Formal analysis of security models for critical systems: Virtualization platforms and mobile devices

Carlos Luna

Grupo de Seguridad Informática, InCo
Facultad de Ingeniería, Universidad de la República, Uruguay

Formal analysis of security models for critical systems

Areas of safety-critical applications:

- *Virtualization platforms*
- *Mobile devices*
- *Domain name systems*

Formal analysis of security models for critical systems

Areas of safety-critical applications:

- *Virtualization platforms*
- *Mobile devices*
- *Domain name systems*

Research projects involved:

1. Mecanismos autónomos de seguridad certificados para sistemas computacionales móviles (ANII–Clemente Estable, Uruguay, 2015-2018);
2. VirtualCert: Towards a Certified Virtualization Platform - Phase II (UDELAR-CSIC I+D, Uruguay, 2013-2015);
3. VirtualCert: Towards a Certified Virtualization Platform (ANII-Clemente Estable, PR-FCE-2009-1-2568, Uruguay, 2010-2012);
4. Especificación Formal y Verificación de Sistemas Críticos (SeCyT-FCEIA ING266, UNR, Argentina, 2009-2010);
5. STEVE: Security Through Verifiable Evidence (PDT 63/118, FCE 2006, DINACYT, Uruguay, 2007-2009);
6. ReSeCo: Reliability and Security of Distributed Software Components (STIC-AMSUD, 2006-2009);

The Calculus of (Co)Inductive Constructions (CIC) and Coq

CIC is an extension of the simple-typed lambda calculus with:

- Polymorphic types [$(\lambda\, x\, .\, x) : A \to A$]
- Higher-order types [$A \to A : * : \Box$]
- Dependent types [$(\lambda\, a : A\, .\, f\ a) : (\forall\, a : A\, .\, B_a)$]

- Implemented in Coq
  Type checker + Proof assistant
- Can encode higher-order predicate logic
- (Co)Inductive definitions
- Curry-Howard isomorphism

types	$\leftrightarrow$	propositions
terms	$\leftrightarrow$	proofs

Outline

1. VirtualCert: an idealized model of virtualization
2. A certified idealized hypervisor
3. Conclusion and work in progress

Part I

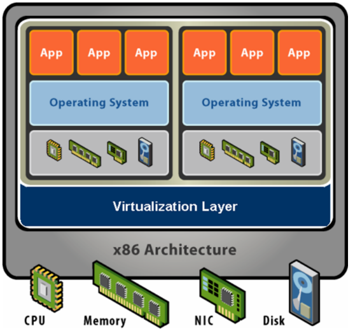
VirtualCert

OS verification

- OS verification since 1970
  - Hand written proofs
  - Type systems and program logics
  - Proof assistants
- OS verification is the next frontier
  - Tremendous advances in proof assistant technology
  - PL verification is becoming ubiquitous
- Flagship projects:
  - L4.verified: formal verification of seL4 kernel (G. Klein et al, NICTA)
  - Hyper-V: formal verification of Microsoft hypervisor (E. Cohen et al, MSR)

Virtualization

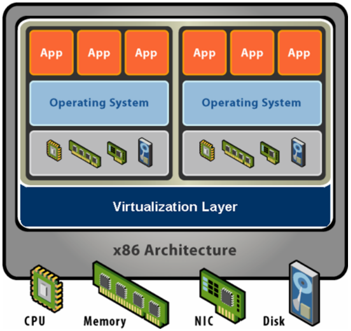
bare-metal hypervisors



- 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

Virtualization

bare-metal 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
  - protection in safety-critical and embedded systems
  - secure provisioning of infrastructures in cloud computing

Hypervisors are a priority target of formal specification and verification

Motivation and challenge

- Main focus of L4.verified and Hyper-V on functional correctness
- We focus on non-functional properties:
  - Isolation
  - Transparency
  - Availability (maximizing resources under constraints)

  Both properties go beyond safety:
  - Isolation and transparency are 2-safety properties
  - Availability is a liveness property

Motivation and challenge

- Main focus of L4.verified and Hyper-V on functional correctness
- We focus on non-functional properties:
  - Isolation
  - Transparency
  - Availability (maximizing resources under constraints)

  Both properties go beyond safety:
  - Isolation and transparency are 2-safety properties
  - Availability is a liveness property
- We reason about classes of systems

