Specifying and Verifying Hardware-based Security Enforcement mechanisms

Thomas Letan

Laboratoire Sécurité du Logiciel (LSL)
ANSSI

January 30, 2020

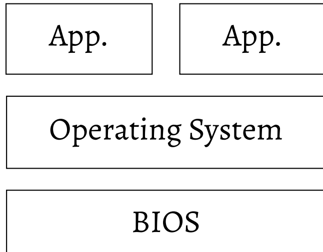
Specifying and Verifying Hardware-based Security Enforcement mechanisms

Thomas Letan

Laboratoire Architectures Matérielles et logicielles (LAM)
ANSSI

January 30, 2020

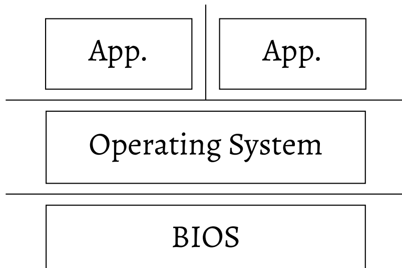
Security of Low Level Software Components



Software components of

- Various origin
- Various quality
- Various level of trust

Security of Low Level Software Components



Software components of

- Various origin
- Various quality
- Various level of trust

Each layer has to protect itself against upper (untrusted) layers

Hardware-based Security Enforcement Mechanisms

Trusted Software Components
configure
Hardware Components
to constrain
Untrusted Software Components
wrt.
Security Policies

Hardware-based Security Enforcement Mechanisms

Trusted Software Components
configure
Hardware Components
to constrain
Untrusted Software Components
wrt.
Security Policies

Operating System
sets up
Page Tables and Rings
to constrain
End-user Applications
to remain
Inside a Sandbox

The BIOS

The BIOS is provided by the hardware manufacturer

Boot sequence: Initialize the hardware platform

Runtime: Keep the platform in a working state

From a security perspective

- Shall continue to operate even in presence of a compromised software stack
- Most privileged software components of the stack

BIOS is the Root of Trust

BIOS HSE Mechanism

SMM	The most privileged x86 operating mode
SMRAM	Dedicated memory region of the DRAM protected by the Memory Controller
SMI	Hardware, non-maskable interrupt

BIOS HSE Mechanism

SMM
: The most privileged x86 operating mode

SMRAM
: Dedicated memory region of the DRAM protected by the Memory Controller

SMI
: Hardware, non-maskable interrupt

SMRAM : Integrity and Confidentiality

SMI : Availability

Selection of SMM Vulnerabilities

Prior 2008	Incorrect configuration of `SMRAMC`
2009	SMRAM Cache Poisoning Attack
2014	Sinkhole
2015	Speed Racer, `SENTER` Sandman
Until today	Incorrect configuration of `BIOS_CNTL`

Selection of SMM Vulnerabilities

2009 SMRAM Cache Poisoning Attack

2014 Sinkhole

2015 Speed Racer, SENTER Sandman

Compositional Attacks

- Only legitimate (wrt. specifications) use of hardware features
- Flaws in hardware specifications

Selection of SMM Vulnerabilities

Prior 2008 Incorrect configuration of `SMRAMC`

Until today Incorrect configuration of `BIOS_CNTL`

Misconfiguration Vulnerabilities

- Hardware features are available
- But are not correctly used by manufacturers

Challenges of HSE Mechanisms

A HSE mechanism implementation is correct if

- Hardware components expose sound APIs
- Trusted software components correctly use them

In between several verification domains

- Hardware verification
- Machine code verification
- System software verification

Goal: Formal Specifications

Formal specification and verification of HSE mechanisms to address

1. Compositional attacks
2. Hardware misconfiguration

Software Developers Perspective

- Unambiguous list of requirements
- Focus on security

Hardware Designers Perspective

- Foundation of a verification process

Contributions

A theory of HSE mechanisms to formally specify and verify them

- *SpecCert: Specifying and Verify Hardware-based Security Enforcement Mechanisms*, Formal Methods 2016
- Proof of correctness of the HSE mechanism implemented by x86 BIOSes at runtime in Coq

