A Case study on Formal Verification

CFPTT 2022

InCo, Facultad de Ingeniería, Universidad de la República, Uruguay

Montevideo, Uruguay, November 2022

Outlines

- Background and Motivation
- Part I: An Overview on Virtualization
- Part II: Verification of Security Models and Security Policies
- Part III: Isolation and Availability in an idealized model of virtualization

Background: OS Verification

- OS verification since 1970
- Tremendous advances in proof technology
- PL verification is becoming ubiquitous: OS verification is the next frontier
- Flagship projects:
    - L4.verified: formal verification of seL4 exokernel (G. Klein et al, NICTA)
    - Hyper-V: formal verification of Microsoft hypervisor (E. Cohen et al, MSR)

Motivation and challenge

- Main focus of L4.verified and Hyper-V on functional correctness
- But non-functional properties are equally important
  - Confidentiality and Integrity
    - Virtualization platforms must ensure isolation
    - Security evaluations (CC)
  - Availability
    - Virtualization platforms must respect availability constraints
    - Certification bodies (DO178)
- Beyond safety properties
  - Isolation properties are 2-safety properties
  - Availability properties are liveness properties

VirtualLogix (now Red Bend Software)

- Provided informal requirements at initial stages
- Suggested focus on Xen-like paravirtualization platforms

Part I

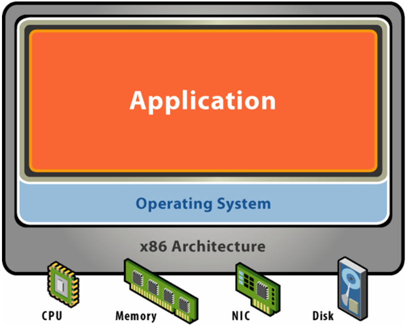
An Overview on Virtualization [Waldspurger 2007]

What is virtualization?

In computing, is a broad term that refers to the abstraction of computer resources

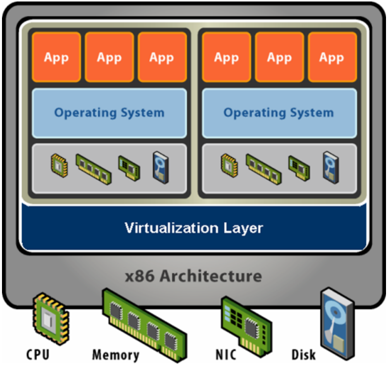
- Virtual systems
  - Abstract physical components using logical objects
  - Dynamically bind logical objects to physical configurations
- Examples
  - Network: Virtual LAN (VLAN), Virtual Private Network (VPN)
  - Storage: Storage Area Network (SAN), LUN
  - Computer: Virtual Machine (VM), simulator

Starting point: A Physical Machine



- Physical Hardware
  - Processors, memory, chipset, I/O bus and devices, etc.
  - Physical resources often underutilized
- Software
  - Tightly coupled to hardware
  - Single active OS images
  - OS controls hardware

What is a Virtual Machine?



- Hardware-Level Abstraction
  - Virtual hardware: processors, memory, chipset, I/O devices, etc.
  - Encapsulates all OS and application state
- Virtualization Software
  - Extra level of indirection decouples hardware and OS
  - Multiplexes physical hardware across multiple *guest* VMs
  - Strong isolation between VMs
  - Manages physical resources, improves utilization

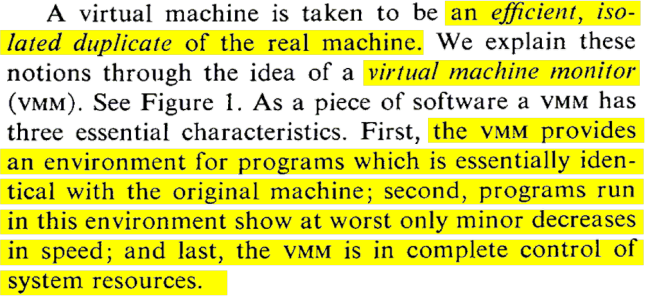
VM Isolation

- Secure Multiplexing
  - Run multiple VMs on single physical host
  - Processor hardware isolates VMs, e.g. MMU
- Strong Guarantees
  - Software bugs, crashes, viruses within one VM cannot affect other VMs
- Performance Isolation
  - Partition system resources
  - Example: VMware controls for reservation, limit, shares

Common Virtualization Uses Today

- **Test and Development** - Rapidly provision test and development servers; store libraries of pre-configured test machines
- **Server Consolidation and Containment** - Eliminate server sprawl by deploying systems into virtual machines that can run safely and move transparently across shared hardware
- **Business Continuity** - Reduce cost and complexity by encapsulating entire systems into single files that can be replicated and restored onto any target server
- **Enterprise Desktop** - Secure unmanaged PCs without compromising end-user autonomy by layering a security policy in software around desktop virtual machines

What is a Virtual Machine Monitor?



