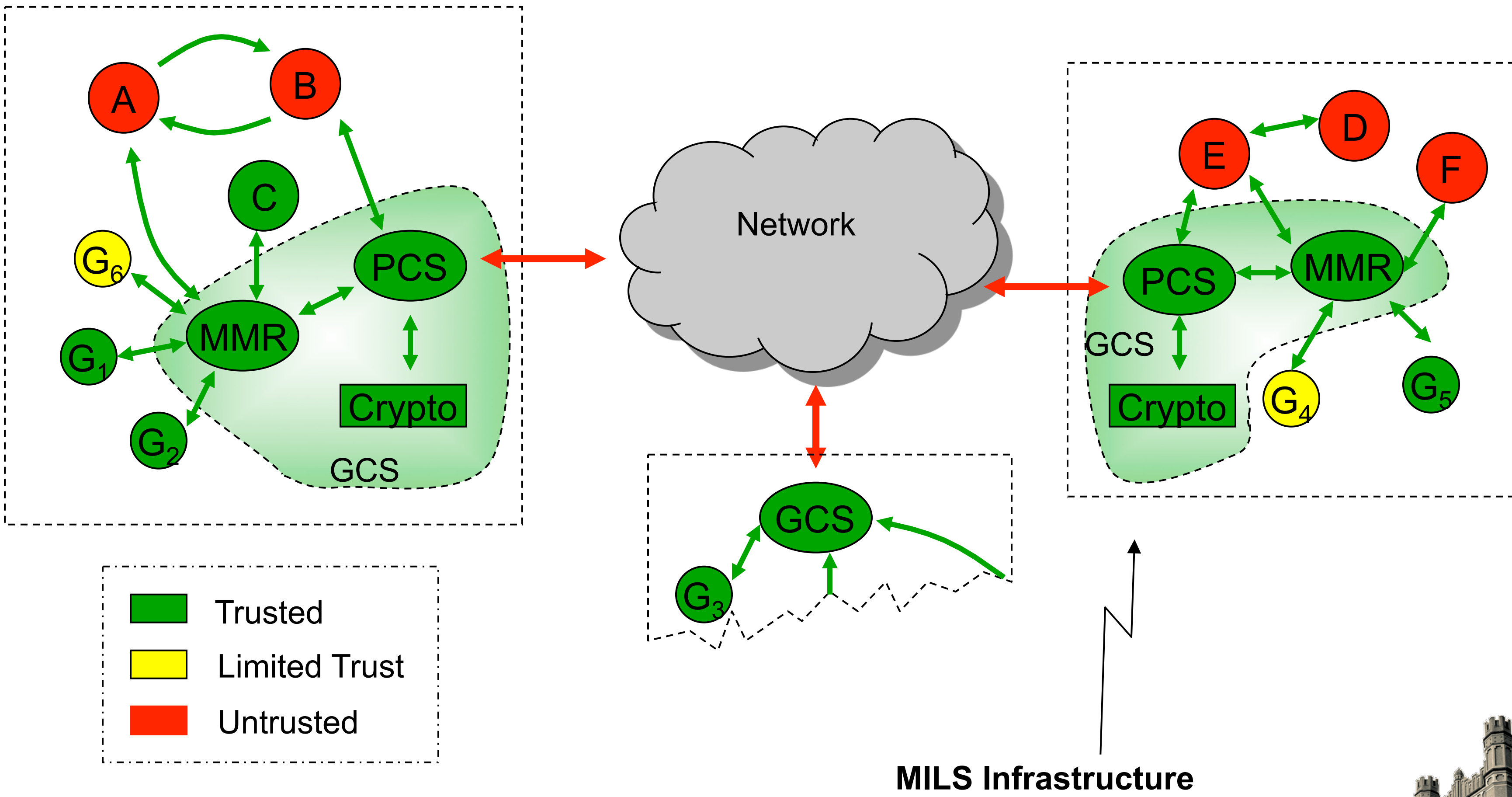
Layered Assurance Scheme for Multi-Core Architectures

