Towards proving security in the presence of large untrusted components

June Andronick
David Greenaway
Kevin Elphinstone
Computers and Trust
Computers and Trust
Computers and Trust
Computers and Trust
Computers and Trust

• Advances in formal methods techniques give us hope

• The seL4 microkernel is one such example: around 10 thousand lines of code formally proven
  – approximately 25 person years of effort
Computers and Trust

• Advances in formal methods techniques give us hope

• The seL4 microkernel is one such example: around 10 thousand lines of code formally proven
  – approximately 25 person years of effort

• A typical smartphone consists of over 10 million lines of code
Computers and Trust

• Advances in formal methods techniques give us hope

• The seL4 microkernel is one such example: around 10 thousand lines of code formally proven – approximately 25 person years of effort

• A typical smartphone consists of over 10 million lines of code
Computers and Trust

• Advances in formal methods techniques give us hope

• The seL4 microkernel is one such example: around 10 thousand lines of code formally proven
  – approximately 25 person years of effort

• A typical smartphone consists of over 10 million lines of code

How can we provide any formal assurance to real-world systems of such size?
Our Vision

• Provide full system guarantees for targeted properties

• Isolate the software parts that are not critical to the target property
  – And then prove that nothing more needs to be said about it

• Formally prove that the remaining parts satisfy the target property
Case Study: Secure Access Controller

Classified Networks

1  2  3  4

User Terminal
Case Study: Secure Access Controller

Classified Networks

1  2  3  4

User Terminal
Case Study: Secure Access Controller
Case Study: Secure Access Controller

- Data from one classified network must not reach another

- Assumptions:
  - User terminal will not leak data
  - Only verify overt communication channels
  - All networks are otherwise malicious
Case Study: Secure Access Controller

SAC
Case Study: Secure Access Controller

Gigabit Network Card Drivers

10,000 Loc

SAC
Case Study: Secure Access Controller

Gigabit Network Card Drivers

10,000 LOC

SAC

TCP/IP Stack

20,000 LOC
Case Study: Secure Access Controller

- Gigabit Network Card Drivers (10,000 LoC)
- Network Address Translation (10,000 LoC)
- TCP/IP Stack (20,000 LoC)

SAC
Case Study: Secure Access Controller

- Gigabit Network Card Drivers: 10,000 LoC
- Web Server: 5,000 LoC
- TCP/IP Stack: 20,000 LoC
- Network Address Translation: 10,000 LoC

SAC
Case Study: Secure Access Controller

Gigabit Network Card Drivers
10,000 LoC

SSL Support
25,000 LoC

Network Address Translation
10,000 LoC

Web Server
5000 LoC

TCP/IP Stack
20,000 LoC

SAC

9
Case Study: Secure Access Controller

Gigabit Network Card Drivers
10,000 LoC

SSL Support
25,000 LoC

Network Address Translation
10,000 LoC

Web Server
5000 LoC

TCP/IP Stack
20,000 LoC

DHCP
3000 LoC

SAC
Case Study: Secure Access Controller

Control Network

Confidential Networks

Nic-C

Nic-A

Nic-B

Nic-D

User Terminal
Case Study: Secure Access Controller
Case Study: Secure Access Controller

Control Network

Confidential Networks

Nic-C

Nic-A

Nic-B

Nic-D

User Terminal
Case Study: Secure Access Controller

Control Network

Nic-C

Confidential Networks

Nic-A

Nic-B

Nic-D

User Terminal

X

X
Case Study: Secure Access Controller

- Verification of all code in the system is infeasible

- Instead, split up code into components
  - Trusted / untrusted components
  - Only give components access to resources they need
  - Principle of least privilege

- To do this, we need some mechanism to enforce this split
seL4 Microkernel

- Small operating system kernel
  - Threads
  - Address Spaces
  - Communication primitives

- Capability based
  - All system resources require a cap to be accessed
  - Provides access control, allowing threads to be isolated by using an appropriate cap distribution

- Proven functionally correct
  - seL4’s C code shown to correctly implement its specification
    - Assumes correctness of hardware, compiler, initialisation code, assembly paths
SAC Security Architecture

