FROM RESEARCH TO INDUSTRY

cea tech

Secure data processing: Blind Hypervision

P. Dubrulle, R. Sirdey, E. Ohayon, P. Dore and M. Aichouch

CEA LIST

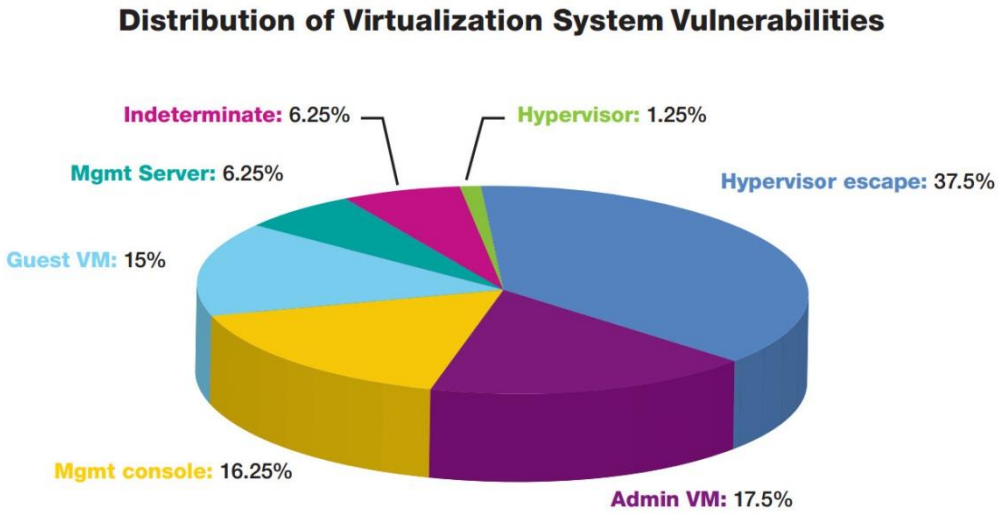
Contact : *paul.dubrulle@cea.fr*

www.cea.fr

leti & list

Introduction

- Well over a third of virtualization vulnerabilities reside in the hypervisor



Source: IBM X-Force Trend and risk report, 2010

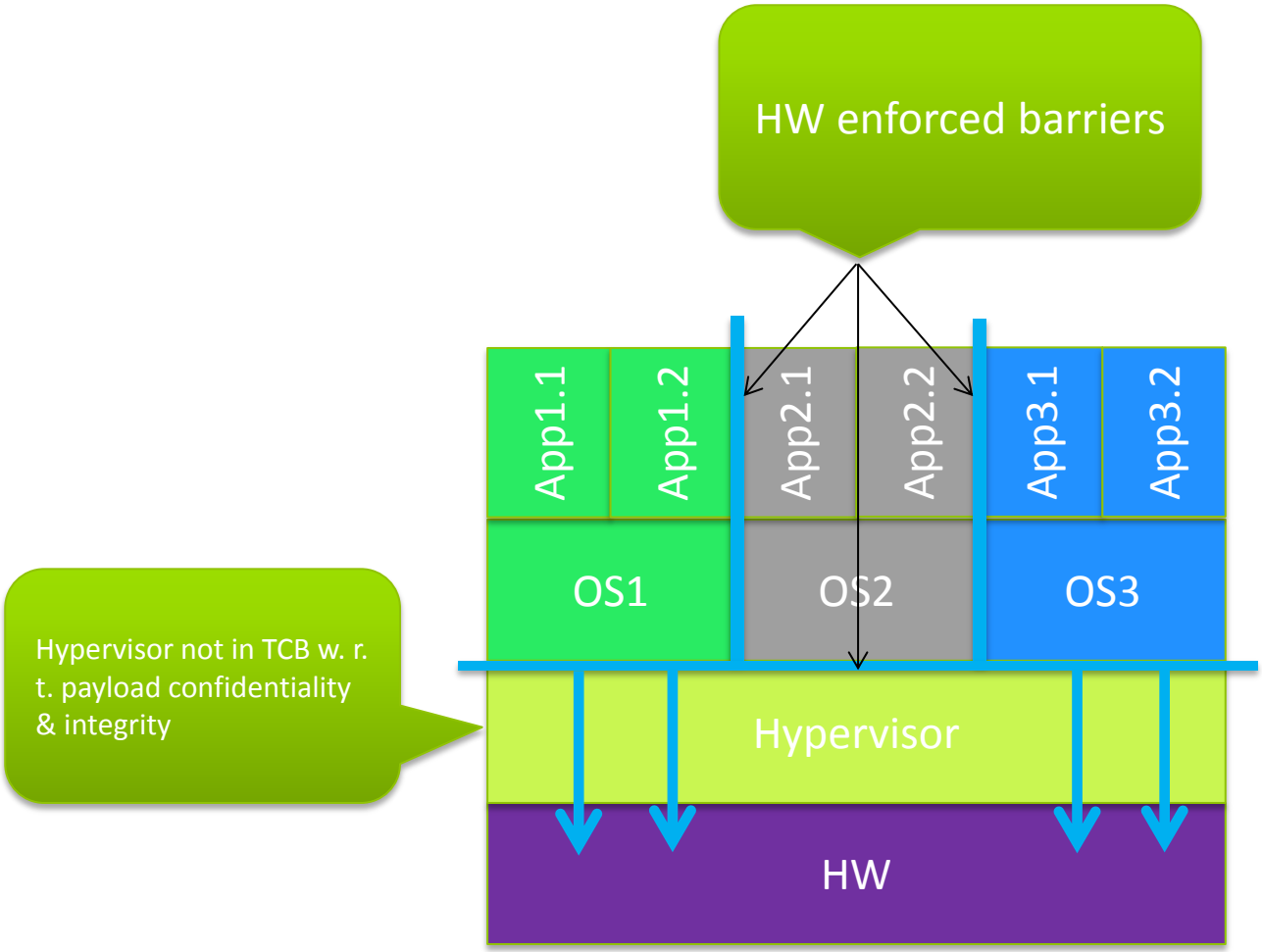
- How to protect VM data privacy from hypervisor escape attacks?

Existing solutions

- Virtual isolated processor runs critical code in complete isolation [ARM,2009]
  - Guarantee security properties for some sporadic critical operations
  - Cannot run a whole VM
- Root security: administrator is denied the right to read or write the sensitive data stored on the host
  - Hypervisor in Trusted Code Base [Garfinkel,2003]
    - No impact on performance
    - Requirement hard to maintain