A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine. We explain these notions through the idea of a *virtual machine monitor* (VMM). See Figure 1. As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

- An Old Concept
  - Classic definition from [Popek and Goldberg 1974]
  - IBM mainframes since the 60s

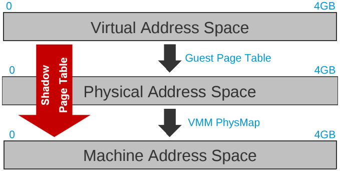
- VMM Characteristics
  - Fidelity
  - Performance
  - Isolation / safety

VMM Platform Types

- Hosted Architecture
  - Install as application on existing x86 *host* OS, e.g. Windows, Linux, OS X
  - Small context-switching driver
  - Leverage host I/O stack and resource management
  - Examples: VMware Player/Workstation/Server, Microsoft Virtual PC/Server, Parallels Desktop
- Bare-Metal Architecture
  - *Hypervisor* installs directly on hardware
  - Acknowledged as preferred architecture for high-end servers
  - Examples: VMware ESX Server, Xen, Microsoft HyperV

Virtualized Address Spaces

With shadow page tables



- Traditional VMM Approach
- Extra Level of Indirection
  - *Virtual* → *Physical*: Guest maps VAS to PAS using primary page tables
  - *Physical* → *Machine*: VMM maps PAS to MAS
- Shadow Page Table
  - Composite of two mappings
  - For ordinary memory references Hardware maps VAS to MAS
  - Cached by physical TLB

What is Paravirtualization?

- Full Virtualization
  - No modifications to guest OS
  - Excellent compatibility, good performance, but complex
- Paravirtualization Exports Simpler Architecture
  - Term coined by Denali project in 2001, popularized by Xen
  - Modify guest OS to be aware of virtualization layering (Hypercalls)
  - Remove non-virtualizable parts of architecture
  - Avoid rediscovery of knowledge in hypervisors
  - Excellent performance and simple, but poor compatibility

Hypervisors

- Allow several operating systems to coexist on commodity hardware
- Provide support for multiple applications to run seamlessly on the guest operating systems they manage
- Provide a means to guarantee that applications with different security policies can execute securely in parallel
- They are increasingly used as a means to improve system flexibility and security

Hypervisors are a priority target of formal specification and verification

Hypervisors

- Allow several operating systems to coexist on commodity hardware
- Provide support for multiple applications to run seamlessly on the guest operating systems they manage
- Provide a means to guarantee that applications with different security policies can execute securely in parallel
- They are increasingly used as a means to improve system flexibility and security

Hypervisors are a priority target of formal specification and verification

Hypervisors

- Allow several operating systems to coexist on commodity hardware
- Provide support for multiple applications to run seamlessly on the guest operating systems they manage
- Provide a means to guarantee that applications with different security policies can execute securely in parallel
- They are increasingly used as a means to improve system flexibility and security

Hypervisors are a priority target of formal specification and verification

Hypervisors

- Allow several operating systems to coexist on commodity hardware
- Provide support for multiple applications to run seamlessly on the guest operating systems they manage
- Provide a means to guarantee that applications with different security policies can execute securely in parallel
- They are increasingly used as a means to improve system flexibility and security

Hypervisors are a priority target of formal specification and verification

Hypervisors

- Allow several operating systems to coexist on commodity hardware
- Provide support for multiple applications to run seamlessly on the guest operating systems they manage
- Provide a means to guarantee that applications with different security policies can execute securely in parallel
- They are increasingly used as a means to improve system flexibility and security

Hypervisors are a priority target of formal specification and verification

Hypervisors

- Allow several operating systems to coexist on commodity hardware
- Provide support for multiple applications to run seamlessly on the guest operating systems they manage
- Provide a means to guarantee that applications with different security policies can execute securely in parallel
- They are increasingly used as a means to improve system flexibility and security

