10 Years of Trustworthy Systems

David Cock

July 29, 2014
Outline

1 seL4
   The C Kernel
   Bitfield DSL

2 Lyrebird
   Simulation for Verification
   Modelling ARM

3 Probability & Security
   Side Channels
   Remote Exploits
   pGCL

4 Projects
   Vlibc
   Opportunistic Verification

5 Questions
First ever verified kernel.

- Written in C — high-performance.
- Verified in Isabelle/HOL.
First ever verified kernel.

- Written in C — high-performance.
- Verified in Isabelle/HOL.
- Open source from yesterday!

http://sel4.systems
The C kernel implements the high-level specification.

- Initial implementation — 2 weeks.
- Small — 8,700 lines.
- Fast — 224cyc one-way IPC.
- DSL automation — bitfields.
The seL4 Call Graph
The seL4 Call Graph
• 573 functions.

• Not modular — No SCCs.......except those leaves.
• 573 functions.
• Not modular — No SCCs... except those leaves.
• 198 of these: 35% of functions, 16% of LOC.
• 573 functions.
• Not modular — No SCCs......except those leaves.
• 198 of these: 35% of functions, 16% of LOC.
• Generated and proved automatically from a DSL.
• **DSL:**

```plaintext
base 32
block B { padding 3 field Y 13 field Z 16 }
```
● DSL:

base 32
block B { padding 3 field Y 13 field Z 16 }

● C:

static inline void
B_ptr_set_X(B_t *B_ptr, uint32_t v) {
    B_ptr->words[0] &= ~0x1fff0000;
    B_ptr->words[0] |= (v<<16)&0x1fff0000;
}
- **DSL:**

  base 32
  block B { padding 3 field Y 13 field Z 16 }

- **C:**

  ```c
  static inline void
  B_ptr_set_X(B_t *B_ptr, uint32_t v) {
      B_ptr->words[0] &= ~0x1fff0000;
      B_ptr->words[0] |= (v<<16)&0x1fff0000;
  }
  ```

- **HOL:**

  \[
  B_{\text{lift}} \equiv (\ |
  B_{\text{CL}}.X_C L = ((\text{index} \ (B_{\text{C}}.\text{words}_{\text{C}} B) \ 0) \gg 16) \ \text{AND} \ 8191, \\
  B_{\text{CL}}.Y_C L = ((\text{index} \ (B_{\text{C}}.\text{words}_{\text{C}} B) \ 0) \gg 0) \ \text{AND} \ 65535)
  \]
Automation Helps!

- 35% of the functions in seL4 were proved automatically.
- The tool is now widely used in NICTA.
- It’s used by engineers, not formal methods people.
- Many features not mentioned: tagged unions, multilevel decoding, . . . .
Further Reading

For more see:

- **Running the manual: An approach to high-assurance microkernel development**, Haskell Workshop ’06.
- **Bitfields and tagged unions in C: verification through automatic generation**, VERIFY’08.
- **Secure microkernels, state monads and scalable refinement**, TPHOLS’08.
- **Mind the gap: A verification framework for low-level C**, TPHOLS’09.
- **seL4: Formal verification of an OS kernel**, SOSP’09.
Outline

1. seL4
   - The C Kernel
   - Bitfield DSL

2. Lyrebird
   - Simulation for Verification
   - Modelling ARM

3. Probability & Security
   - Side Channels
   - Remote Exploits
   - pGCL

4. Projects
   - Vlibc
   - Opportunistic Verification

5. Questions
Lyrebird

- A DSL for CPU/system modelling.
- High performance simulator.
- Automatic formal model.
- Used to prototype seL4.
Program proof is important, but there’s more to do.
Program proof is important, but there’s more to do.
Program proof is important, but there’s more to do.

Any statement "P is True" is incomplete: It must be read as "I, under Q - my model of the world".

Goal

*Development outcomes: program, proof and model.*
Program proof is important, but there’s more to do.
Program proof is important, but there’s more to do.

