La sécurité dans les couches basses du logiciel : attaques et contre-mesures

Eric Alata, Yves Deswarte, Mohamed Kaâniche, Vincent Nicomette

INSA Toulouse - LAAS/CNRS

PRé-GDR Sécurité - Sécurité des Systèmes logiciels
Paris - 31 Mai 2017

Pourquoi s'intéresser aux couches basses du logiciel ?

Méthodes de défense

- Méthodes formelles pour la spécification, le développement et la vérification (ISO/IEC 15408 Common Criteria notamment)
- Mécanismes et outils de protection : pare-feux, réseaux privés, détection d'intrusion, authentification, contrôle d'accès, etc.
- La recherche de vulnérabilités et la proposition de contre-mesures $\Rightarrow$ en particulier dans les couches basses du logiciel

Limites des méthodes formelles dans les couches basses

- Difficulté d'exprimer des propriétés de haut niveau (confidentialité, intégrité, disponibilité) avec des mécanismes subtiles (modes d'exécution, caches, interruptions, etc)
- Difficulté de modéliser un noyau complet : les certifications EAL6 ou 7 ne concernent que des noyaux minimalistes
- Le processus de certification est long ; pendant ce temps, le développement continue ...

1 Sécurité des noyaux d'OS : *rootkits* noyau et contre-mesures

2 Les attaques par entrées-sorties

3 Architecture mixte logicielle/matérielle de sécurité

Rookits noyau et contre-mesures
(2006-2009)

Thèse de d'Eric Lacombe
Collaboration EADS/IW

Problématique (1/2)

Vulnérabilité des systèmes d'exploitation

- Les systèmes d'exploitation sont en majorité des COTS (*Components Off The Shelf*)
- Ces composants génériques sont très complexes : susceptibles de contenir des vulnérabilités
- Système d'exploitation : espace utilisateur et espace noyau $\Rightarrow$ Corruption de l'espace noyau particulièrement rentable

Objectif

Renforcer la sécurité des noyaux de système d'exploitation (travaux sur noyau Linux, architecture x86)

Problématique (2/2)

Le logiciel est contournable

- Majorité des mécanismes permettant d'empêcher la corruption d'un noyau : mécanismes s'exécutant dans le même niveau de privilège que le noyau lui-même
- Mécanismes inefficaces vis-à-vis de logiciels s'exécutant au niveau de privilège du noyau (*ring 0 sur Intel*) : en particulier, les rootkits noyau

Objectif

Utilisation de mécanismes matériels que nous considérons "inviolables et incontournables"

Les rootkits

Définition

Ensemble de modifications permettant à un attaquant de maintenir dans le temps un contrôle frauduleux sur un système informatique.

Evolution des rootkits

- D'abord modification des commandes du système
- Puis modification de librairies partagées
- Enfin au sein de l'entité partagée par tous les éléments du système : le noyau

Architecture fonctionnelle d'un rootkit

Quatre éléments fonctionnnels

- Un injecteur : installer le rootkit
- Un module de protection : protéger le rootkit et les activités de l'attaquant
- Une backdoor : communiquer avec le rootkit
- Des services : effectuer des opérations malveillantes

Comment protéger les objets contraints par le noyau

Kernel-Constrained Object (KCO)

- Objet contraint par le noyau
- Toujours dans un état fixe ou prédictible **par spécification**

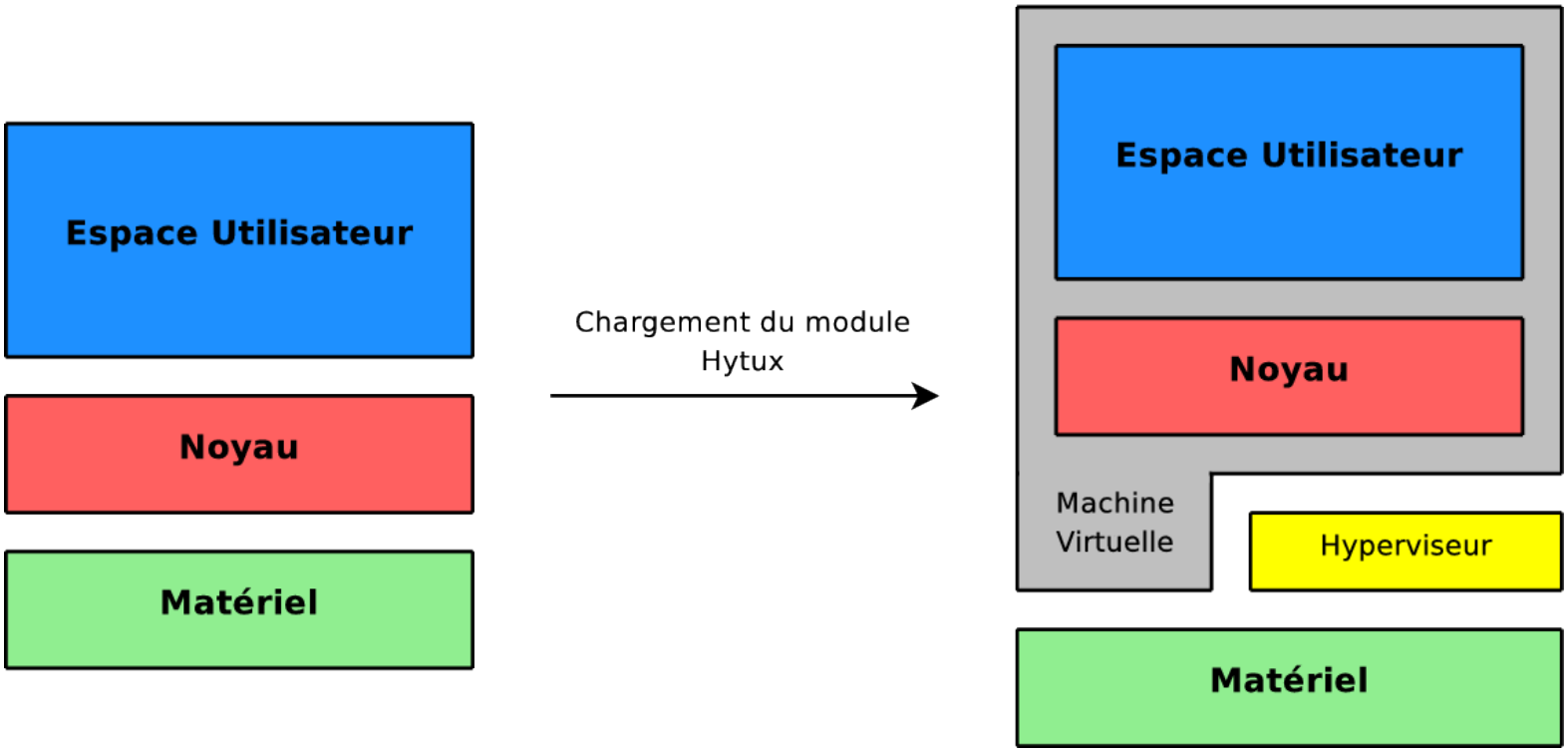
Exemples

- `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

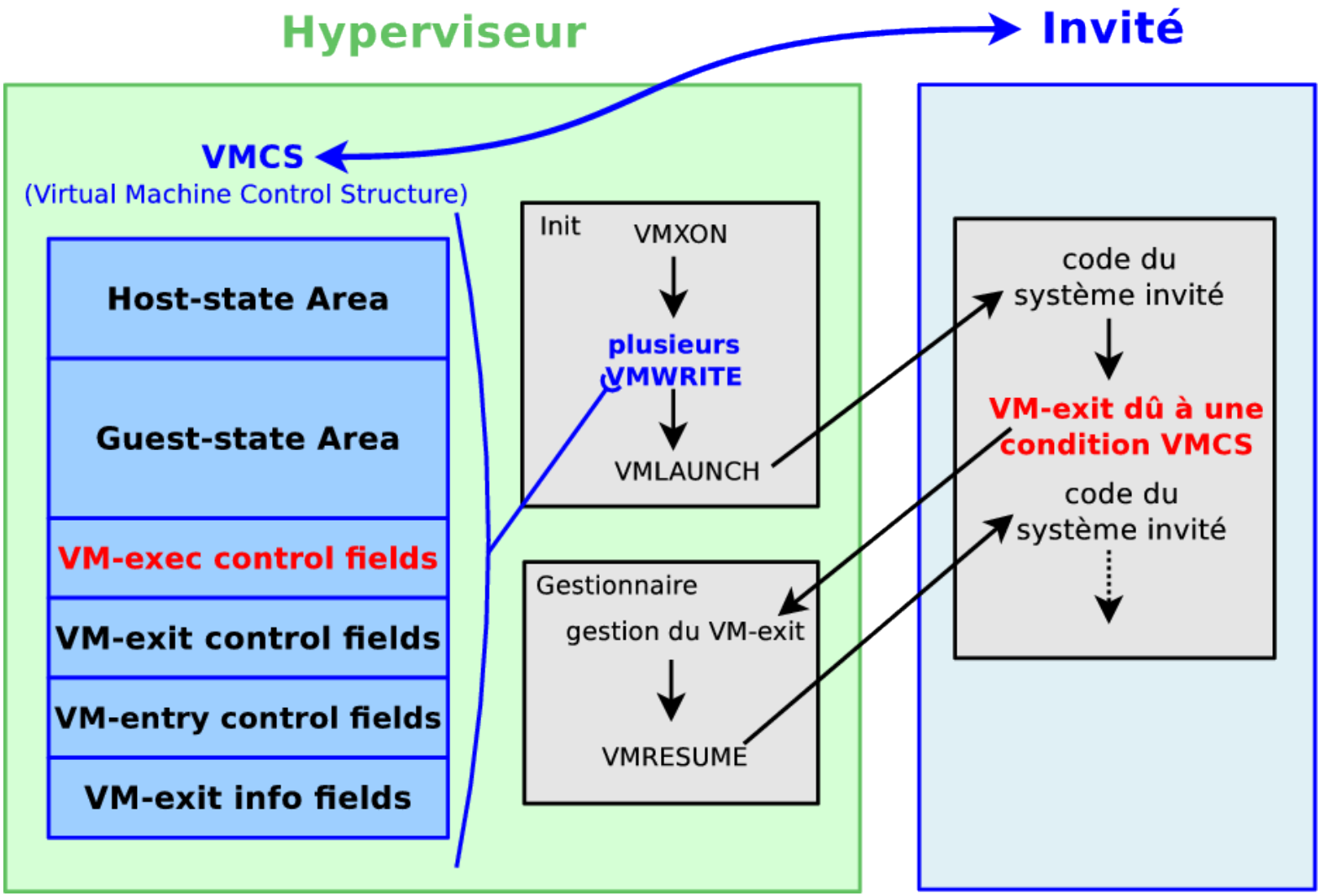
Protection par un hyperviseur assisté par le matériel

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

Aperçu de Hytux



Aperçu d'Intel-VTx



Exemple : empêcher idtr d'être modifié

L'objet contraint

- `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

La protection

- Hytux empêche les modifications de l'`idtr` :
  - Intercepte les accès en écriture (grâce aux contrôles de VM-exit)
  - Emule 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.

