The seL4® Report
An Update From seL4 Land

Gernot Heiser
Chair, seL4 Foundation

gernot@sel4.systems
The Highlights of the Year

seL4 is verified on RISC-V!

2020/06/09

Sounds great! But what does it mean?

seL4

seL4 (https://sel4.systems/) (pronounced e-ell-fo) is arguably the world’s most secure operating system kernel. The OS kernel is the lowest level of software running on a computer system. It is the code that executes in privileged mode (S-mode in RISC-V; M-mode is reserved for microcode). The kernel is ultimately responsible for the security of a computer system.

To accelerate seL4 microkernel developments.

The Linux Foundation is set to host a new global not-for-profit foundation established by the CSIRO’s Data61 to promote and fund the development of its security-focused microkernel, seL4.
The Highlights of the Year

- **The seL4 Foundation** (June’s talk right after this):
  - Open governance for the seL4 ecosystem
  - Trademark registration in Australia and US, others in progress
- **Verification**: RISC-V (RV64) functional correctness proof done!
  - Binary verification (translation correctness) progressing
  - MCS verification progressing (see my FOSDEM’20 talk)
- **seL4 System** development
  - RFCs for seL4 Core, seL4 Core Platform
  - soon: RFC for seL4 driver framework
- **Research**:
  - Verifying *time protection*
  - Secure multi-server OS
Background

What is seL4?
Background: What is \texttt{sel4}?

\texttt{sel4} is an open source, high-assurance, high-performance operating system microkernel

- Available on GitHub under GPLv2 license
- World’s most comprehensive mathematical proofs of correctness and security
- World’s fastest microkernel
- Piece of software that runs at the heart of any system and controls all accesses to resources

Critical non-critical, untrusted

FOSDEM, Feb’21
Gernot Heiser: The sel4 Report
What is seL4?

seL4 is the most trustworthy foundation for safety- and security-critical systems.

Already in use across many domains:
automotive, aviation, space, defence, critical infrastructure, cyber-physical systems, IoT, industry 4.0, certified security...
Unique Verification by Mathematical Proof

Confidentiality

Integrity

Availability

Security Enforcement

Proof

Abstract Model

Functional Correctness

Now done for RISC-V!

C Implementation

Translation Correctness

A few more months for RISC-V

Binary code

Confidentiality

Integrity

Availability

Security Enforcement

Proof

Abstract Model

Functional Correctness

Now done for RISC-V!

C Implementation

Translation Correctness

A few more months for RISC-V

Binary code

Complete proof chain for ARMv7 (32-bit)

seL4: Still only capability-based OS kernel with correctness proof!

seL4: The only OS on RISC-V with correctness proof

RISC-V Summit, Dec'20

Gernot Heiser: seL4 on RISC-V
... and Performance

Latency (in cycles) of a round-trip cross-address-space IPC on x64

<table>
<thead>
<tr>
<th>Source</th>
<th>seL4</th>
<th>Fiasco.OC</th>
<th>Zircon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mi et al, 2019</td>
<td>986</td>
<td>2717</td>
<td>8157</td>
</tr>
<tr>
<td>Gu et al, 2020</td>
<td>1450</td>
<td>3057</td>
<td>8151</td>
</tr>
<tr>
<td>seL4.systems, Nov’20</td>
<td>797</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Still the world’s fastest microkernel!

Temporary performance regression in Dec’19

Sources:
- Zeyu Mi, Dingji Li, Zihan Yang, Xinran Wang, Haibo Chen: “SkyBridge: Fast and Secure Inter-Process Communication for Microkernels”, EuroSys, April 2020
- Jinyu Gu, Xinyue Wu, Wentai Li, Nian Liu, Zeyu Mi, Yubin Xia, Haibo Chen: “Harmonizing Performance and Isolation in Microkernels with Efficient Intra-kernel Isolation and Communication”, Usenix ATC, June 2020
- seL4 Performance, [https://sel4.systems/About/Performance/](https://sel4.systems/About/Performance/), accessed 2020-11-08
Making seL4 Easier to Use

The seL4 Core Platform
Why seL4 Core Platform?