  - Hardware enforced with eXecute-Only-Memory (XOM) [Lie,2003]
    - Slowdown due to on-the-fly encryption
    - Low software requirements

Proposed approach: Blind Hypervision

- A hardware/software co-design to enforce data security for VMs sharing a multi-core processor



Blind hypervision overview

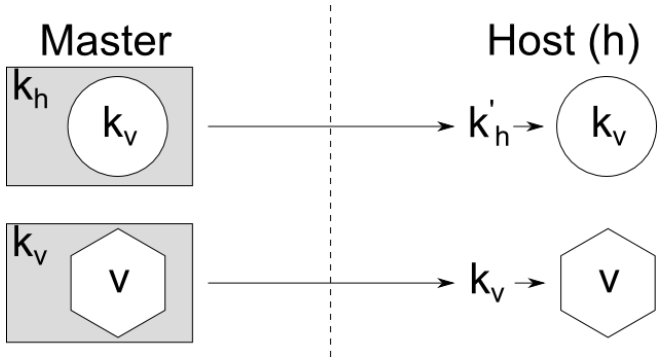
- Blind Hypervision to warrant SW code and data privacy
  - Strong memory isolation prevent VM/Hypervisor accessing other VMs or the hypervisor itself
  - Protect against privilege escalation attacks
  - Do not rely on trusted hypervisor
- No impact on performance:
  - Both code and data are stored in clear text
  - No on-the-fly encryption required
  - Ciphering takes place during VM loads/migrations only
- Protection scope:
  - SW attacks from within loaded code including hypervisor
  - SW attacks through equipment interfaces
  - Do not address probing or other physical attacks