1 Sécurité des noyaux d'OS : *rootkits* noyau et contre-mesures

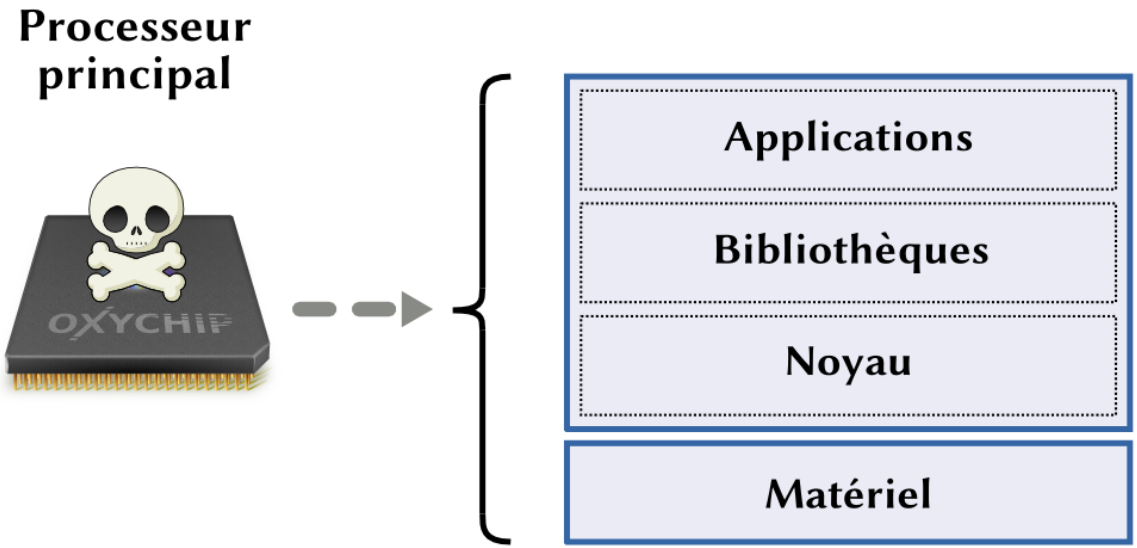
2 Les attaques par entrées-sorties

3 Architecture mixte logicielle/matérielle de sécurité

Attaques par entrées-sorties
(2009-2012)

Thèse de Fernand Lone Sang
Etude sur fonds propres

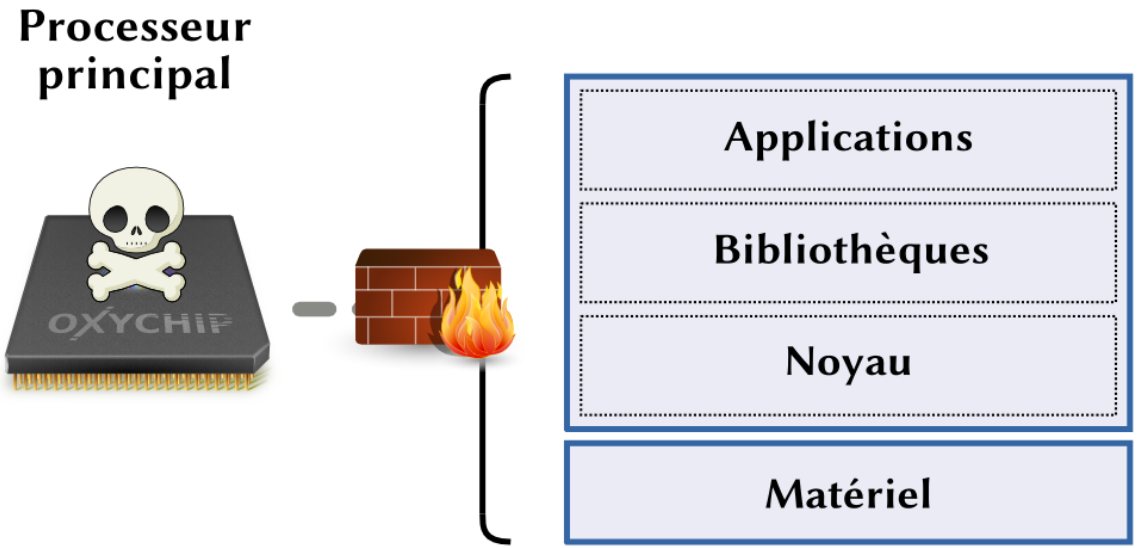
Contexte et problématique (1/3)



Logiciel malveillant s'exécutant sur le processeur principal

- Exploitation d'une vulnérabilité logicielle : débordement de tampon, chaîne de format, ...
- Abus de fonctionnalités légitimes du système : chargeur de modules noyau, périphériques virtuels, ...

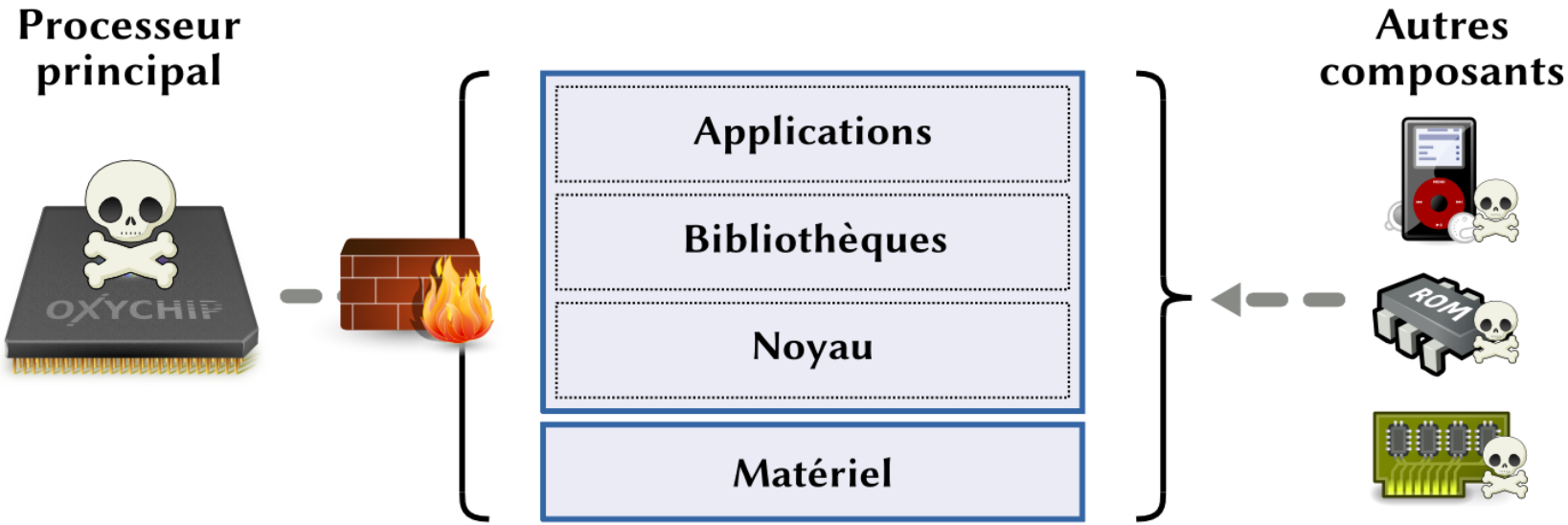
Contexte et problématique (1/3)



Contre-mesures à ces logiciels malveillants

- Axées sur les vulnérabilités : les détecter et les éliminer, empêcher leur exploitation, ...
- Axées sur les logiciels malveillants : les détecter et les supprimer, empêcher leur exécution, ...