Our approach is a language framework: *Lyrebird*.
10 Years of Trustworthy Systems

David Cock

seL4
The C Kernel
Bitfield DSL

Lyrebird
Simulation for Verification
Modeling
ARM

Probability & Security
Side Channels
Remote Exploits
pGCL

Projects
Vlibc
Opportunistic Verification

Questions
Abstract
Haskell
C
Proof
Model
Lyrebird
λ
10 Years of Trustworthy Systems

David Cock

seL4
The C Kernel
Bitfield DSL

Lyrebird
Simulation for Verification
Modelling
ARM

Probability & Security
Side Channels
Remote Exploits
pGCL

Projects
Vlibc
Opportunistic Verification

Questions
A simple model of a CPU connected to RAM.
Modules are written in Lyrebird.
The cycle specifies asynchronous behaviour.
Modules export instructions.
All behaviour is built from register transfers.
module vsr;
cycle {
  Memory.Read[[PC, Instr]];
  decode_execute VSR;
}

instruction ADD {
  execute { Ra <- Rb + Rc; }
}

instruction LDR {
  execute { Memory.Read[[Rb,Ra]]; }
}

Modules are linked by **interfaces**.
Interfaces define **transactions**.
Transactions access the **datapath**.
Interfaces and modules allow different implementations.
Lyrebird can also be used to model devices.
Register types have explicit width.
Type-checked macros minimize duplication.
Transactions are implemented by modules.
• We have an ARMv6 user-level integer instruction model.
• Floating-point and vector operations are excluded.
• The complete model is approximately 1600 lines.
• We used it to validate the seL4 Haskell prototype.
Register transfer is easy to simulate.
The simulator is portable and fast — 10MIPS for ARMv6 user.
The output is a single C module;
It is easily incorporated into larger simulations.
Further Reading

For more see:

- *Lyrebird* — *assigning meanings to machines*, SSV’10
Outline

1. seL4
   The C Kernel
   Bitfield DSL

2. Lyrebird
   Simulation for Verification
   Modelling ARM

3. Probability & Security
   Side Channels
   Remote Exploits
   pGCL

4. Projects
   Vlibc
   Opportunistic Verification

5. Questions
The attacker tries to guess the lock combination.
After $n$ tries he’s locked out.
The Problem

Every guess leaks something about the combination.
It's easy to spot a cache miss.

Cache contention forms a channel.

This is a big problem in crypto e.g. AES.

We ran a large empirical evaluation:

• 3 channels, 2 countermeasures and 5 platforms.
• 6 months of observations.
• Integrated with regression tests.

The Cache Channel
The Cache Channel

- It's easy to spot a cache miss.
• It’s easy to spot a cache miss.
• Cache contention forms a channel.
The Cache Channel

- It's easy to spot a cache miss.
- Cache contention forms a channel.
- This is a big problem in crypto e.g. AES.
The Cache Channel

- It’s easy to spot a cache miss.
- Cache contention forms a channel.
- This is a big problem in crypto e.g. AES.

We ran a large empirical evaluation:
- 3 channels, 2 countermeasures and 5 platforms.
- 6 months of observations
- Integrated with regression tests.
<table>
<thead>
<tr>
<th>Processor</th>
<th>iMX.31</th>
<th>E6550</th>
<th>DM3730</th>
<th>AM3358</th>
<th>Exynos4412</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Freescale</td>
<td>Intel</td>
<td>TI</td>
<td>TI</td>
<td>Samsung</td>
</tr>
<tr>
<td>Architecture</td>
<td>ARMv6</td>
<td>x86-64</td>
<td>ARMv7</td>
<td>ARMv7</td>
<td>Cortex A9</td>
</tr>
<tr>
<td>Core type</td>
<td>ARM1136JF-S</td>
<td>Conroe</td>
<td>Cortex A8</td>
<td>Cortex A8</td>
<td>Cortex A9</td>
</tr>
<tr>
<td>Released</td>
<td>2005</td>
<td>2007</td>
<td>2010</td>
<td>2011</td>
<td>2012</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Clock rate</td>
<td>532 MHz</td>
<td>2.33 GHz</td>
<td>1 GHz</td>
<td>720 MHz</td>
<td>1.4 GHz</td>
</tr>
<tr>
<td>Timeslice</td>
<td>1 ms</td>
<td>2 ms</td>
<td>1 ms</td>
<td>1 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>RAM</td>
<td>128 MiB</td>
<td>1024 MiB</td>
<td>512 MiB</td>
<td>256 MiB</td>
<td>1024 MiB</td>
</tr>
</tbody>
</table>