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 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 provide the right level of abstraction

- Many details of behavior are irrelevant for specific property
- Idealization helps comparing different alternatives
- Proofs are more focused, and achievable within reasonable time

Our focus: Xen on ARM

A popular bare-metal hypervisor initially developed at U. Cambridge

Architecture

A computer running the Xen hypervisor contains three components:

- The Xen Hypervisor (software component)
- The privileged Domain ($Dom0$): privileged guest running on the hypervisor with direct hardware access and management responsibilities
- Multiple Unprivileged Domain Guests ($DomU$): unprivileged guests running on the hypervisor, and executing hypercalls (access to services mediated by the hypervisor)

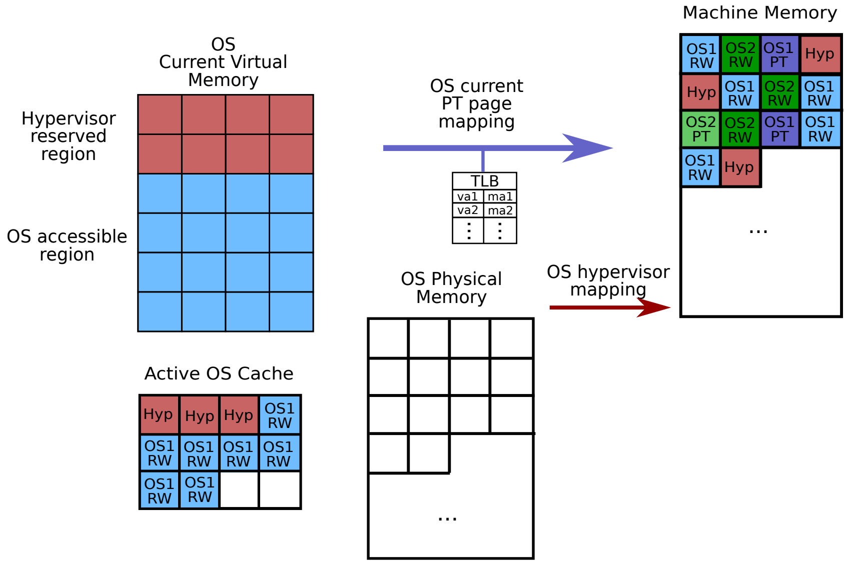
Xen on ARM

- Suggested during initial collaboration with VirtualLogix (now Red Bend Software)
- In turn, determines some modelling choices, e.g. for the cache

VirtualCert - Idealized model

- Abstract model written in Coq
- Focus on memory management
- Model of the hypervisor: based on Xen
- Model of the host machine: based on ARM

Memory model



States

$$
\mathit{State} \overset{\text{def}}{=} \left\{
\begin{array}{ll}
\mathit{active\_os} & : \mathit{os\_ident}, \\
\mathit{aos\_exec\_mode} & : \mathit{exec\_mode}, \\
\mathit{aos\_activity} & : \mathit{os\_activity}, \\
\mathit{oss} & : \mathit{os\_ident} \mapsto \mathit{os\_info}, \\
\mathit{hypervisor} & : \mathit{os\_ident} \mapsto (\mathit{padd} \mapsto \mathit{madd}), \\
\mathit{memory} & : \mathit{madd} \mapsto \mathit{page} \\
\mathit{cache} & : \mathit{vadd} \mapsto_{\mathit{size\_cache}} \mathit{page}, \\
\mathit{tlb} & : \mathit{vadd} \mapsto_{\mathit{size\_tlb}} \mathit{madd}
\end{array}
\right\}
$$

OS information and pages

$$\mathit{os\_info} \overset{\text{def}}{=} \{\, \mathit{curr\_page} : \mathit{padd}, \mathit{hcall} : \mathit{option}\ \mathit{Hyper\_call} \,\}$$

$$\mathit{page} \overset{\text{def}}{=} \{\, \mathit{page\_content} : \mathit{content}, \mathit{page\_owned\_by} : \mathit{page\_owner} \,\}$$

$$\mathit{content} \overset{\text{def}}{=} \{\, \mathit{RW}\ (\mathit{option}\ \mathit{Value}) \mid \mathit{PT}\ (\mathit{vadd} \mapsto \mathit{madd}) \mid \mathit{Other} \,\}$$

$$\mathit{page\_owner} \overset{\text{def}}{=} \{\, \mathit{Hyp} \mid \mathit{Os}\ (\mathit{os\_ident}) \mid \mathit{No\_Owner} \,\}$$