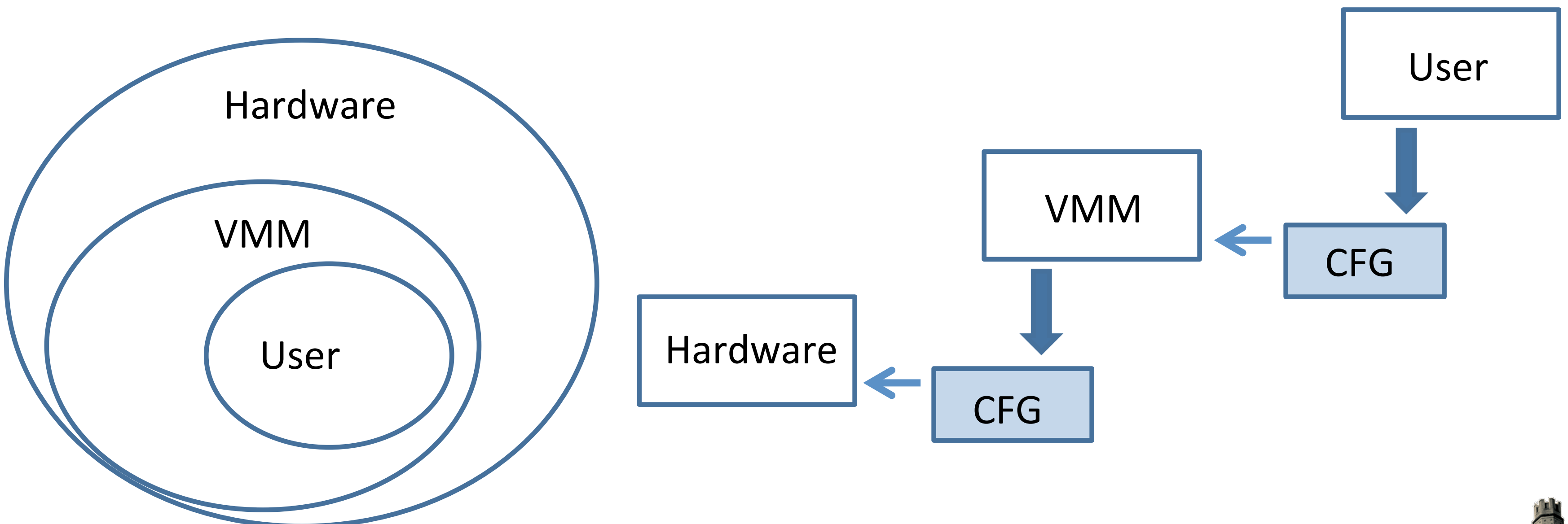
J. Alves-Foss, X. He and J. Song
Center for Secure and Dependable Systems
University of Idaho
jimaf@uidaho.edu,[xhhe,song3202]
@vandals.uidaho.edu

MILS Architecture



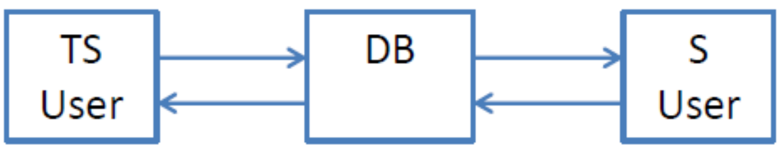
A 3-Level Layered Framework

- Identify and examine multi-core hardware features;
- Decompose security policy in VMM level into pieces of components that can be mapped into hardware level;
- Verify that VMM- and HW- level security policy satisfies user-level security requirements.