Hypervisors are a priority target of formal specification and verification

Part II

Formal verification of security policies

Security policies and security models

- Distinction between model and policy [Goguen and Meseguer 1982]
- A model describes the system
    - a high level specification or an abstract machine description of what the system does
    - that paper uses a state transition systems with focus on operations and outputs
- A security policy
    - defines the security requirements for a given system
    - Verification shows that a policy is satisfied by a system

An abstract system model

[Goguen and Meseguer 1982]

- A set of states S
- A set of subjects U
- A set of state commands SC
- A set of all possible outputs Out
- $do : S \times U \times SC \rightarrow S$ and $out : S \times U \rightarrow Out$ where
  - $do(s_i, u, c) = s_j$ means that at state s_i, when u performs command c, the resulting state is s_j
  - $out(s, u)$ gives the output that u sees at state s
- $s_0 : S$ is an initial state

Security Policies

[Goguen and Meseguer 1982]

Definition

A security policy is a set of noninterference assertions

Noninterference

Given two group of users G_0 and G_1, we say G_0 does not interfere with G_1 if for any sequence of commands $w, View\_G_1(w) = View\_G_1(P_{G_0}(w))$, where

- $View\_G_1(w)$ denotes what users in G_1 may observe after the execution of w
- $P_{G_0}(w)$ is w with commands initiated by users in G_0 removed.

Basic notions (revisited)

[Rushby 1992], [von Oheimb 2004]

- System model:
  - $\mathit{step} : \mathit{action} \times \mathit{state} \rightarrow \mathit{state}$
  - $\mathit{run} : [\mathit{action}] \times \mathit{state} \rightarrow \mathit{state}$
  - also nondeterministic variants
- Security model:
  - domain - secrecy level/area
  - $\mathit{obs} : \mathit{domain} \times \mathit{state} \rightarrow \mathit{output}$
  - $\mathit{dom} : \mathit{action} \rightarrow \mathit{domain}$ - input domain
- Policy or interference relation:
  - $\rightsquigarrow : \mathit{domain} \rightarrow \mathit{domain} \rightarrow \mathit{Prop}$
  - always reflexive, possibly intransitive
  - difference between confidentiality and integrity requirements is the direction in which security domains must not interfere.
- Noninterference relation: $\not\rightsquigarrow$ (the activities in source domain are confidential for the target domain)

Basic notions (revisited)

[Rushby 1992], [von Oheimb 2004]

- System model:
  - $step : action \times state \rightarrow state$
  - $run : [action] \times state \rightarrow state$
  - also nondeterministic variants
- Security model:
  - domain - secrecy level/area
  - $obs : domain \times state \rightarrow output$
  - $dom : action \rightarrow domain$ - input domain
- Policy or interference relation:
  - $\rightsquigarrow$: $domain \rightarrow domain \rightarrow Prop$
  - always reflexive, possibly intransitive
  - difference between confidentiality and integrity requirements is the direction in which security domains must not interfere.
- Noninterference relation: $\not\rightsquigarrow$ (the activities in source domain are confidential for the target domain)

Basic notions (revisited)

[Rushby 1992], [von Oheimb 2004]

- System model:
  - $step : action \times state \rightarrow state$
  - $run : [action] \times state \rightarrow state$
  - also nondeterministic variants
- Security model:
  - domain - secrecy level/area
  - $obs : domain \times state \rightarrow output$
  - $dom : action \rightarrow domain$ - input domain
- Policy or interference relation:
  - $\rightsquigarrow : domain \rightarrow domain \rightarrow Prop$
  - always reflexive, possibly intransitive
  - difference between confidentiality and integrity requirements is the direction in which security domains must not interfere.
- Noninterference relation: $\not\rightsquigarrow$ (the activities in source domain are confidential for the target domain)

Basic notions (revisited)

[Rushby 1992], [von Oheimb 2004]

- System model:
  - $step : action \times state \to state$
  - $run : [action] \times state \to state$
  - also nondeterministic variants
- Security model:
  - domain - secrecy level/area
  - $obs : domain \times state \to output$
  - $dom : action \to domain$ - input domain
- Policy or interference relation:
  - $\leadsto$: $domain \to domain \to Prop$
  - always reflexive, possibly intransitive
  - difference between confidentiality and integrity requirements is the direction in which security domains must not interfere.
- Noninterference relation: $\not\leadsto$ (the activities in source domain are confidential for the target domain)