Control Network

Confidential Networks

User Terminal

Nic-C

Nic-A

Nic-B

Nic-D
SAC Security Architecture

Control Network

Nic-C

Confidential Networks

Nic-A

Nic-B

Router

User Terminal

Nic-D
SAC Security Architecture

Control Network

Confidential Networks

Nic-C

Nic-A

Nic-B

Router

Nic-D

User Terminal
SAC Security Architecture

Control Network

Nic-C

Confidential Networks

Nic-A

Nic-B

User Terminal

Nic-D
SAC Security Architecture

Control Network
Nic-C

Confidential Networks
Nic-A
Nic-B

Router

User Terminal
Nic-D
SAC Security Architecture

Control Network

Nic-C

Confidential Networks

Nic-A

Nic-B

Router Manager

Router

Nic-D

User Terminal
SAC Security Architecture

Control Network

Nic-C

SAC Controller

Router Manager

Confidential Networks

Nic-A

Nic-B

Router

Nic-D

User Terminal
SAC Security Architecture

Control Network

Nic-C

SAC Controller

Router Manager

User Terminal

Confidential Networks

Nic-A

Router

Nic-B

Nic-D

Timer Server
SAC Prototype

Secure Access Controller

Currently selected connection: Network 2

Available Networks:
- Disconnect from all
- Network 1
- Network 2

Switch Network
SAC Prototype

- **Router**
  - Virtualised Linux
  - Routing Code / NAT

- **SAC Controller**
  - Virtualised Linux
  - mini-httpd / OpenSSL

- **Timer**
  - Hand-written C

- **Router Manager**
  - Hand-written C

- **seL4 Kernel**
  - Hand-written C
SAC Prototype

• Router
  – Virtualised Linux
  – Routing Code / NAT
  } 10,000,000 LoC

• SAC Controller
  – Virtualised Linux
  – mini-httpd / OpenSSL
  } 10,000,000 LoC

• Timer
  – Hand-written C
  } 300 LoC

• Router Manager
  – Hand-written C
  } 1500 LoC

• seL4 Kernel
  – Hand-written C
  } 8300 LoC
SAC Prototype

- Router
  - Virtualised Linux
  - Routing Code / NAT
  \[10,000,000\text{ LoC}\]

- SAC Controller
  - Virtualised Linux
  - mini-httpd / OpenSSL
  \[10,000,000\text{ LoC}\]

- Timer
  - Hand-written C
  \[300\text{ LoC}\]

- Router Manager
  - Hand-written C
  \[1500\text{ LoC}\]

- seL4 Kernel
  - Hand-written C
  \[8300\text{ LoC}\]

\(\sim 20,000,000\text{ lines of code}\)
\(\sim 10,000\text{ lines of code}\)
Full System Verification

- Merely *reducing* the amount of code isn’t sufficient to provide any security guarantee
- Our goal is to provide a formal guarantee
- How can we achieve this?
Full System Verification
Full System Verification
Full System Verification
Full System Verification

- Hardware
- seL4 kernel
- Components
- Formal Cap

Components

seL4 kernel

Hardware

NICTA Copyright 2010

From imagination to impact
Full System Verification
Full System Verification

- Components
- seL4 kernel
- Hardware

1. Formal Cap Distribution
2. Security Architecture
3. Trusted Component Behaviour
4. Kernel Security Model

- Trusted Component
- Component Code

NICTA Copyright 2010
From imagination to impact
Full System Verification

- Component Code
- seL4 kernel
- Hardware
- Components
- Trusted Component
- ① Formal Cap Distribution
- ② Security Architecture
- ③ Trusted Component Behaviour
- ④ Kernel Security Model
- ⑤ Formal Security Property
High Level System Model

RM_id  -> Some (\{rw_to_NIC_A, rw_to_NIC_B, ...\}, not_contaminated)
SAC_C_id -> Some (\{rw_to_NIC_C, w_to_RM, ...\}, not_contaminated)
TIMER_id  -> Some (\{w_to_SAC_C, w_to_RM, ...\}, not_contaminated)
ROUTER_id -> None
NIC_A_id  -> Some (\{}, contaminated\}
NIC_B_id  -> Some (\{}, contaminated\}
NIC_C_id  -> Some (\{}, not_contaminated\}
NIC_D_id  -> Some (\{}, not_contaminated\}
High Level System Model

Control Network

- Nic-C
  - SAC Controller
  - Router Manager
  - Timer Server

Classified Networks

- Nic-A
  - Nic-B
  - Router

User Terminal

```
RM_id       -> Some ({rw_to_NIC_A, rw_to_NIC_B, ...}, not_contaminated)
SAC_C_id    -> Some ({rw_to_NIC_C, w_to_RM, ...}, not_contaminated)
TIMER_id    -> Some ({w_to_SAC_C, w_to_RM, ...}, not_contaminated)
ROUTER_id   -> None
NIC_A_id    -> Some ({}, not_contaminated)
NIC_B_id    -> Some ({}, contaminated)
NIC_C_id    -> Some ({}, not_contaminated)
NIC_D_id    -> Some ({}, not_contaminated)
```
High Level System Model

2. Security Architecture

3. Trusted Component Behaviour

4. Kernel Security Model

5. Formal Security Property

---

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM_id</td>
<td>Some ({rw_to_NIC_A, rw_to_NIC_B, \ldots}, not_contaminated)</td>
</tr>
<tr>
<td>SAC_C_id</td>
<td>Some ({rw_to_NIC_C, w_to_RM, \ldots}, not_contaminated)</td>
</tr>
<tr>
<td>TIMER_id</td>
<td>Some ({w_to_SAC_C, w_to_RM, \ldots}, not_contaminated)</td>
</tr>
<tr>
<td>ROUTER_id</td>
<td>None</td>
</tr>
<tr>
<td>NIC_A_id</td>
<td>Some ({}, not_contaminated)</td>
</tr>
<tr>
<td>NIC_B_id</td>
<td>Some ({}, contaminated)</td>
</tr>
<tr>
<td>NIC_C_id</td>
<td>Some ({}, not_contaminated)</td>
</tr>
<tr>
<td>NIC_D_id</td>
<td>Some ({}, not_contaminated)</td>
</tr>
</tbody>
</table>
UNTRUSTED prg ≡ [ AnyLegalOperation ]

