Tests d'intégrité d'hyperviseurs de machines virtuelles à distance et assisté par le matériel

Benoît Morgan, **Éric Alata**, Vincent Nicomette

LAAS-CNRS, INSA Toulouse

14 juin 2014

Plan

Contexte des travaux

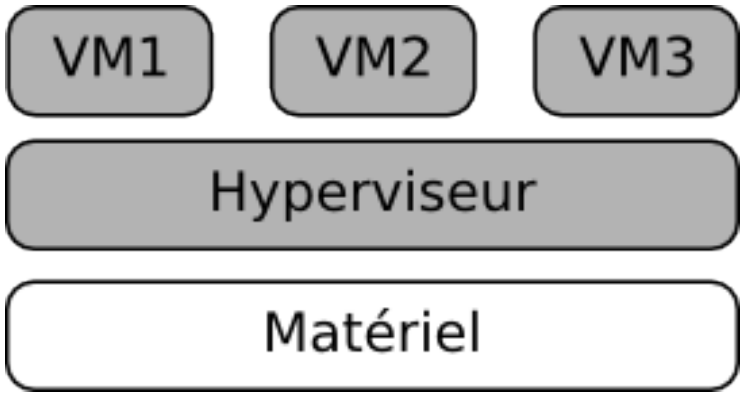
Projet SVC – *Secure Virtual Cloud*

- Projet Investissement d'Avenir
  Itrust, *Bull*, *Gosis*, *Eneed*, *Secludit*, *Eurogiciel*, *Val Informatique*, *Blue Mind*, *LAAS-CNRS*, *IRIT*,
- Coordinateur du projet : Bull

Contributions du LAAS – 3 thèses

- Évaluation de la sûreté de fonctionnement dans le *cloud*
- Évaluation et analyse de la sécurité dans le *cloud*
- **Sécurisation de gestionnaires de machines virtuelles**

Virtualisation dans le Cloud



Hypothèses

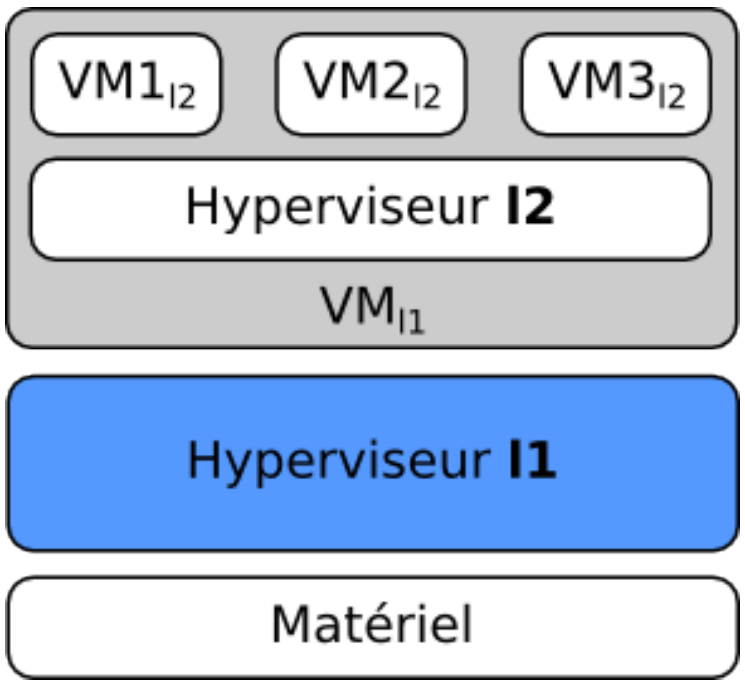
- Exécution, au dessus du même *hyperviseur*, de machines virtuelles appartenant à des clients différents
- Utilisation de plateformes Intel

Risques à prendre en compte : machine virtuelle malveillante

- Compromission de l'*hyperviseur*
- Désactivation de l'*hyperviseur*
- Installation d'un *hyperviseur* "malveillant"

