Formal Modeling and Security Analysis for Intra-level Privilege Separation

Yinggang Guo, Zicheng Wang, Bingnan Zhong, Qingkai Zeng
State Key Laboratory for Novel Software Technology

ACSAC 2022
Background

• Trustworthy systems require intra-level privilege separation.

- Privilege-based domain isolation
- Secure domain switching
Background

Problems besides security:

- Performance overhead
- Semantic gap
- Hardware dependency

The privileged software cannot be stacked higher and higher.

✓ Intra-level privilege separation!

- Nested Kernel [ASPLOS’15]
- SKEE [NDSS’16]
- Hilps [NDSS’17]
- SelMon [MobiSys’20]
Problems

The lack of formalization has several consequences:

1. *For system designers*
   - cannot formally state security guarantees of complex systems
     - ✓ Design error detection

2. *For system users*
   - cannot formally reason about potential threats and defenses
     - ✓ Attack scenario simulation
Problems

The lack of formalization has several consequences:

3. *For comparison*

- difficult to compare the security of different solutions
- difficult to evaluate potential improvements

✓ A general and extensible formal framework
Challenges

• Faithful abstraction of the intra-level privilege separation model
  ➢ complex system software, subtle hardware mechanisms
    ✓ Privilege-Centric Model (PCM)

• Standardized definitions of the security properties
  ✓ Security invariants based on the privilege differences

• Inherent difficulties in formal analysis and verification
  ➢ exhausting manual effort, state explosion, expressiveness ...
    ✓ A two-step verification strategy
Overview

Security Analysis

Model Checking

Model Checking

PRIV_CONFIG

SETS
TAG
MEM
MEM_PRIV
REG
REG_PRIV
...

CONSTANTS
mem_priv_diff
reg_priv_diff
...

PRIV_DOMAIN

VARIABLES
tag
mem
mem_priv
reg_priv
...

INVARIANT
...

INITIALISATION
...

OPERATIONS
Write(m)
P2N_switch()
N2P_switch()
...

transition function
\( \delta, \psi_{priv} \)

states

A

.:.

P-stack

B

.:.

N-stack

input statements
Threat Model

• Follow the common threat model in prior work
  ➢ The system software contains exploitable vulnerabilities

• No security assumptions for the normal domain

• The privileged domain is partially trusted
  ➢ Security contracts

• All hardware components are trusted
Formal Framework: Privilege-Centric Model

We define state $\sigma$ to be a valuation of all variables in Vars, including

- **tag**: \{ normal, gate, privileged \}

- **mem**: ( MEM $\times$ M_TAG $\times$ M_TYPE $\times$ M_PRIV $\times$ M_INTEGRITY )

- **regs**: $\text{addr\_regs} \cup \text{bit\_regs}$
  - **addr\_regs**: ( MEM $\times$ R_PRIV ), e.g. PC, SP, CR3, TTBR
  - **bit\_regs**: ( CONTROL_BIT $\times$ R_PRIV ), e.g. CR0, TTBCR

- **flags**: \{ interrupt_flag, ... \}
The transition function $\delta$ will be like:

$$
\delta(\sigma, stmt, \gamma_i) = \begin{cases} 
\{ (\sigma', \gamma'_i) \}, & \text{iff } \psi_{priv} \\
\emptyset, & \text{otherwise}
\end{cases}
$$

where $i = \begin{cases} 
1, & \text{if } \sigma.tag = \text{privileged} \\
2, & \text{if } \sigma.tag = \text{normal}
\end{cases}$

e.g. $\psi_{write} \triangleq \text{control}_{-}\text{bit}(wp) = \text{FALSE} \lor \text{write} \in m_{-}\text{priv}(m)$
Formal Framework: Security Properties

Security invariants for privilege separation:

1. \( \text{tag} = \text{privileged} \Rightarrow \text{MEM}_{\text{PRIV}}_{\text{DIFF}} \subseteq \text{MEM}_{\text{PRIV}}_{\text{SET}} \)
2. \( \text{tag} = \text{normal} \Rightarrow \text{MEM}_{\text{PRIV}}_{\text{SET}} \cap \text{MEM}_{\text{PRIV}}_{\text{DIFF}} = \emptyset \)
3. \( \text{tag} = \text{privileged} \Rightarrow \text{REG}_{\text{PRIV}}_{\text{DIFF}} \subseteq \text{REG}_{\text{PRIV}}_{\text{SET}} \)
4. \( \text{tag} = \text{normal} \Rightarrow \text{REG}_{\text{PRIV}}_{\text{SET}} \cap \text{REG}_{\text{PRIV}}_{\text{DIFF}} = \emptyset \)
5. \( \text{m}_{\text{tag}}(m) = \text{privileged} \Rightarrow \text{m}_{\text{integ}}(m) = \text{TRUE} \)
Formal Framework: Security Properties

Security contracts for the privileged domain:

C1. $\text{tag} = \text{privileged} \land \text{write}(m) \Rightarrow \text{execute} \notin m_{\text{priv}}(m)$

C2. $\text{tag} = \text{privileged} \land \text{write}(m) \Rightarrow m_{\text{tag}}(m) \neq \text{normal}$

C3. $\text{tag} = \text{privileged} \land \text{execute}(m) \Rightarrow m_{\text{tag}}(m) \neq \text{normal}$
Instantiation

