Experiments on automation of formal verification of devices at the binary level

Thomas Lacroix

INSA Lyon
Soutenance de PFE (Option R&D)

19/06/2019

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Section 1

Motivation

Table of Contents

Security critical systems

Privacy

- Smartphones
- Smart TVs

Integrity

- Hospital equipment
- Traffic control systems
- Power plants

Security critical systems

Privacy

- Smartphones
- Smart TVs

Integrity

- Hospital equipment
- Traffic control systems
- Power plants

Problem: complex systems almost always contain bugs

Security critical systems - vulnerable

Security critical systems - vulnerable

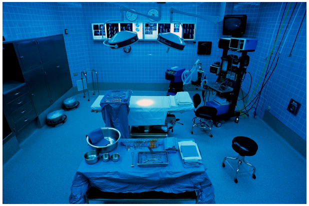


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

Security critical systems - vulnerable

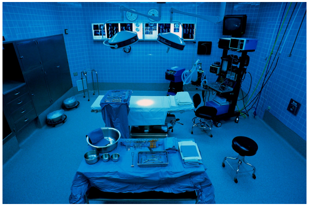


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

- Remote control of equipment

Security critical systems - vulnerable

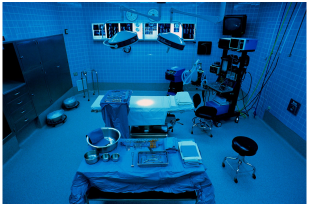


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

Figure: "Remote Exploitation of an Unaltered Passenger Vehicle" [1, 2]

- Remote control of equipment

Security critical systems - vulnerable

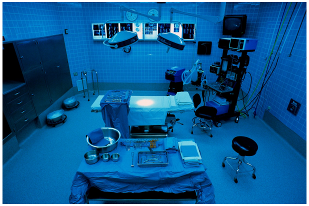


Figure: "It's Insanely Easy to Hack Hospital Equipment" [4]

Figure: "Remote Exploitation of an Unaltered Passenger Vehicle" [1, 2]

- Remote control of equipment
- Total control of drive system

Secure operating systems

seL4

Security. Performance. Proof.

[1] https://sel4.systems/Info/FAQ/proof.pml

Secure operating systems

Formal proof[1]:

- The binary code correctly implements its **abstract specification**.
- The specification guarantees **integrity** and **confidentiality**.

[1] `https://sel4.systems/Info/FAQ/proof.pml`

Secure operating systems

Security. Performance. Proof.

Formal proof[1]:

- The binary code correctly implements its **abstract specification**.
- The specification guarantees **integrity** and **confidentiality**.

- **Integrity**: data cannot be *changed* without permission.
- **Confidentiality**: data cannot be *read* without permission.

[1] `https://sel4.systems/Info/FAQ/proof.pml`

Secure operating systems

Proof assumptions[2]:

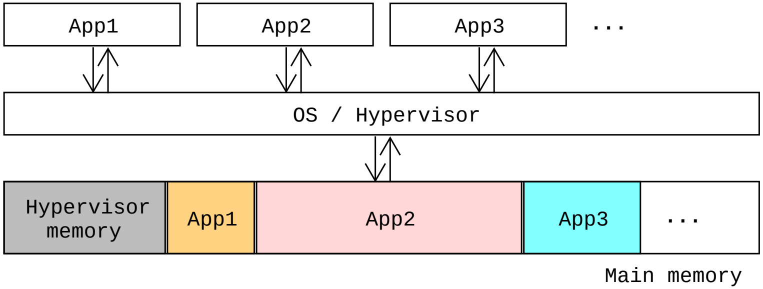
2 https://docs.sel4.systems/FrequentlyAskedQuestions#is-sel4-proven-secure

Secure operating systems

Proof assumptions[2]:

- Use of Direct Memory Access (DMA) is excluded, or only allowed for **trusted drivers that have to be formally verified by the user**.

