A Verified OS Kernel. Now What?

Gerwin Klein
The Team
The Team
L4 Verified

1 microkernel
8,700 lines of C
0 bugs*

qed

*conditions apply
An exception 06 has occurred at 0028:C11B3ADC in \xD DiskTSD(03) + 00001660. This was called from 0028:C11B40C8 in \xD voltrack(04) + 00000000. It may be possible to continue normally.

* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue
The Problem
Small Kernels

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...

- High assurance components in presence of other components

seL4 API:
- IPC
- Threads
- VM
- IRQ
- Capabilities

Untrusted

Legacy Apps

Linux Server

Trusted

Sensitive App

Trusted Service

Hardware
Small Kernels

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...

- High assurance components in presence of other components

seL4 API:
- IPC
- Threads
- VM
- IRQ
- Capabilities

Untrusted

- Legacy Apps
- Linux Server

Trusted

- Sensitive App
- Trusted Service

seL4

Hardware
The Proof
Functional Correctness

Proof

Specification

Code
Functional Correctness

What

Proof

specification

definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread

Functional Correctness

What

Proof

How

Specification

definition
schedule :: unit s_monad where
schedule ≡ do
  threads ← allActiveTCBs;
  thread ← select threads;
  switch_to_thread thread
od
OR switch_to_idle_thread

void
schedule(void) {
  switch ((word_t)ksSchedulerAction) {
    case (word_t)SchedulerAction.ResumeCurrentThread:
      break;
    case (word_t)SchedulerAction.ChooseNewThread:
      chooseThread();
      ksSchedulerAction = SchedulerAction.ResumeCurrentThread;
      break;
    default: /* SwitchToThread */
      switchToThread(ksSchedulerAction);
      ksSchedulerAction = SchedulerAction.ResumeCurrentThread;
      break;
  }
}

void
chooseThread(void) {
  prio_t prio;
  tcb_t *thread, *next;
*conditions apply
*conditions apply
*conditions apply

Assume correct:
- compiler + linker (wrt. C op-sem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)
Execution always defined:

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion

Not implied:

- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis
Implications

Execution always defined:
• no null pointer de-reference
• no buffer overflows
• no code injection
• no memory leaks/out of kernel memory
• no div by zero, no undefined shift
• no undefined execution
• no infinite loops/recursion

Not implied:
• “secure” (define secure)
• zero bugs from expectation to physical world
• covert channel analysis
Implications

Execution always defined:

- no null pointer de-reference
- no buffer overflows

Not implied:

- "secure" (define secure)
- zero bugs from expectation to physical world
- covert channel analysis
Implications

Execution always defined:

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion

Not implied:

- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis
Proof Architecture

Specification

Proof

C Code
Proof Architecture

Specification

Design

C Code
Proof Architecture

Access Control Spec

Specification

Design

C Code

Confinement
Proof Architecture

Access Control Spec

Specification

Design

C Code

Confinement

Haskell Prototype

Specification

Design

C Code

Access Control Spec

Confinement

Haskell Prototype
Proof Architecture

Access Control Spec

Confinement

Specification

definition
schedule :: unit s_monad where
schedule ≡ do
  threads ← allActiveTCBs;
  thread ← select threads;
  switch_to_thread thread
od
OR switch_to_idle_thread

Design

C Code

Haskell Prototype
Proof Architecture

Access Control Spec  ⊳  Confinement

Specification

Design

C Code

```
schedule :: Kernel ()
schedule = do
  action <- getSchedulerAction
  case action of
    ResumeCurrentThread -> return ()
    ChooseNewThread -> do
      chooseThread
      setSchedulerAction ResumeCurrentThread
    SwitchToThread t -> do
      switchToThread t
      setSchedulerAction ResumeCurrentThread

chooseThread :: Kernel ()
chooseThread = do
  r <- findM chooseThread' (reverse [minBound .. maxBound])
  when (r == Nothing) $ switchToIdleThread
  where
```
Proof Architecture

Access Control Spec

Confinement

Specification

Design

C Code

void schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChOOSE_NEW_THREAD:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
}
Did you find any Bugs?

Bugs found

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs

Effort

- Haskell design: 2 py
- First C impl.: 2 weeks
- Debugging/Testing: 2 months
- Kernel verification: 12 py
- Formal frameworks: 10 py
- Total: 25 py
Did you find any Bugs?