Noninterference

Aim: secrecy of the presence/absence of actions

A definition

$$
\begin{aligned}
&\mathit{noninterference} \equiv \\
&\quad \forall\, \alpha\ u.\ \mathit{obs}(u, \mathit{run}(\alpha, s_0)) \;=\; \mathit{obs}(u, \mathit{run}(\mathit{ipurge}(u, \alpha), s_0))
\end{aligned}
$$

ipurge

$$
\begin{aligned}
\mathit{ipurge} \quad &: \quad \mathit{domain} \to [\mathit{action}] \to [\mathit{action}] \\
\mathit{ipurge}(u, [\,]) \quad &= \quad [\,] \\
\mathit{ipurge}(u, a :: \alpha) \quad &= \quad \mathit{if}\ \mathit{dom}(a) \in\ \mathit{sources}(a :: \alpha, u) \\
&\qquad \mathit{then}\ a :: \mathit{ipurge}(u, \alpha)\ \mathit{else}\ \mathit{ipurge}(u, \alpha)
\end{aligned}
$$

remove from the sequence α all actions that may not influence u, directly or via the domains of subsequent actions within α

sources

$\mathit{sources}(\alpha, u) =$ all domains of actions in α that may influence u, directly or via the domains of subsequent actions within α.

$$
\begin{aligned}
&v\ \in\ \mathit{sources}(a_1 :: a_2 :: a_3 :: a_4, u) \\
&\mathit{if}\ v\ =\ \mathit{dom}(a_2) \rightsquigarrow\ \mathit{dom}(a_4) \rightsquigarrow\ u\ (\mathit{even\ if}\ \mathit{dom}(v) \not\rightsquigarrow u)
\end{aligned}
$$

Noninterference

Aim: secrecy of the presence/absence of actions

A definition

$$\begin{aligned}&\mathit{noninterference} \equiv\\ &\quad \forall\, \alpha\ u.\ \mathit{obs}(u, \mathit{run}(\alpha, s_0)) \ = \ \mathit{obs}(u, \mathit{run}(\mathit{ipurge}(u, \alpha), s_0))\end{aligned}$$

ipurge

$$\begin{array}{lcl} \mathit{ipurge} & : & \mathit{domain} \rightarrow [\mathit{action}] \rightarrow [\mathit{action}] \\ \mathit{ipurge}(u, []) & = & [] \\ \mathit{ipurge}(u, a :: \alpha) & = & \mathit{if}\ \mathit{dom}(a) \in \ \mathit{sources}(a :: \alpha, u) \\ & & \mathit{then}\ a :: \mathit{ipurge}(u, \alpha)\ \mathit{else}\ \mathit{ipurge}(u, \alpha) \end{array}$$

remove from the sequence α all actions that may not influence u, directly or via the domains of subsequent actions within α

sources

$\mathit{sources}(\alpha, u) =$ all domains of actions in α that may influence u, directly or via the domains of subsequent actions within α.

$$\begin{array}{lcl} v & \in & \mathit{sources}(a_1 :: a_2 :: a_3 :: a_4, u) \\ \mathit{if}\ v & = & \mathit{dom}(a_2) \rightsquigarrow \ \mathit{dom}(a_4) \rightsquigarrow \ u\ (\mathit{even\ if}\ \mathit{dom}(v) \not\rightsquigarrow u) \end{array}$$

Noninterference

Aim: secrecy of the presence/absence of actions

A definition

$$\begin{array}{l} \mathit{noninterference} \equiv \\ \quad \forall\, \alpha\; u.\; \mathit{obs}(u, \mathit{run}(\alpha, s_0)) \;=\; \mathit{obs}(u, \mathit{run}(\mathit{ipurge}(u, \alpha), s_0)) \end{array}$$

ipurge

$$\begin{array}{lcl} \mathit{ipurge} & : & \mathit{domain} \to [\mathit{action}] \to [\mathit{action}] \\ \mathit{ipurge}(u, []) & = & [] \\ \mathit{ipurge}(u, a :: \alpha) & = & \mathit{if}\ \mathit{dom}(a) \in\ \mathit{sources}(a :: \alpha, u) \\ & & \mathit{then}\ a :: \mathit{ipurge}(u, \alpha)\ \mathit{else}\ \mathit{ipurge}(u, \alpha) \end{array}$$

remove from the sequence α all actions that may not influence u, directly or via the domains of subsequent actions within α

sources

$\mathit{sources}(\alpha, u) =$ all domains of actions in α that may influence u, directly or via the domains of subsequent actions within α.