A Compositional Verification Framework

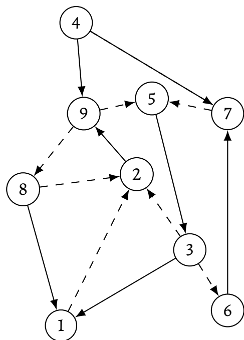
- *Modular Verification of Program with Effects and Effect Handlers in Coq*, Formal Methods 2018
- FreeSpec, a general-purpose framework for the Coq theorem prover

Agenda

A Theory of HSE Mechanisms

A Compositional Verification Framework

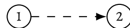
Prerequisite: Hardware Model

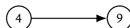


Hardware model as a Labeled Transition System

- Set of states

  Registers values, RAM content

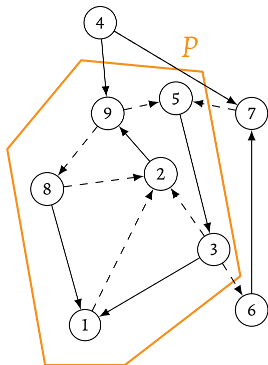
- Set of labeled transitions
  - Software transitions 

    ISA semantics
  - Hardware transitions 

    Hardware interrupts

$$\Sigma \triangleq \langle H, L_H, L_S, T\rangle$$

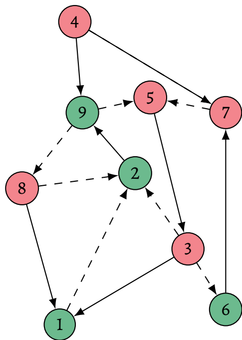
Step 1. Security Policy



Goal: Enforcing a security policy

- Safety and liveness properties
- $P \subseteq \mathcal{R}(\Sigma)$, where $\mathcal{R}(\Sigma)$ the set of traces of Σ

Step 2. HSE Mechanism



- S the set of software components

  $S = \{\texttt{bios}, \texttt{os}, \texttt{app}_1, \texttt{app}_2\}$

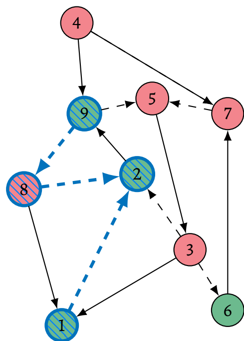
- $T \subseteq S$ the subset of trusted software components **which implements the HSE mechanism**

  $T = \{\texttt{bios}\}$ $\qquad U = S \backslash T$

- $context : H \rightarrow S$ to determine which software component in executed in a given state

  $context(s) = \texttt{bios}$ *iff the core is in SMM*

Step 2. HSE Mechanism



List of requirements

- over states: safe hardware configurations

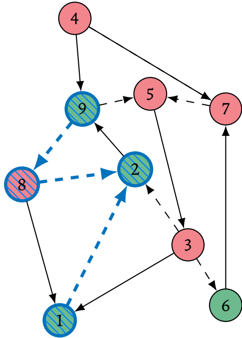
  The SMRAM has to be locked

- over software transitions: safe software executions

  Do not execute code outside of the SMRAM

$$\Delta \triangleq \langle S, T, context, state_req, trans_req \rangle$$

HSE Laws



Attacker Model

We do not make hypotheses about the behavior of untrusted softwre components

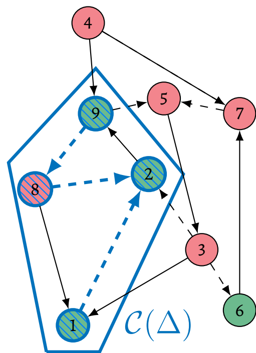
$$\forall (h, l) \in H \times L_S, \\ context(h) \notin T \Rightarrow trans_req(h, l)$$

Requirements consistency

Requirements over transitions preserves requirements over states

$$\forall (h, l, h') \in \mathcal{T}(\Sigma), \\ state_req(h) \\ \wedge (l \in L_S \Rightarrow trans_req(h, l)) \\ \Rightarrow state_req(h')$$

