NICTA

seL4 Enforces Integrity

Thomas Sewell Simon Winwood Peter Gammie
Toby Murray June Andronick Gerwin Klein

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Introduction

- seL4 is an operating system. It comes with some proofs.

Introduction

- seL4 is an operating system. It comes with some proofs.
- Integrity is a textbook security property. It composes with the existing proofs.

Introduction

- seL4 is an operating system. It comes with some proofs.
- Integrity is a textbook security property. It composes with the existing proofs.
- Proving integrity holds in seL4 is a good proof exercise.

Introduction (again)

- seL4 is

Introduction (again)

- seL4 is a microkernel, not an operating system. It provides no policy.

Introduction (again)

- seL4 is a microkernel, not an operating system. It provides no policy.
- Integrity for seL4 looks quite different to a textbook security policy.

Introduction (again)

- seL4 is a microkernel, not an operating system. It provides no policy.
- Integrity for seL4 looks quite different to a textbook security policy.
- Proving integrity holds in seL4 is a good exercise in frustration.

Overview

Talk overview:

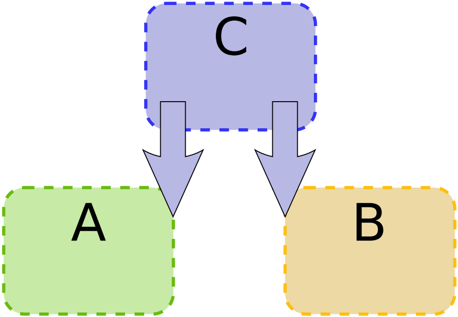
1. Integrity overview
2. Previous work: Composability and Comparability
3. Integrity for seL4
4. Proof of integrity

Integrity

Integrity is the property which says that things do not change when they should not.

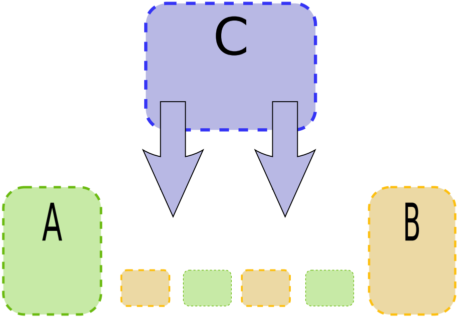
Integrity

Integrity is the property which says that things do not change when they should not.



Integrity

Integrity is the property which says that things do not change when they should not.



Previous Work: L4.verified

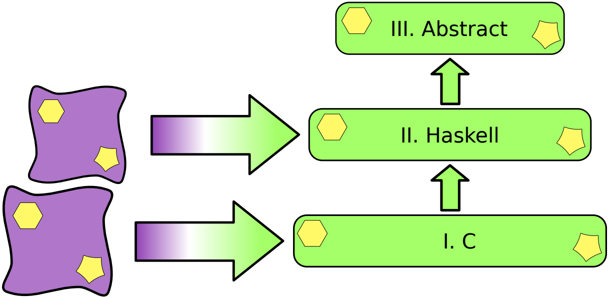
Previous work on seL4 includes the L4.verified project, which proved its functional correctness.

Harvey Tuch, Gerwin Klein and Gernot Heiser. **OS Verification - Now!** In Proceedings, 10th HotOS 2005.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Michael Norrish, Rafal Kolanski, Thomas Sewell, Harvey Tuch and Simon Winwood. **seL4: Formal verification of an OS Kernel.** In Proceedings, 22nd SOSP 2009.

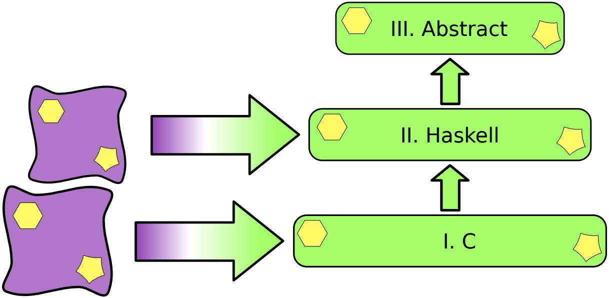
Previous Work: L4.verified

Previous work on seL4 includes the L4.verified project, which proved its functional correctness.
This was proven as a stack of refinement proofs.



Previous Work: L4.verified

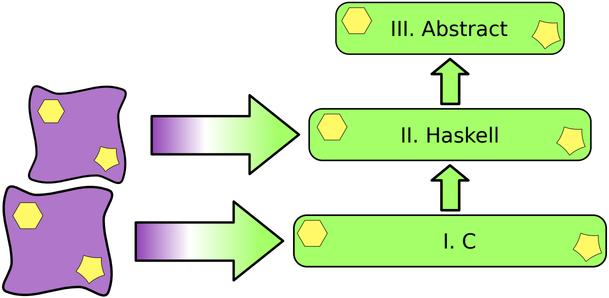
Previous work on seL4 includes the L4.verified project, which proved its functional correctness.
This was proven as a stack of refinement proofs.



