ex : Firewire)
  → Solution générique : filtrage des accès DMA par une I/O MMU
- Pilotes malveillants qui commandent de mauvaises transactions DMA

Note : l'utilisation d'un vecteur d'accès peut en ouvrir d'autres
→ Exemple :

1. Désactivation/Altération de l'I/O MMU (AV de type CPU)
2. lecture/écriture DMA

Plan

Classe 1 - Modification inappropriée des chemins d'exécution du mode noyau (Code)

Provoque la modification du comportement du noyau, par l'altération de son code :

- **(Classe 1.1)** Ajout d'une région de code noyau malveillant et atteignable
- **(Classe 1.2)** Écrasement d'une région de code noyau existante avec du code malveillant
- **(Classe 1.3)** Injection de code malveillant et atteignable dans une région de données noyau
- **(Classe 1.4)** Injection de code malveillant et atteignable dans une région n'appartenant pas au noyau

Classe 2 - Modification inappropriée des variables du noyau

Provoque la modification du comportement du noyau, par l'altération des données qu'il utilise :

- **(Classe 2.1)** Altération des variables d'état (impacte le flux d'exécution) :
  - Donnée de contrôle dans la pile
  - Donnée employée dans une condition de branchement
  - Attribut d'une entrée de table de pages
  - Valeur du registre `idtr`
  - etc.
- **(Classe 2.2)** Altération des variables auxiliaires :
  - Description d'une erreur affichée par `printk()`
  - ...

Plan

Un noyau rempli de contraintes

Kernel-Constrained Object (KCO)

- $\in$ Variables du noyau
- Toujours dans un état fixe ou prédictible par spécification

Par exemple :

- `IDT` (Interrupt Descriptor Table) et le registre `idtr` sont des **KCO**
- L'agencement de l'espace d'adressage du noyau est composé de plusieurs **KCO**

Comment protéger les objets contraints par le noyau

- Doit être fait à un niveau de privilège matériel supérieur à celui du noyau
  $\longrightarrow$ Hyperviseur assisté par le matériel
- Hyperviseur très léger
  $\longrightarrow$ Vérification de sa correction plus facile

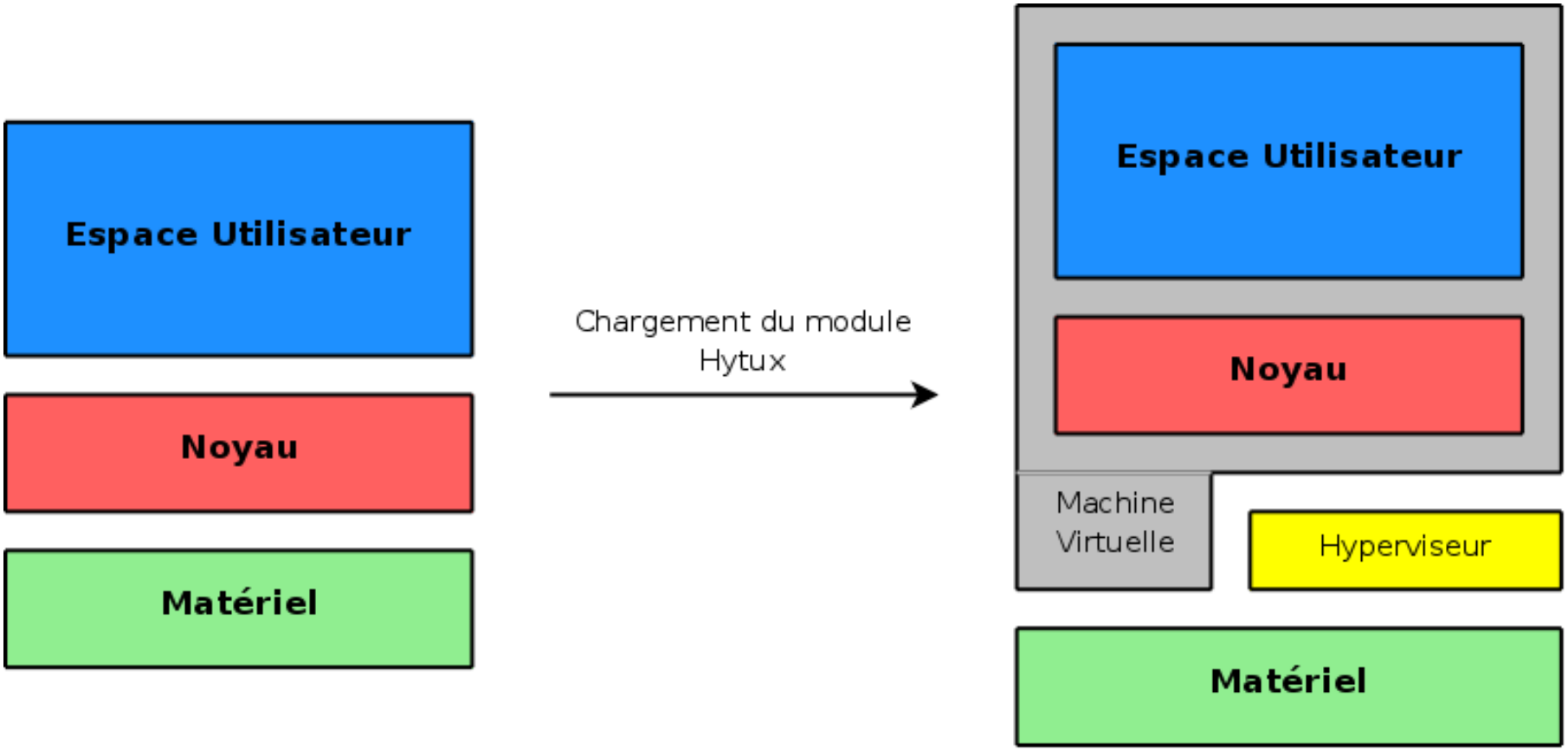
Avantages :