RM prg ≡
[ (* 00: Wait for command, delete Router. *)
  SysOp (SysRead cap_R_to_SAC_C),
  SysOp (SysRemoveAll cap_C_to_R),
  SysOp (SysDelete cap_C_to_R),
  SysOp (SysWriteZero cap_RW_to_NIC_D).
  ...
(* 09: Non-deterministic “goto” *)
  Jump [0, 10, 19],
(* 10: Setup Router between NIC-A and NIC-D *)
  SysOp (SysCreate cap_C_to_R),
  SysOp (SysNormalWrite cap_RWGC_to_R),
  ...
]
High Level System Model

- Security Architecture
- Trusted Component Behaviour
- Kernel Security Model
- Formal Security Property

### What operations do user system calls perform?

\[
\text{step state } e \text{ (SysRead } c) = \text{ write-operation (entity } c) e \text{ state}
\]

### When is a system call allowed by the kernel?

\[
\text{legal } s \text{ e (SysRead } cap) = \\
\text{(is-entity s e} \wedge \text{is-entity s (entity cap)} \wedge cap \in \text{entity_caps_in_state s e} \wedge \text{Read } \in \text{rights cap)}
\]

### What effect do system calls have?

\[
\text{write-operation source target ss } = \\
\text{(case (ss target) of} \\
\text{Some target_entity } \Rightarrow \\
\text{ss(target } \to \text{ target_entity(} \\
\text{contaminated } := \\
\text{is_contaminated ss target} \vee \text{is_contaminated ss source)} \\
\text{l } _{-} \Rightarrow \text{ss})
\]
High Level System Model

- Security Architecture
- Trusted Component Behaviour
- Kernel Security Model
- Formal Security Property

Diagram:
- Control Network
  - Nic-C
  - SAC Controller
  - Router Manager
  - Timer Server
- Classified Networks
  - Nic-A
  - Nic-B
  - Router
  - Nic-D
- User Terminal
High Level System Model

1. Security Architecture
2. Trusted Component Behaviour
3. Kernel Security Model
4. Formal Security Property

Control Network
- Nic-C
  - SAC Controller
  - Router Manager
  - Timer Server

Classified Networks
- Nic-A
- Nic-B
- Router
- Nic-D

User Terminal
High Level System Model

- Security Architecture
- Trusted Component Behaviour
- Kernel Security Model
- Formal Security Property
High Level System Model

② Security Architecture
③ Trusted Component Behaviour
④ Kernel Security Model
⑤ Formal Security Property

Control Network
- Nic-C
  - SAC Controller
  - Router Manager
  - Timer Server

Classified Networks
- Nic-A
- Nic-B
- Nic-D

User Terminal
High Level System Model

② Security Architecture

③ Trusted Component Behaviour

④ Kernel Security Model

⑤ Formal Security Property

Control Network

Classified Networks

Nic-C

SAC Controller

Router Manager

Nic-A

Timer Server

Nic-B

Nic-D

User Terminal
Theorem: \( \text{sac\_is\_secure}: (\text{SAC\_startup} \to \text{ss}) \Rightarrow \neg \text{is\_contaminated} (\text{sac\_entity\_ss}) \text{NicA} \)
theorem sac_is_secure:
(SAC-startup \to* ss) \implies \neg is_contaminated (sac-entity-ss) NicA
High Level System Model

 theorem sac_is_secure:
  (SAC-startup →* ss) → ¬ is_contaminated (sac-entity-ss) NicA

② Security Architecture

③ Trusted Component Behaviour

④ Kernel Security Model

⑤ Formal Security Property
Progress

1. Formal Cap Distribution
2. Security Architecture
3. Trusted Component Behaviour
4. Kernel Security Model
5. Formal Security Property
Progress

- Components
- seL4 kernel
- Hardware

1. Formal Cap Distribution
2. Security Architecture
3. Trusted Component Behaviour
4. Kernel Security Model
5. Formal Security Property
Progress

Components

seL4 kernel

Hardware

① Formal Cap Distribution

② Security Architecture

③ Trusted Component Behaviour

④ Kernel Security Model

⑤ Formal Security Property
Progress

1. Formal Cap Distribution
2. Security Architecture
3. Trusted Component Behaviour
4. Kernel Security Model
5. Formal Security Property
Progress

Components
seL4 kernel
Hardware

1. Formal Cap Distribution
2. Security Architecture
3. Trusted Component Behaviour
4. Kernel Security Model
5. Formal Security Property
Conclusion

• Full system verification of modern systems infeasible
  – But verification of specific, targeted properties feasible

• Presented a framework for proving security
  – Break code into components, avoid needing to trust the bulk of our functionality
  – Formally verify components capable of violating desired property

• Built SAC as a case-study
  – Uses seL4 microkernel as a secure foundation
  – Showed a model of the system is secure

• Ongoing work is to join security model with existing seL4 proof
Thank You