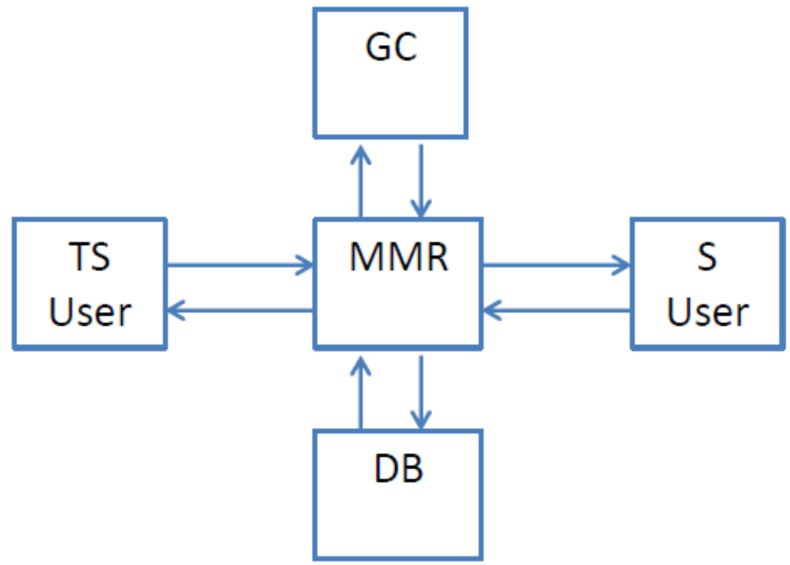


3-Level framework for secure multi-core systems

A 3-Level Layered Framework



(a) Highest Level View



(b) Application Level Security View

HW-Level Security Mechanisms

- Protection Rings
- Instructions (VM Exits)
- Memory Virtualization (EPT & VPID)

- Covert Channels Analysis
  - Processor Caches (CR0.CD)
  - Registers (TSS, MOV_DR)
  - Instructions (UD2)

VMM-Level Security Mechanisms

	VM Memory	*VMM Data*	*VMM Code*
VMX Non-root Operation	RWX	W	-
VMX Root Operation	RWX	RW	RX

Table 6.1: Protection page table enforced on IAVMM

VMM configures underlying hardware mechanisms to provide separation between VMs and protection of VMM

User-Level Security Policy

- Access control security
  - Bell_LaPadula Model: No “read-up, write-down” policy

- Information flow security
  - Some security properties
    - Non-interference -- defined interference by viewing changes in outputs in an event system model
    - Separability is an example of perfect security, but too strong
  - A weakest security property, Perfect Security Property (Zakinthinos,1997)
    - Allow high-level outputs to be dependent on low-level events, but low-level user still will not know how he has influenced high-level outputs.

Perfect Security Property

- For any low level observation:
  - All interleavings of high level input sequences must be possible;
  - High level outputs can be inserted anywhere in the trace and can depend on low level activity.
- PSP Equation

$$\forall \tau : traces(S) \cdot \tau|L \in LLES(\tau, S) \wedge \forall p, s : p\hat{\ }s \in LLES(\tau, S) \wedge s|H = \langle\rangle \cdot \forall \alpha : H \cdot p\hat{\ }\langle\alpha\rangle \in traces(S) \Rightarrow p\hat{\ }\langle\alpha\rangle\hat{\ }s \in LLES(\tau, S)$$

  - wherein

$$LLES(\tau, S) = \{s|\ \tau|L = s|L \wedge s \in traces(S)\}$$

Formal Model of VMS

Definition 7.1. The formal model of a state machine M is defined as:

- $M = \langle \Sigma, \sigma_0, T \rangle$
- Σ is the set of states of the system
- Initial State: $\sigma_0 \in \Sigma$
- $T : \Sigma \rightarrow \Sigma$ defines the allowed transitions between states.
- The notation $\sigma(p)$ denotes the substate of σ that corresponds to the named resource, p, in the system.