Approche envisagée

⇒ Utilisation de la *nested virtualisation*



Quid des attaques provenant du matériel ?
Nécessité d'un composant matériel de confiance

Composant utilisable pour réaliser les tests d'intégrité

- Utilisation de Intel AMT
- Instrumentation du mode SMM
- Utilisation de Intel SGX
- **Utilisation d'un périphérique PCIe dédié**
  Développement simplifié à partir d'un FPGA

$\Rightarrow$ Test de l'intégrité à distance avec un périphérique PCIe/FPGA

Plan

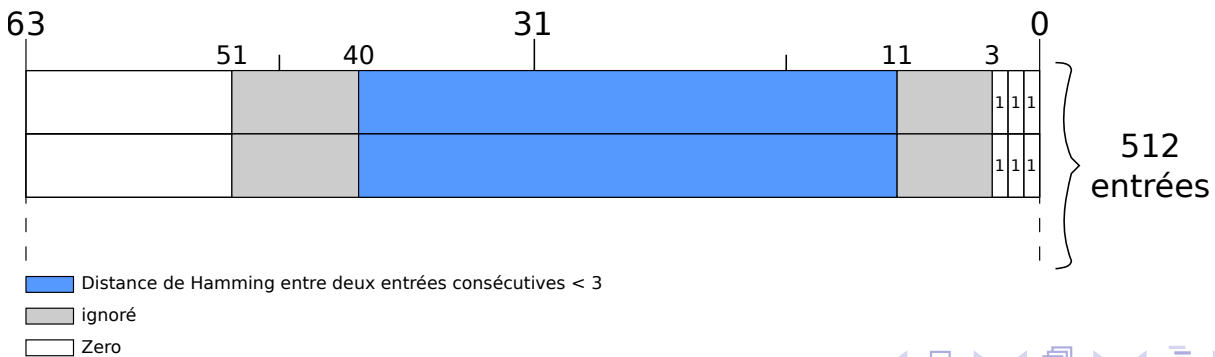
Empreinte mémoire d'un hyperviseur

Espace mémoire utilisé par un hyperviseur

- Code.
- Données.
- Structures de contrôle du processeur.
  - Table de pages et tables de page EPT.
  - Table de vecteurs d'interruptions.
  - Structure de contrôle de machine virtuelle (VMCS).

$\Rightarrow$ Construction d'empreintes mémoire.

Exemple : table de page EPT en *identity mapping*.



Vérification

- Modifications de l'hyperviseur.

$\Rightarrow$ Modification du code, des données, des structures de contrôle.