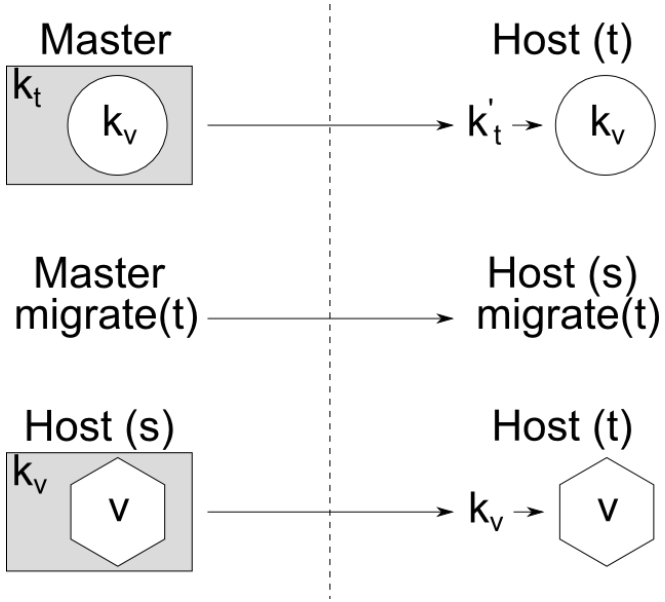
Blind hypervision principles

- A two-level Secured MMU (S-MMU)
  - Level 1: MPU partitioning memory – not configurable by Hypervisor/Guest
  - Level 2: MMU for partition management – context based, configurable only from partition owner
- A combined DMA and encryption engine (S-DMA) for VM load/migration
  - VM images deciphered using their own symmetrical key (K_v)
  - Critical data (K_v and CRC) deciphered using the host's public key K_h (protected register of the S-DMA)
- Tailored processor modes rather than hierarchical ones
  - Three legacy levels refined:
    - HYPERVISOR: S-MMU1/S-MMU2/S-DMA
    - VM-SUPERVISOR: S-MMU1/S-MMU2/S-DMA
    - VM-USER: S-MMU1/S-MMU2/S-DMA
  - One new level added:
    - INITIALIZATION: S-MMU1/S-MMU2/S-DMA

Deployment architecture

- A secured master
  - Cipher VMs with its symmetrical K_v
  - Cipher VM critical info with public K_h of target host (VMK, CRC)
  - Introduce VMs or request migrations
- One or more Blind Hypervision Host(s)



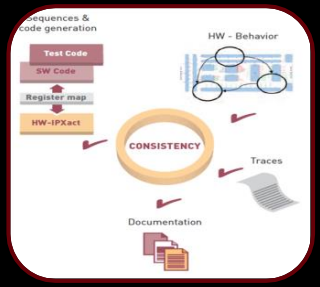
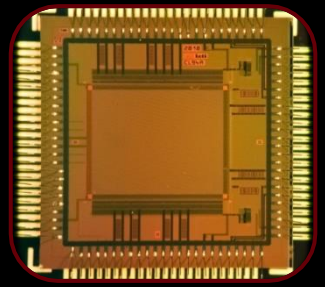


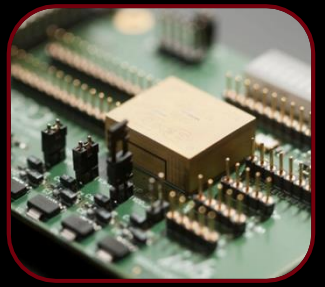
HW/SW trade-offs

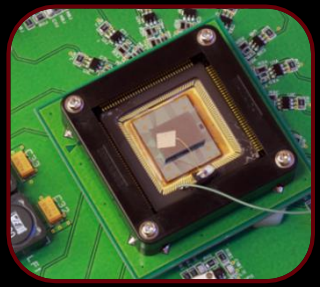
- Depending on the architecture starting point:
  - S-DMA as TCB could be SW residing in isolated virtual processor (e.g. ARM Trustzone)
  - S-MPU as a specialized Mondrian memory protection
  - Processor modes require HW assistance
- Deciding on how some techniques will be implemented depends on:
  - The already available security engine
  - The silicium cost
  - The SW complexity and validation cost