With seL4 we achieved unprecedented levels of security and user-unfriendliness [2015]

The seL4 API is (for good reason):
• very general
• very low-level
• architecture-dependent
• very spartan
... and **requires a lot of expertise to use correctly**

See https://microkerneldude.wordpress.com/2020/03/11/sel4-design-principles/

Almost all present deployments are:
• embedded/cyber-physical systems
• simple, static architectures
Aims of the seL4 Core Platform

Small OS for IoT, cyber-physical and other embedded use cases

- Ease development and deployment
- Provide reasonable degree of application portability, defined HW interfaces
- Support implementation diversity through well-defined interfaces
- Support code re-use between related deployments
- Simple programming model ensures “correct” use of seL4 mechanisms
- Retain near-minimal trusted computing base (TCB)
- Leverage seL4-enforced isolation for strong security/safety
- Retain seL4’s superior performance
- Be amenable to formal verification of the TCB
The seL4 Core Platform is POSIX-Compliant

Of Course Not

Posix is past its use-by date, and too inefficient for seL4.
See Curtis Millar’s seL4 Summit talk

Core Platform properties:
- Simple execution model
- Simple communication model
- Real-time capable
- Efficient
- Deadlock-free
- Some integrity properties enforced by build tools
- Suitable for formal reasoning

For legacy software use virtual machines!
Target Hardware: Embedded SoCs

- Homogenous multicore
- Shared L2 cache
- Single system image
- Uniform memory access
- Accelerators (GPUs etc) are “devices”
Core Platform Abstractions

- Memory Region (MR)
- Protection Domain (PD)
- Communication Channel (CC)
- Notification

Protected Procedure Call (PPC)

```
f(){
  ...
}
f(..);
```
Abstractions: Protection Domain

Superficially similar to Unix process
- ... but much more lightweight
Consists of several seL4 objects:
- VSpace – virtual memory map
- CSpace – access right
- Scheduling context (SC)
- Priority (fixed)
- Notification

- All three execute atomically with respect to each other!
- They may
  - signal another PD’s notification
  - call another PD’s PP

- Runs once, on
- PD’s prio and SC

Protected Procedure Call (PPC)

Contains:
- Init procedure
- Notification procedure
- Optional: Protected procedure (PP)

- Invoked when PD’s notification is signalled
- Runs on PD’s prio and SC
- Bound to a Core!

- Invoked from other PD
- Runs on PD’s prio but caller PD’s SC

See https://microkerneldude.wordpress.com/2019/03/07/how-to-and-how-not-to-use-sel4-ipc/
Abstractions: Memory Region

Memory Region (MR)

May be mapped into one or more PDs
• at a virtual address
• with defined caching attributes
• with specific permissions

Protection Domain (PD)

May be attached to a CC
• shared buffer
• zero-copy communication

Protected Procedure Call (PPC)

f();

Communication Channel (CC)

Represents physical memory
• contiguous
• integer multiple of page size
• page-aligned

f(…);

Protected Procedure Call (PPC)
Abstractions: Communication Channels

Supported channel operations:
- reading/writing the channel’s MR – if mapped into the accessing PD
- referencing channel buffer locations
- signalling the other PD’s Notification
- calling the other PD’s PP (if available)

Supports communication between PDs
- connects exactly two PDs
- any pair of PDs has at most one common CC
- data flow can be uni- or bidirectional
- information flow is bi-directional (no data diode!)

In general:
- no trust relationship implied
- CCs form non-directed, cyclic graph

Potentially cross-core

Using reference wrappers, not plain pointers!