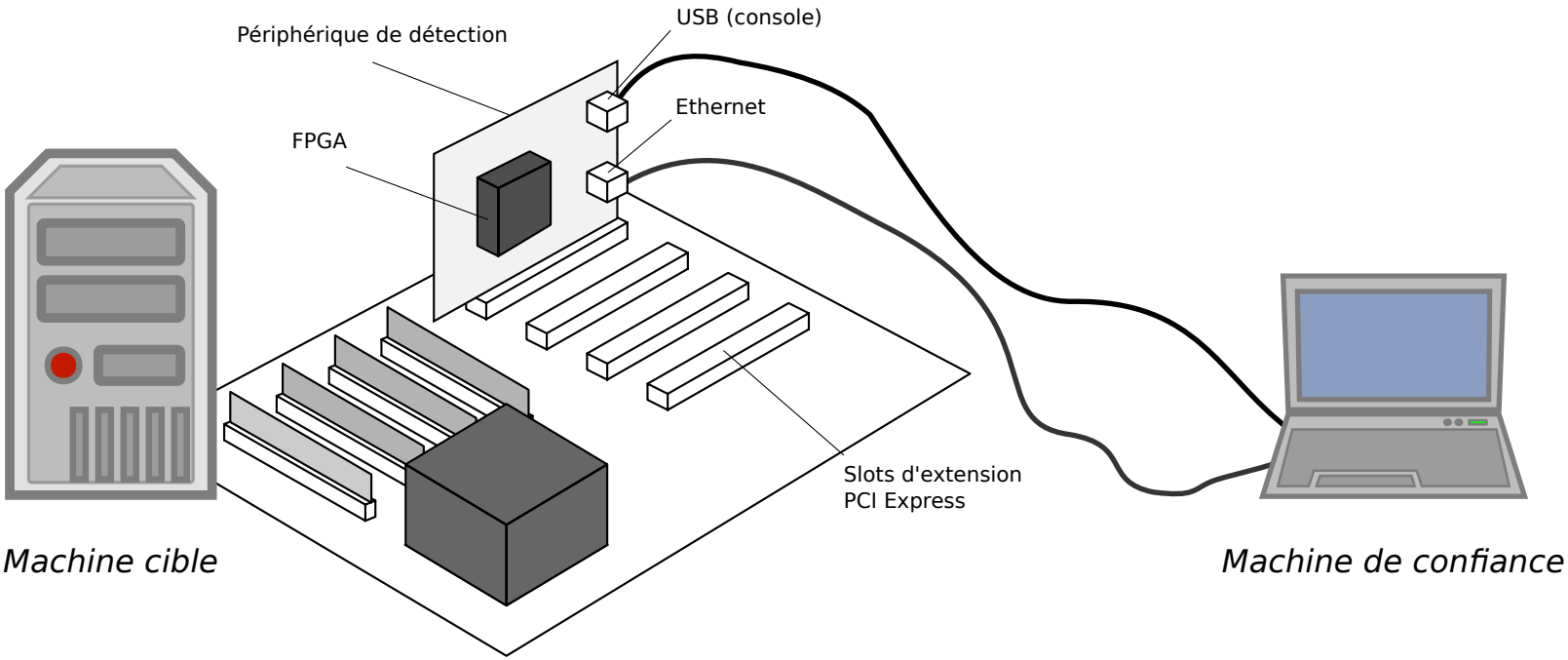
- Désactivation de l'hyperviseur.

$\Rightarrow$ Non évolution significative des données et des VMCS.

- Relocalisation de l'hyperviseur.

$\Rightarrow$ détection de nouveaux espaces mémoire de logiciels bas niveau.

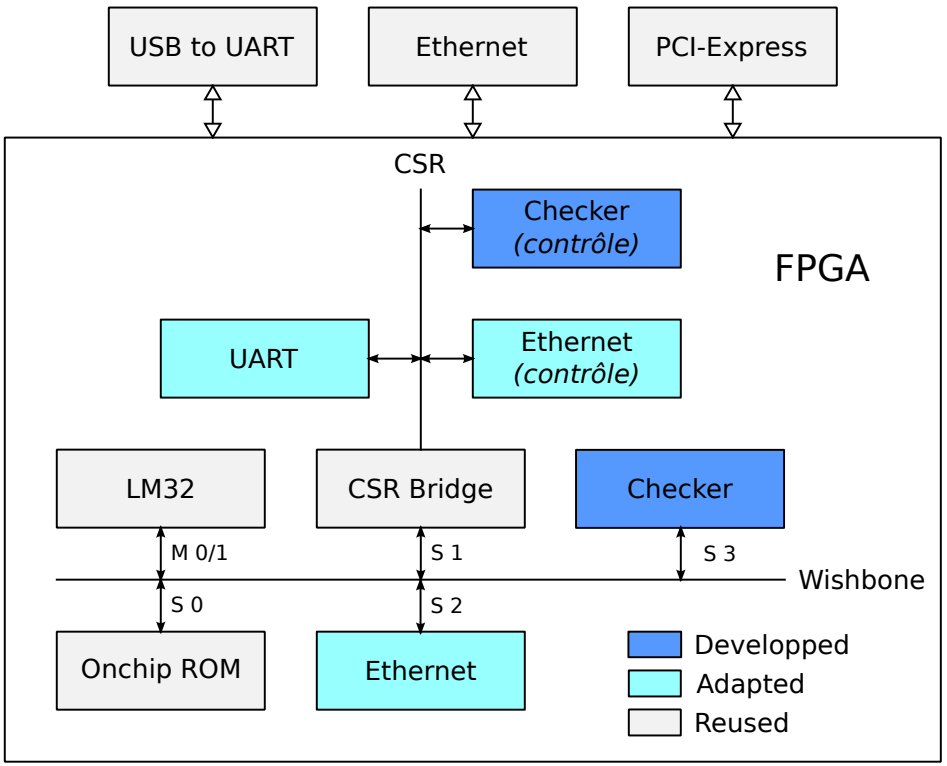
Utilisation du périphérique de détection