$$\begin{array}{l} v\ \in\ \mathit{sources}(a_1 :: a_2 :: a_3 :: a_4, u) \\ \mathit{if}\ v\ =\ \mathit{dom}(a_2) \rightsquigarrow \mathit{dom}(a_4) \rightsquigarrow u\ (\mathit{even\ if}\ \mathit{dom}(v) \not\rightsquigarrow u) \end{array}$$

Noninterference

Observational relation

Observational equivalence/relation

Parameterized by an observing domain and induced by the *obs* function.

$$s \lhd \alpha \overset{u}{\approxeq} t \lhd \beta \equiv obs(u, run(\alpha, s)) = obs(u, run(\beta, t))$$

An alternative Definition

$$noninterference \equiv \forall\, \alpha\; u.\; s_0 \lhd \alpha \overset{u}{\approxeq} s_0 \lhd ipurge(u, \alpha)$$

- Noninterference is a global property of sequences of actions and state transitions
- To inductively reason on action sequences we need to derive conditions on individual state transitions

Noninterference

Observational relation

Observational equivalence/relation

Parameterized by an observing domain and induced by the *obs* function.

$$s \lhd \alpha \overset{u}{\approx} t \lhd \beta \;\equiv\; obs(u, run(\alpha, s)) \;=\; obs(u, run(\beta, t))$$

An alternative Definition

$$noninterference \;\equiv\; \forall\, \alpha\; u.\; s_0 \lhd \alpha \overset{u}{\approx} s_0 \lhd ipurge(u, \alpha)$$

- Noninterference is a global property of sequences of actions and state transitions
- To inductively reason on action sequences we need to derive conditions on individual state transitions

Noninterference

Observational relation

Observational equivalence/relation

Parameterized by an observing domain and induced by the *obs* function.

$$s \lhd \alpha \overset{u}{\approx} t \lhd \beta \equiv obs(u, run(\alpha, s)) = obs(u, run(\beta, t))$$

An alternative Definition

$$noninterference \equiv \forall\, \alpha\, u.\ s_0 \lhd \alpha \overset{u}{\approx} s_0 \lhd ipurge(u, \alpha)$$

- Noninterference is a global property of sequences of actions and state transitions
- To inductively reason on action sequences we need to derive conditions on individual state transitions

Noninterference

Observational relation

Observational equivalence/relation

Parameterized by an observing domain and induced by the *obs* function.

$$s \lhd \alpha \overset{u}{\approxeq} t \lhd \beta \equiv obs(u, run(\alpha, s)) = obs(u, run(\beta, t))$$

An alternative Definition

$$noninterference \equiv \forall\, \alpha\; u.\; s_0 \lhd \alpha \overset{u}{\approxeq} s_0 \lhd ipurge(u, \alpha)$$

- Noninterference is a global property of sequences of actions and state transitions
- To inductively reason on action sequences we need to derive conditions on individual state transitions

Proving Noninterference

- The essence of noniterference is that an observer cannot tell the difference between any system run and the variant of it obtained by removing (*purging*) all events that he is not allowed to notice, directly or indirectly
- The use of unwinding reduces this global property to a set of local, step-wise properties, in particular the two complementing ones introduced in [Rushby 1992]:

Step consistency

$$\mathit{step\_consistency} \equiv \forall\, a\; u\; s\; t.\; s \overset{u}{\sim} t \to \mathit{step}(a, s) \overset{u}{\sim} \mathit{step}(a, t)$$

Local respect

$$\mathit{local\_respect} \equiv \forall\, a\; u\; s.\; \mathit{dom}(a) \not\rightsquigarrow u \to s \overset{u}{\sim} \mathit{step}(a, s)$$

Proof sketch

Theorem goal: $obs(u, run(\alpha, s_0)) = obs(u, run(ipurge(u, \alpha), s_0))$

Main Lemma:

$\forall\ u\ s\ t.\ s \overset{sources(\alpha, u)}{\approx} t \rightarrow run(\alpha, s) \overset{u}{\sim} run(ipurge(u, \alpha), t)$