B-method

- first-order logic + set theory
- highly expressive and readable
- suitable for modular modeling
- good tool support: Atelier B, ProB

```plaintext
1   MACHINE Untitled
2   SETS
3   | ID={aa, bb} |
4   CONSTANTS iv
5   PROPERTIES
6   | iv:ID |
7   VARIABLES xx
8   INVARIANT
9   | xx:ID |
10  INITIALISATION xx:=iv
11  OPERATIONS
12  | Set(yy) = PRE yy:ID THEN xx:= yy END |
13  END
```

The abstract machine structure of B-method
Instantiation

From developers

• Privilege differences
• Privilege switch \( (\psi_{priv}) \)

Customized invariants

Reusable

• Other definitions
• Operational semantics
Instantiation

N2P Switch (vice versa for P2N Switch)

1. Jump to the gateway
2. Turn off the interrupt
3. Change the privilege configuration and check again
4. Switch the stack
5. Jump to the privileged domain

For Nested Kernel (x86), clear CR0.WP to turn off write protection

For Hilps (ARM), reconfigure TCR_ELx.TxSZ to expand the virtual address
Model Checking

- Intel(R) Core(TM) i7-10700 with 16GB RAM
- Ubuntu-20.04.4, ProB-1.11.1

Table 1: Model Checking Results of Nested Kernel and Hilps

<table>
<thead>
<tr>
<th>Abstract Machine</th>
<th>States</th>
<th>Transitions</th>
<th>Time(s)</th>
<th>Memory(MB)</th>
<th>Deadlock</th>
<th>Invariant Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested Kernel</td>
<td>2554</td>
<td>4886</td>
<td>2.559</td>
<td>174.039</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Hilps</td>
<td>44546</td>
<td>105218</td>
<td>47.745</td>
<td>238.102</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>
Security Analysis

• Consequences of missing security contracts
  ➢ Invariant violations or state explosion

• Design error detection
  ➢ Memory: PT, IOPT, IDT, stack, …
  ➢ Register: CR0, CR3, IDTR, …

• Attack scenario simulation
  ➢ Attack surfaces: memory, register, control flow

• Security comparison
  ➢ *Hilps* is stricter on confidentiality
  ➢ Perform similarly against attacks
Security Analysis

• Consequences of missing security contracts
  ➢ Invariant violations or state explosion

• Design error detection
  ➢ Memory: PT, IOPT, IDT, stack, ...
  ➢ Register: CR0, CR3, IDTR, ...

• Attack scenario simulation
  ➢ Attack surfaces: memory, register, control flow

• Security comparison
  ➢ Hilps is stricter on confidentiality
  ➢ Perform similarly against attacks
Security Analysis

• Consequences of missing security contracts
  ➢ Invariant violations or state explosion

• Design error detection
  ➢ Memory: PT, IOPT, IDT, stack, ...
  ➢ Register: CR0, CR3, IDTR, ...

• Attack scenario simulation
  ➢ Attack surfaces: memory, register, control flow

• Security comparison
  ➢ Hilps is stricter on confidentiality
  ➢ Perform similarly against attacks
Security Analysis

• Consequences of missing security contracts
  ➢ Invariant violations or state explosion

• Design error detection
  ➢ Memory: PT, IOPT, IDT, stack, ...
  ➢ Register: CR0, CR3, IDTR, ...

• Attack scenario simulation
  ➢ Attack surfaces: memory, register, control flow

• Security comparison
  ➢ *Hilps* is stricter on confidentiality
  ➢ Perform similarly against attacks
Case Study

- Interrupt-execution attack
- Jump-to-the-middle attack

<table>
<thead>
<tr>
<th>No.</th>
<th>Attack Target</th>
<th>Attack Scenario</th>
<th>Processed States (All Known States/Transitions)</th>
<th>Time(s)</th>
<th>Memory(MB)</th>
<th>Invariant Violations</th>
<th>Operation History</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Switch gate</td>
<td>Malicious interrupt bypassing the stack switch.</td>
<td>142(265/362)</td>
<td>0.340</td>
<td>169.737</td>
<td>I1 I3</td>
<td>1.Init(); 2.P2N_switch(); 3.Interrupt(); 4.Ret(); 1.Init(); 2.P2N_switch(); 3.Interrupt(); 1.Init(); 2.P2N_switch(); 3.Set_cr0(wp, false); 4.Jump(n_code);</td>
</tr>
<tr>
<td>6</td>
<td>Switch gate</td>
<td>Malicious interrupt bypassing privilege settings and checks.</td>
<td>64(1210/1282)</td>
<td>0.551</td>
<td>171.452</td>
<td>I2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Switch gate</td>
<td>Malicious jump to exploit privilege settings.</td>
<td>331(1517/1618)</td>
<td>0.744</td>
<td>171.874</td>
<td>I2</td>
<td></td>
</tr>
</tbody>
</table>
Case Study

The state space of Nested Kernel in Attack 5

The state space of Nested Kernel in the lack of C3
Conclusion

- A general and extensible formal framework
  - Privilege-Centric Model (PCM)
  - Security invariants based on privilege differences
- Nested Kernel and Hilps instantiations in B
- Security analysis based on model checking

Thank You!

Q & A

https://github.com/gyg128/Privilege-Centric-Model
Reach me at gyg@smail.nju.edu.cn