Compliant Traces



A trace $\rho \in \mathcal{R}(\Sigma)$ complies with Δ when

- Its initial state satisfies the requirements over states

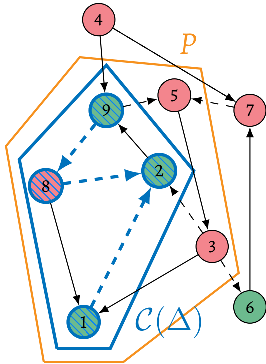
$$state_req(init(\rho))$$

- Trusted software components satisfy the requirements over software transitions

$$\forall (h, l, h') \in trans(\rho), \\ l \in L_S \Rightarrow trans_req(h, l)$$

$\mathcal{C}(\Delta)$ is the set of compliant traces

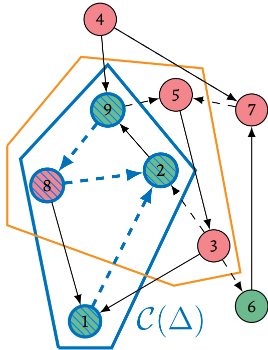
Step 3. Correctness



Implementing Δ is a sufficient condition in order to enforce P

$$\forall \rho \in \mathcal{C}(\Delta), \rho \in P$$

Step 3. Correctness



Implementing Δ is a sufficient condition in order to enforce P

$$\forall \rho \in \mathcal{C}(\Delta), \rho \in P$$

Typically, compositional attacks will prevent to conclude the correctness proof.

BIOS HSE Mechanism Overview

Requirements over states

- PC contains a SMRAM address if in SMM
- SMBASE contains a SMRAM address
- SMRAM contains code owned by the BIOS
- SMRAMC register has been locked
- SMRR registers are correctly configured
- Cached SMRAM is owned by the BIOS

Requirements over transitions

When CPU is in SMM (BIOS)

- Do not modify the SMRR
- Do not jump outside of the SMRAM

Code Injection Policy

When a CPU in SMM fetches an instruction, this instruction is owned by the BIOS

SpecCert Implementation

- 2 000 lines of definitions, 2 500 lines of proofs
- 150 lemmas and theorems
- Available as a free software (CeCILL-B)

`https://github.com/lthms/SpecCert`

Lessons Learned

Our theory allows for

- Reasoning about hardware/software co-designs
- Without modeling software components

A hardware model remains mandatory

- Practicable
- Reusable

Agenda

A Theory of HSE Mechanisms

A Compositional Verification Framework

Goal: Compositional Reasoning

Ease the reasoning about component-based systems

Our contributions is twofold

- Modular verification approach for component-based systems
  - In isolation *and* in composition
- An implementation of this approach in Coq
  - Components as programs with effects (implemented using a Free monad)

Interface Contracts

Given an Interface, we define two classes of requirements

PRECONDITION Which operations can be used at a given time?

POSTCONDITION What guarantee to expect from their result?

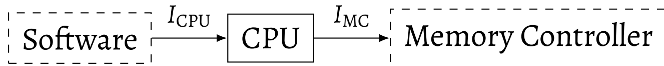
$$\text{PRECONDITION} \Rightarrow \text{POSTCONDITION}$$

Illustration of the Proposed Formalism

CPU

Illustration of the Proposed Formalism

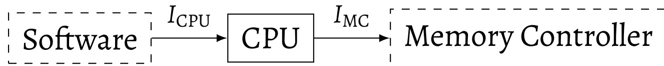
The CPU executes a **Software Component** thanks to a **Memory Controller**.



$$\frac{I_{\mathrm{CPU}} \xrightarrow{CPU} I_{\mathrm{MC}}}{}$$

Illustration of the Proposed Formalism

We want to prove the CPU complies with a couple of pre and postconditions $(\mathbb{P}, \mathbb{Q})$.

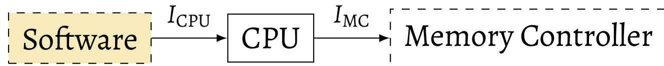


$$\frac{I_{\text{CPU}} \xrightarrow{\text{CPU}} I_{\text{MC}}}{\mathbb{P}}$$

$$\mathbb{Q}$$

Illustration of the Proposed Formalism

We assume the Software Component will enforce the precondition $\mathbb{P}$.