Proof of Theorem: specialize by $s = t = s_0$, use $s_0 \overset{sources(\alpha, u)}{\approx} s_0$ and apply output consistency

Proof of Main Lemma:

- induction on the actions sequence α,
- IF $dom(a) \in sources(a :: \alpha, u)$
- THEN apply *step_consistency*
- ELSE apply *local_respect*

Part III

An idealized model of virtualization

Formalization of an idealized model of virtualization

- Focus on the memory management policy of a paravirtualization style hypervisor
- Formally establish that the hypervisor
    - ensures strong isolation properties between the guest operating systems
    - guarantees the requests from guest operating systems are eventually attended
- Model and proofs developed using the Coq proof-assistant

Presented in 17th International Symposium on Formal Methods (FM 2011)
[Barthe, Betarte, Campo and Luna 2011]

Idealized models vs. implementations

Reasoning about implementations

- Give the strongest guarantees
- Is feasible for *some* exokernels and hypervisors
- May be feasible for *some* baseline properties of *some* systems
- Is out of reach in general (Linux Kernel)
- May not be required for evaluation purposes

Idealized models

- Many details of OS behavior are irrelevant for security
- Idealized models can provide a right level of abstraction. Proofs are more focused, and achievable within reasonable time
- Con: idealized models may not capture all relevant details. But: covert channels are also ignored if verifying implementations

A Xen like hypervisor

- A computer running the Xen hypervisor contains three components:
  - The Xen Hypervisor (software component)
  - The privileged Domain (*Dom*0): privileged guest running on the hypervisor with direct hardware access and management responsibilities
  - Multiple Unprivileged Domain Guests (*DomU*): unprivileged guests running on the hypervisor
- unprivileged guests execute hypercalls (access to services mediated by the hypervisor)

Virtualized memory

Abstract view

- Partitioning of memory
- Not fixed: allocation & deallocation
- Not total: memory may not belong to any OS

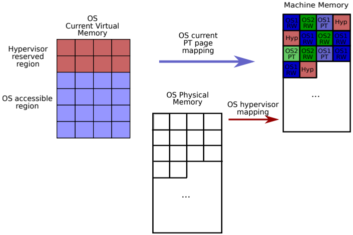
Virtualized memory

Abstract view

- Partitioning of memory
- Not fixed: allocation & deallocation
- Not total: memory may not belong to any OS

Idealized model

- Addresses: va, pa and ma
- Mappings between addresses
- Pages hold RW values or page tables



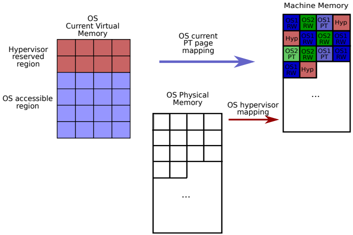
Virtualized memory

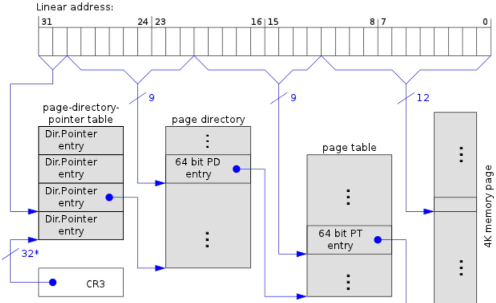
Idealized model

- Addresses: . . .
- Mappings . . .
- Pages . . .

In reality

- Multi-level page tables
- Cache and TLB
- Devices
- . . .





*) 32 bits aligned to a 32-Byte boundary

Context and States

$$
\mathit{Context} \stackrel{\text{def}}{=} \{ \begin{array}{ll} \mathit{vadd\_accessible} & : \mathit{vadd} \to \mathit{bool}, \\ \mathit{guests} & : \mathit{os\_ident} \to \mathit{bool} \} \end{array}
$$

$$
\mathit{State} \stackrel{\text{def}}{=} \{ \begin{array}{ll} \mathit{active\_os} & : \mathit{os\_ident}, \\ \mathit{aos\_exec\_mode} & : \mathit{exec\_mode}, \\ \mathit{aos\_activity} & : \mathit{os\_activity}, \\ \mathit{oss} & : \mathit{os\_ident} \mapsto \mathit{os\_info}, \\ \mathit{hypervisor} & : \mathit{os\_ident} \mapsto (\mathit{padd} \mapsto \mathit{madd}), \\ \mathit{memory} & : \mathit{madd} \mapsto \mathit{page} \} \end{array}
$$