**L1 D-cache**

| size | 16 KiB | 32 KiB | 32 KiB | 32 KiB | 32 KiB |
| index | virtual | physical | virtual | virtual | virtual |
| tag | physical | physical | physical | physical | physical |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 512 | 512 | 512 | 1024 | 512 |
| associativity | 4 | 8 | 4 | 4 | 4 |
| sets | 128 | 64 | 128 | 128 | 256 |

**L2 cache**

| size | 128 KiB | 4096 KiB | 256 KiB | 256 KiB | 1024 KiB |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 4096 | 65.536 | 4096 | 4096 | 32,768 |
| associativity | 8 | 16 | 8 | 8 | 16 |
| sets | 512 | 4096 | 512 | 512 | 2048 |
| colours | 4 | 64 | 8 | 8 | 16 |

**Table:** Experimental platforms.
<table>
<thead>
<tr>
<th>Processor</th>
<th>iMX.31</th>
<th>E6550</th>
<th>DM3730</th>
<th>AM3358</th>
<th>Exynos4412</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Freescale</td>
<td>Intel</td>
<td>TI</td>
<td>TI</td>
<td>Samsung</td>
</tr>
<tr>
<td>Architecture</td>
<td>ARMv6</td>
<td>x86-64</td>
<td>ARMv7</td>
<td>ARMv7</td>
<td>ARMv7</td>
</tr>
<tr>
<td>Core type</td>
<td>ARM1136JF-S</td>
<td>Conroe</td>
<td>Cortex A8</td>
<td>Cortex A8</td>
<td>Cortex A9</td>
</tr>
<tr>
<td>Released</td>
<td>2005</td>
<td>2007</td>
<td>2010</td>
<td>2011</td>
<td>2012</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Clock rate</td>
<td>532 MHz</td>
<td>2.33 GHz</td>
<td>1 GHz</td>
<td>720 MHz</td>
<td>1.4 GHz</td>
</tr>
<tr>
<td>Timeslice</td>
<td>1 ms</td>
<td>2 ms</td>
<td>1 ms</td>
<td>1 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>RAM</td>
<td>128 MiB</td>
<td>1024 MiB</td>
<td>512 MiB</td>
<td>256 MiB</td>
<td>1024 MiB</td>
</tr>
</tbody>
</table>

**L1 D-cache**

| size | 16 KiB | 32 KiB | 32 KiB | 32 KiB | 32 KiB |
| index | virtual | physical | virtual | virtual | virtual |
| tag | physical | physical | physical | physical | physical |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 512 | 512 | 512 | 1024 | 512 |
| associativity | 4 | 8 | 4 | 4 | 4 |
| sets | 128 | 64 | 128 | 128 | 256 |

**L2 cache**

| size | 128 KiB | 4096 KiB | 256 KiB | 256 KiB | 1024 KiB |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 4096 | 65,536 | 4096 | 4096 | 32,768 |
| associativity | 8 | 16 | 8 | 8 | 16 |
| sets | 512 | 4096 | 512 | 512 | 2048 |
| colours | 4 | 64 | 8 | 8 | 16 |

**Table:** Experimental platforms.
Exynos4 Cache Channel

Bandwidth: 7.04kb/s
Cache Colouring

address set wayline
{
offset page {
colour 7
1 2 3
1
2
3
4
5
6
7
6
5
4
3
2
1
0

line
0
1
2
3
Way

offset
0
1
2
3

colour
1 2 3 4
1
2
3

page
5
6
7

address

set
0
1
2
3

Coloured Cache Channel

Bandwidth: 81.3 b/s
Residual TLB Channel

Lines touched -12290
No TLB flush
TLB flush

Stalls/line
No TLB flush
TLB flush
The Lucky-13 Attack

This is a recent vulnerability in OpenSSL TLS.

- Runtime depends on *unvalidated* user input.
- Can be used as a decryption oracle.
- ‘Fixed’ with a constant-time algorithm.
- We reproduced the attack on seL4...
The Lucky-13 Attack

This is a recent vulnerability in OpenSSL TLS.