Execution: State transformers

read va	Guest OS reads virtual address va.
write va val	Guest OS writes value val in va.
read_hyper va	Hypervisor reads virtual address va.
write_hyper va val	Hypervisor writes value val in virtual address va.
hcall c	Guest OS requires privileged service c to the hypervisor.
new o va pa	Hypervisor extends os memory with $va \mapsto ma$.
del o va	Hypervisor deletes mapping for va from current memory mapping of o.
lswitch o pa	Hypervisor changes the current memory mapping of the active OS, to be the one located at physical address pa.
switch o	Hypervisor sets o to be the active OS.
ret_ctrl	Returns control to the hypervisor.
chmod	Hypervisor changes execution mode from supervisor to user mode, and gives control to the 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
Axiomatic specification

- Pre-condition $Pre : State \to Action \to Prop$
- Post-condition $Post : State \to Action \to State \to Prop$
- Focus on normal execution: no semantics for error cases
- Alternatives (write through/write back, replacement and flushing policies)
- One step execution:

$$s \xhookrightarrow{a} s' \stackrel{\text{def}}{=} valid\_state(s) \wedge Pre\ s\ a \wedge Post\ s\ a\ s'$$

- Traces:

$$s_0 \xhookrightarrow{a_0} s_1 \xhookrightarrow{a_1} s_2 \xhookrightarrow{a_2} s_3 \ldots$$

- Valid state:
  - invariant under execution
  - key to isolation results

Valid state

Many conditions, 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
- ...

Semantics

Write Action

$$\begin{aligned}
&\mathit{Pre}\ s\ (\mathit{write}\ \mathit{va}\ \mathit{val}) \overset{\text{def}}{=} \exists \mathit{ma}, \mathit{pg} \\
&\quad \mathit{os\_accessible}(\mathit{va})\ \wedge \\
&\quad s.\mathit{aos\_activity} \ = \ \mathit{running}\ \wedge \\
&\quad \mathit{va\_mapped\_to\_ma}(s, \mathit{va}, \mathit{ma})\ \wedge \\
&\quad \mathit{va\_mapped\_to\_pg}(s, \mathit{va}, \mathit{pg})\ \wedge \\
&\quad \mathit{is\_RW}(\mathit{pg})
\end{aligned}$$

$$\begin{aligned}
&\mathit{Post}\ s\ (\mathit{write}\ \mathit{va}\ \mathit{val})\ s' \ \overset{\text{def}}{=} \\
&\quad \text{let } (\mathit{new\_pg} : \mathit{page} = \ \langle \mathit{RW}(\mathit{Some}\ \mathit{val}), \mathit{pg}.\mathit{page\_owned\_by}\rangle) \text{ in} \\
&\quad s' = s \cdot \left[\begin{array}{rcl}
\mathit{memory} & := & (s.\mathit{memory}[\mathit{ma} := \mathit{new\_pg}]), \\
\mathit{cache} & := & \mathit{cache\_add}(\mathit{fix\_cache\_syn}(s, s.\mathit{cache}, \mathit{ma}), \mathit{va}, \mathit{new\_pg}), \\
\mathit{tlb} & := & \mathit{tlb\_add}(s.\mathit{tlb}, \mathit{va}, \mathit{ma})
\end{array}\right]
\end{aligned}$$

Equivalence w.r.t. an OS

Two states s_1 and s_2 are osi-equivalent, written $s_1 \equiv_{osi} s_2$, iff:

1. osi is the active OS in both states and the processor mode is the same, or the active OS is different to osi in both states
2. osi has the same hypercall in both states, or no hypercall in both states
3. the current page tables of osi are the same in both states
4. all page table mappings of osi that map a virtual address to a RW page in one state, must map that address to a page with the same content in the other
5. the hypervisor mappings of osi in both states are such that if a given physical address maps to some RW page, it must map to a page with the same content on the other state