Hoare triples {P} f {Q} compose down these refinement proofs

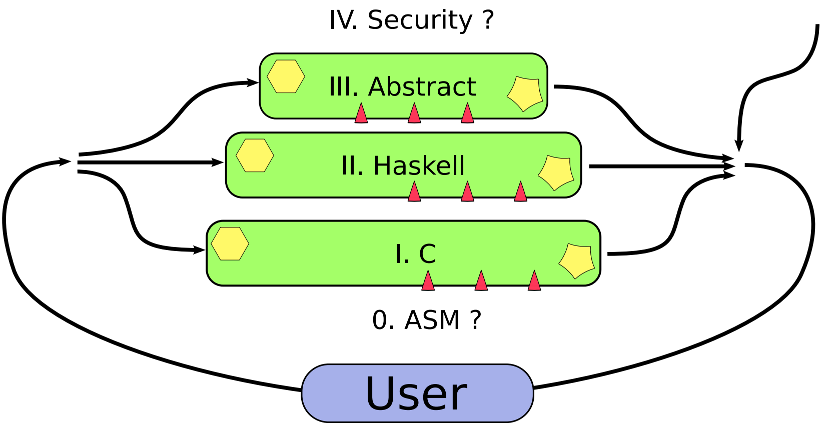
Previous Work: L4.verified

Previous work on seL4 includes the L4.verified project, which proved its functional correctness.
This was proven as a stack of refinement proofs.



Hoare triples $\{P\}$ f $\{Q\}$ compose down these refinement proofs modulo the abstraction/refinement relation.

Previous Work: L4.verified



IV. Security ?

III. Abstract

II. Haskell

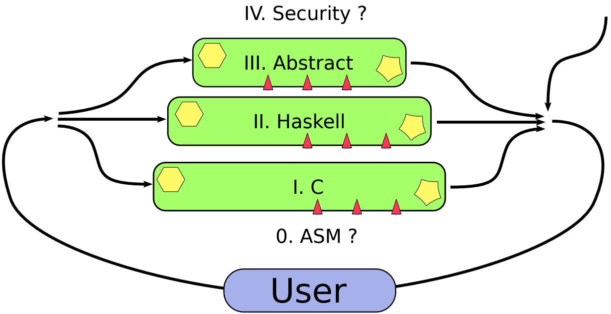
I. C

0. ASM ?

User

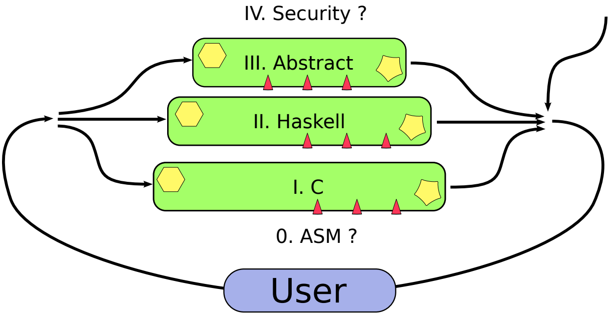
Previous Work: Verisoft

The Verisoft project addressed all the hardware-related issues by designing the hardware.



Previous Work: Verisoft

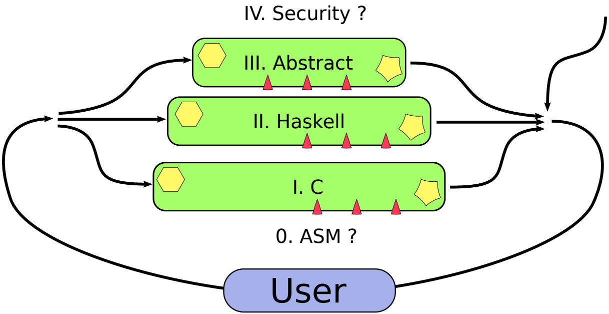
The Verisoft project addressed all the hardware-related issues by designing the hardware.



Verisoft also had defining property for their kernel: simulating multiple machines.

Previous Work: Verisoft

The Verisoft project addressed all the hardware-related issues by designing the hardware.



Verisoft also had defining property for their kernel: simulating multiple machines. This is a **policy**.

The seL4 Permission Model

The seL4 “Secure Embedded L4” microkernel is a member of the L4 microkernel family. It is designed to be as general-purpose as possible while providing security guarantees.

The seL4 Permission Model

The seL4 “Secure Embedded L4” microkernel is a member of the L4 microkernel family. It is designed to be as general-purpose as possible while providing security guarantees.

It provides objects for threads, virtual memory and communication.