Bugs found

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs

Effort

- Haskell design: 2 py
- First C impl.: 2 weeks
- Debugging/Testing: 2 months
- Kernel verification: 12 py
- Formal frameworks: 10 py
- Total: 25 py
Access Control
Access Control
Proof Architecture

Access Control Spec

Specification

Design

C Code

Confinement

Haskell Prototype

Access Control Spec

Specification

Design

C Code
Lipton and Snyder:

- entities represented as nodes of a graph
- capabilities represented as edges of a graph
- rights are contained in capabilities

The Rights:

- Read
- Write
- Create
- Take
- Grant
Create $\in c_1$

Create new entity

Create $e^{(n)}$
Operations - Create

Create \( \in c_1 \)

Create new entity
Operations - Grant

Grant $\in c_1$ to $e_1$

Grant $c_2$ to $e_1$
with mask $R$
Operations - Grant

\[ \text{Grant} \in c_1 \]

\[ \text{Grant} \quad c_2 \text{ to } e_1 \]

\[ \text{diminish } c_2 R \]

\[ \text{with mask } R \]
Operations - Remove/Delete

Remove capability $c_2$

Delete entity $e_2$
Operations - Remove/Delete

**Remove** capability $c_2$

$$e \xrightarrow{c_1} e_1 \xrightarrow{} e_2$$

**Delete** entity $e_2$

$$e \xrightarrow{c_1} e_1 \xrightarrow{\text{Create } \in c_2} e_2$$
Operations - Remove/Delete

Remove capability $c_2$

Delete entity $e_2$
Questions

For any state in the future:

• Can E gain authority to do X?
• Can E gain more authority than it has?
• How much more?
• Can information flow from A to B?
Example
Example

Authority Barrier
Example

Diagram showing information flow and authority barrier.
Now What?
Current Proof
Even More Assurance?
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Compiler Verification
CompCert

Assembly Code
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
  - hardware (ARMv6)
  - cache and TLB management
  - boot code (1,200 loc)

Compiler Verification
CompCert

Assembly Code

Assembly Verification
ARM model, Fox et al
Verisoft
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
  - hardware (ARMv6)
  - cache and TLB management
  - boot code (1,200 loc)

Compiler Verification
CompCert

Assembly Code

Calling Conventions
Frame Conditions

Assembly Verification
ARM model, Fox et al
Verisoft
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
  - hardware (ARMv6)
  - cache and TLB management
  - boot code (1,200 loc)

Compiler Verification
CompCert

Assembly Verification
ARM model, Fox et al
Verisoft

Assembly Code
Coq
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Assembly Code  Cache/TLB model  Verisoft XT
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Assembly Code

Cache/TLB model

Verisoft XT
Even More Assurance?

Assume correct:
- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

boot result verification

Assembly Code

Cache/TLB model

Hardware Verification

VHDL?

Verisoft XT

VAMP
FM9001
Intel i7
Even More Assurance?

- compiler + linker (wrt. C opsem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

Assembly Code

Cache/TLB model

Boot result verification

Hardware Verification

VHDL?

Verisoft XT

VAMP
FM9001
Intel i7
Systems On Top
Systems On Top

Describes Binary Interface
Systems On Top

C/C++ progs

seL4/Linux

C system call bindings

Exists:
- standard seL4 library
- used in seL4/Linux
- not hard to formally verify
- verification scheduled
Systems On Top

C/C++ progs  seL4/Linux

C system call bindings

Haskell Programs

Haskell runtime

seL4/Haskell:
- early prototype Haskell runtime
- has seL4 systems call bindings
- verification hard
- runtime verification progress in HASP project @ PSU & Galois
Systems On Top

C/C++ progs

seL4/Linux

C system call bindings

Haskell Programs

Haskell runtime

Java Programs

JVM

seL4/JVM:
- any takers?
- JVM extensively formalised
- widely used
- EAL7 smart card implementations exist
Other Architectures
Other Architectures

**seL4/x86:**
- x86 version exists, supports Linux
- verification likely, not started yet

Intel 32bit
Other Architectures

**seL4/x86:**
- x86 version exists, supports Linux
- verification likely, not started yet
- Intel VT-d/IOMMU implemented
- enables untrusted device DMA
- verification possible

Intel 32bit + IOMMU
Other Architectures

Intel 32bit + IOMMU + multi core

**seL4/x86:**
- x86 version exists, supports Linux
- verification likely, not started yet

- Intel VT-d/IOMMU implemented
- enables untrusted device DMA
- verification possible

- experimental multi processor version
- initial proofs exist
Other Architectures

Intel 32bit + IOMMU + multi core

**seL4/x86:**
- x86 version exists, supports Linux
- verification likely, not started yet
- Intel VT-d/IOMMU implemented
- enables untrusted device DMA
- verification possible
- experimental multi processor version
- initial proofs exist

**New Architectures**
- ca 1/3 of seL4 arch dependent
- close to ARM
  - easy to implement and verify
Looking Forward
Looking Forward
Trustworthy Embedded Systems

- L4.verified: functional correctness 10,000 loc
- Next step: formal guarantees for > 1,000,000 loc
Exploit:

- seL4 isolation
- verified properties
- MILS architectures
Challenges

• Find right architecture

• Security analysis
  – identify trusted components
  – ideally take-grant style
  – behaviour of trusted components