$\longrightarrow$ Capacité unique pour restreindre/contrôler le mode noyau

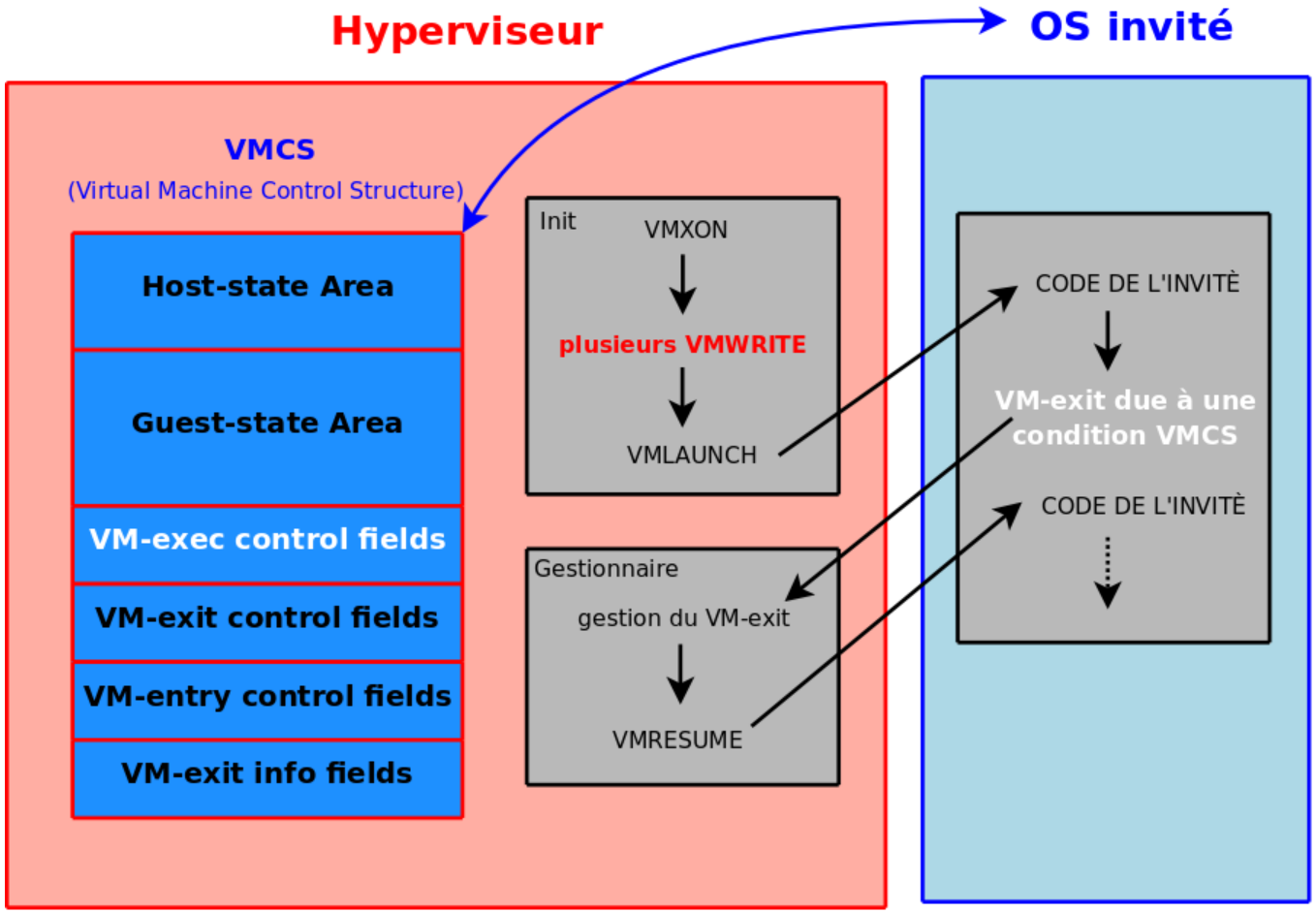
$\longrightarrow$ Couvre de nombreuses actions malveillantes en empêchant la violation de contraintes.