The seL4 Permission Model

The seL4 “Secure Embedded L4” microkernel is a member of the L4 microkernel family. It is designed to be as general-purpose as possible while providing security guarantees.

It provides objects for threads, virtual memory and communication.

Its permission model is based on **capabilities** which explicitly grant a thread authority over some object.

The seL4 Permission Model

The seL4 “Secure Embedded L4” microkernel is a member of the L4 microkernel family. It is designed to be as general-purpose as possible while providing security guarantees.

It provides objects for threads, virtual memory and communication.

Its permission model is based on **capabilities** which explicitly grant a thread authority over some object.

There is **no policy**. Threads, their memory and their capability storage may overlap arbitrarily.

The seL4 Permission Model

The seL4 “Secure Embedded L4” microkernel is a member of the L4 microkernel family. It is designed to be as general-purpose as possible while providing security guarantees.

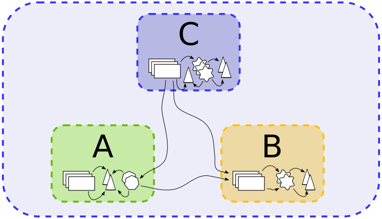
It provides objects for threads, virtual memory and communication.

Its permission model is based on **capabilities** which explicitly grant a thread authority over some object.

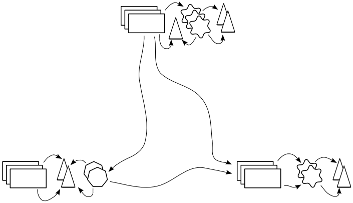
There is **no policy.** Threads, their memory and their capability storage may overlap arbitrarily.

Capabilities can be created, moved, sent through communcation channels and shared between threads.

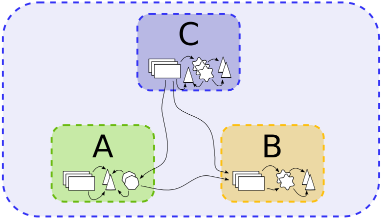
Integrity Property



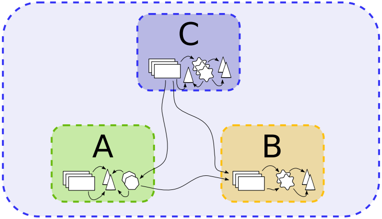
Integrity Property



Integrity Property



Integrity Property

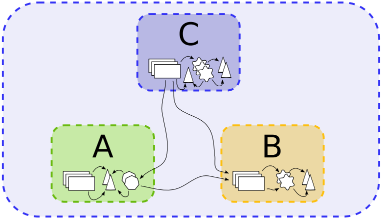


abstraction :: obj-ref ⇒ α

policy :: $(\alpha \times \text{auth} \times \alpha)$ set

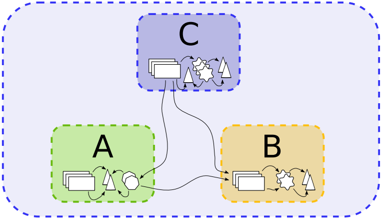
Integrity Property: Requirements

The kernel comes with **no explicit policy** about the way untrusting components may interact.



Integrity Property: Requirements

The kernel comes with **no explicit policy** about the way untrusting components may interact.



We require:

1. Only communcation endpoints and memory may be shared.
2. Capabilities may not be transferred.

Integrity Property: Authority type

We require:

1. Only communcation endpoints and memory may be shared.
2. Capabilities may not be transferred.

We map authority to communication endpoints and memory into the constructors Send, Receive, Read and Write of the auth type. All other authority we map to the Control constructor.

Integrity Property: Authority type

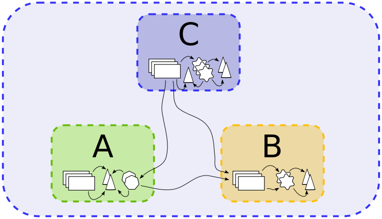
We require:

1. Only communcation endpoints and memory may be shared.
2. Capabilities may not be transferred.

We map authority to communication endpoints and memory into the constructors Send, Receive, Read and Write of the auth type. All other authority we map to the Control constructor.

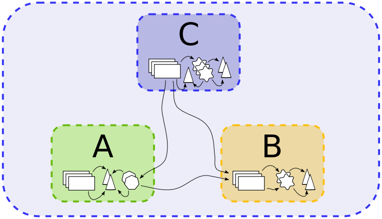
There are also Grant and Reset constructors. See the paper.

Integrity Policy: Controllers



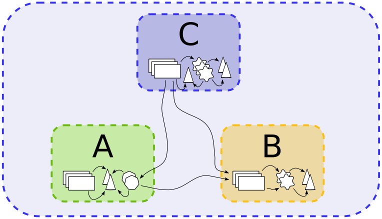
A & B	Integrity & Authority Confinement
C	??