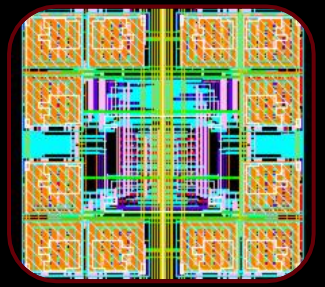
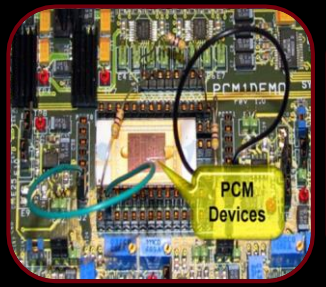
Conclusion and perspectives

- Blind Hypervision removes the on-the-fly encryption usually used to achieve root security
  - Preserved security level
  - Should perform better
- Prototype implementation will be realized to evaluate HW/SW trade-offs and performance gain
  - ARM based prototype using TrustZone
  - µServer based using Kalray MPPA

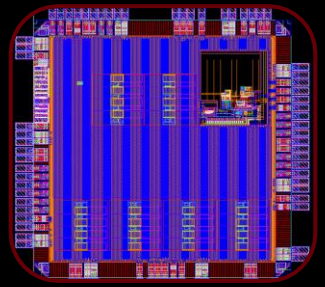
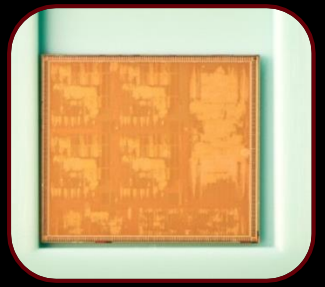
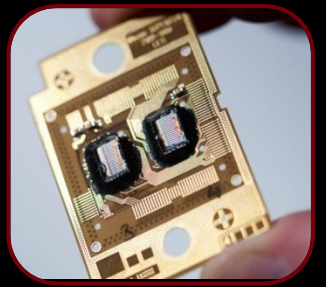
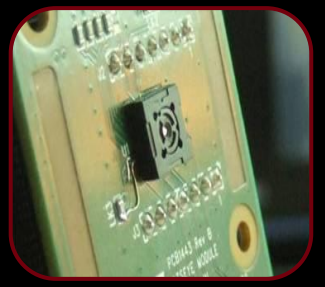
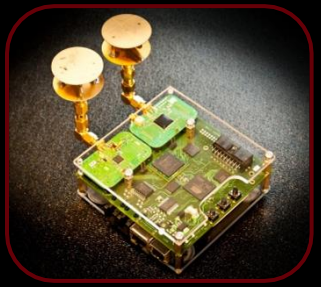
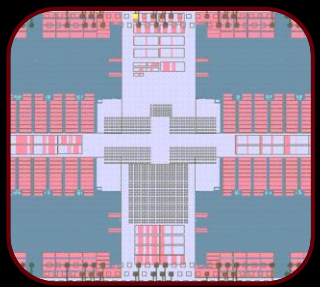
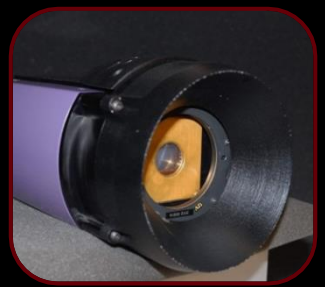