Isolation properties

Read isolation

No OS can read memory that does not belong to it

Isolation properties

Read isolation

No OS can read memory that does not belong to it

Write isolation

An OS cannot modify memory that it does not own

Isolation properties

Read isolation

No OS can read memory that does not belong to it

Write isolation

An OS cannot modify memory that it does not own

OS isolation (on traces)

$$\begin{array}{l} \forall\ (t_1\ t_2 : \mathit{Trace})\ (\mathit{osi} : \mathit{os\_ident}), \\ \mathit{same\_os\_actions}(\mathit{osi}, t_1, t_2) \to \\ (t_1[0] \equiv_{\mathit{osi}} t_2[0]) \to \\ \square(\equiv_{\mathit{osi}}, t_1, t_2) \end{array}$$

Transparency

- A guest OS is unable to distinguish between executing together with other OSs and executing alone on the platform

Transparency

- A guest OS is unable to distinguish between executing together with other OSs and executing alone on the platform
- Given a trace, erase all state components that do not correspond to osi and “silence” all actions not performed by osi
- Similar to isolation, but the execution of the OS must be valid in the erased trace

Lemmas

$$\forall\ (s : State),\ valid\_state(s)\ \rightarrow\ valid\_state(s\backslash_{osi})\ \wedge\ s\ \overset{w}{\equiv}_{osi}\ s\backslash_{osi}$$
$$\forall\ (s\ s' : State)(a : Action),\ s \xhookrightarrow{a} s'\ \rightarrow\ s\backslash_{osi} \xhookrightarrow{a\backslash_{osi}} s'\backslash_{osi}$$

Theorem

$$\forall\ (t : Trace),\ t\ \overset{w}{\approx}_{osi}\ t\backslash_{osi}$$

Availability

- IF the hypervisor only performs `chmod` actions whenever no hypercall is pending
- AND the hypervisor returns control to guest operating systems infinitely often
- THEN no OS blocks indefinitely waiting for its hypercalls to be attended

$$\begin{array}{l} \forall\ (t : Trace), \neg\ hcall(t[0]) \rightarrow \\ \square(chmod\_nohcall, t) \rightarrow \\ \square(\lozenge \neg\ hyper\_running, t) \rightarrow \\ \square(\lozenge \neg\ hcall, t) \end{array}$$

Fairness and other properties

- Does not guarantee that every OS will eventually get attended
- Many other policies may be considered

Part II

A certified idealized hypervisor

Implementation in Coq

- We present an implementation of an hypervisor in the programming language of Coq
- The implementation is total, in the sense that it computes for every state and action a new state or an error. Thus, soundness is proved with respect to an extended axiomatic semantics in which transitions may lead to errors

Error management

$$ErrorMsg : State \rightarrow Action \rightarrow ErrorCode \rightarrow Prop$$

Action	**Failure**	**Error Code**
`write` $va\ val$	$s.aos\_activity \neq running$	$wrong\_os\_activity$
	$\neg\ va\_mapped\_to\_ma(s, va, ma)$	$invalid\_vadd$
	$\neg\ os\_accessible(va)$	$no\_access\_va\_os$
	$\neg\ is\_RW(s.memory[ma].page\_content)$	$wrong\_page\_type$

Table: Preconditions and error codes

Executions with error management

$$\frac{valid\_state(s) \qquad Pre(s,a) \qquad Post(s,a,s')}{s \xhookrightarrow{a/ok} s'}$$

$$\frac{valid\_state(s) \qquad ErrorMsg(s,a,ec)}{s \xhookrightarrow{a/error\ ec} s}$$

$$\begin{aligned} \textsf{Response} \stackrel{\text{def}}{=}\ & ok : Response \\ \mid\ & error : ErrorCode \to Response \end{aligned}$$

Executions with error management

$$\frac{valid\_state(s) \qquad Pre(s,a) \qquad Post(s,a,s')}{s \xhookrightarrow{a/ok} s'}$$

$$\frac{valid\_state(s) \qquad ErrorMsg(s,a,ec)}{s \xhookrightarrow{a/error\ ec} s}$$

$$\begin{aligned} \textsf{Response} \stackrel{\text{def}}{=}\ & ok : Response \\ & \mid error : ErrorCode \rightarrow Response \end{aligned}$$

Lemma (Validity is invariant)

$$\forall\ (s\ s' : State)(a : Action)(r : Response),$$
$$valid\_state(s) \ \rightarrow s \xhookrightarrow{a/r} s' \rightarrow valid\_state(s')$$

