seL4 Enforces Integrity

Thomas Sewell  Simon Winwood  Peter Gammie
Toby Murray  June Andronick  Gerwin Klein
• seL4 is an operating system. It comes with some proofs.
seL4 is an operating system. It comes with some proofs.
Integrity is a textbook security property. It composes with the existing proofs.
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.
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.
seL4 is a microkernel, not an operating system. It provides no policy.
seL4 is a microkernel, not an operating system. It provides no policy.

Integrity for seL4 looks quite different to a textbook security policy.
- 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 is the property which says that things do not change when they should not.
Integrity is the property which says that things do not change when they should not.
Integrity is the property which says that things do not change when they should not.
Previous work on seL4 includes the L4.verified project, which proved its functional correctness.


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 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 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

I. C

II. Haskell

III. Abstract

IV. Security

0. ASM

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

Verisoft also had defining property for their kernel: simulating multiple machines.
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 “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 communication channels and shared between threads.
Integrity Property

\[ \text{abstraction} :: \text{obj-ref} \Rightarrow \alpha \]

\[ \text{policy} :: (\alpha \times \text{auth} \times \alpha) \text{set} \]
Integrity Property

\[ \text{obj-ref} \Rightarrow \alpha \]

Policy:
\[ (\alpha \times \text{auth} \times \alpha) \text{ set} \]
Integrity Property

\[ \text{abstraction} :: \text{obj-ref} \Rightarrow \alpha \]

\[ \text{policy} :: (\alpha \times \text{auth} \times \alpha) \text{ set} \]
Integrity Property

abstraction :: obj-ref $\Rightarrow$ $\alpha$

policy :: ($\alpha \times$ auth $\times$ $\alpha$) set
The kernel comes with no explicit policy about the way untrusting components may interact.
The kernel comes with **no explicit policy** about the way untrusting components may interact.

We require:

1. Only communication endpoints and memory may be shared.
2. Capabilities may not be transferred.
We require:

1. Only communication 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.
We require:

1. Only communication 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.
Fine grained analyses like Take-Grant deal poorly with this case. We can handle some dynamic cases this way.
Integrity Policy: Controllers

Fine grained analyses like Take-Grant deal poorly with this case. We can handle some dynamic cases this way.

<table>
<thead>
<tr>
<th>A &amp; B</th>
<th>Integrity &amp; Authority Confinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>SEP</td>
</tr>
</tbody>
</table>
Integrity Policy: Controllers

<table>
<thead>
<tr>
<th>A &amp; B</th>
<th>Integrity &amp; Authority Confinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Someone Else’s Problem</td>
</tr>
</tbody>
</table>
Fine grained analyses like Take-Grant deal poorly with this case.
Fine grained analyses like Take-Grant deal poorly with this case.

We can handle some dynamic cases this way.
We define the PAS record:

\[
\text{record } \alpha \text{ PAS } = \\
\text{ pasPolicy } :: (\alpha \times \text{auth} \times \alpha) \text{ set} \\
\text{ pasAbs } :: \text{obj-ref } \Rightarrow \alpha \\
\text{ pasSubject } :: \alpha
\]
We define the PAS record:

\[
\text{record } \alpha \text{ PAS } = \begin{align*}
\text{pasPolicy} & : (\alpha \times \text{auth} \times \alpha) \text{ set} \\
\text{pasAbs} & : \text{obj-ref} \Rightarrow \alpha \\
\text{pasSubject} & : \alpha
\end{align*}
\]

The PAS record is a \textbf{constant} parameter to all analysis.
Definition

\[ \text{pas-wellformed} \, \text{pas} \equiv \]
\[ \forall \ y. \ (\text{pasSubject} \, \text{pas}, \text{Control}, \ y) \in \text{pasPolicy} \, \text{pas} \]
\[ \rightarrow \ y = \text{pasSubject} \, \text{pas} \]

The current subject cannot have Control authority over any other.
Definition

pas-refined \( pas \ s \equiv \)
\[
\forall (x, auth, y) \in \text{system-auth } s \\
\rightarrow (\text{pasAbs } pas \ x, auth, \text{pasAbs } pas \ y) \in \text{pasPolicy } pas
\]

All authority in the system must be permitted in the policy.
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.
We set out to prove two Hoare triples.

Integrity:
\[ \forall \text{pas e. pas-wellformed pas} \rightarrow \text{pas-refined pas s} \rightarrow \{ s \} \text{ call-kernel e } \{ s'. \text{ integrity pas s s'} \} \]

Confinement:
\[ \forall \text{pas e. } \{ s. \text{ pas-wellformed pas } \land \text{ pas-refined pas s}\} \text{ call-kernel e } \{ s. \text{ pas-refined pas s}\} \]
Proofs

Lemma receive-async-ipc-pas-refined:
\[\forall \text{cap. pas-refined pas s} \land (\forall \text{aepptr } \in \text{obj-refs cap. pasAbs pas t, Receive, pasAbs pas aepptr}) \in \text{pasPolicy pas})\]

Lemma receive-async-ipc-integrity:
\[\forall \text{st. pas-refined pas s} \land \text{valid-objs s} \land \text{pasAbs pas t} = \text{pasSubject pas} \land (\forall \text{aepptr } \in \text{obj-refs cap. pasAbs pas t, Receive, pasAbs pas aepptr}) \in \text{pasPolicy pas})\]
Lemma receive-async-ipc-pas-refined:
\[\forall \text{ pas cap. } \{s. \text{ pas-refined pas s} \land \langle \forall \text{ aepptr } \in \text{ obj-refs cap. pasAbs pas t, Receive, pasAbs pas aepptr} \rangle \in \text{ pasPolicy pas} \rangle \}\]
receive-async-ipc t cap
\{s. \text{ pas-refined pas s}\}

Lemma receive-async-ipc-integrity:
\[\forall \text{ pas cap st. } \{s. \text{ integrity pas st s} \land \text{ pas-refined pas s} \land \text{ valid-objs s} \land \text{ pasAbs pas t = pasSubject pas} \land \langle \forall \text{ aepptr } \in \text{ obj-refs cap. pasAbs pas t, Receive, pasAbs pas aepptr} \rangle \in \text{ pasPolicy pas} \}\]\nreceive-async-ipc t cap
\{s. \text{ integrity pas st s}\}
We’ve done this before.

Conclusions

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