$$
\begin{array}{lll}
\mathit{os\_info} & \stackrel{\text{def}}{=} & \{\mathit{curr\_page} : \mathit{padd}, \mathit{hcall} : \mathit{option\ Hyper\_call}\} \\
\mathit{content} & \stackrel{\text{def}}{=} & \mathit{RW}\ (v : \mathit{option\ Value}) \mid \mathit{PT}\ (\mathit{va\_to\_ma} : \mathit{vadd} \mapsto \mathit{madd}) \mid \mathit{Other} \\
\mathit{page\_owner} & \stackrel{\text{def}}{=} & \mathit{Hyp} \mid \mathit{Os}\ (\mathit{osi} : \mathit{os\_ident}) \mid \mathit{No\_Owner} \\
\mathit{page} & \stackrel{\text{def}}{=} & \{\mathit{page\_content} : \mathit{content}, \mathit{page\_owned\_by} : \mathit{page\_owner}\}
\end{array}
$$

$s \sim_{map,idx} s' \equiv s$ *and* s' *differ at most in the value associated to the index idx of the component map in the state* s'

Valid state

- Many conditions (see [Barthe, Betarte, Campo and Luna 2011]), e.g:
    - if the hypervisor or a trusted OS is running the processor must be in supervisor mode;
    - if an untrusted OS is running the processor must be in user mode;
    - all page tables of an OS *o* map accessible virtual addresses to pages owned by *o* and not accessible ones to pages owned by the hypervisor;
    - the current page table of any OS is owned by that OS;
    - any machine address *ma* which is associated to a virtual address in a page table has a corresponding pre-image, which is a physical address, in the hypervisor mapping.

Actions

`read` va	Guest OS reads virtual address va.
`write` va val	Guest OS writes value val in va.
`new` o va pa	Hypervisor extends memory of o with $va \mapsto ma$.
`del` o va	Hypervisor deletes mapping for va from current memory mapping of o.
`switch` o	Hypervisor sets o to be the active OS.
`hcall` c	Untrusted OS requires privileged service c to hypervisor.
`ret_ctrl`	Returns control to hypervisor.
`chmod`	Hypervisor changes execution mode from supervisor to user mode, and gives control to active OS.
`page_pin` o pa t	Registers memory page of type t at address pa.
`page_unpin` o pa	Memory page at pa is un-registered.

Semantics

- Pre-condition $\mathit{Pre} : \mathit{State} \rightarrow \mathit{Action} \rightarrow \mathit{Prop}$
- Post-condition $\mathit{Post} : \mathit{State} \rightarrow \mathit{Action} \rightarrow \mathit{State} \rightarrow \mathit{Prop}$
- Focus on normal execution: no semantics for error cases

Semantics of write action

$$
\begin{array}{l}
\mathit{Pre}\ s\ (\texttt{write}\ \mathit{va}\ \mathit{val}) \stackrel{\text{def}}{=} \\
\quad \mathit{os\_accessible}(\mathit{va}) \wedge \\
\quad s.\mathit{aos\_activity} \ = \ \mathit{running} \wedge \\
\quad \exists\ \mathit{ma} : \mathit{madd},\ \ \mathit{va\_mapped\_to\_ma}(s, \mathit{va}, \mathit{ma}) \wedge \\
\quad \mathit{is\_RW}((s.\mathit{memory}[\mathit{ma}]).\mathit{page\_content})
\end{array}
$$

$$
\begin{array}{l}
\mathit{Post}\ s\ (\texttt{write}\ \mathit{va}\ \mathit{val})\ s' \stackrel{\text{def}}{=} \\
\exists\ \mathit{ma} : \mathit{madd},\ \ \mathit{va\_mapped\_to\_ma}(s, \mathit{va}, \mathit{ma}) \wedge \\
\quad s'.\mathit{memory} \ = \ (s.\mathit{memory}[\mathit{ma} := \langle \mathit{RW}(\mathit{Some}\ \mathit{val}), s.\mathit{active\_os} \rangle]) \wedge \\
\quad s \sim_{\mathit{memory}, \mathit{ma}} s'
\end{array}
$$

Execution

Step and Invariance

The execution of an action is specified by the relation $\hookrightarrow$:

$$\frac{\mathit{valid\_state}(s) \quad \mathit{Pre}\ s\ \mathtt{a} \quad \mathit{Post}\ s\ \mathtt{a}\ s'}{s \overset{\mathtt{a}}{\hookrightarrow} s'}$$