Sequences &
code generation
HW - Behavior
Test Code
SW Code
Register map
HW-IPXact
CONSISTENCY
Traces
Documentation



cea leti


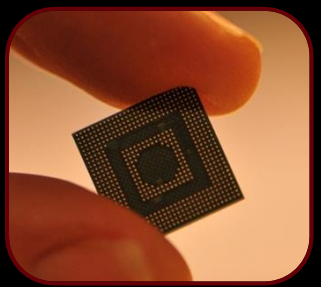


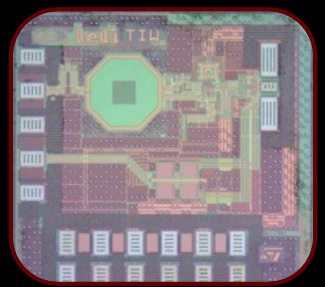
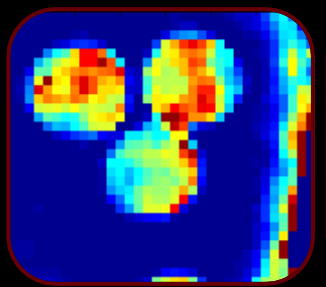


PCM
Devices








CEA-LETI
GENEPY
S65C12_1


Low-level Secured MPU (S-MPU)

- As soon as it has been configured memory is shown as statically partitioned:
  - One partition for the hypervisor
  - One partition for each possible VM
  - No overlay between partitions except for IO sub system
- S-MPU behavior:
  - Only the active partition is accessible by the CPU at a given time
  - Perform address translation to show up partition starting at @0
  - Regular MPU/MMU operate on top of S MPU within the active partition

Processor modes (1/2)

- The three legacy modes require to be slightly reworked:
  - The legacy VM-USER, VM_SUPERVISOR operate in the usual way except:
    - S-MPU restricts memory accesses to the VM partition
    - S-DMA usage is not allowed
    - Switching to/from VM MONITOR mode behavior reworked to respect privacy
  - VM-Monitor (hypervisor) mode:
    - S-MPU insure hypervisor runs in its dedicated partition
    - S-DMA usage available in this mode only
    - Only VM SUPERVISOR is reachable from VM_MONITOR one

Processor modes (2/2)

- System booting and initialization required adding a 4th mode called INITIALIZATION:
  - Default mode after processor reset:
    - Memory shall be cleared during reset handling
    - S MPU disabled with no remaining data in its registers
    - S-DMA unavailable
  - Dedicated to system init:
    - IO setting
    - S-MPU configuration (HYPERVISOR and VMs partition sizes)
    - Amount of VMs depend on assigned partition size
  - One-shot transition to VM MONITOR mode:
    - Done through a dedicated instruction which result in hypervisor start
    - Activate security features (S-MPU and S-DMA)
    - No preloaded VM allowed

Processor modes allowed transitions

Processor modes		Trigger	S-MPU	S-DMA
From	To			
INITIALIZATION	VM_MONITOR	LaunchHV	Enabling	Available
VM_MONITOR	VM_SUPERVISOR	LaunchVM	Switch to VM partition	Unavailable
VM-SUPERVISOR	VM-USER	OS operation	Unchanged partition	Unavailable
VM-USER	VM-SUPERVISOR	OS operation	Unchanged partition	Unavailable
VM-USER	VM_MONITOR	Yield	Switch to hypervisor partition	Available
VM-SUPERVISOR	VM_MONITOR	Yield	Switch to hypervisor partition	Available
*	INITIALIZATION	Reset	Disabling	Unavailable

Mode switching instructions

- Three instructions are dedicated to processor mode switching:
  - LaunchHV:
    - Instruct the processor to behave in a secure manner
    - Returning to a non-secure behavior require a reset action
    - Enable the VM MONITOR partition and start the hypervisor
  - LaunchVM:
    - Allow a VM monitor to Switch to a given VM without accessing its data
    - Ensure no data exchange between HV and VM
    - Prevent switching to a fake VM (uninitialized VM partition)
  - Yield (include HV preemption through unmaskable interrupt):
    - Switch back to HV from a running VM
    - Prevent data exchange between VM and HV
    - Ensure switching success