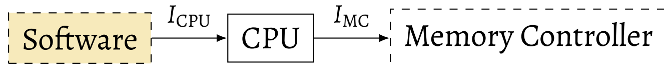


$$\frac{I_{\text{CPU}} \xrightarrow{CPU} I_{\text{MC}}}{\mathbb{P}}$$

$\mathbb{Q}$

Illustration of the Proposed Formalism

We want to verify that, according to the CPU model, the postcondition $\mathbb{Q}$ holds.



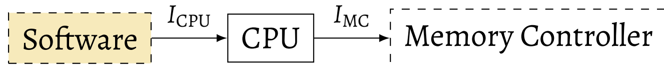
$$\frac{I_{\text{CPU}} \xrightarrow{CPU} I_{\text{MC}}}{\mathbb{P}}$$

$$\Downarrow$$

$$\mathbb{Q}$$

Illustration of the Proposed Formalism

Rather than relying on the Memory Controller model, we'd rather identify a sufficient couple $(\mathbb{P}', \mathbb{Q}')$, and abstract away the MCH.

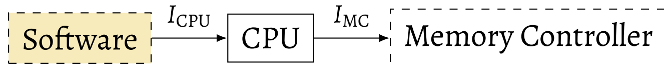


$$\frac{I_{\mathrm{CPU}} \quad \xrightarrow{CPU} \quad I_{\mathrm{MC}}}{\mathbb{P} \qquad\qquad \mathbb{P}'}$$

$$\mathbb{Q} \qquad\qquad \mathbb{Q}'$$

Illustration of the Proposed Formalism

We prove that, because $\mathbb{P}$ is assumed and thanks to the CPU model, then the assumptions $\mathbb{P}'$ are met.

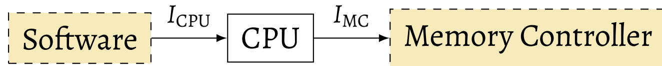


$$\frac{I_{\mathrm{CPU}} \quad \xrightarrow{CPU} \quad I_{\mathrm{MC}}}{\mathbb{P} \quad \Longrightarrow \quad \mathbb{P}'}$$

$$\mathbb{Q} \qquad\qquad \mathbb{Q}'$$

Illustration of the Proposed Formalism

We assume the Memory Controller enforces the postcondition $\mathbb{Q}'$.



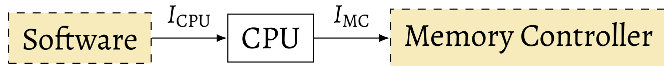
$$\frac{I_{\text{CPU}} \xrightarrow{CPU} I_{\text{MC}}}{\mathbb{P} \Longrightarrow \mathbb{P}'}$$

$$\mathbb{P}' \Downarrow \mathbb{Q}'$$

$$\mathbb{Q} \qquad \mathbb{Q}'$$

Illustration of the Proposed Formalism

We prove that, because $\mathbb{Q}'$ and thanks to CPU model, then $\mathbb{Q}'$ is verified.



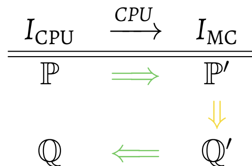
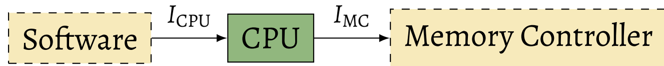


Illustration of the Proposed Formalism

In other words, our CPU model enforces $\mathbb{Q}$ as long as the Software component complies to $\mathbb{P}$.



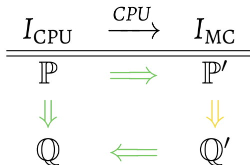


Illustration of the Proposed Formalism

We can then have a similar work with the Memory Controller model.