One-step execution preserves valid states:

$$\forall\ (s\ s' : \mathit{State})\ (a : \mathit{Action}),\ s \overset{a}{\hookrightarrow} s' \ \rightarrow\ \mathit{valid\_state}(s')$$

- The (long and tedious) proof of this property follows by an inductive argument over action $\mathtt{a}$
- Key to isolation and availability results

Isolation properties

- Read isolation: no OS can read memory not belonging to it
- Write isolation: an OS cannot modify memory not owned by it
- OS isolation: the behavior of an OS does not depend on others

Isolation properties

Read isolation

Read isolation captures the intuition that no OS can read memory that does not belong to it:

$$
\begin{array}{l}
\mathit{read\_isolation} \equiv \\
\quad \forall\ (s\ s' : \mathit{State})\ (\mathit{va} : \mathit{vadd}), \\
\qquad s \xhookrightarrow{\mathit{read}\ \ \mathit{va}} s' \ \rightarrow \\
\qquad \exists\ \mathit{ma} : \mathit{madd},\ \ \mathit{va\_mapped\_to\_ma}(s, \mathit{va}, \mathit{ma}) \wedge \\
\qquad \exists\ \mathit{pg} : \mathit{page},\ \ \mathit{pg} = s.\mathit{memory}[\mathit{ma}]\ \wedge\ \mathit{pg}.\mathit{page\_owned\_by} = s.\mathit{active\_os}
\end{array}
$$

The execution of a `read` *va* action requires that the virtual address *va*

- is mapped to a machine address *ma* that belongs to the current memory mapping of active OS, and
- it is owned by the active OS.

Statistics

Size of the Coq code corresponding to the core model:

Model and basic lemmas	4.8kLOC
Valid state invariance	8.0kLOC
Read and write isolation	0.6kLOC
OS Isolation	6.0kLOC
Availability	1.0kLOC
Total	**20.4kLOC**

The extension with cache and TLB adds further **12kLOC**.

Conclusions

- There exist well-understood theoretical tools to formalize and verify security models
- Notions of information flow security can be directly applied to reason on isolation and properties of virtualization platforms
- The formal development in Coq forms a suitable basis for reasoning about hypervisors

References

Carl Waldspurger.
Lecture 1: Virtualization 101.
VMware R&D, 2007.

Gerald J. Popek, Robert P. Goldberg.
Formal requirements for virtualizable third generation architectures.
Communications of the ACM, volume 17, pages 412–421, 1974.

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield.
Xen and the art of virtualization.
In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

J. A. Goguen, J. Meseguer.
Security Policies and Security Models.
IEEE Symposium on Security and Privacy, 1982.

J. M. Rushby.
Noninterference, Transitivity, and Channel-Control Security Policies.
Technical Report CSL-92-02, SRI International, 1992.

David von Oheimb.
Information flow control revisited: Noninfluence = Noninterference + Nonleakage.
In P. Samarati, P. Ryan, D. Gollmann, and R. Molva, editors, *Computer Security – ESORICS 2004*, volume 3193 of *LNCS*, pages 225–243. Springer, 2004.

References (Cont.)

Heiko Mantel.
A Uniform Framework for the Formal Specification and Verification of Information Flow Security.
PhD Thesis, Univ. des Saarlandes, 2003.

T. Garfinkel, A. Warfield.
What virtualization can do for security.
;login: The USENIX Magazine, 32, 2007.

G. Barthe, G. Betarte, J.D. Campo, C. Luna.
Formally verifying isolation and availability in an idealized model of virtualization.
In Proceedings of FM2011: 17th International Symposium on Formal Methods, Lecture Notes in Computer Science, vol. 6664, pp 231-245, 2011.

Gustavo Betarte.
Formal verification of an idealized model of virtualization.
Invited tutorial in 9th International Conference on Software Engineering and Formal Methods (SEFM 2011), Montevideo, Uruguay, November 2011.

G. Barthe, G. Betarte, J.D. Campo, C. Luna.
Cache-leakage resilience in an idealized model of virtualization.
In Proceedings of CSF12: 25th IEEE Computer Security Foundations Symposium, IEEE Computer Society Press, pp 231-245, Harvard University, Massachusetts, USA, June 2012.