[2] `https://docs.sel4.systems/FrequentlyAskedQuestions#is-sel4-proven-secure`


App1
App2
App3
...
OS / Hypervisor
Hypervisor memory
App1
App2
App3
...
Main memory

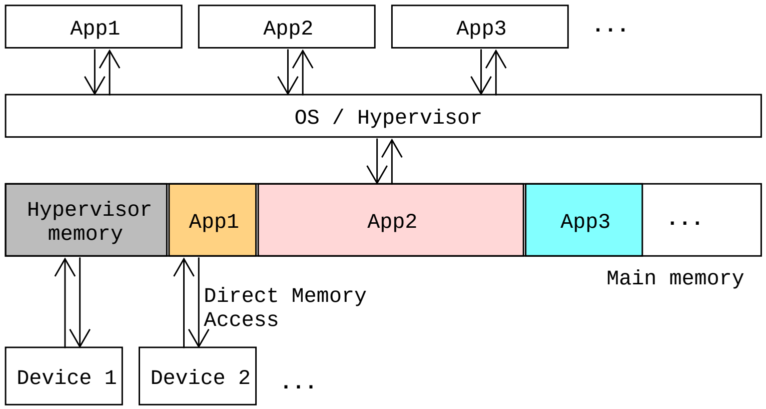

App1
App2
App3
...
OS / Hypervisor
Hypervisor memory
App1
App2
App3
...
Main memory
Direct Memory Access
Device 1
Device 2
...

Table of Contents

System software verification

Objective: show absence of errors in modelisation of real systems

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof

machine checkable proofs using
rigorous semantic

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof

machine checkable proofs using rigorous semantic

Non proof-producing verification

specialized programs or procedures that check a given property

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof

machine checkable proofs using rigorous semantic

Use small reliable kernels
$\rightarrow$ produced theorems are trustworthy

Non proof-producing verification

specialized programs or procedures that check a given property

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof

machine checkable proofs using rigorous semantic

Use small reliable kernels
$\rightarrow$ produced theorems are trustworthy

Non proof-producing verification

specialized programs or procedures that check a given property

Classic bug-prone software
$\rightarrow$ need tests, less trustworthy

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof

machine checkable proofs using rigorous semantic

Use small reliable kernels
$\rightarrow$ produced theorems are trustworthy

Examples: HOL4, Coq, Isabelle

Non proof-producing verification

specialized programs or procedures that check a given property

Classic bug-prone software
$\rightarrow$ need tests, less trustworthy

System software verification

Objective: show absence of errors in modelisation of real systems

Formal proof	**Non proof-producing verification**
machine checkable proofs using rigorous semantic	specialized programs or procedures that check a given property
Use small reliable kernels $\rightarrow$ produced theorems are trustworthy	Classic bug-prone software $\rightarrow$ need tests, less trustworthy
Examples: HOL4, Coq, Isabelle	SMT solvers, model checkers

Table of Contents

Network Interface Controller (NIC)

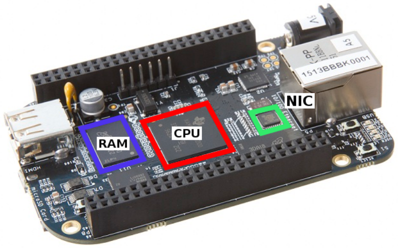
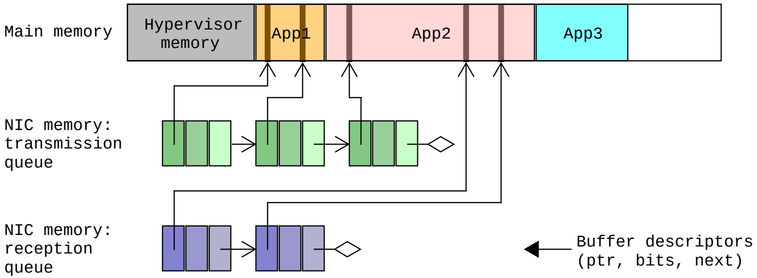
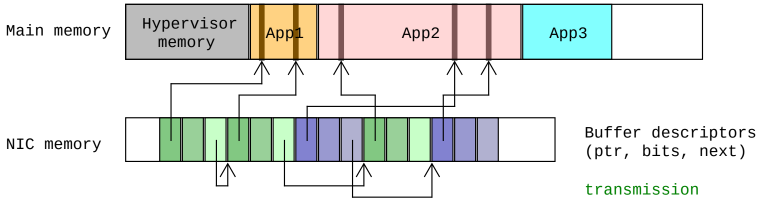


Figure: BeagleBone Black.