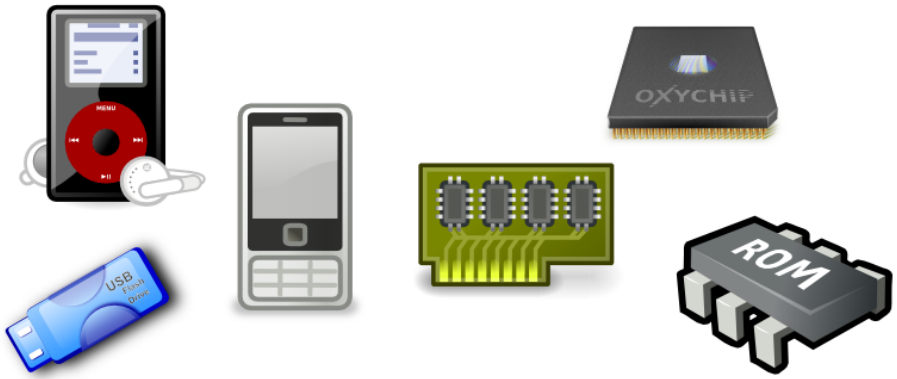
Contexte et problématique (1/3)



Logiciel malveillant impliquant les composants matériels

- Modification du comportement
- Altération de la configuration matérielle

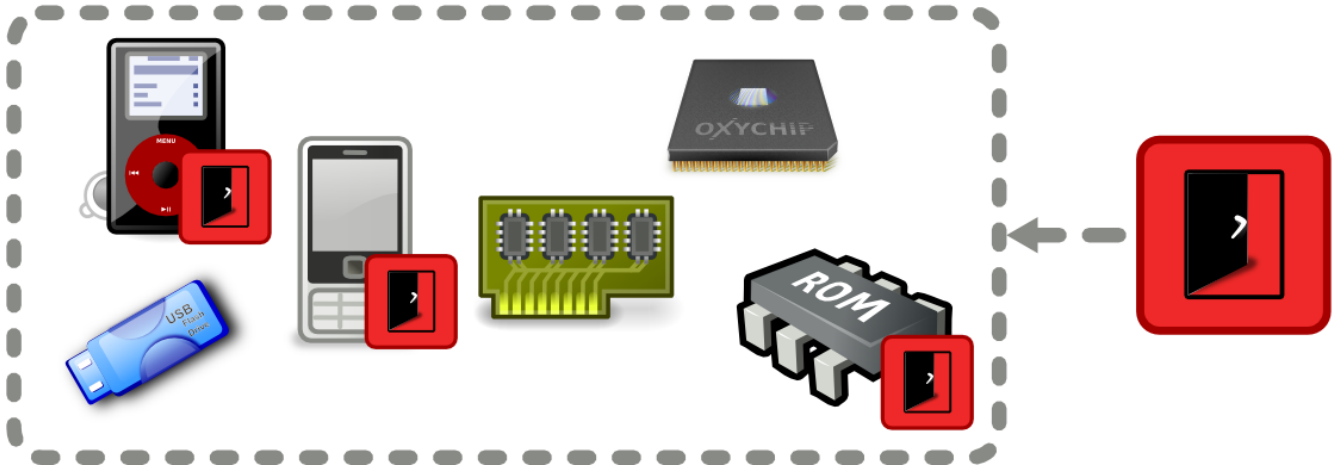
Contexte et problématique (2/3)



Hypothèses d'attaques considérées

- Composants vulnérables qui peuvent être modifiés
- Composants conçus pour contenir des fonctions cachées

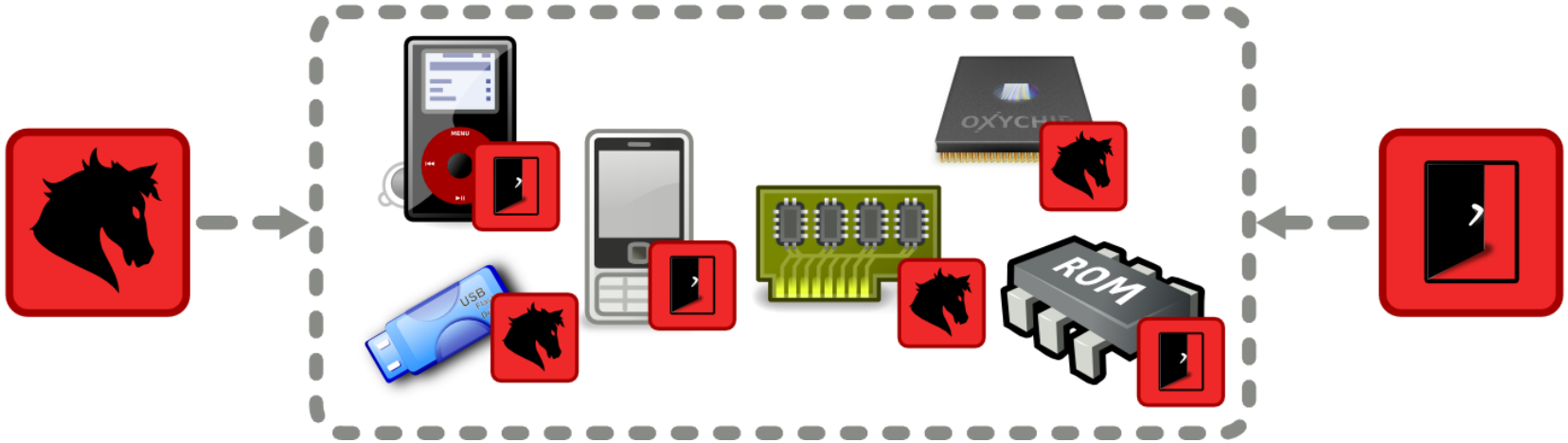
Contexte et problématique (2/3)



Hypothèses d'attaques considérées

- Composants vulnérables qui peuvent être modifiés
- Composants conçus pour contenir des fonctions cachées

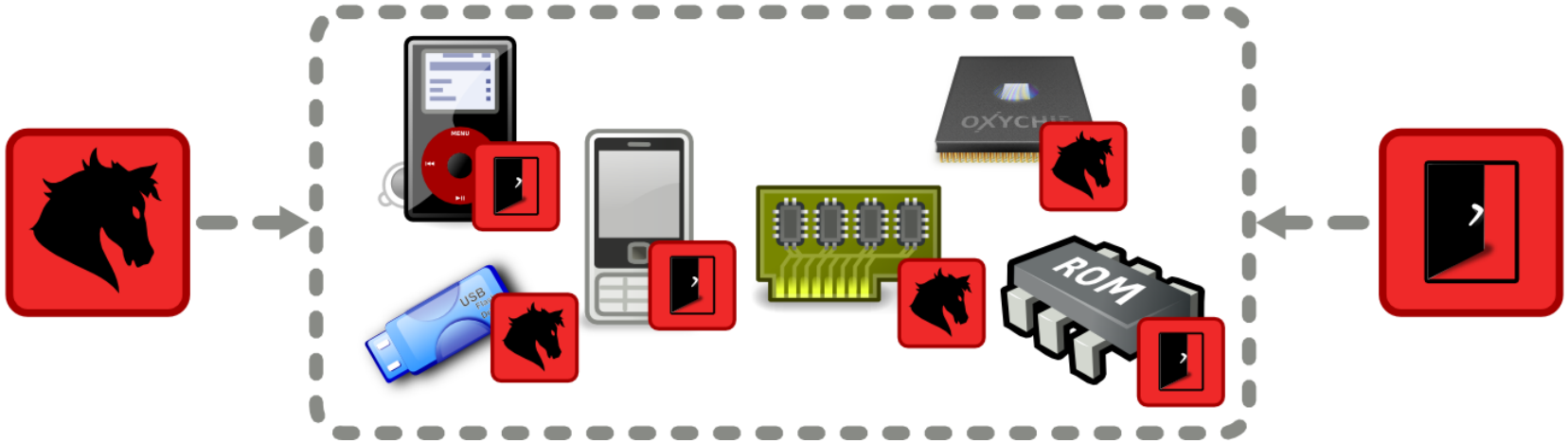
Contexte et problématique (2/3)



Hypothèses d'attaques considérées

- Composants vulnérables qui peuvent être modifiés
- Composants conçus pour contenir des fonctions cachées

Contexte et problématique (3/3)



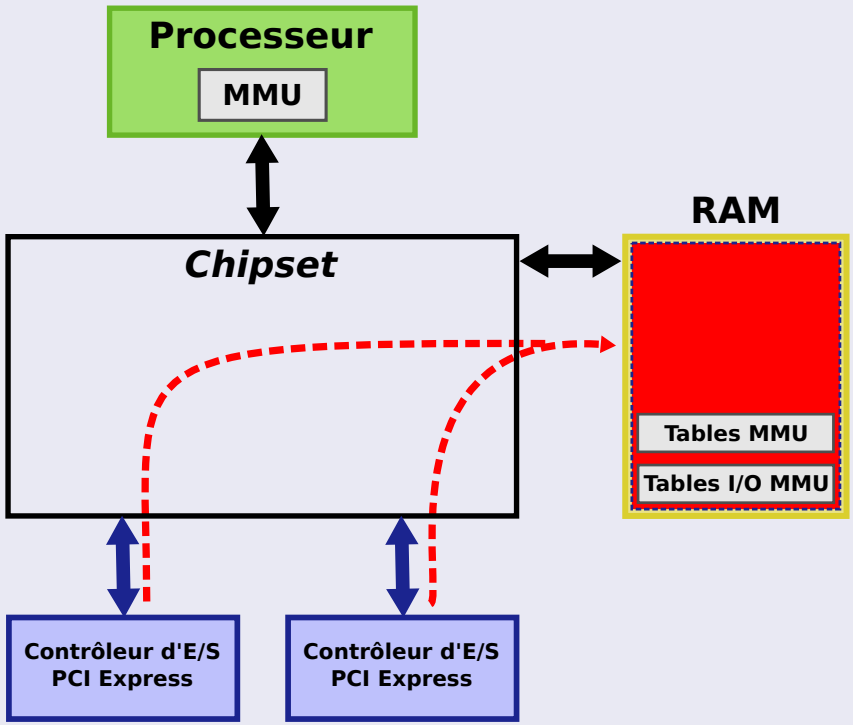
Problématiques sur lesquels portent nos travaux

- Actions malveillantes possibles pour un attaquant
- Efficacité des contre-mesures actuelles
- Contre-mesures complémentaires pour s'en protéger

Attaques vers la mémoire (1/4)