- Nombre d'empreintes mémoire variable et élevé.
- Balayage de tout l'espace mémoire.
- Besoin d'un processeur dédié.

$\Rightarrow$ FPGA.

Architecture du *System On Chip*



ERIC (Electronic Remote Integrity Checker)

Basé sur IronHide et Milkymist

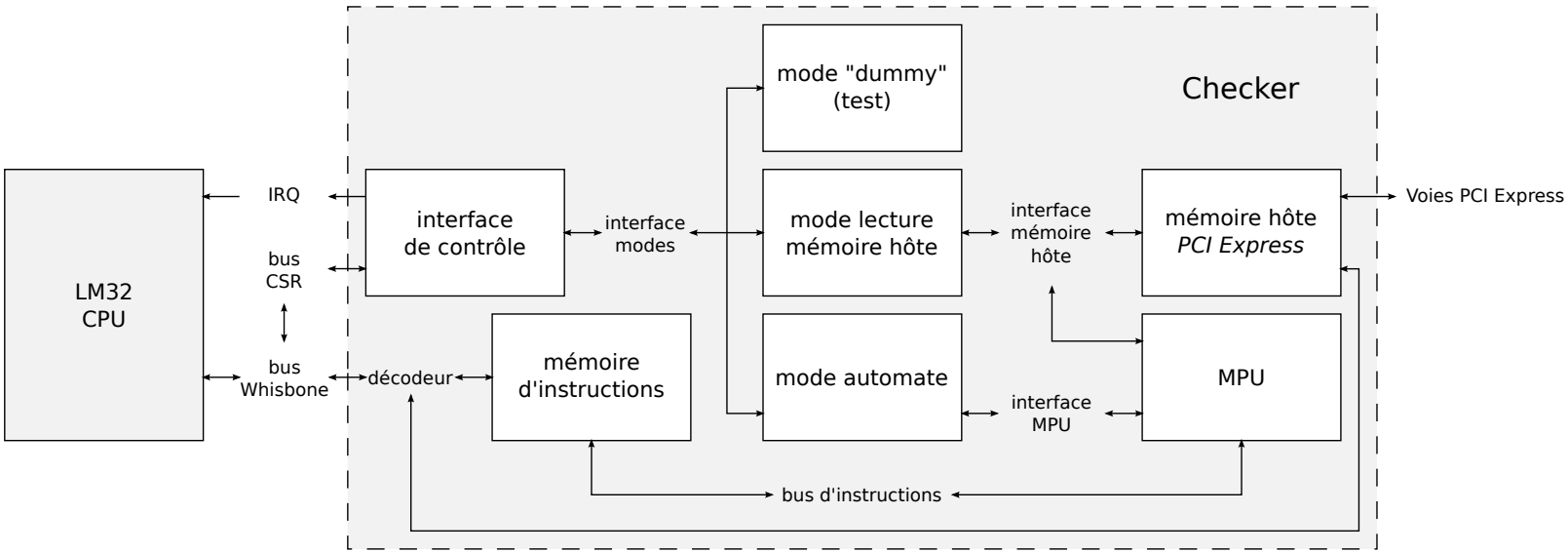
Composants

- Processeur généraliste.
- ROM (BIOS).
- Checker.
- Interface Ethernet.
- UART.