Plan

Aperçu de Hytux



Une rapide introduction à Intel VT-x



Plan

Empêcher idtr d'être modifié

- `idtr` = registre du processeur, contient l'adresse de l'IDT
- Besoin de le charger *seulement* à l'initialisation du système
- Mais peut être modifié par des maliciels noyau

- Hytux empêche les modifications de l'`idtr` :
  - Intercepte les accès en écriture (grâce aux contrôles de VM-exit)
  - Émule cet accès → ne modifie pas l'`idtr`, met à jour seulement le compteur d'instructions de l'invité

- D'autres registres sont protégés de façon similaire : `gdtr`, `cr0` and `cr4`, MSR, etc.

Plan

Préserver l'agencement de l'espace d'adressage du noyau

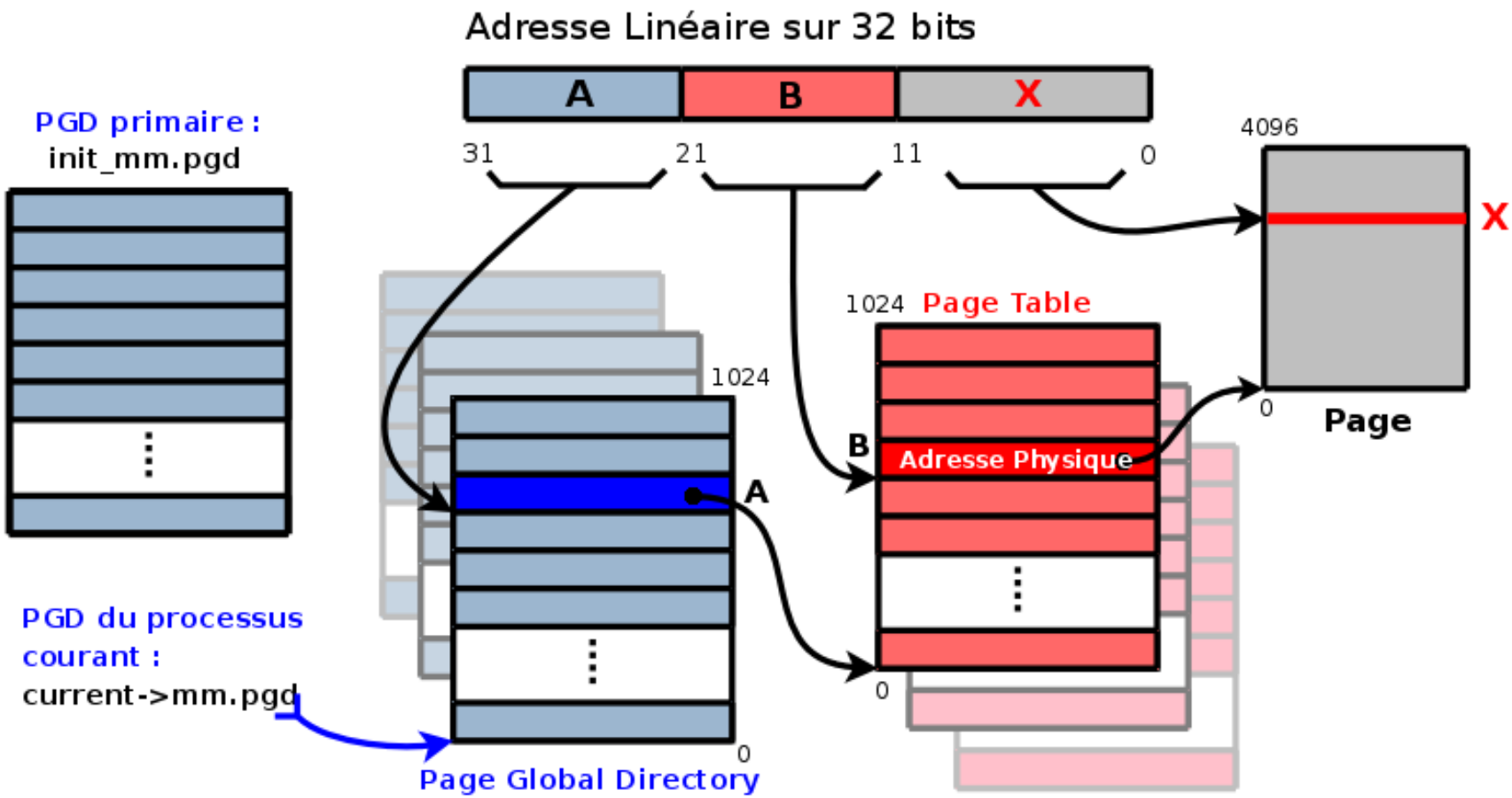
L'idée

Préserver les contraintes sur l'agencement de l'espace d'adressage du noyau

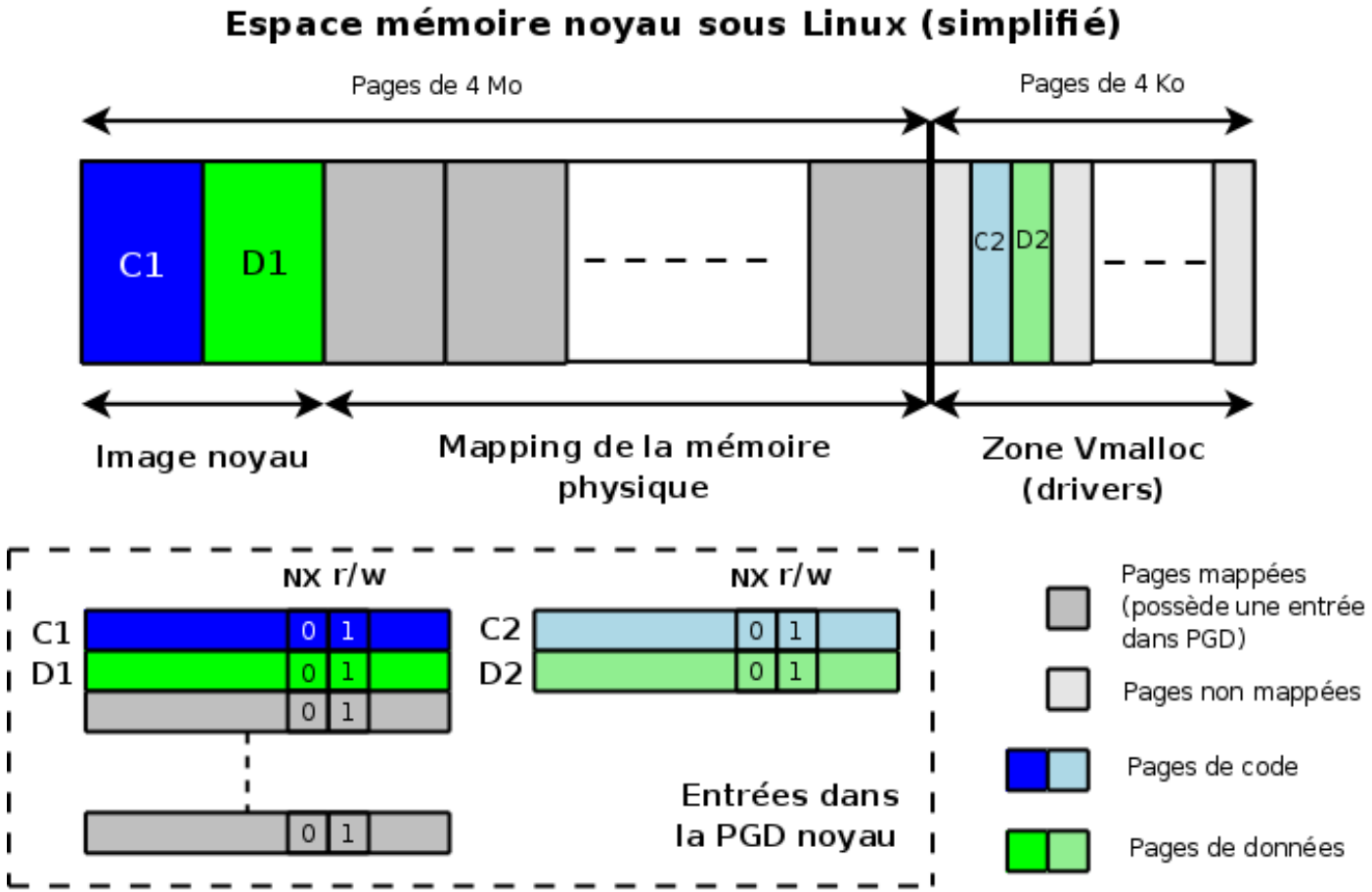
$\Rightarrow$ Protège le système au moins contre les classes 1.2 et 1.3