- Runtime depends on *unvalidated* user input.
- Can be used as a decryption oracle.
- ‘Fixed’ with a constant-time algorithm.
- We reproduced the attack on seL4...
- ...and fixed it with better performance!
- Required no modifications to OpenSSL.
The Lucky-13 Attack

![Graph showing response time vs. probability](image-url)
10 Years of
Trustworthy
Systems

David Cock

seL4
The C Kernel
Bitfield DSL

Lyrebird
Simulation for Verification
Modelling
ARM

Probability & Security
Side Channels
Remote Exploits

pGCL

Projects
Vlibc
Opportunistic Verification

Questions

Intercontinental Attack

![Graph showing response time (ms) vs. p/ms for M0 and M1]
Scheduled Delivery

\[
\text{in} \rightarrow \text{SSL_read} \rightarrow \text{handler} \rightarrow \text{SSL_write} \leftarrow \text{server} \leftarrow \text{out}
\]

\[
\text{network} \quad \text{TLS} \quad \text{application}
\]
Scheduled Delivery

network → SSL_read → handler

out ← SSL_write ← server

1

SSL_read

handler

SSL_write

server

network

TLS

application
Scheduled Delivery

Scheduled Delivery

in -> SSL_read -> handler

1

SSL_write <- server

out <-

network TLS application
Scheduled Delivery

1. in → SSL_read → handler
2. C
3. network

SSL_write ← Server

network TLS application
Scheduled Delivery

network \quad TLS \quad application
10 Years of Trustworthy Systems

David Cock

seL4
The C Kernel
Bitfield DSL

Lyrebird
Simulation for Verification
Modelling
ARM

Probability & Security
Side Channels
Remote Exploits
pGCL

Projects
Vlibc
Opportunistic Verification

Questions

Scheduled Delivery

network → TLS → application

in → SSL_read → handler

out ← SSL_write ← server

1 3 4

+Δt

2 5
Lucky-13 Mitigated

Response time (µs)

M0 VL
M1 VL
M0 CT
M1 CT
M0 SD
M1 SD
Load Performance

![Load Performance Graph](image)

- **1.0.1c**
- **1.0.1e**
- **1.0.1c-sd 1 thread**
- **1.0.1c-sd 2 thread**

**Axes:**
- **CPU load** vs **Ingress rate (p/s)**
Further Reading

For more see:

- *Exploitation as an inference problem, AISEC,11.*
- pGCL is a language of probabilistic automata.
- It models both demonic and probabilistic choice.
- My Isabelle/HOL formalisation is now in the Archive of Formal Proofs.
- Used to formally verify probabilistic security properties e.g. side channel leakage.
For more see:

- **Verifying probabilistic correctness in Isabelle with pGCL**, SSV’12
- **From probabilistic operational semantics to information theory - side channels with pGCL in isabelle**, ITP’14
- **pGCL for Isabelle**, Archive of Formal Proofs, 2014
10 Years of Trustworthy Systems

David Cock

Outline

1. seL4
   - The C Kernel
   - Bitfield DSL

2. Lyrebird
   - Simulation for Verification
   - Modelling ARM

3. Probability & Security
   - Side Channels
   - Remote Exploits
   - pGCL

4. Projects
   - Vlibc
   - Opportunistic Verification

5. Questions
Can We Verify the C Library?

An open project:

- Work in a public repository.
- Code only accepted with proof.
- Self-contained student projects.
Can We Verify the C Library?

An open project:

- Work in a public repository.
- Code only accepted with proof.
- Self-contained student projects.

Applications:

- Systems on a verified kernel.
- Library spec for symbolic execution (no tracing libc!).
- Verified compiler (CompCert) needs a verified runtime.
Getting Value out of FM

You don’t have to do seL4 to benefit from FM:

- Go for bang/buck.
- Focus on things likely to be wrong.
- Provide a toolset to programmers.
You don’t have to do seL4 to benefit from FM:

- Go for bang/buck.
- Focus on things likely to be wrong.
- Provide a toolset to programmers.

**DSLs provide a convenient interface:**

- We’ve seen examples: Bitfields, Lyrebird, ...
- Match to tool to the job.
- Full formalism isn’t exposed to programmers.
- Don’t force everything into a single framework: provide tools!
Outline

1. seL4
   The C Kernel
   Bitfield DSL

2. Lyrebird
   Simulation for Verification
   Modelling ARM

3. Probability & Security
   Side Channels
   Remote Exploits
   pGCL

4. Projects
   Vlibc
   Opportunistic Verification

5. Questions
Questions?