Fonctionnalités

- Débuguer.
- Charger et exécuter du logiciel.
- Caractériser les zones mémoire.
- Remonter des alertes.

Principales fonctions du *Checker*



- Accéder aux pages mémoire hôte.
- Exécuter des automates de détection de motifs mémoire.
- Interrompre le LM32.

Co-processeur MPU

Code	mnémonique	Description
0x1	mask	Vérifie les bit autorisés à 0 et 1, saute si faux
0x2	equ	Compare deux registres, saute si différent
0x3	inf	Compare deux registres, saute si supérieur ou égal
0x4	add	Addition
0x5	hamm	Calcule la distance de Hamming entre deux registres
0xc	int	Interruption logicielle
0xd	mload	Chargement d'un mot mémoire hôte
0xe	load	Chargement d'une valeur immédiate
0xf	jmp	Saut inconditionnel

- Format 1 : OP, r1, [r2, [r3, [r4]]].
- Format 2 : OP, r1, valeur immédiate (8, 16, 32).
- Jeux d'instructions spécialisé.

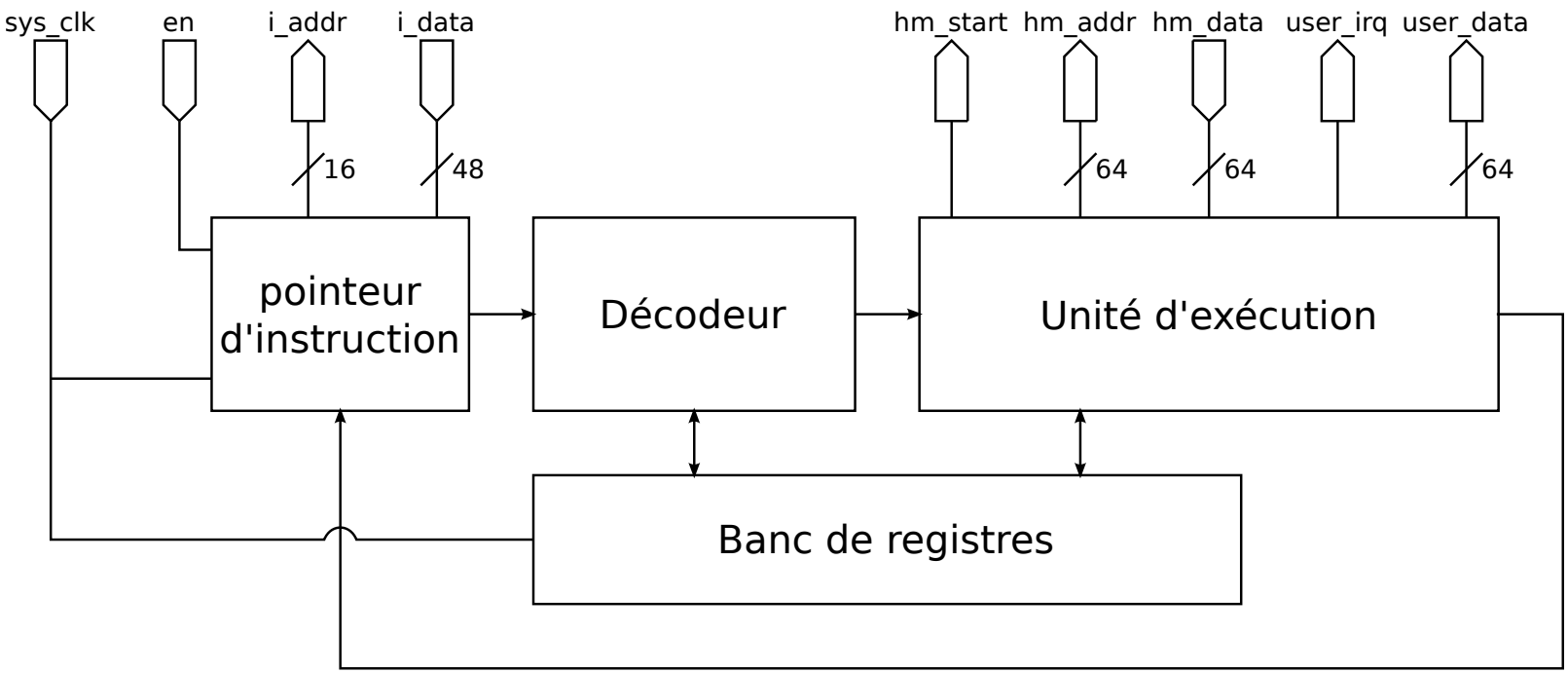
$\Rightarrow$ Gain de performances.