Integrity Policy: Controllers



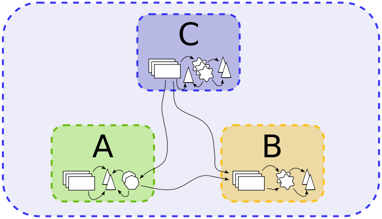
A & B	Integrity & Authority Confinement
C	SEP

Integrity Policy: Controllers



A & B	Integrity & Authority Confinement
C	Someone Else's Problem

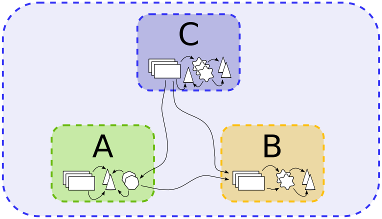
Integrity Policy: Controllers



A & B	Integrity & Authority Confinement
C	Someone Else’s Problem

Fine grained analyses like Take-Grant deal poorly with this case.

Integrity Policy: Controllers



A & B	Integrity & Authority Confinement
C	Someone Else’s Problem

Fine grained analyses like Take-Grant deal poorly with this case.

We can handle some dynamic cases this way.

Integrity Property: PAS

We define the PAS record:

$$\begin{array}{lrcl} \text{record } \alpha \text{ PAS } = & \text{pasPolicy} & :: & (\alpha \times \text{auth} \times \alpha) \text{ set} \\ & \text{pasAbs} & :: & \text{obj-ref} \Rightarrow \alpha \\ & \text{pasSubject} & :: & \alpha \end{array}$$

Integrity Property: PAS

We define the PAS record:

$$\begin{array}{rrcl} \text{record } \alpha \text{ PAS} \; = & \text{pasPolicy} & :: & (\alpha \times \text{auth} \times \alpha) \text{ set} \\ & \text{pasAbs} & :: & \text{obj-ref} \Rightarrow \alpha \\ & \text{pasSubject} & :: & \alpha \end{array}$$

The PAS record is a **constant** parameter to all analysis.

Integrity Property: Definition 1

Definition

pas-wellformed *pas* $\equiv$
$\forall\, y.\ (\text{pasSubject } pas, \text{Control}, y) \in \text{pasPolicy } pas$
$\longrightarrow y = \text{pasSubject } pas$

The current subject cannot have Control authority over any other.

Integrity Property: Definition 2

Definition

pas-refined *pas* *s* $\equiv$
    $\forall$ (x, *auth*, y) $\in$ system-auth *s*
        $\rightarrow$ (pasAbs *pas* x, *auth*, pasAbs *pas* y) $\in$ pasPolicy *pas*

All authority in the system must be permitted in the policy.

Integrity Property: Definition 3

Definition

integrity *pas s s'* $\equiv \ldots$

The subject is allowed to cause this transition. Describes what is allowed by Read, Write, Send, Receive and Control.

More details are in the paper.

Integrity Property: Hypothesis

We set out to prove two Hoare triples.

Integrity:

$\forall \mathit{pas}\ s\ e.$ pas-wellformed $\mathit{pas} \rightarrow$ pas-refined $\mathit{pas}\ s \rightarrow$

$\quad \{s\}$ call-kernel e $\{s'.$ integrity $\mathit{pas}\ s\ s'\}$

Confinement:

$\forall \mathit{pas}\ e.$ $\{s.$ pas-wellformed $\mathit{pas} \wedge$ pas-refined $\mathit{pas}\ s\}$

$\quad$ call-kernel e $\{s.$ pas-refined $\mathit{pas}\ s\}$

Proofs

Proofs

Lemma receive-async-ipc-pas-refined:

∀ *pas cap*. {*s*. pas-refined *pas s* ∧
(∀ *aepptr* ∈ obj-refs *cap*. pasAbs *pas t*, Receive, pasAbs *pas aepptr*)
∈ pasPolicy *pas*)}
receive-async-ipc *t cap*
{*s*. pas-refined *pas s*}

Lemma receive-async-ipc-integrity:

∀ *pas cap st*. {*s*. integrity *pas st s* ∧ pas-refined *pas s* ∧ valid-objs *s*
∧ pasAbs *pas t* = pasSubject *pas* ∧ (∀ *aepptr* ∈ obj-refs *cap*. pasAbs
pas t, Receive, pasAbs *pas aepptr*) ∈ pasPolicy *pas*)}
receive-async-ipc *t cap*
{*s*. integrity *pas st s*}

Proofs: History

We've done this before.

David Cock, Gerwin Klein and Thomas Sewell. **Secure Microkernels, State Monads and Scalable Refinement.** In Proceedings TPHOLs 2008.

Conclusions

- Defined Integrity for seL4, and not the textbook way.
- Proven that seL4 Enforces Integrity.