Network Interface Controller (NIC)


Main memory
Hypervisor memory
App1
App2
App3
NIC memory: transmission queue
NIC memory: reception queue
Buffer descriptors (ptr, bits, next)

Network Interface Controller (NIC)

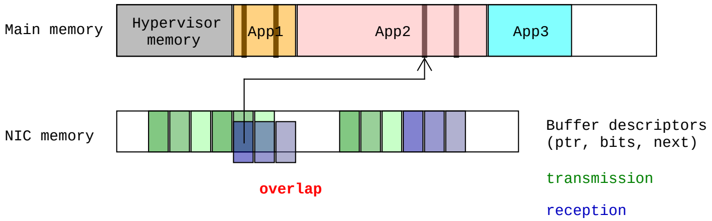


Buffer descriptors
(ptr, bits, next)

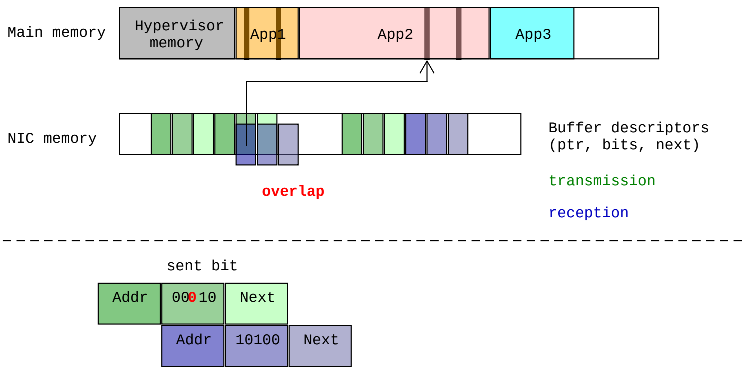
transmission

reception

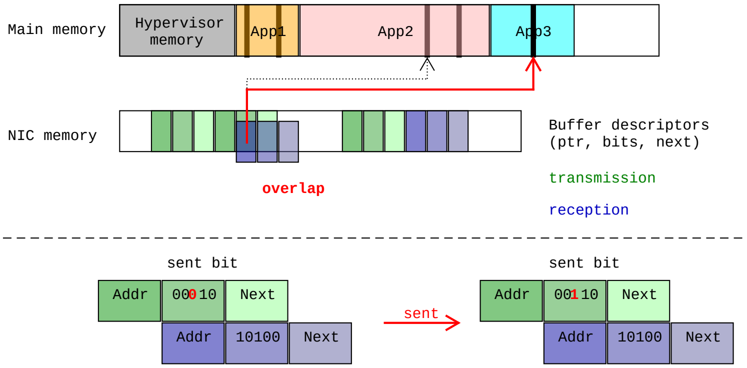
Network Interface Controller (NIC)



Network Interface Controller (NIC)



Network Interface Controller (NIC)



Research question

Can we apply traditional software verification techniques and tools to show security properties of hardware devices?

Section 2

Automatic contract-based verification pipeline

Table of Contents

Title

- Transition system
- Loop-free
- Verification = prove invariants
- (Show CFG?)

Table of Contents

Title

HT - WP -> SMT

Title

Example: invariant

Table of Contents

Title

Section 3

Proof-producing verification

Table of Contents

Title

Section 4

Conclusion

Questions

References I

Andy Greenberg.
Hackers remotely kill a jeep on the highway—with me in it.

Dr Charlie Miller and Chris Valasek.
Remote exploitation of an unaltered passenger vehicle.
page 91.

Thomas Tuerk.
Interactive theorem proving (ITP) course.

Kim Zetter.
It's insanely easy to hack hospital equipment.