Distance de Hamming en C

```
uint8_t hamdist(uint64_t x, uint64_t y, uint64_t bs) {
  uint8_t dist = 0;
  uint64_t val = (x & bs) ^ (y & bs);
  // Count the number of set bits
  while (val) {
    ++dist;
    val &= val - 1;
  }
  return dist;
}
```

64 itérations dans le pire des cas.

Architecture du MPU



Cas d'utilisation

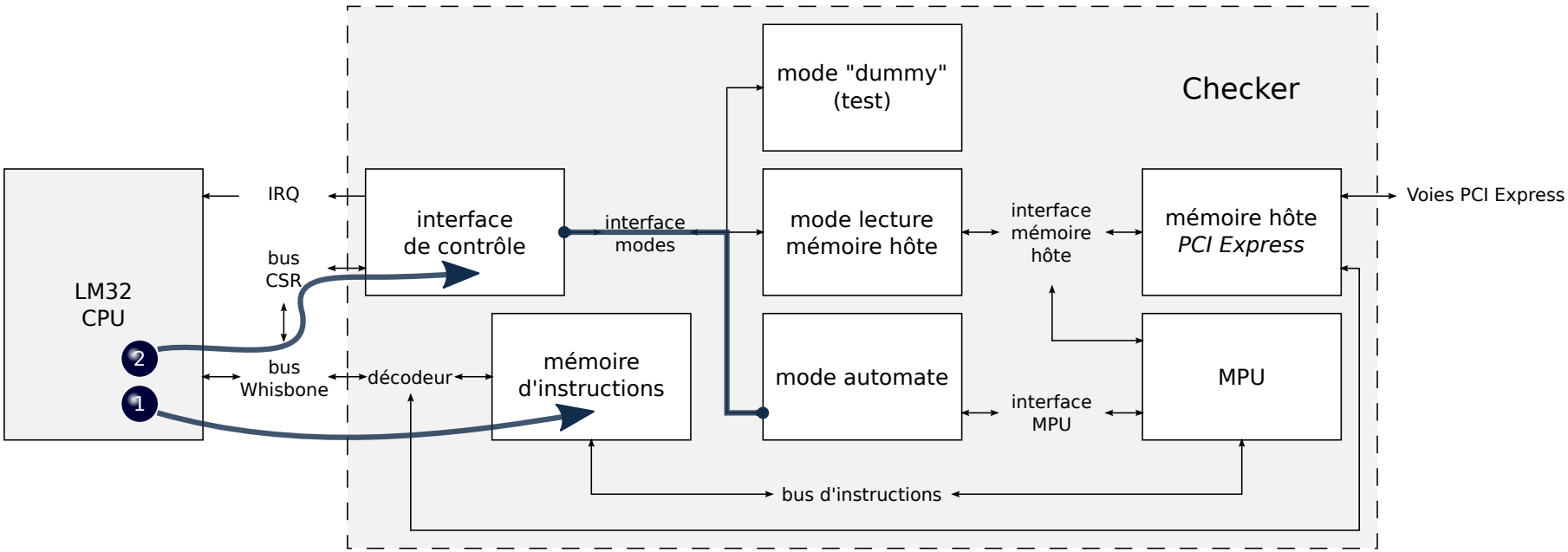
Recherche d'espaces mémoire de malwares au travers de la reconnaissance de structures de contrôle des cœurs du processeur.