Action execution

```
Definition step s a :=
  match a with
      | ... ⇒ ...
      | Write va val ⇒ write_safe(s, va, val)
      | ... ⇒ ...
  end.
```

$$Result \overset{\text{def}}{=} \{resp : Response, st : State\}$$

Execution of `write` action

```
Definition $write\_safe$ $(s : state)$ $(va : vadd)$ $(val : value)$ : $Result$ :=
  match $write\_pre(s, va, val)$ with
          | $Some\ ec \Rightarrow \langle error(ec), s\rangle$
          | $None \Rightarrow \langle ok, write\_post(s, va, val)\rangle$
   end.

Definition $write\_pre$ $(s : state)$ $(va : vadd)$ $(val : value)$ : $option\ ErrorCode$ :=
  match $get\_os\_ma(s, va)$ with
    | $None \Rightarrow Some\ invalid\_vadd$
    | $Some\ ma$
      $\Rightarrow$ match $page\_type(s.memory, ma)$ with
         | $Some\ RW$
           $\Rightarrow$ match $aos\_activity(s)$ with
              | $Waiting \Rightarrow Some\ wrong\_os\_activity$
              | $Running$
                $\Rightarrow$ if $vadd\_accessible(s, va)$
                   then $None$
                   else $Some\ no\_access\_va\_os$
              end
         | $\_ \Rightarrow Some\ wrong\_page\_type$
      end end.
```

Effect of `write` execution

```
Definition write_post (s : state) (va : vadd) (val : value) : state :=
  match s.cache[va] with
  | Value old_pg ⇒
    let new_pg := Page (RW_c (Some val)) (page_owned_by old_pg) in
    let val_ma := va_mapped_to_ma_system(s, va) in
    match val_ma with
    | Value ma ⇒
      s · [ mem := s.memory[ma := new_pg],
            cache := fcache_add(fix_cache_syn(s, s.cache, ma), va, new_pg) ]
    | Error _ ⇒ s
    end
  | Error _ ⇒
    match s.tlb[va] with
    | Value ma ⇒
      match s.memory[ma] with
      | Value old_pg ⇒
        let new_pg := Page (RW_c (Some val)) (page_owned_by old_pg) in
        s · [ mem := s.memory[ma := new_pg],
              cache := fcache_add(fix_cache_syn(s, s.cache, ma), va, new_pg) ]
      | Error _ ⇒ s
      end
```

Effect of `write` execution (II)

```
| $Error\ \_ \Rightarrow$
  match $va\_mapped\_to\_ma\_currentPT(s, va)$ with
  | $Value\ ma \Rightarrow$
    match $s.memory[ma]$ with
    | $Value\ old\_pg \Rightarrow$
      let $new\_pg := Page\ (RW\_c\ (Some\ val))\ (page\_owned\_by\ old\_pg)$ in
      $s \cdot [\ mem := s.memory[ma := new\_pg],$
          $cache := fcache\_add(fix\_cache\_syn(s, s.cache, ma), va, new\_pg),$
          $tlb := ftlb\_add(s.tlb, va, ma)\ ]$
    | $Error\ \_ \Rightarrow\ s$
    end
  | $Error\ \_ \Rightarrow\ s$
  end
end
end.
```

Soundness

Theorem (Soundness of hypervisor implementation)

$$\forall\ (s : State)\ (a : Action),\ valid\_state(s) \to$$
$$s \xhookrightarrow{a/step(s,a).resp} step(s,a).st$$

Soundness

Theorem (Soundness of hypervisor implementation)

$$\forall\ (s : State)\ (a : Action),\ valid\_state(s) \to$$
$$s \xhookrightarrow{a/step(s,a).resp} step(s,a).st$$

Lemma (Soundness of error execution)

$$\forall\ (s : State)\ (a : Action),$$
$$valid\_state(s) \to\ \neg Pre(s,a) \to\ \exists\ (ec : ErrorCode),$$
$$step(s,a).st = s\ \wedge\ step(s,a).resp = ec\ \wedge\ ErrorMsg(s,a,ec)$$

Lemma (Soundness of valid execution)

$$\forall\ (s : State)\ (a : Action),\ valid\_state(s)\ \to\ Pre(s,a)\ \to$$
$$s \xhookrightarrow{a/ok} step(s,a).st\ \wedge\ step(s,a).resp = ok$$