Formal Model of VMS

Definition (composite state machines):

- $M = (M^1, M^2, \ldots, M^n)$ n-tuple representing the individual state machines in the composite machine, where $M^i = \langle \Sigma^i, \sigma_0^i, T^i \rangle$
- $\forall \sigma \in \Sigma : \sigma = cs(\sigma^1, \sigma^2, \ldots, \sigma^n)$ where $\sigma^i \in \Sigma^i$
- Initial State: $\sigma_0 = cs(\sigma_o^1, \sigma_o^2, \ldots, \sigma_o^n)$,
- The notation $cs(s_1, \ldots, s_n)$ denotes the composite state of the system.
- The extraction function $\mathcal{S}^i(\sigma) = \sigma_i$ returns the portion of the composite state relevant to sub-machine i.
- $T(\sigma) = cs(\tau^1(\mathcal{S}^1(\sigma)), \tau^2(\mathcal{S}^2(\sigma)), \ldots, \tau^n(\mathcal{S}^n(\sigma)))$ where $\tau^i \in T^i$

Formal Model of VMS

Definition (composite state machines):

- State Machine Policy 1:
  - The intersection of substates must be restricted such that execution of τ^i does not interact with substate σ^j in violation of the security property.

- State Machine Policy 2:
  - If the execution of τ^i modifies a component of substate σ^j ($j \neq i$), then the transition τ^i in τ must also specify that modification.

Event System Model

- $\mathcal{E} = \{(a, s, r, w) \mid a \in \mathcal{A}, s \in \mathcal{S}, r, w \in \mathcal{P}(\mathcal{O})\}$ is the set of events of the system.
- $\mathcal{S}$ is the set of subjects
- $\mathcal{O}$ is the set of objects and $\mathcal{P}(\mathcal{O})$ is powerset (set of subsets) of O.
- r and w are two (not-necessarily disjoint) subsets of objects that are accessed by action a; the *read-set* and the *write-set*.
- $\mathcal{E}$ is the set of events, where events correspond to state transition chains

  let $\tau = \tau_0, \tau_1, \ldots, \tau_n \in T^*$ be a sequence of state transitions corresponding to event $e = (a, s, r, w)$ such that:

$$\tau(x) = \tau_n(\tau_{n-1}(\ldots \tau_1(\tau_0(x)) \ldots))$$

  let σ be the state of the system prior to execution of event e and $\sigma' = \tau(\sigma)$ be the state after execution of τ.

Event System Model

- Events are classifed as *atomic* or *synchronizing*

The formal model for events in a composite system are:

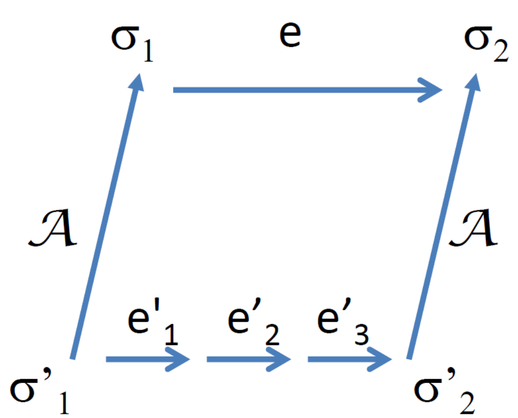
- $\mathbb{E} \subseteq \mathcal{P}(E)$. At any time there may be any number of "active" events in the system.
- Let $\overline{e} \in \mathbb{E}$ be a set of active events, and $e_i, e_j \in \overline{e}$ be two different active events.
  - If e_i and e_j are atomic events, then $(e_i.r \cap e_j.w) = (e_i.w \cap e_j.w) = (e_i.w \cap e_j.r) = (e_i.r \cap e_j.r) = \emptyset$
  - If e_i is a synchronizing event, and $(e_i.r \cap e_j.w) \neq \emptyset \vee (e_i.w \cap e_j.w) \neq \emptyset \vee (e_i.w \cap e_j.r)$ then e_j and e_i must be *partner synchronizing events.*

Event Policy 1: $\forall o \in \mathcal{O} : o \notin w \Rightarrow \sigma'(o) = \sigma(o)$

Event Policy 2: $\forall \sigma_1, \sigma_2 \in \Sigma : (\forall o \in r \cup w : \sigma_1(o) = \sigma_2(o)) \Rightarrow (\forall p \in w : \sigma_1'(p) = \sigma_2'(p))$

Event System Model

- A layered approach to using the event model:
- $\mathcal{A}$ is a abstraction function that maps the state of the lower level to the higher level of abstraction.