Accès DMA Bus Master

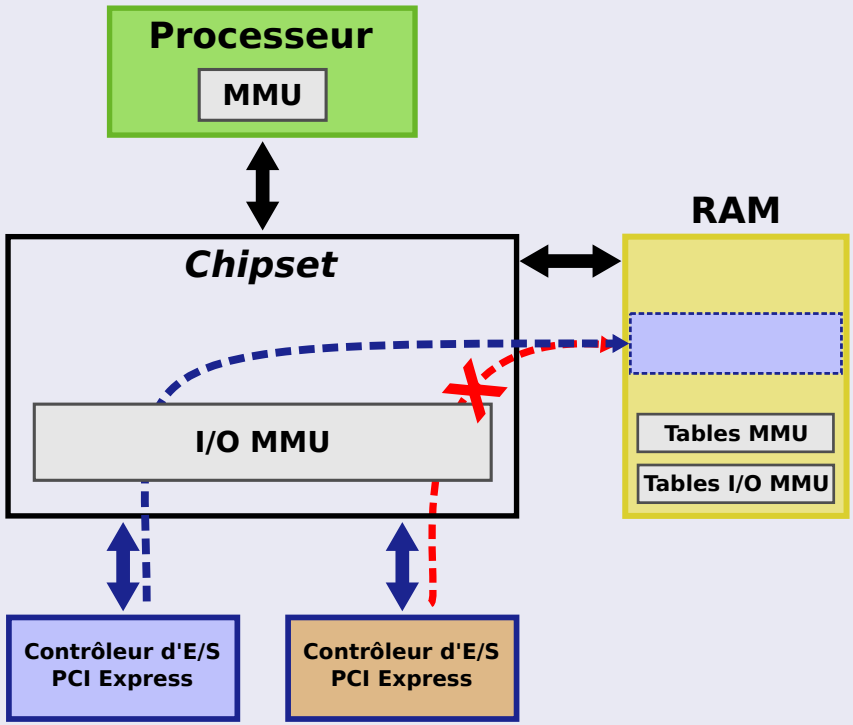
- Accès DMA sur la mémoire possible à l'aide de périphériques de type "Bus Master" : possible d'écraser n'importe quelle zone de mémoire y compris celle du noyau !



Attaques vers la mémoire (2/4)

Contre-mesure : l'IOMMU

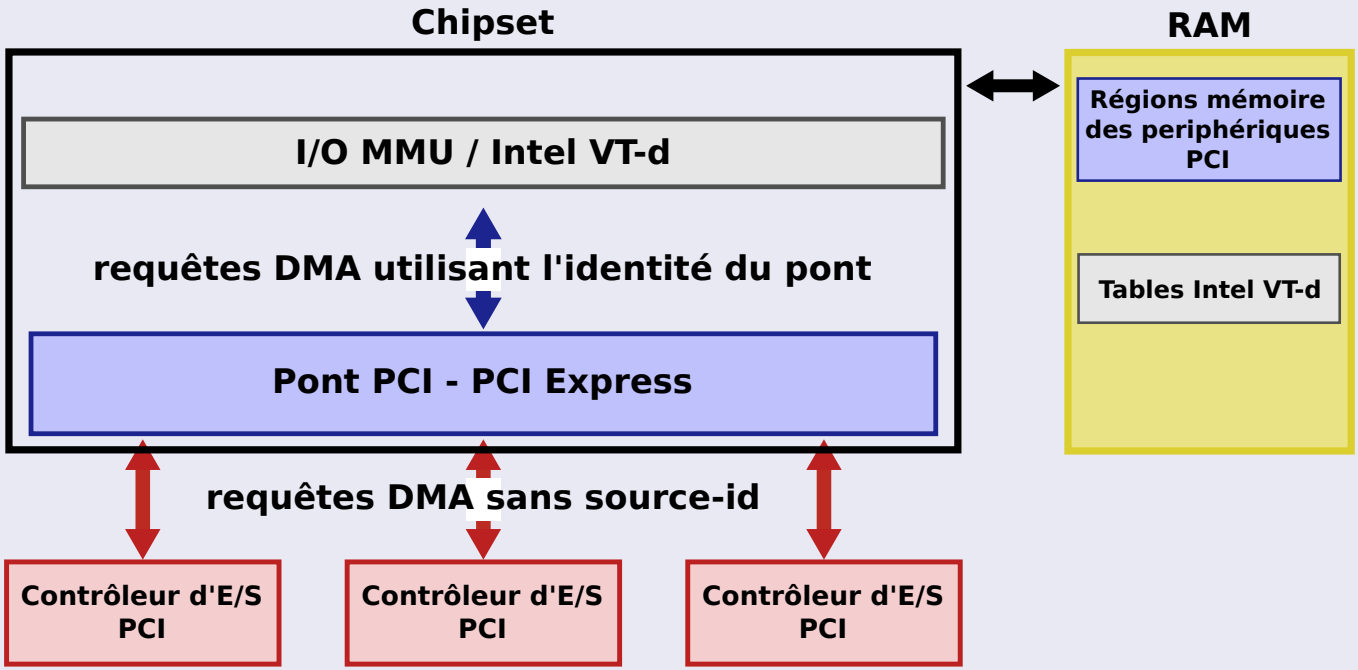
- Même fonctionnalité qu'une MMU mais pour les accès de type entrée-sortie : sur les processeurs récents seulement ... de l'époque :)
- S'intercale entre un système d'E/S et la mémoire : traduction d'adresse et contrôle d'accès



Attaques vers la mémoire (3/4)

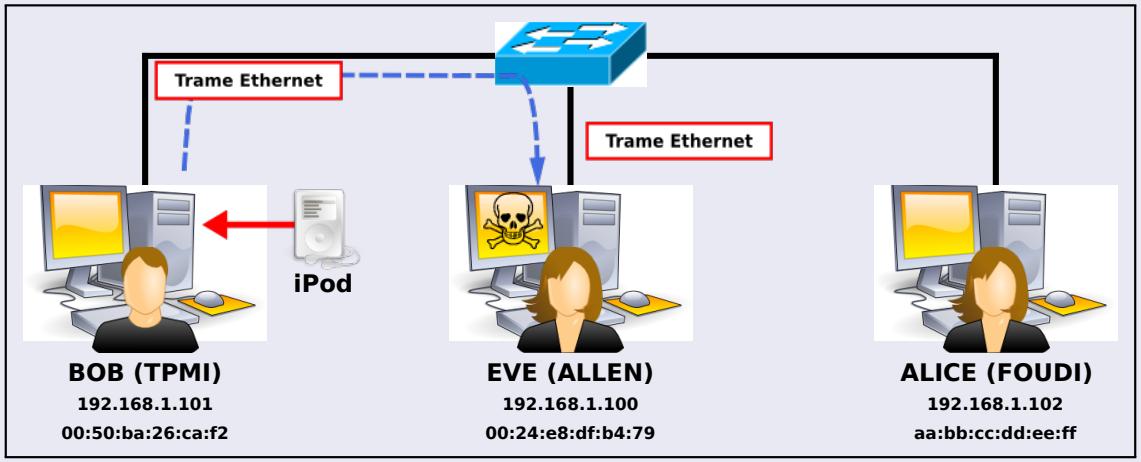
Limites de l'IOMMU

- Périphériques PCI sont associés au même identifiant PCI-Express et peuvent donc accéder à la même zone de mémoire



Attaques vers la mémoire (4/4)

L'attaque



1. Mise en place d'une attaque exploitant la limitation de l'IO/MMU
   - Forger des paquets ARP à injecter
   - Localiser les régions mémoire utilisées par un contrôleur Ethernet
   - Programmer un iPod pour injecter les trames malveillantes
2. Rejeu sur la machine de Bob
   - Prêter l'iPod à Bob
   - Recevoir les paquets réseaux détournés

Attaques peer-to-peer (1/2)

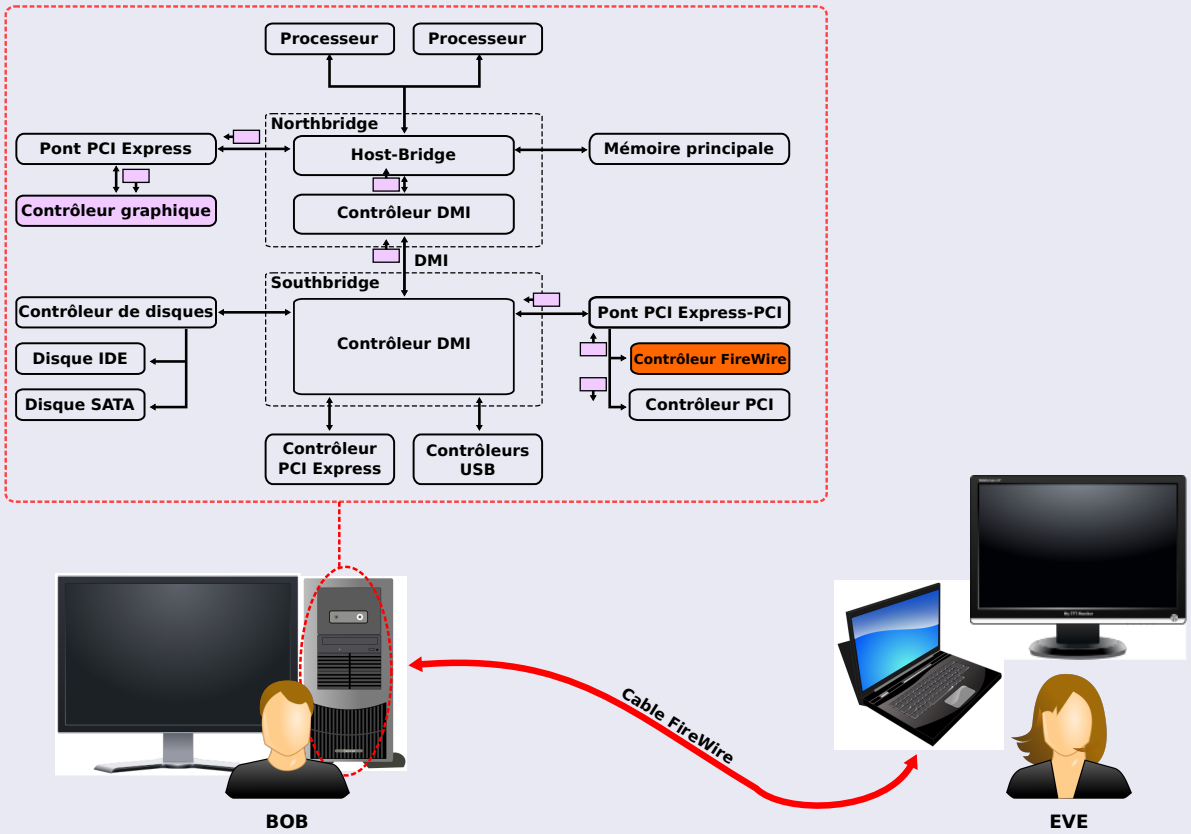
Le principe

- Accès par des périphériques à d'autres périphériques par E/S
- Lecture/modification des registres et des mémoires internes
- L'IOMMU est cette fois encore contournable (située dans le northbridge et ne peut donc contrôler les accès au sein du southbridge)