Switching step by step

- Switching from one processor mode to another looks like this sequence:
  - Current context saving
  - Cache flush and invalidation
  - Processor context wipe out
  - Memory partition switching
  - Processor mode switching
  - Context restoration
- Remarks:
  - Only mode switching involving security follow this scheme
  - Context saving has to be trusted (either hardwired or implemented through incorruptible code)

Privacy and VM load/export

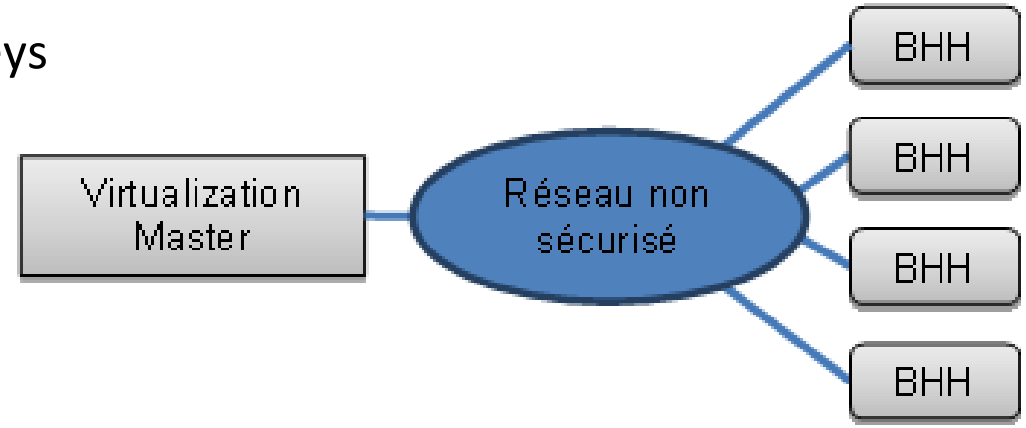
- Encryption mechanisms will be used to ensure privacy during VM loading and migration:
  - VM images will be ciphered using their own symmetrical key:
    - Do not involve whole image encryption prior to download
    - Ensure image efficient deciphering at Blind Hypervision Host (BHH)
  - Critical data (VM cipherKey and CRC) will be ciphered using the public BHH key:
    - Complex ciphering/deciphering operations take place on small data
    - BHH private key is accessible by the S-DMA only

VM monitoring API

- The Blind Hypervision Host (BHH) supplies an API to manipulate VM without accessing them directly.
- Most important primitive are:
  - vmlNewContext(cipherKey) to get an empty VM partition
  - vmlWriteContext(id, data, size): decipher and write a chunk of VM data into a partition
  - vmlCloseContext(id, CRC): ends VM loading and perform integrity checks
- Remarks:
  - VM cipherKey and CRC are ciphered using the BHH public key
  - VM image data are ciphered using the symmetrical cipherKey
  - The above API could be extended to support VM migration

Example of deployment architecture

- Such IT consists of:
  - A BHH set insuring privacy of executed VMs
  - A VM Master responsible for VM image storage and VM to BHH assignment
- Only the VM master has to be trusted because of its access to:
  - VM images
  - VM ciphering keys
  - BHH public ciphering keys



Application to other architectures

- Those security techniques could be of interest on architectures like:
  - DSP where a compromised boot loader may get access to valuable IP
  - µ-server capable to run one VM only
  - Manycores where a VM will be assigned to one tile/cluster
- Those architectures shall be easier to custom for security:
  - DSP or µ-server will rely on less processor modes
  - Manycore will rely on natural cluster/tile isolation for S-MPU

HW/SW trade-offs

- This secure device relies on several techniques. Depending on the architecture starting point some of them could be implemented in SW:
  - S-DMA as TCB could reside in Trustzone
  - S-MPU could be, but interaction with legacy MPU/MMU may be tricky
  - INITIALIZATON processor mode specificities requires HW assistance
- Deciding on how some techniques will be implemented depends on:
  - The already available security engine
  - The silicium cost
  - The SW complexity and validation cost