$$\tau(\sigma_1) = \sigma_2 \quad \wedge \quad (\tau'_3 \circ \tau'_2 \circ \tau'_1)(\sigma'_1) = \sigma'_2$$

$$\mathcal{A}(\sigma'_1) = \sigma_1 \quad \wedge \quad \mathcal{A}(\sigma'_2) = \sigma_2$$

Event System Model

- An exemplary 3-level layered assurance:
  - HW layer
    - Execution mode and available resources
  - VMM layer
    - Authorize a set of the allowable states
  - Application/User layer
    - The highest layer of abstraction, user view of the world

HW Layer

- Supports the concepts of execution in a context.
  - defined as the execution mode (e.g., supervisor/user, privilege ring, VM status) and the set of available resources (e.g., the memory maps in the MMU).
  - mechanisms to set and change the configurations of contexts, and to perform context changes.
- Subjects mapped to the contexts
- Events are bound to the current executing subject of the hardware.
  - In a multi-core model, there may be multiple subjects, one running on each logical processor, or on a collection of processors.
- The objects of the hardware are the physical resources of the hardware,
  - memory, devices, registers, the MMU, etc.
- Exports a model of an executing set of systems, the individual logical processors, and current executing contexts.

HW Layer

- Hw security policy does not directly map to the concepts of high and low-level users
- However, we can map security policies to allowable sequences of events
  - $e_1^{hw}, e_2^{hw} \ldots , e_3^{hw}$
  - Each will be sequence of events for current contexts, or the context switch events.
- Set of traces will only contain events that are allowable within the current contexts.
  - If supports virtual memory and MMU-based memory maps, the events will correspond to available virtual memory accesses and MMU maps.
- Verification of the correct behavior of the system includes verification that the hardware supports the configuration data and does not violate the contexts.
- Security property must clearly specify the limitations of the HW execution model and configuration.
- We should not attempt to model security levels, or user intent at this level, just the correct implementation of the configuration data.

VMM Layer

- The VMM layer is responsible for defining the contexts of the hardware, and thus will only authorize a subset of the allowable states with a more restrictive policy.
- The hardware provides the basic security mechanisms of isolation:
  - virtualization, virtual memory, memory management, and context switching. It also supports multiple execution units.
- The hardware supports execution of each logical CPU core in either root-mode or in a guest (virtual machine) mode.
- The VMM configures:
  - the memory maps,
  - assigns usage of the cores to the VMs
  - Schedules VMs
  - manages transitions in and out of virtualization mode.

VM Layer

- Events of the VMM correspond to actions of the individual VMs
  - Control events of the VM (configuring the hardware, establishing the VM contexts).
  - will mostly still be at the same granularity as the hardware level, with additions for VMM specific actions (e.g., creation of a VM, swapping in/out a VM).
- Subjects in the VMM are now mapped to individual VMs and the VMM
- Objects of the system are still mostly the hardware resources, but also now the VMM data structures representing:
  - context's of the VMs, possible buffers and other internal resources.
  - We need a mapping of these objects onto the hardware and a mapping of the VMM-specific events into hardware events.
- VMM exports a model of an executing set of virtual systems, individual VMs, and current executing contexts for those VMs.
- We still can not model security levels, but
  - we have now separated the hardware (time and space) into VM contexts
  - have VMM rules for behavior of those VMs
  - Need to verify VMM satisfies configuration

Application Layer

- System designer "draws" interaction graph, and defines rules for communication.
- Designer refines implementation down to individual components
  - each mapped to a VM
  - authorized communication between entities specified using VMM configuration
- Subjects, Objects and Events are now those seen defined in the top-level security policy
  - Need mappings between these abstract entities and their implementations in the VMM and HW
- Verification requires
  - validation of the mappings
    - Can assume separation exists, if lower level exports separation policy
  - validation of design (access control/authorized communication)

Layered Framework Conclusion

- A layered assurance scheme for secure multi-core architectures and formalize security policy.
  - A 3-level layered framework for secure multi-core architectures.
  - Formalize security policies, specify and verify a layered assurance scheme for multi-core architectures.