In general:
- no trust relationship implied
- CCs form non-directed, cyclic graph

Using reference wrappers, not plain pointers!
Abstractions: Notifications

- Notifications are binary semaphores
  - Multiple signals from same PD may not invoke *notification procedure* multiple times
- Processing of signals from multiple PDs should happen in priority order
  - Ideally enforced by Core Platform tooling
  - For now limit of 64 CCs per PD (seL4 limitation)

Support triggering of events:
- can signal PD’s Notification through CC
- this invokes target PD’s *notification procedure*
- Platform provides source PD’s identity
  - uses seL4’s badged capabilities
- Signalling is asynchronous

Notifications are associated with a PD
- exactly one per PD
Abstractions: Protected Procedure Calls

Supports execution of code in a different PD
- Callee ("server") must have a PP
- Caller ("client") and callee must share a CC
- PPC arguments may reference locations in the CC’s MR (using reference wrappers)
- Arguments limited to 16 words in total
- PPCs may nest

Asymmetric relationship:
- Client trusts server
- PPC must not block
- Server must be higher prio than client
- Remember: PPC runs on client’s SC and thus the client’s core!
- PPCs form acyclic, directed graph

Enforceable by build tools!
A VM is a PD with extra attributes
- supports extra execution mode (guest mode)
- else behaves like any PD
  - may share MRs
  - may signal/be signalled
  - may be client or server of a PPC

- Not yet fully specified
- Not supported in first version

A VM is an abstraction that offers additional capabilities beyond a regular PD (Protection Domain). It supports extra execution modes, such as guest mode, and can share memory regions, signal other domains, and act as clients or servers in communication channels. Functional specifications and support vary across different versions.
Initially all PDs will be single-core
  - Single scheduling context means single core

This restriction will be removed in the near future for pure client PDs
  - PDs without a PPC

Supporting multi-threaded servers is possible
  - a bit more complicated
  - \( \leq 1 \) thread per core

For now targeting static architectures:
  - All PDs known at build time

Will (eventually) support late loading/re-loading of (known) PDs
Summary: seL4 Core Platform

- Designed to ease construction of well-designed seL4-based embedded systems
- Design mostly complete: RFC-5
- Will integrate with the seL4 Driver Framework
- We’ll provide best-practice training material
Research Update

Verifying time protection
Secure multi-server OS
What’s the Issue with Temporal Isolation?

Safety: Timeliness
- Execution interference

Security: Confidentiality
- Leakage via timing channels

Affect execution speed:
Integrity violation – deadline miss

Observe execution speed:
Confidentiality violation

Addressed by MCS kernel [FOSDEM’20]

Addressed by time protection
Cause: Competition for HW Resources

- Inter-process interference
- Competing access to micro-architectural features
- Hidden by the HW-SW contract!

Solution: *Time Protection* – Eliminate interference by preventing sharing
Time Protection: Partition all Hardware State

**High**

Cache

**Low**

Temporally partition

Flush

**High**

Cache

**Low**

Spatially partition

Flushing useless for concurrent access
- HW threads
- cores

Need both!

Cannot spatially partition on-core caches (L1, TLB, branch predictor, pre-fetchers)
- virtually-indexed
- OS cannot control

More details:
- [Heiser, FOSDEM’20]
- [Ge et al, EuroSys’19]
Measuring Leakage: Channel Matrix

D-cache channel on x86 Haswell, no mitigation

Variation along a horizontal line indicates a channel

Channel matrix:
- Conditional probability of observing output signal \( t \), given input \( n \)
- Represented as heat map:
  - bright: high probability
  - dark: low probability
Measuring Leakage: Channel Matrix

D-cache channel on x86 Haswell, no mitigation

Variation along a horizontal line indicates a channel

Channel

D-cache channel on Haswell, time protection

No channel
Challenge: Broken Hardware

BHB channel on x86 Sky Lake, no mitigation

BHB channel on x86 Sky Lake, time protection

Small channel!
Challenge: Broken Hardware

Systematic study of COTS Hardware [Ge et al, APSys’18]:
- contemporary processors hold state that cannot be reset
- need a new hardware-software contract to enable real security

Gernot Heiser: The sel4 Report

Best Paper Award
RISC-V To The Rescue!

New instruction fence.t: flush of all micro-architectural state in ETH Ariane processor and evaluated channels on FPGA implementation.