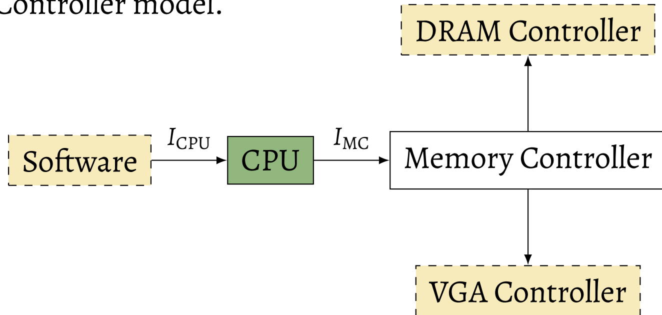


Illustration of the Proposed Formalism

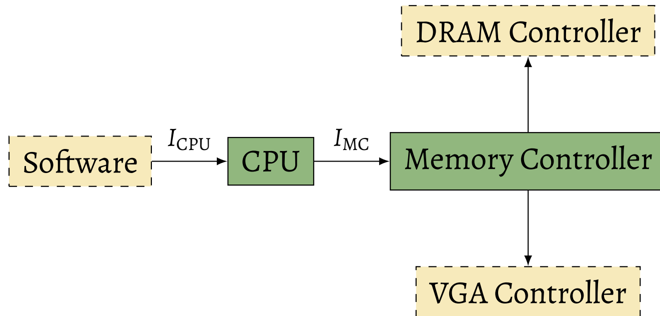


Illustration of the Proposed Formalism

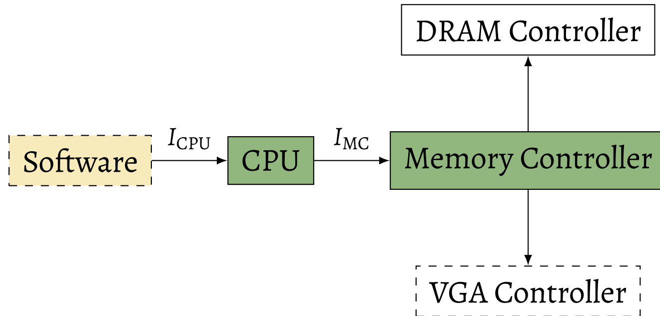


Illustration of the Proposed Formalism

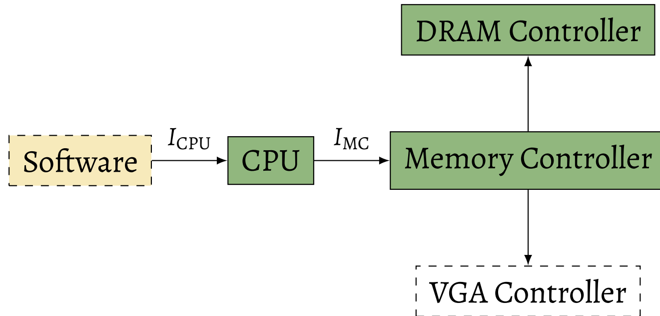
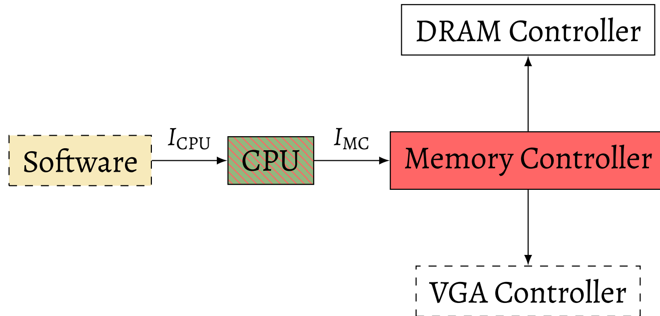


Illustration of the Proposed Formalism



FreeSpec

A general-purpose framework for implementing (with a Free monad) and certifying (with interface contracs) impure computations in Coq.

- A first iteration of the framework ("in-the-large")
- Verification of a simplified Memory Controller

Still an active research project (CPP'20)

`https://github.com/ANSSI-FR/coq-prelude`
`https://github.com/ANSSI-FR/FreeSpec`

- SpecCert: Specifying and Verify Hardware-based Security Enforcement Mechanisms
  Thomas Letan, Pierre Chifflier, Guillaume Hiet, Pierre Néron, Benjamin Morin, Formal Methods 2016
- Modular Verification of Program with Effects and Effect Handlers in Coq
  Thomas Letan, Yann Régis-Gianas, Pierre Chifflier, Guillaume Hiet, Formal Methods 2018

`https://github.com/lthms/SpecCert`
`https://github.com/ANSSI-FR/FreeSpec`

Questions?