- région de code : NX=0, R/W=0
- région de données : NX=1, R/W=1
- région de données en lecture seule : NX=1, R/W=0

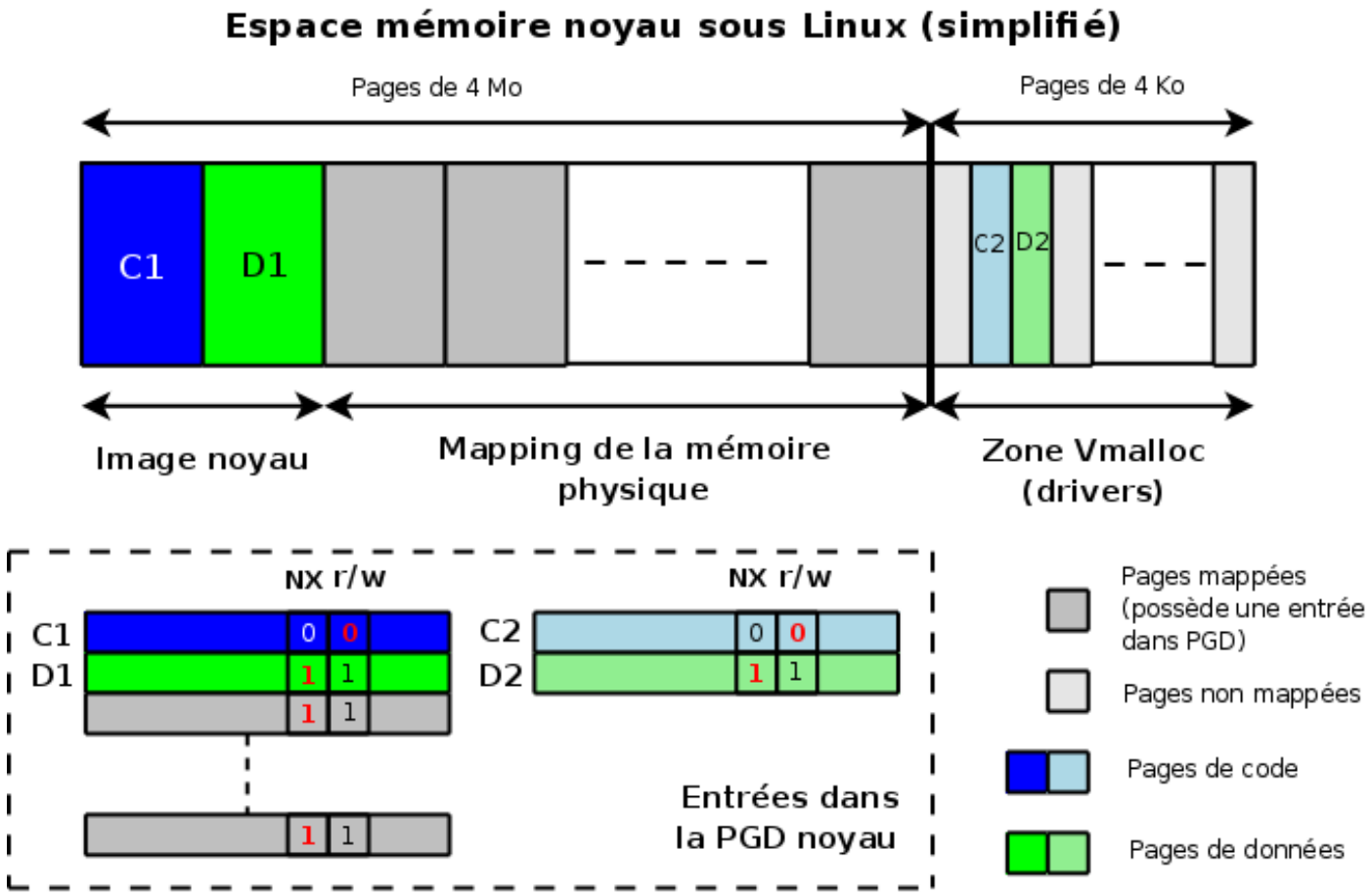
Rappel sur le mécanisme de pagination ...