• Code-level theorem in the end
  – connect to kernel proof
  – ideally prove trusted component only
Example System

• Scenario:
Example System

• Scenario:
Example System

- Multilevel Secure Access Device
Example System

• Multilevel Secure Access Device

Secure Access Controller

Currently selected connection: No Active Connection

Switch to Network:
- US
- EU
- AUS
- ASIA
- WWW

Login as: hotspy007
Logout
Example System

• Multilevel Secure Access Device

Secure Access Controller

Currently selected connection: No Active Connection

Switch to Network:
- US
- EU
- AUS
- ASIA
- WWW

Login as: hotspy007

Logout
Example System

- Multilevel Secure Access Device

Secure Access Controller

Currently selected connection: No Active Connection

Switch to Network:
- US
- EU
- AUS
- ASIA
- WWW

Login as: hotspy007
Logout
Example System

- Multilevel Secure Access Device

Secure Access Controller

Login as: hotspy007
Logout

The Network has been successfully switched to EU
Currently selected connection: EU

Switch to Network:
- US
- EU
- AUS
- ASIA
- WWW

SAC
Example System

• Multilevel Secure Access Device

Secure Access Controller

The Network has been successfully switched to EU
Currently selected connection: EU

Switch to Network:
- US
- EU
- AUS
- ASIA
- WWW

Login as: hotspy007
Logout

The Network has been successfully switched to EU
Currently selected connection: EU
Example System

• Multilevel Secure Access Device
Example System

- Multilevel Secure Access Device

Secure Access Controller

- The Network has been successfully switched to **WWW**
- Currently selected connection: **WWW**

Switch to Network:

- US
- EU
- AUS
- ASIA
- WWW

Login as: hotspy007
Logout

Currently selected connection: **WWW**
The Network has been successfully switched to **WWW**
SAC System

Components
SAC System

Components

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
SAC System

Components

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal

Web server

SAC System

Net-A

Net-B

NIC-A

NIC-B

NIC-C

NIC-D

CT

DT
**SAC System**

**Components**

Net-A = Network A  
Net-B = Network B  
NIC-A = Network Card for Network A  
NIC-B = Network Card for Network B  
NIC-C = Control Network Card  
NIC-D = Data Network Card  
CT = Control Terminal  
DT = Data Terminal

Web server  
packet routing
SAC System

Components

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
SAC System

Desired Property

No information flow between providers A and B through SAC even if they collaborate.

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
SAC System

Desired Property

No information flow between providers A and B through SAC even if they collaborate

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
**SAC System**

**Desired Property**

No information flow between providers A and B through SAC even if they collaborate.

**Proving all this correct?**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net-A</td>
<td>Network A</td>
</tr>
<tr>
<td>Net-B</td>
<td>Network B</td>
</tr>
<tr>
<td>NIC-A</td>
<td>Network Card for Network A</td>
</tr>
<tr>
<td>NIC-B</td>
<td>Network Card for Network B</td>
</tr>
<tr>
<td>NIC-C</td>
<td>Control Network Card</td>
</tr>
<tr>
<td>NIC-D</td>
<td>Data Network Card</td>
</tr>
<tr>
<td>CT</td>
<td>Control Terminal</td>
</tr>
<tr>
<td>DT</td>
<td>Data Terminal</td>
</tr>
</tbody>
</table>

**Web server**

**Packet routing**

**driverA**

**driverB**

**driverD**
SAC System

Desired Property

No information flow between providers A and B through SAC even if they collaborate

Proving all this correct?

NO!

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
SAC System

Design

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
SAC System

Design

SAC-Controller: Embedded Linux + Web Server UI

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
The SAC System design includes the following components:

- **Net-A**: Network A
- **Net-B**: Network B
- **NIC-A**: Network Card for Network A
- **NIC-B**: Network Card for Network B
- **NIC-C**: Control Network Card
- **NIC-D**: Data Network Card
- **CT**: Control Terminal
- **DT**: Data Terminal
- **SAC-C**: SAC Controller
- **R**: Router
- **RM**: Router Manager
- **Embedded Linux**: Used for routing and driver support

The SAC System utilizes a router (R) that runs Embedded Linux to manage network routing and driver operations.
SAC System

Design

Net-A = Network A  NIC-C = Control Network Card  R = Router
Net-B = Network B  NIC-D = Data Network Card  RM = Router Manager
NIC-A = Network Card for Network A  CT = Control Terminal  SAC-C = SAC Controller
NIC-B = Network Card for Network B  DT = Data Terminal
SAC System

Design

Net-A = Network A  NIC-C = Control Network Card  R= Router
Net-B = Network B  NIC-D = Data Network Card  RM = Router Manager
NIC-A = Network Card for Network A  CT = Control Terminal  SAC-C = SAC Controller
NIC-B = Network Card for Network B  DT = Data Terminal
SAC System