Attaques peer-to-peer (2/2)

L'attaque

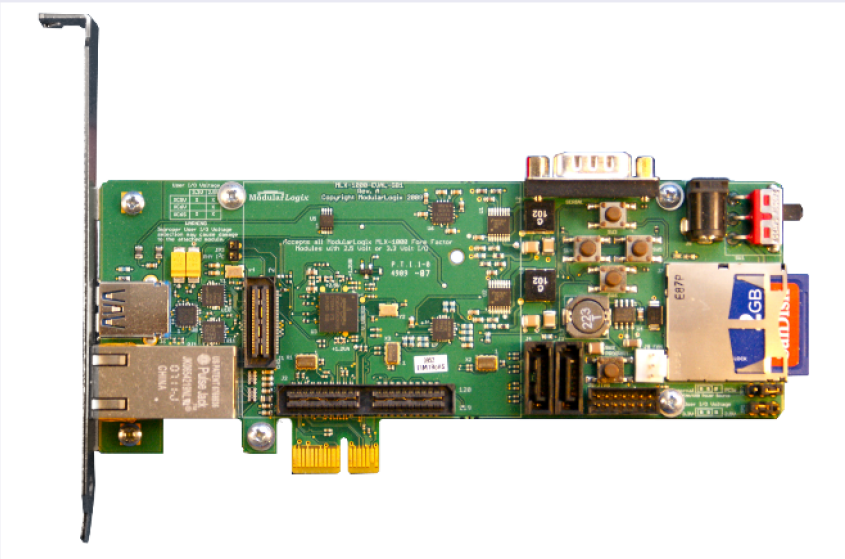
- Preuve de concept : recopie de la mémoire vidéo d'une machine grâce à un périphérique firewire



IronHide (1/2)

Description

- IronHide est une carte FPGA connectée à un bus PCI-Express et capable d'émettre tout type de requête sur ce Bus
- Requêtes légitimes mais aussi illégitimes
- Peut être utile pour : générer des attaques, mais aussi faire de la détection d'intrusions
- Expérimentation : fuzzing sur le bus PCI-Express



IronHide (2/2)

Résultats de tests de fuzzing

		Eaglelake Lakeport	Tylersburg
TLP Valide	**Requête normale**	-	-
	Requête inhabituelle	Déni de service sur un segment de bus PCI-Express amenant à un *kernel panic*	-
TLP invalide	**Altération** `Address`	Erreur non détectée	Erreur détectée
	Altération `Length`	Déni de service sur un segment de bus PCI-Express amenant à un *kernel panic*	Erreur détectée
	Altération `Reserved`	Erreur non détectée	Erreur détectée

1 Sécurité des noyaux d'OS : *rootkits* noyau et contre-mesures

2 Les attaques par entrées-sorties

3 Architecture mixte logicielle/matérielle de sécurité

Architecture de sécurité mixte logicielle/matérielle (2012-2015)

Thèse de Benoit Morgan

Projets Investissements d'Avenir SVC

Limitation des solutions de sécurité actuelles

- Il existe toujours un logiciel privilégié sur lequel repose toute la sécurité
- Les protections les plus privilégiées ont été contournées
  - Attaques sur la virtualisation, les caches (canaux de fuite, etc)
  - *"Bypassing IOMMU protection against I/O Attacks", LADC 2016, B. Morgan, E. Alata, V. Nicomette et M. Kaâniche*

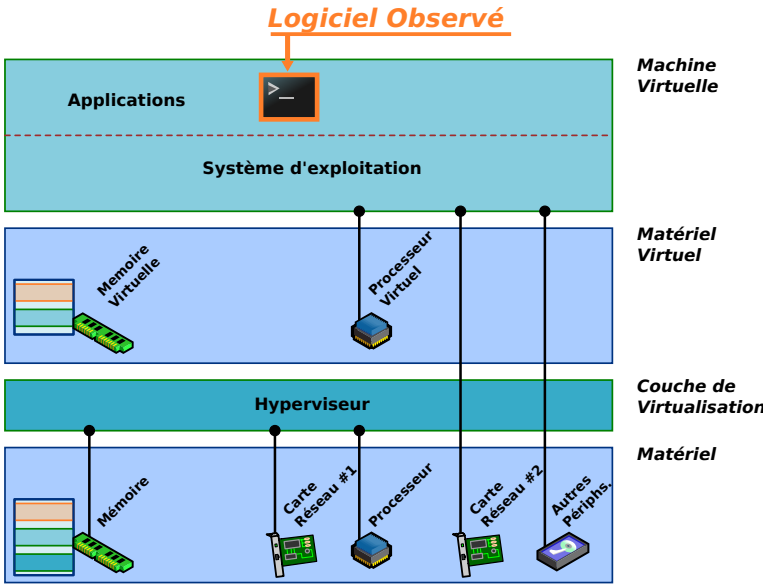
$\Rightarrow$ Le processeur peut être compromis $\Rightarrow$ le logiciel privilégié aussi

Problématique de la thèse

Comment s'assurer de l'intégrité d'un logiciel sans faire confiance au processeur ?

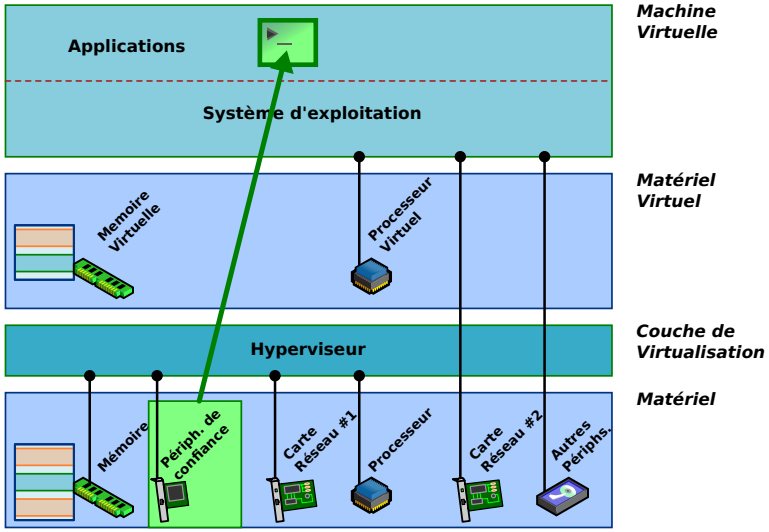
- La sécurité doit être assurée par un composant dédié
- Il ne faut pas que ce composant soit compromis suite à une compromission du processeur
- Architecture de sécurité assistée par un composant indépendant
- Focalisation des travaux sur les architectures Intel

Test de l'intégrité de logiciels



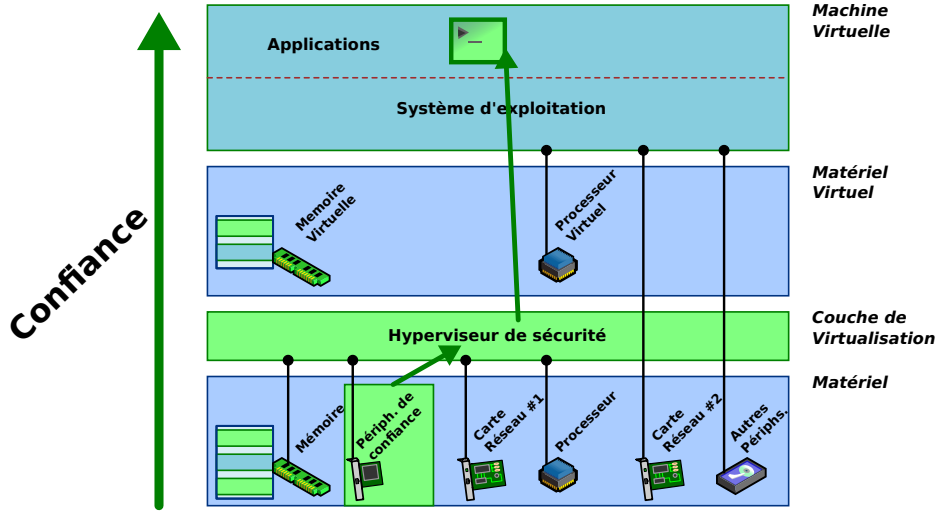
- Comment s'assurer de l'intégrité d'un logiciel ?