... et sur l'agencement de l'espace d'adressage du noyau



Une première (mauvaise) solution



Une première (mauvaise) solution (suite)

Pourquoi est elle incorrecte ?

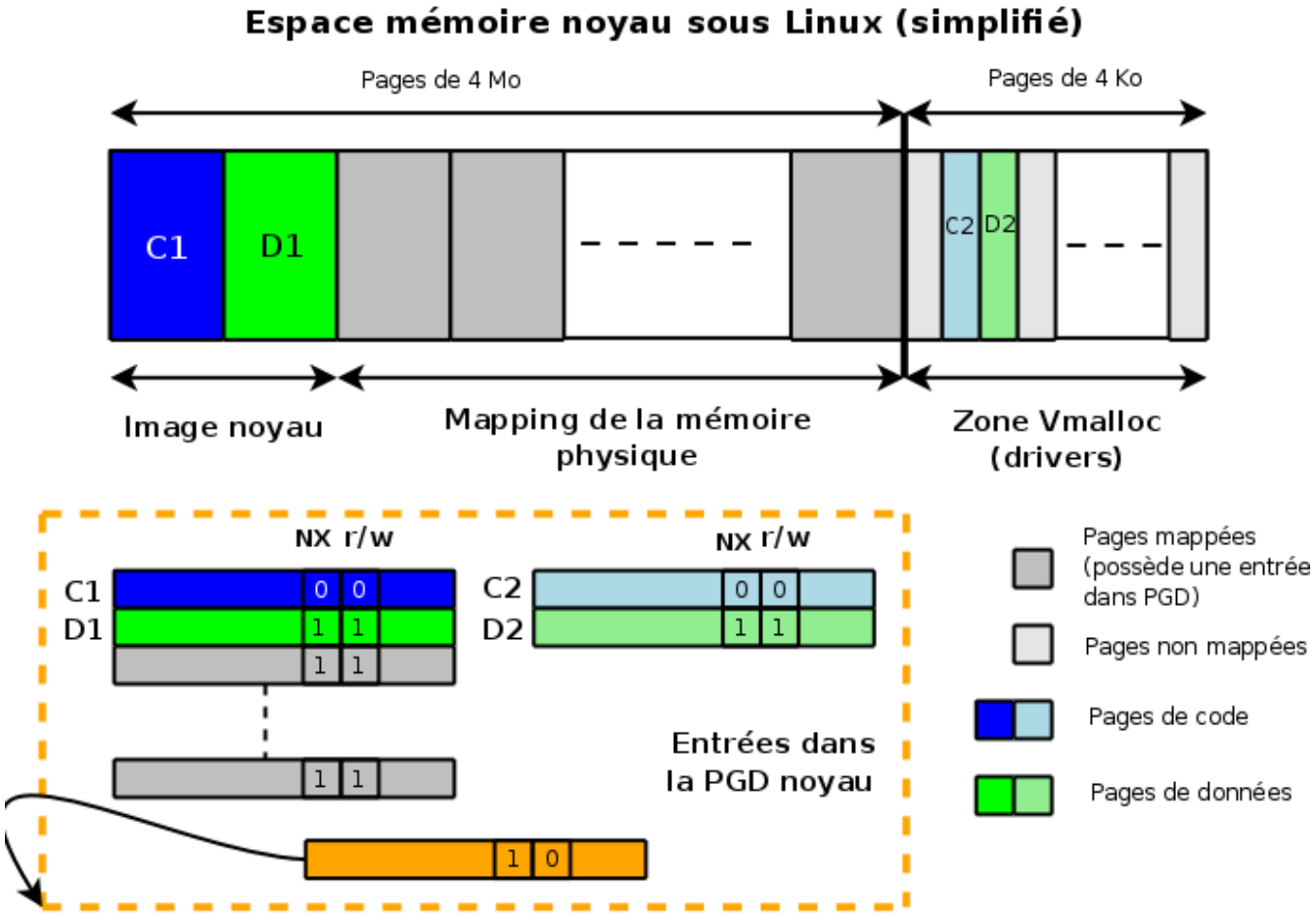
- Les tables de pages du noyau ont besoin d'être modifiées durant l'exécution du système
  $\longrightarrow$ Chargement dynamique de pilotes de périphériques (région VMALLOC)
- Doit rester dans une région mémoire R/W
- Mais l'attaquant peut alors modifier les attributs des tables de pages et casser les contraintes