1. Initialisation des composants.
2. Calcul de l'adresse de la prochaine page à analyser.
3. Exécution du MPU sur la page.
4. Prise de décision (alertes, statistiques).
5. Retour à l'étape 2.

Automate de reconnaissance d'une table de pages EPT

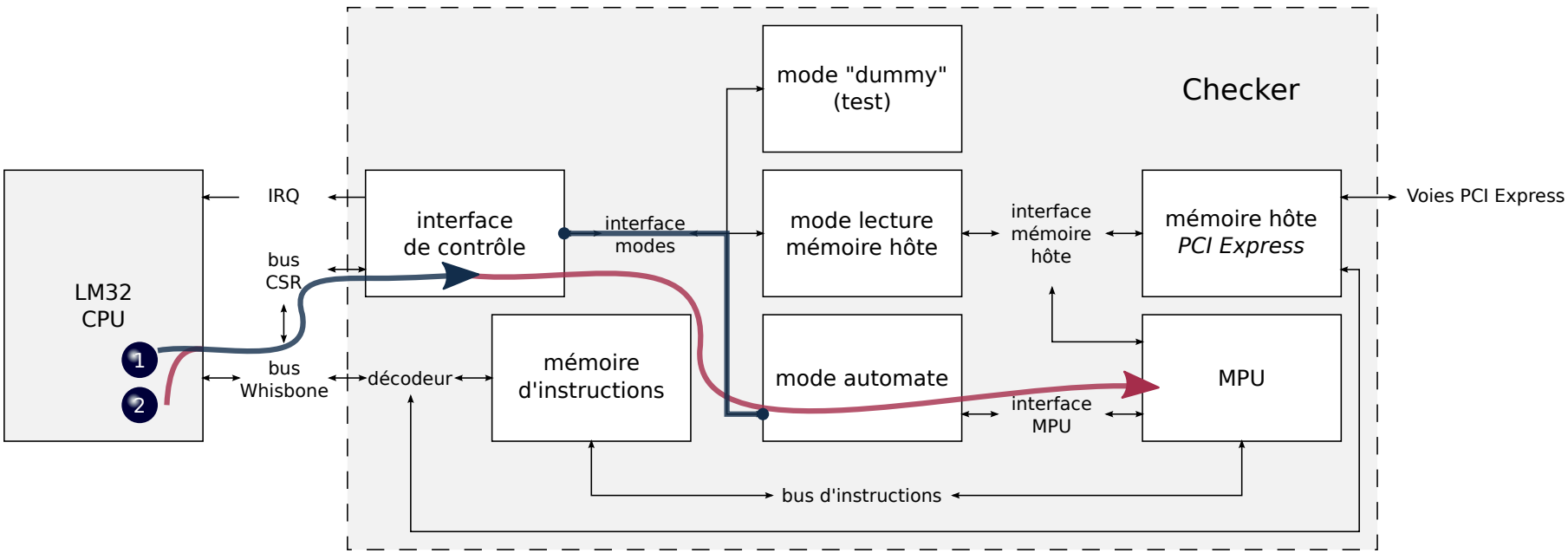
```
start:
  loadw r2, $0x1000 // Maximum page offset
  loadw r3, $0x3 // Maximum expected Hamming distance : 2 + 1
  loadd r6_0, $0xfffff000 // Bit select for Hamming distance
  loadd r6_1, $0x0000000f
  loadw r7, $no_ept_end
  loadw r8, $entry
  loadw r9, $ept_end
  loadb r10, $0x8 // Increment
entry:
  infw r1, r2, r9 // Loop stop condition
  movq r5, r4 // Remember the last entry
  mloadq r4, r1 // Load the current entry
  hammq r10, r4, r5, r6 // Compute Hamming distance
  infb r10, r3, r7 // Is Hamming < 3 ?
  add r1, r1, r10 // Go to next entry
  jmpw r8 // Loop
ept_end:
  intq r3 // Notifies that EPT 4k page table entry is found
no_ept_end:
  intq r0 // intq $0x0, end of execution
```