Composant externe seul



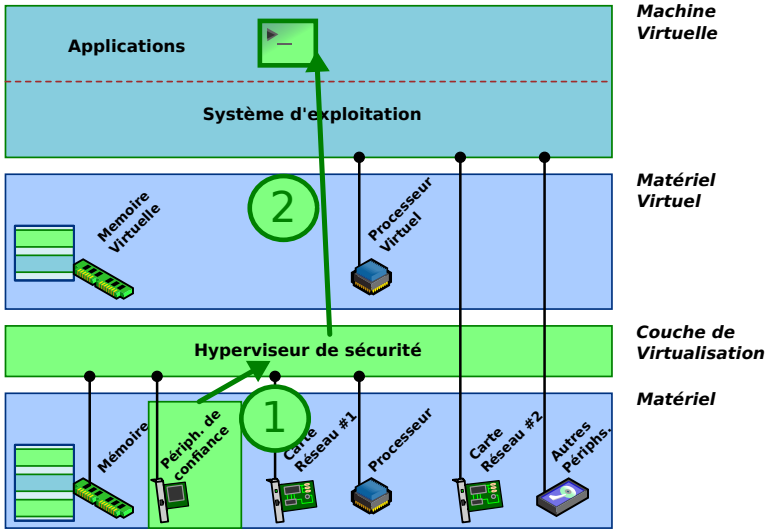
- Fossé sémantique : difficile de déterminer l'état réel des applications
  - Difficile de retrouver la vision mémoire du processeur
  - Impossible d'accéder à l'état interne du processeur

Architecture proposée



- Architecture assistée par le matériel pour la surveillance de l'intégrité de logiciels

Protocole de test d'intégrité

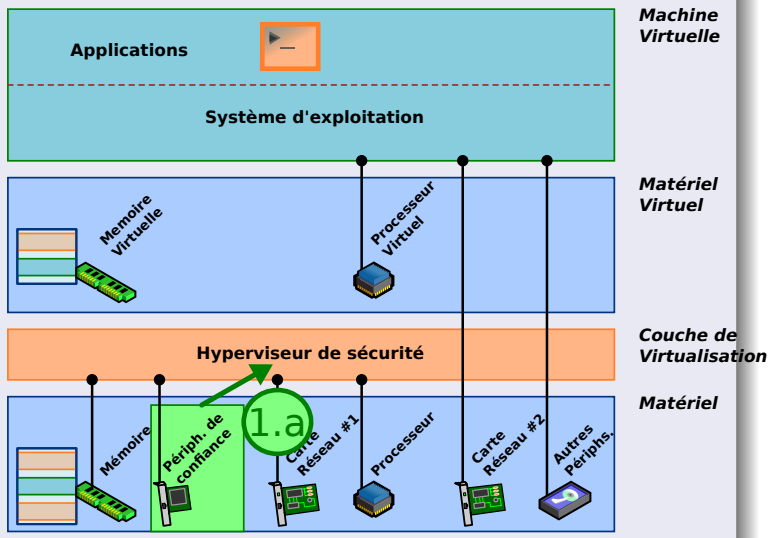


- Phase 1 : périphérique de confiance
  - Teste l'intégrité de l'hyperviseur de sécurité
  a) Challenges
  b) Tests d'environnement
- Phase 2 : hyperviseur de sécurité
  - Teste l'intégrité du logiciel observé
  - Approche boite noire : fonction de test f, apportée par le développeur du logiciel surveillé

Phase 1.a : les challenges

Objectif : s'assurer que l'hyperviseur de sécurité contrôle le matériel

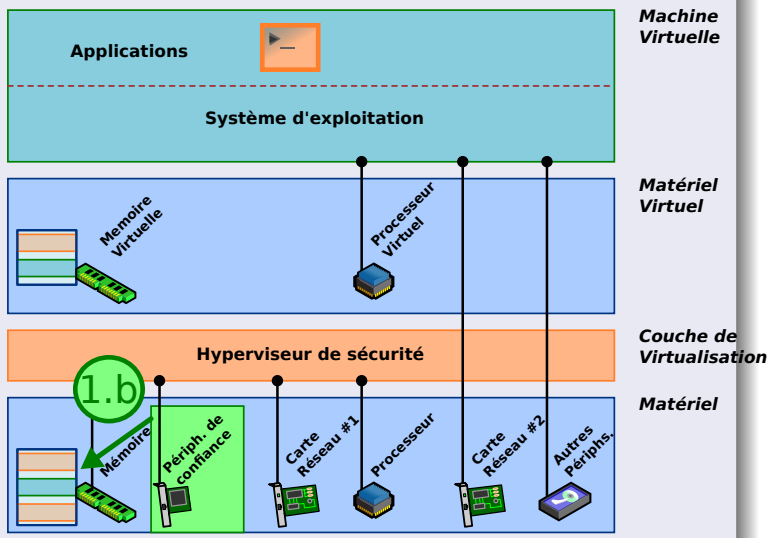
- Un challenge est :
  - Un algorithme généré aléatoirement
  - Produisant un résultat attendu
  - Dans un temps d'exécution borné
- Forme d'un challenge : code exécutable x86
  - Instructions d'identification du processeur
  - Lecture de registres spécifiques au matériel
  - Lecture des structures de contrôle de machines virtuelles



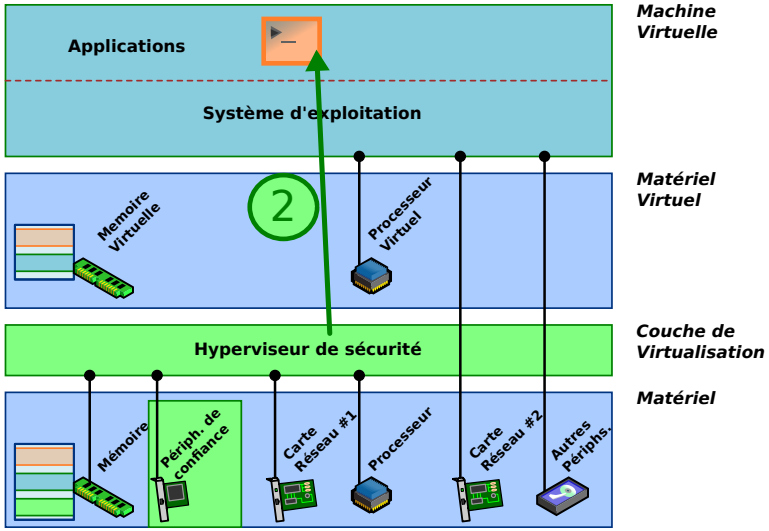
Phase 1.b : tests d'environnement

Objectif : contrôler l'environnement d'exécution de l'hyperviseur de sécurité

- Test des propriétés de zones mémoire
- Comment ?
  - Calcul et vérification de l'empreinte du code et des données statiques
  - Vérification de structures mémoire de contrôle du matériel à l'aide d'un processeur dédié