Similar result for all other channels [Wistoff et al, DATE’21]

BHB channel Ariane, no mitigation

BHB channel Ariane, time protection

Large channel

No channel!
On-Going Work

Prove: no leakage

Combine with temporal integrity (MCS)

Time protection prototype

Verify efficacy

Make complete

Fix hardware

Make usable

Include fence.t in RISC-V ISA spec

Assumes sane (non-existent) hardware

Stay tuned!
Research: Secure Multi-Server OS

Aim: A truly secure, general-purpose OS
✔ Support wide class of use cases, fully dynamic
✔ Support wide class of security policies
✔ Support changes of security policy during execution
✔ Support least privilege (aka principle of least authority, POLA)
✔ Support formal verification of security policy enforcement
  ➢ Incl confidentiality, integrity, availability
✔ Performance comparable to monolithic systems
Secure Multi-Server OS Features

✓ Policy-mechanism separation:
  ➢ Servers implement abstractions independent of security policy
  ➢ Policy is encapsulated in a single security server

✓ Dynamic information-flow control:
  ➢ Communication limited by security policy

✓ Resource-availability guarantee through resource donation

✓ Performance by minimising security overhead
  ➢ Checks only on connection establishment
  ➢ Connection removed on policy change

✓ Design for formal verification

**Stay tuned for detailed white paper!**
Take-Aways:

✔ seL4 Foundation takes seL4 to the next level
  ➢ open development
  ➢ open governance
  ➢ community funding
  ➢ maturing ecosystem
  ➢ increasing deployments

✔ RISC-V is now a first-class seL4 architecture
  ➢ functional correctness done, other verification in progress

✔ Ambitious research agenda:
  ➢ provably eliminate timing channels
  ➢ secure, general-purpose multi-server OS

seL4: Defining the state of the art in secure OS since 2009
Questions?
Licensing: What Does the GPL Imply?

Valuable IP

Your application code

Foundation system services: libs, drivers, NW stacks...

Platform port:
- timer
- serial
- IRQ controller
- platform init

Boiler plate

GPL v2

Foundation kernel source

Any

Your system services: libs, drivers, ...

Any

kernel.org source
- core kernel
- device drivers
- NW stacks
- file systems
- ...

GPL v2

Platform port:
- device drivers
- platform init
- ...

GPL v2

Your system services: libs, drivers, ...

Your application code

BSD
What Does This Mean?

Kinds of properties proved

- Behaviour of C code is fully captured by abstract model
- Behaviour of C code is fully captured by executable model
- Kernel never fails, behaviour is always well-defined
  - assertions never fail
  - will never de-reference null pointer
  - will never access array out of bounds
  - cannot be subverted by misformed input
  - ...
- All syscalls terminate, reclaiming memory is safe, ...
- Well typed references, aligned objects, kernel always mapped...
- Access control is decidable

Can prove further properties on abstract level!
How Can I Use It?

✓ Open source (GPL v2): Download from https://github.com/sel4
✓ But keep in mind: seL4 is an OS microkernel and hypervisor, not an OS!
✓ Many OS components available on the seL4 GitHub
How Can I Use It?

- Open source (GPL v2): Download from https://github.com/sel4
- But keep in mind: seL4 is an OS microkernel and hypervisor, not an OS!
- Many OS components available on the seL4 GitHub
- Alternative: HENSOLDT Cyber’s TRENTO

![Diagram showing components]

TRENTO Components:
- Logger
- License Manager
- Secure Update
- TLS Stack
- Key Store
- Loader
- File System
- TCP/IP
- Secure Boot
- Crypto
- SD Driver
- E/N Driver

**Verified!**

MiG-V: HENSOLDT-Cyber RV64 Processor (based on Ariane)
So, Why Aren’t We Done?

Still only capability-based OS kernel with functional correctness proof

Still the world’s fastest microkernel!

- seL4’s verification provides the best possible guarantee of *spatial isolation*
- It says nothing about *temporal isolation*
What’s the Issue with Temporal Isolation?