Étape 1 : initialisation des composants



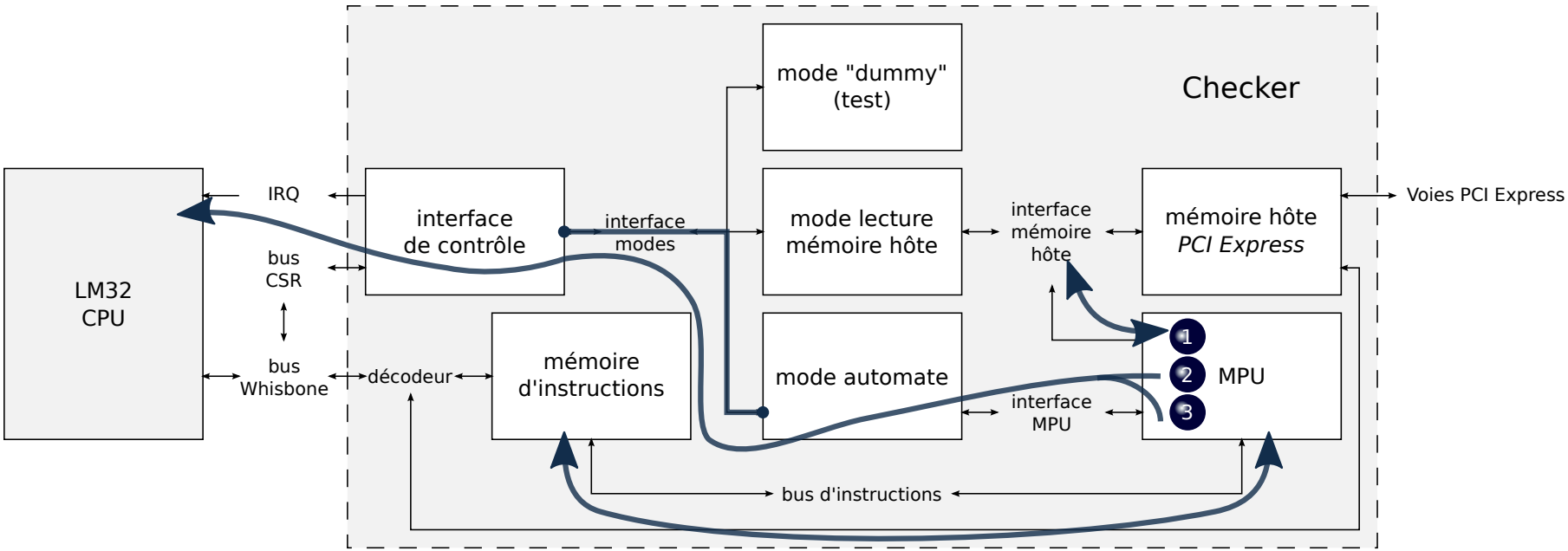
1. Téléchargement du binaire du MPU : motifs mémoire.
2. Sélection du mode d'exécution.

Étape 2 : Configuration de la page à analyser



1. Configuration de l'adresse de la page.
2. Démarrage du checker en mode automate.

Étape 3 : exécution et interruptions



1. Accès mémoire hôte.
2. Interruptions utilisateur.
3. Fin d'exécution du MPU.

Plan

Développements

Composant	État
UART	✗
Ethernet	✗
Checker	✗
MPU	✗
Binutils MPU	✗
Mémoire Hôte	

Automate	État
Caractérisation d'un hyperviseur	
Construction d'empreintes	
Superposition des empreintes	

Perspectives

- Finalisation du composant de réception.
- Preuves de concepts sur différentes attaques.

- Détection de malwares ?
- Sniffer PCI Express ?

Tests d'intégrité d'hyperviseurs de machines virtuelles à distance et assisté par le matériel

Benoît Morgan, **Éric Alata**, Vincent Nicomette

LAAS-CNRS, INSA Toulouse

14 juin 2014