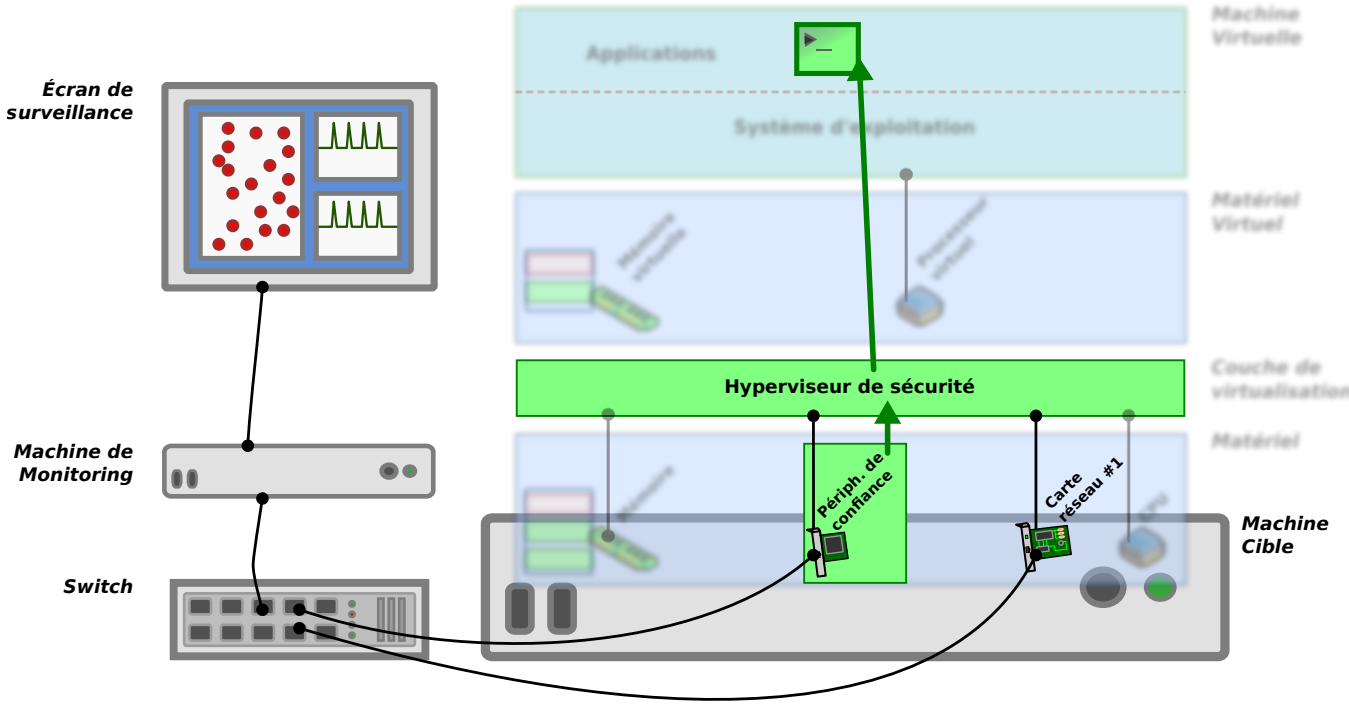


Phase 2 : test de l'intégrité du logiciel observé

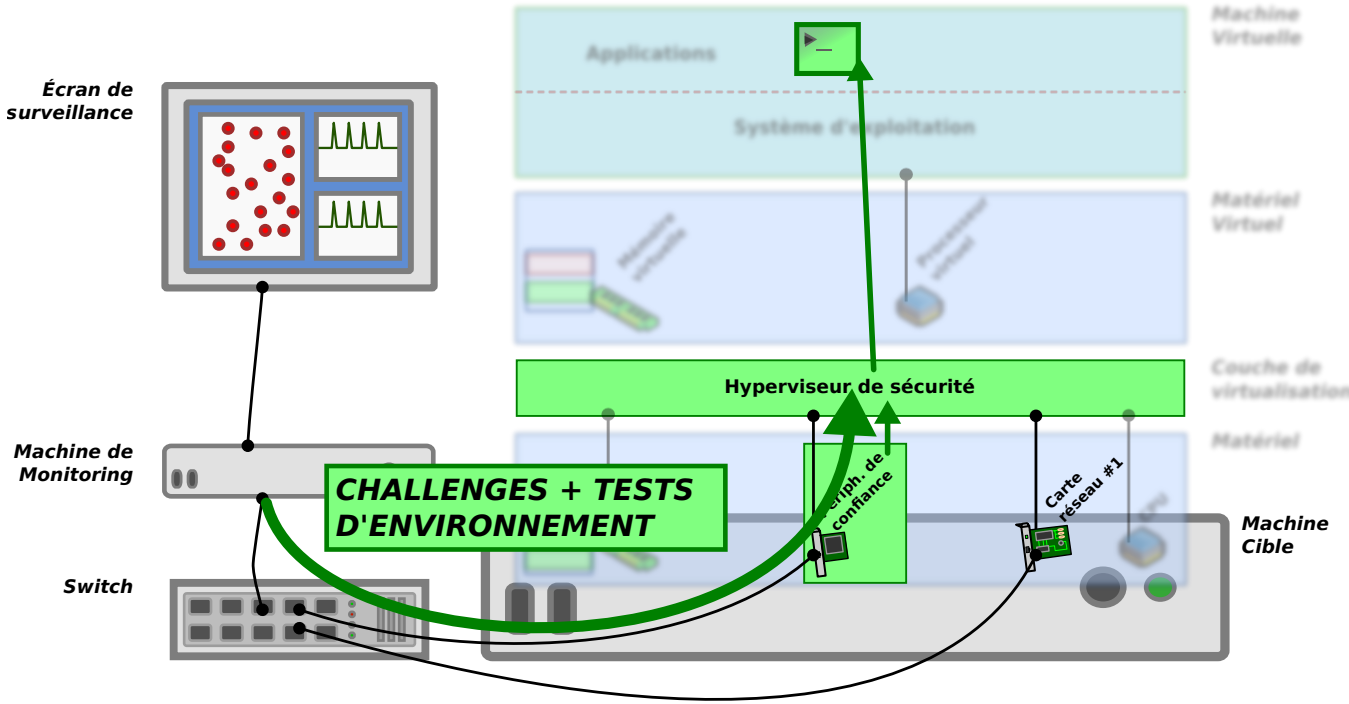


- Aucune compromission de l'hyperviseur de sécurité n'a été detectée
- Exécution de la fonction de test d'intégrité f, fournie par le développeur