Une solution correcte

Concept

- Mettre à 0 l'attribut R/W des pages qui contiennent les tables de pages noyau
- Quand le noyau souhaite modifier les tables de pages noyau :
  1. Une faute de page va se produire
  2. Une VM-exit est alors déclenchée avant la faute
  3. Hytux prend la main sur le processeur
  4. Il vérifie que les modifications souhaitées ne modifient pas les contraintes (de l'agencement qu'il sait être sûr)
- Quand le noyau souhaite charger le registre `cr3` (référence des tables de pages) :
  1. Une VM-exit est déclenchée
  2. Hytux vérifie que les dernières entrées des tables de pages (espace noyau) sont correctes

Une solution correcte (suite)



Une solution correcte (suite)

Notes

- L'agencement de l'espace noyau peut être modifié dans la région VMALLOC (insertion drivers/modules)
  $\Rightarrow$ `vmalloc()/vfree()` doivent informer Hytux du changement (via `vmcall` qui déclenche une VM-exit)

- Hytux vérifie le point d'appel (via `rip` lors de la VM-exit)

Contributions

- Proposition d'une classification sur les actions malveillantes ciblant le noyau
- Nouveau concept pour la sécurité noyau : protection des objets contraints par le noyau
  $\longrightarrow$ Peut être appliquée à toutes les classes sauf la classe 1.1
- Protection de l'agencement de l'espace d'adressage du noyau
  $\longrightarrow$ Couvre au moins les classes 1.2 and 1.3
- Première utilisation des technologies matérielles de virtualisation pour protéger l'OS :
  - Capacité unique pour restreindre/contrôler le mode noyau
  - En résulte la protection du noyau vis-à-vis de nombreux maliciels

Limites

- Tous les objets du système ne peuvent être assimilés à des KCO
  $\longrightarrow$ Les contraintes n'existent pas ou ne sont pas vérifiables
  $\Rightarrow$ Les actions malveillantes peuvent profiter de ces objets

- Les actions malveillantes seraient-elles toujours possibles dans un monde idéal, rempli de KCO ?

Merci de votre attention ...

... Des questions ?