Design

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
SAC-C = SAC Controller
R = Router
RM = Router Manager
SAC-C = SAC Controller

COMPONENTS

has access to
SAC System

Trusted Components

Router Manager:
< 2kloc
only trusted component

Net-A = Network A
Net-B = Network B
NIC-A = Network Card for Network A
NIC-B = Network Card for Network B
NIC-C = Control Network Card
NIC-D = Data Network Card
CT = Control Terminal
DT = Data Terminal
RM = Router Manager
SAC-C = SAC Controller
R = Router

Net-A
Net-B
NIC-A
NIC-B
NIC-C
NIC-D
TIMER
R
SAC-C
RM
CT
DT

has access to
Low-Level Design
Goal: No information flowing between providers A and B
Assumption: Info flow through front-end terminal is trusted
Plain Take-Grant Analysis
Plain Take-Grant Analysis
Plain Take-Grant Analysis

Need to know trusted component behaviour
Security Goal

Approach:
- label-based security
  - tag as ‘contaminated’ if may contain data from Net-A
  - NIC-A always contaminated
- Goal: prove NIC-B always ‘not contaminated’
Security Analysis

Rules:

A → B

A → B

A → B

A → B

A → B

A → B

A → B

A → B

A → B

A → B
Life Cycle

[Diagram showing the life cycle with nodes labeled RM, SAC-C, NIC-C, NIC-D, Net-A, Net-B, R-code, (R-code), and TIMER CHIP. Arrows indicate the flow of information or processes between these nodes.]
Life Cycle

CT authenticates with SAC-C

CT sends a request to switch to Net-A
Life Cycle

RM receives request to switch to Net-A
Life Cycle

RM creates a new Router instance R
(and gets full rights to the newly created object)
Life Cycle

RM initializes the Router instance.
RM grants to R its rights to NIC-A, NIC-D, R-mem, R-code, TIMER, and itself.
DT starts to communicate with Net-A, through R. This may imply the creation of new objects using R-mem and granting caps to them.
CT sends a request to switch to Net-B (while DT still communicates with Net-A through R)
Life Cycle

RM receives request to switch to Net-B
RM revokes all caps given to R (using the cap used to create R)
Life Cycle

RM revokes caps of R-mem

(using create cap to R-mem)
Life Cycle

RM deletes R and R-mem

NIC-A

NIC-B

Net-A

Net-B

RM

NIC-A

NIC-B

SAC-C

NIC-C

NIC-D

CT

DT

R-code

(R)

(R-mem)

R-code

R

RW

TIMER

CHIP

RW
Life Cycle

RM flushes NIC-A, NIC-B and NIC-D

Net-A

Net-B

NIC-A

NIC-B

NIC-C

NIC-D

RM

SAC-C

R-code

(R-mem)

R-code

(R)

RW

RW

RW

RW

TIMER

TIMER

CHIP

CT

DT

RW

RW

RW

RW

rw

rw

rw

rw

r

r

r

r

c

c

R

R

R

R

w

w

w

w

R

R

R

R
RM creates a new Router instance \( R \)

(and gets full rights to the newly created object)
RM grants to R its rights to NIC-B, NIC-D, NIC-A, NIC-A, and TIMER and itself.
Life Cycle

RM grants to R its rights to NIC-B, NIC-D, R-mem, R-code, TIMER and itself.
DT starts to communicate with Net-B, through R.

This may imply the creation of new objects using R-mem and granting some caps to them.
So far
So far

• Can build systems with
  – large untrusted components
  – plus few small, trusted components
  – trusted = needs behaviour spec
So far

• Can build systems with
  – large untrusted components
  – plus few small, trusted components
  – trusted = needs behaviour spec

• Use take-grant to model security
  – can simulate system
  – modelling already finds bugs
  – high-level proof in Isabelle/HOL or SPIN
  – includes behaviour of trusted component
Future
Future

- Need to verify low-level design
Future

- Need to verify low-level design

- Building tool-chain for:
  - describing cap layout (capDL)
  - generating booter
  - generating booter proof
  - abstraction to take-grant

NICTA Copyright 2010

From imagination to impact
More Future
More Future

- Verify Trusted Component
More Future

• Verify Trusted Component
  – interface with kernel
  – use most abstract level possible
  – make sure sec property preserved by refinement
Summary
Summary

Formal proof all the way from spec to C.

- **200kloc** handwritten, machine-checked proof
- ~**460** bugs (160 in C)
- Verification on **code**, **design**, and **spec**
- Systems with **trusted components**
- **The future**: formal proof for large systems down to code

Formal Code Verification up to 10kloc:

  - It works.
  - It’s feasible.
  - It’s fun.
Thank You
Thank You