Non-influencing execution (errors)

Traces

$$s_0 \xhookrightarrow{a_0/r_0} s_1 \xhookrightarrow{a_1/r_1} s_2 \xhookrightarrow{a_2/r_2} s_3 \ldots$$

Non-influencing execution (errors)

Traces

$$s_0 \xhookrightarrow{a_0/r_0} s_1 \xhookrightarrow{a_1/r_1} s_2 \xhookrightarrow{a_2/r_2} s_3 \ldots$$

$$\frac{t_1 \approx_{osi,cache,tlb} t_2 \qquad \neg\, os\_action(s, a, osi)}{(s \xhookrightarrow{a/r} t_1) \approx_{osi,cache,tlb} t_2}$$

$$\frac{t_1 \approx_{osi,cache,tlb} t_2 \qquad \neg\, os\_action(s, a, osi)}{t_1 \approx_{osi,cache,tlb} (s \xhookrightarrow{a/r} t_2)}$$

$$\frac{t_1 \approx_{osi,cache,tlb} t_2 \qquad os\_action(\{s_1, s_2\}, a, osi) \qquad s_1 \equiv^{cache,tlb}_{osi} s_2}{(s_1 \xhookrightarrow{a/ok} t_1) \approx_{osi,cache,tlb} (s_2 \xhookrightarrow{a/ok} t_2)}$$

Non-influencing execution (errors)

Traces

$$s_0 \xhookrightarrow{a_0/r_0} s_1 \xhookrightarrow{a_1/r_1} s_2 \xhookrightarrow{a_2/r_2} s_3 \dots$$

$$\frac{t_1 \approx_{osi,cache,tlb} t_2 \qquad \neg\, os\_action(s, a, osi)}{(s \xhookrightarrow{a/r} t_1) \approx_{osi,cache,tlb} t_2}$$

$$\frac{t_1 \approx_{osi,cache,tlb} t_2 \qquad \neg\, os\_action(s, a, osi)}{t_1 \approx_{osi,cache,tlb} (s \xhookrightarrow{a/r} t_2)}$$

$$\frac{t_1 \approx_{osi,cache,tlb} t_2 \qquad os\_action(\{s_1, s_2\}, a, osi) \qquad s_1 \equiv^{cache,tlb}_{osi} s_2}{(s_1 \xhookrightarrow{a/ok} t_1) \approx_{osi,cache,tlb} (s_2 \xhookrightarrow{a/ok} t_2)}$$

Cache and TLB equivalences

$s_1 \equiv^{cache,tlb}_{osi} s_2$ iff $s_1 \equiv_{osi} s_2 \wedge s_1 \equiv^{cache}_{osi} s_2 \wedge s_1 \equiv^{tlb}_{osi} s_2$

OS isolation in execution traces (with errors)

Theorem (OS isolation)

$$\begin{aligned}
&\forall\ (t_1\ t_2 : \mathit{Trace})\ (\mathit{osi} : \mathit{os\_ident}),\\
&\mathit{same\_os\_actions}(\mathit{osi}, t_1, t_2) \to\\
&(t_1[0] \equiv_{\mathit{osi}} t_2[0]) \to t_1 \approx_{\mathit{osi},\mathit{cache},\mathit{tlb}} t_2
\end{aligned}$$

OS isolation in execution traces (with errors)

Theorem (OS isolation)

$$\begin{array}{l}\forall\ (t_1\ t_2 : \mathit{Trace})\ (\mathit{osi} : \mathit{os\_ident}),\\ \mathit{same\_os\_actions}(\mathit{osi}, t_1, t_2) \rightarrow\\ (t_1[0] \equiv_{\mathit{osi}} t_2[0]) \rightarrow t_1 \approx_{\mathit{osi},\mathit{cache},\mathit{tlb}} t_2\end{array}$$

Lemma (Locally preserves unwinding lemma)

$$\begin{array}{l}\forall\ (s\ s' : \mathit{State})\ (a : \mathit{Action})\ (r : \mathit{Response})\ (\mathit{osi} : \mathit{os\_ident}),\\ \neg\ \mathit{os\_action}(s, a, \mathit{osi})\ \rightarrow\ s \xhookrightarrow{a/r} s'\ \rightarrow\ s \equiv_{\mathit{osi}}^{\mathit{cache},\mathit{tlb}}\ s'\end{array}$$