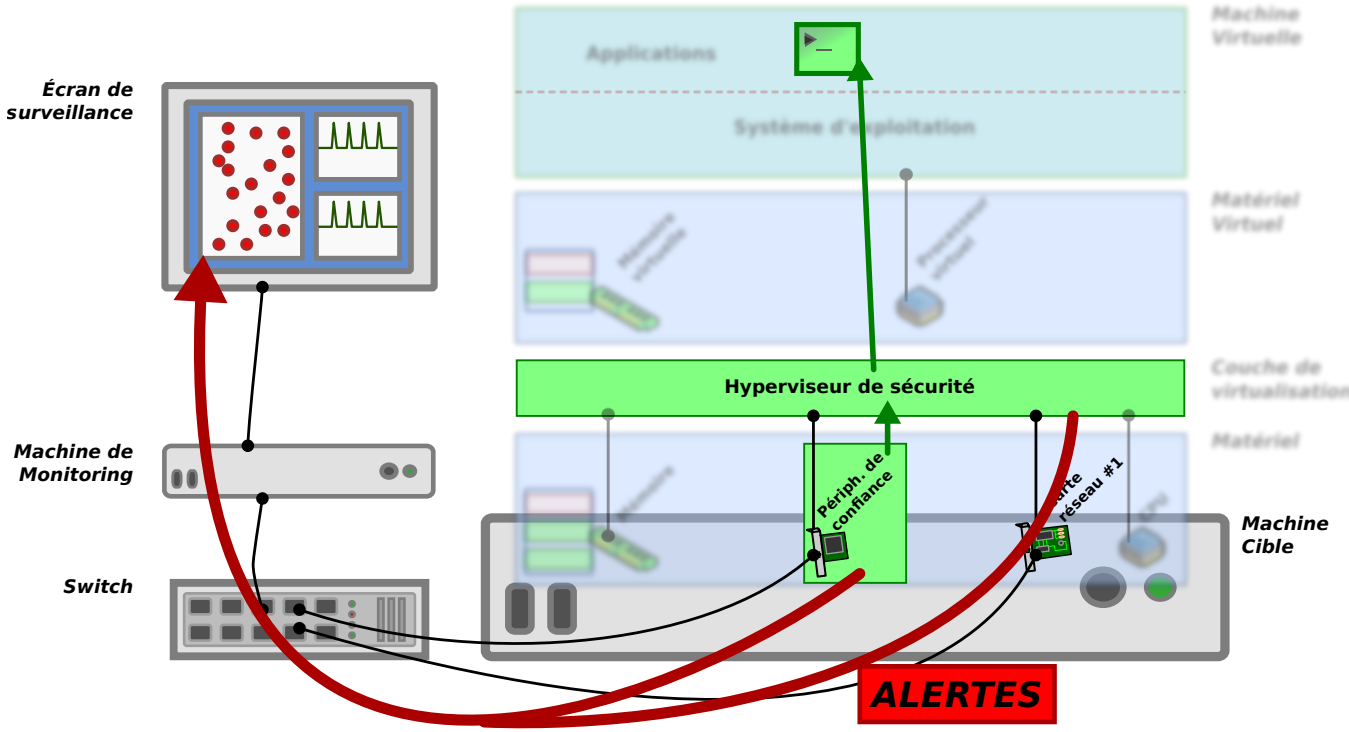
Architecture du prototype



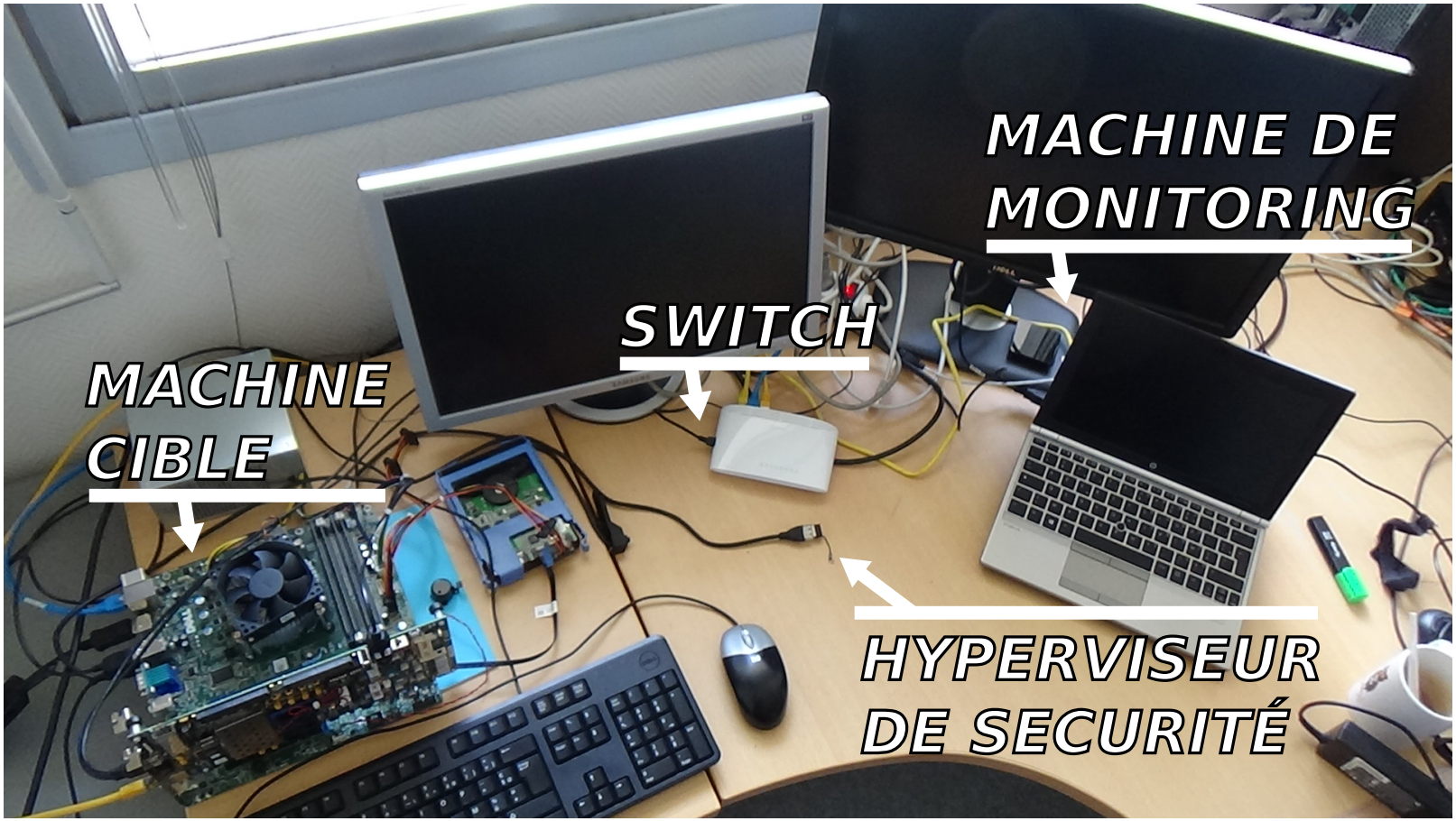
Architecture du prototype



Architecture du prototype



Architecture du prototype



Périphérique de confiance : choix de développement

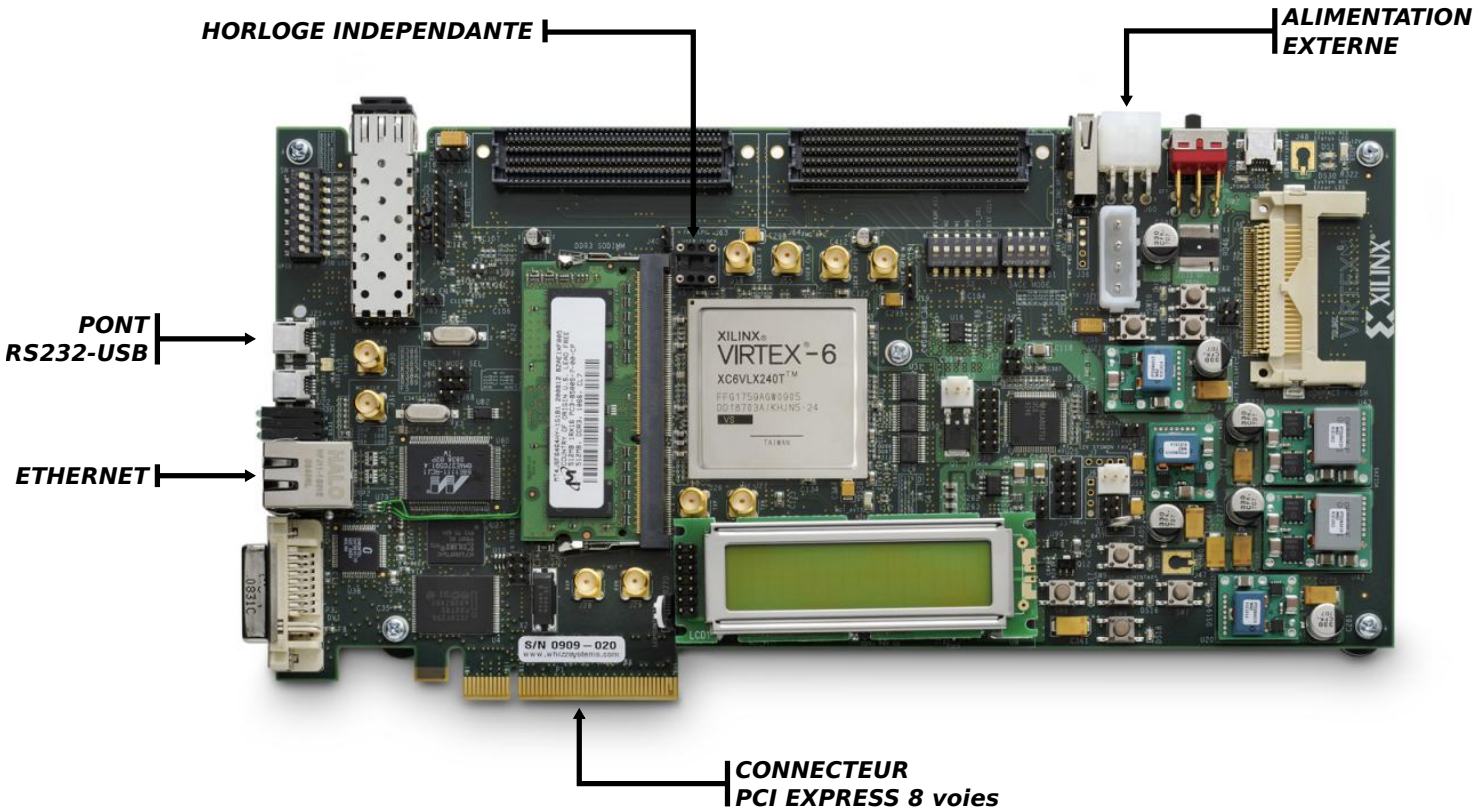
Contraintes fortes

- Minimiser la conception de composants électroniques
- Flexibilité et modularité de l'architecture matérielle / logicielle
- Compatibilité avec la spécification PCI Express 2
- Maîtrise de la couche transactionnelle PCI Express

Solutions

- Carte de développement PCI Express
- Embarquant un circuit programmable de type FPGA
- Architecture matérielle de type *System On Chip*
- Utilisation d'un coeur propriétaire modulaire pour la pile matérielle PCI Express

Carte de développement FPGA sélectionnée



Hyperviseur de sécurité : caractéristiques

- Intégrable facilement dans une architecture existante
- Virtualise le système dans une machine virtuelle
- Minimaliste
- Contrôlable à distance par la machine de *monitoring*
- Utilisation du support matériel à la virtualisation
- Récursif

Hyperviseur de sécurité : détails techniques

Hyperviseur

- Architecture Intel 64
- Intel VT-x (EPT, shadow VMCS)
- Bare-metal ⇒ générique
- Driver UEFI de *runtime*

Contrôle de la machine virtuelle à distance

- Serveur et client de contrôle
- Intégration d'une pile IP
- Driver Ethernet UEFI de *runtime*

Mesure des performances : configuration et tests

Configuration matérielle

- Processeur Intel i7-4770
- 8 Gigaoctets de SDRAM DDR3
- Vitesse du lien PCI Express : 4x
- Contrôleurs Ethernet Gigabit connectés à 100 Mégabits

Types de tests

- 10 calculs de 10 millions de décimales de π
- 10 copies d'un fichier de 512 Mégaoctets avec `dd`, sur le même disque
- 4 transferts à l'aide de `ssh` de fichiers de 512 Mégaoctets sur une machine distante

Mesure des performances : résultats

		Périodes de test d'intégrité avec le protocole de monitoring		
Benchmark	Hyperviseur seul	5s	1s	100ms
Performances brutes (π)	0.1 %	0.4 %	1.2 %	10 %
Disque	-1.3 %	-0.1 %	0.4 %	1.2 %
Réseau	-0.1 %	0.1 %	-0.2 %	-0.4 %

- Impact très raisonnable sur les performances

Capacité de détection

Vulnérabilités injectées

1. Interface de paravirtualisation vulnérable
2. Interface PCI Express vulnérable
3. Périphérique vulnérable piloté à distance

Attaques exécutées

#	Vuln.	Description
1	2	Corruption des données et du code de l'hyperviseur.
2	2	Modification de la mémoire virtuelle de l'hyperviseur.
4	1	Virtualisation de l'hyperviseur.
5	1	Modification de l'ordre de démarrage.
6	2	Relocalisation de l'hyperviseur.
7	3	Contournement du contrôle d'accès sur les périphériques.
8	1	Modification du *driver* Linux Intel e1000e.

Détection des attaques

#	Carac. altérées	Challenge	Test d'env.	Test d'int.
1	1		✓	
2	9		✓	
3	13	✓		
4	3, 5, 9, 14	✓ (3)		
5	2	✓		
6	3, 5, 9, 14	✓		
7	2		✓	
8	15			✓

Détection des attaques

#	Carac. altérées	Challenge	Test d'env.	Test d'int.
1	1		✔	
2	9		✓	
3	13	✓		
4	3, 5, 9, 14	✓ (3)		
5	2	✓		
6	3, 5, 9, 14	✓		
7	2		✓	
8	15			✓

Attaque : corruption des données et du code de l'hyperviseur

- Méthode de détection : test d'environnement
- Les empreintes du code et des données ne correspondent plus à celles attendues

Détection des attaques

#	Carac. altérées	Challenge	Test d'env.	Test d'int.
1	1		✓	
2	9		✓	
3	13	✓		
4	3, 5, 9, 14	✔ (3)		
5	2	✓		
6	3, 5, 9, 14	✓		
7	2		✓	
8	15			✓

Attaque : virtualisation malveillante de l'hyperviseur

- Méthode de détection : challenge
- L'hyperviseur ne s'exécute plus en mode hyperviseur
- La vision des registres est contrôlée par l'hyperviseur malveillant
- Seul le temps d'exécution n'est pas respecté

Détection des attaques

#	Carac. altérées	Challenge	Test d'env.	Test d'int.
1	1		✓	
2	9		✓	
3	13	✓		
4	3, 5, 9, 14	✓ (3)		
5	2	✓		
6	3, 5, 9, 14	✔		
7	2		✓	
8	15			✓

Attaque : relocalisation de l'hyperviseur

- Méthode de détection : challenge
- Notre hyperviseur est désactivé
- La lecture des registres de configuration nous indique les manquements aux caractéristiques 5, 9 et 14

Bilan

La sécurité dans les couches basses

- Est un aspect à ne pas négliger
- En particulier là où les méthodes formelles montrent encore leurs limites
- Sophistication des architectures matérielles ⇒ surface importante d'attaque : canaux de fuite, attaques sur les caches, sur la virtualisation, etc

Perspectives

- Diversifier les architectures : ARM en cours
- Les objets connectés ...