**Safety: Timeliness**
- Execution interference

**Security: Confidentiality**
- Leakage via timing channels

Affect execution speed:
- Integrity violation – deadline miss

Observe execution speed:
- Confidentiality violation

Addressed by MCS kernel

Addressed by time protection
MCS Kernel: Capabilities for Time

Traditional seL4: Capabilities authorise access to spatial resources:
- Memory
- Threads
- Address spaces
- Communication endpoints
- Interrupts
- ...

MCS model: Capabilities also authorise CPU time
- Scheduling objects
Scheduling Contexts

Classical thread attributes

- Priority
- Time slice

New thread attributes

- Priority
- Scheduling context capability

Scheduling context object
T: period
C: budget (≤ T)

Scheduling-context object specifies CPU bandwidth limit

Ensure time available to lower-priority threads
Client is charged for server's time

Server runs on client's scheduling context

Scheduling-context capabilities: a principled, light-weight OS mechanism for managing time [Lyons et al, EuroSys’18]
Generally much cleaner model, cleans up a number of other things ⇒ Use for all new work!

- Verification getting close (Arm v7 and RV64)
- Legacy model will be *archived* once verification is done
Partition Hardware: Page Colouring

- Partitions get frames of disjoint colours preventing interference
- seL4: userland supplies kernel memory \(\Rightarrow\) colouring userland colours dynamic kernel memory
- Per-partition kernel image to colour kernel

[Ge et al. EuroSys’19]

Small amount of static kernel memory needs special handling
Temporal Partitioning: Flush on Switch

1. $T_0 = \text{current\_time()}$
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while ($T_0 + \text{WCET} < \text{current\_time()}$) ;
6. Reprogram timer
7. return

Latency depends on prior execution!

Ensure deterministic execution

Time padding to remove dependency

Must remove any history dependence!
Evaluation: Prime & Probe Attack

1. Fill cache with own data
2. Touch \( n \) cache lines
3. Traverse cache, measure *execution time*

Trojan encodes

Spy observes

Input signal

Output signal
Can We Verify Time Protection?

Assume we have:
• hardware that implements a suitable contract,
• a formal specification of that hardware,
  can we prove that our kernel eliminates all timing channels?
To prove: No two domains share hardware†
- Requires abstract model of partitionable hardware (cache model)
- \textit{Functional property, use existing techniques}

†Remaining shared kernel data needs separate argument

†Core idea: Convert timing channels into storage channels!
Proving Temporal Partitioning

1. \( T_0 = \text{current\_time}() \)
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while \((T0 + \text{WCET} < \text{current\_time}())\) ;
6. Reprogram timer
7. return

Prove: flush all non-partitioned HW
- Needs model of stateful HW
- Somewhat idealised on present HW
  ... but matches our Ariane
- **Functional property**

Prove: access to shared data is deterministic
- Each access sees same cache state
- Needs cache model
- **Functional property**

Prove: padding is correct – how?
Use Minimal Abstraction of Clocks

Abstract clock = monotonically increasing counter

Operations:
• Add constant to clock value
• Compare clock values

To prove: padding loop terminates as soon as $\text{clock} \geq T_0 + \text{WCET}$
• Functional property!
Status

✓ Published analysis of hardware mechanisms (APSys’18) – *Best Paper*
✓ Published time protection design and analysis (EuroSys’19) – *Best Paper*
  ◦ demonstrated effectiveness within limits set by hardware flaws (Arm, x86)
✓ Published planned approach to verification (HotOS’19)
✓ Published minimal hardware support for time protection (CARRV’20)
  ◦ evaluation demonstrated efficacy and performance

➢ Working on:
  ◦ Integrating time-protection mechanisms with clean seL4 model
    ◦ Done: Rebased experimental kernel off latest seL4 mainline (x86, Arm, RISC-V)
    ◦ In progress: Real system model that integrates the mechanisms
  ◦ Proving timing-channel absence (on conforming hardware)
    ◦ Done: Confidentiality proofs for flushing and time padding on simplified HW model
    ◦ In progress: Include pre-fetching of data
    ◦ To do: Extend to realistic hardware model