Lemma (Step-consistent unwinding lemma)

$$\begin{array}{l}\forall\ (s_1\ s_1'\ s_2\ s_2' : \mathit{State})\ (a : \mathit{Action})\ (\mathit{osi} : \mathit{os\_ident}),\\ s_1 \equiv_{\mathit{osi}} s_2\ \rightarrow\ \mathit{os\_action}(s_1, a, \mathit{osi}) \rightarrow\ \mathit{os\_action}(s_2, a, \mathit{osi}) \rightarrow\\ s_1 \xhookrightarrow{a/ok} s_1'\ \rightarrow\ s_2 \xhookrightarrow{a/ok} s_2'\ \rightarrow\ s_1' \equiv_{\mathit{osi}}^{\mathit{cache},\mathit{tlb}}\ s_2'\end{array}$$

Part III

Conclusion and Work in Progress

Conclusion

- Our work shows that it is feasible to analyze formally models of safety-critical applications
- The Coq proof assistant is a useful tool for the verification of critical systems

Conclusion

- Our work shows that it is feasible to analyze formally models of safety-critical applications
- The Coq proof assistant is a useful tool for the verification of critical systems

Virtualization platforms

- Formally verified idealized model of virtualization
- Machine-checked proofs of isolation, availability and transparency
- Certified functional specification of step execution with error handling (and extraction of prototype in a functional programming language)

Statistics

Virtualization platforms

- 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
- The certified prototype of hypervisor adds further 20kLOC

More...

- Extension of the virtualization model to use a VIPT cache and abstract replacement and write policies
- Using the model for reasoning about cache-based attacks and countermeasures

More...

- Extension of the virtualization model to use a VIPT cache and abstract replacement and write policies
- Using the model for reasoning about cache-based attacks and countermeasures

Papers

1. Barthe, G., Betarte, G., Campo, J., Luna, C., Pichardie, D.: System-level non-interference for constant-time cryptography. In: 21st ACM Conference on Computer and Communications Security (2014) 1267–1279;
2. Barthe, G., Betarte, G., Campo, J.D., Chimento, J.M., Luna, C.: Formally verified implementation of an idealized model of virtualization. In TYPES 2013. Volume 26 of Leibniz International Proceedings in Informatics (LIPIcs)., Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2014) 45–63;
3. Barthe, G., Betarte, G., Campo, J., Luna, C.: Cache-Leakage Resilient OS Isolation in an Idealized Model of Virtualization. In: IEEE 25th Computer Security Foundations Symposium (2012) 186–197;
4. Barthe, G., Betarte, G., Campo, J., Luna, C.: Formally verifying isolation and availability in an idealized model of virtualization. In Butler, M., Schulte, W., eds.: Formal Methods 2011. Volume 6664 of LNCS, Springer-Verlag (2011) 231–245;

Work in progress: mobile devices

Android

- Open-source operating system originally designed for mobile devices
- Developed by Google and the Open Handset Alliance (OHA)
- Multi-user Linux system in which each app is a different user
- Any app can invoke another app's functionalities

Work in progress: mobile devices

Android

- Open-source operating system originally designed for mobile devices
- Developed by Google and the Open Handset Alliance (OHA)
- Multi-user Linux system in which each app is a different user
- Any app can invoke another app's functionalities

Permission system

- Permissions granting among applications (installation / access)
- Can be used until revocation
- Different delegation mechanisms

Android security

Work in progress

- Formal analysis of security models for mobile devices: Android 4.x – 6.x
- Vulnerability analysis
- A certified monitor

Android security

Work in progress

- Formal analysis of security models for mobile devices: Android 4.x – 6.x
- Vulnerability analysis
- A certified monitor

Papers

1. Betarte G., Campo J., Luna, C., Romano, A.: Formal Analysis of Android's Permission-Based Security Model. In: Scientific Annals of Computer Science 26(1):27–68 (2016);
2. Betarte, G., Campo, J., Luna, C., Romano, A.: Verifying Android's Permission Model In: ICTAC 2015, 485–504 (2015).

Time for questions

Questions?
Comments?

Thanks!